ELSEVIER

Theoretical Computer Science 185 (1997) 347–377

Theoretical Computer Science

Many-to-many routing on trees via matchings

Grammati E. Pantziou [a,1], Alan Roberts [b], Antonis Symvonis [b,*,2]

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

Abstract

In this paper we present an extensive study of many-to-many routing on trees under the matching routing model. Our study includes on-line and off-line algorithms. 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 that allows us to apply tools which were previously used exclusively in the analysis of hot-potato routing.

Keywords: Hot-potato routing; Matching routing model; Off-line routing; On-line routing; Packet routing

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 (uni-directional, bi-directional), 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. Usually, routing models are described informally.

Packet routing problems can be also classified based on the properties of the collection of packets that participate in the routing. When all packets are available at the

* Corresponding author. E-mail: symvonis@cs.su.oz.au.

[1] The work of Dr. Pantziou was partly supported by the EEC ESPRIT Projects GEPPCOM (contract No. 9072) and ALCOM IT.

[2] The work of Dr. Symvonis was supported by an ARC Institutional Grant.

PII S0304-3975(97)00049-2

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. Routing problems can be further classified. When each node of the graph 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 routing*) 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 $h_1 > 1$ and $h_2 = 1$ we have a *one-to-many* routing problem (*one* node sends packets to *many* other nodes); 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.*

Another classification of the routing problem is based on whether the routing decisions/actions are being made in a centralised or a distributed manner. The routing is said to be *on-line* when the routing actions of each node at a given time are based only on knowledge obtained from the packets that entered the node in previous routing steps. The routing is said to be *off-line* when a *routing schedule* is produced for each packet and then all packets are routed according to their produced schedules. The routing schedule of a packet consists of information which can be used to infer the node at which the packet resides at any time instance.

The *matching model* was defined by Alon et al. [2, 3] when they studied the routing of permutations. In the original 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 (a matching of graph G on which the routing takes place) can occur in a single routing step. These edges are said to be *active* during the routing step. Under the original matching model, 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. At that time, the routing is completed.

The importance of studying the routing using the matching routing model is twofold: Firstly, this routing problem can be considered as a formalisation of a mathematical problem related to the diameter of permutation groups. This becomes obvious if we consider an undirected graph G and a permutation π on its node set and let $rt(G, \pi)$ denote the minimum number of permutations σ_i whose product is π, where each σ_i is a product of disjoint transpositions on pairs of connected nodes. The *routing number* $rt(G)$ is the maximum value of $rt(G, \pi)$, where the maximum is taken over all permutations π. Secondly, from a practical point of view the striking feature of the matching model is that in contrast to the traditional "store-and-forward" approach, it involves no queueing of incoming packets. Additionally, routing on product graphs, including hypercubes and meshes, can be implemented in this model [6, 16, 19].

Most of the work available on the original matching model is devoted to off-line routing. Alon et al. [2, 3] showed that any permutation on a tree of n nodes can be routed in at most $3n$ steps. Roberts et al. [20] 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 [21] 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 [11] (see also [1, 16]). The odd–even transposition method sorts a permutation on a linear array of n elements in at most n steps.

Not allowing for the consumption of packets before the end of the routing limits the application of the original matching model to the routing of permutations. In this paper, we consider the natural extension of the original model which allows for the consumption of packets. Given the fact that this modification of the model is minimal, we continue to refer to the routing model as the *matching model*. (The term *matching with consumption model* was used in [18].) Since we allow for the consumption of packets at their destination, we have to assume that in the case where only one of the nodes at the endpoints of an edge holds a packet, a swap operation on that edge results in moving the packet to the opposite endpoint. Krizanc and Zhang [15] 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. The above method of packet injection into the routing satisfies the explicit requirement of the matching model according to which at most one packet is present at any node at any time instance. In the context of dynamic routing, an injection can be considered as the generation of a new packet. In practice, packets can be generated at any time instance and then they wait to be injected into the routing. In this paper, for simplicity we assume that packets are generated only at time instances in which their injection into the routing is possible.

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 the time 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 [4], no detailed and non-trivial analysis of their routing time was available until recently.

The work of Feige and Raghavan [10] 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 [5, 7, 13, 14, 17]. Borodin et al. [8] 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 [12] and Brassil and Cruz [9] for hypercubes.

1.1. Our results

In this paper, we present an extensive study of many-to-many packet routing on n-node trees under the matching routing model. We limit the investigation of the matching model to trees, however, the same results apply to undirected graphs since the routing can be performed on a spanning tree of the graph. Our study covers both on-line and off-line routing. More specifically:

(i) We show that there exists a family of permutation problems on n-node trees of maximum degree d which require $\Omega(dn)$ steps for their routing by any on-line routing algorithm which considers the edges incident to any tree node in a fixed order.

(ii) We provide an on-line algorithm which completes the routing of any many-to-many routing problem on an n node tree of maximum degree d in $d(k-1)+d\cdot dist$ routing steps where, k is the number of packets which participate in the routing and *dist* is the maximum distance some packet has to travel.

(iii) We provide an off-line algorithm which completes the routing of any many-to-many routing problem on an n node tree in $2(k-1)+dist$ routing steps where k is the number of packets which participate in the routing and *dist* is the maximum distance some packet has to travel. The algorithm routes any many-to-one routing problem in at most $3n-3$ routing steps and it significantly improves upon the previous algorithm of Krizanc and Zhang [15] which routed any many-to-one problem in $9n$ steps.

Another innovation of our work is that we establish a closed relationship between the matching and the hot-potato routing models and we exploit it in the analysis of the algorithms presented in this paper. More specifically, we show how our algorithms on tree T can be simulated by hot-potato routing on a graph G_T that is obtained through a transformation of tree T. This allows us to use tools that were developed for the analysis of hot-potato routing in our analysis of the matching routing.

The rest of the paper is organised as follows: In the next section we present terminology and preliminary results which are used in the paper. In Sections 3 and 4, we study on-line and off-line routing, respectively. We conclude in Section 5 with open problems.

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, i.e., nodes of V can be compared by the operator "$<$". 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 $dist_G(u,v)$ the distance (i.e., the length of the shortest path) from u to v on G.

A static routing problem $\mathscr{R}$ can be defined to be a tuple $\mathscr{R}=(G,S)$ 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 $orig(p) \neq dest(p)$. For dynamic routing problems, we have to augment the definition to incorporate the time at which a packet is generated. There are several ways to formalise the notion of the generation of a packet. For the purposes of this paper, in a dynamic routing problem, a packet p is considered to be a triple $p=(orig,dest,birth)$, where *birth* denotes the time that packet p is generated (*orig* and *dest* again denote the origin and the destination of packet p, respectively). The notation $birth(p)$ is also used to denote the time that p is generated. Note that, the time at which a packet is generated does not necessarily coincide with the time it is injected into the routing. A packet that is generated at time t, might be injected into the routing at a later time. We also allow set S to grow with time since, in general, it is not possible to specify before the start of the routing the number of packets that will be generated nor the specific times of the generation.

Both the matching and the hot-potato models assume a synchronous mode of communication. Thus, we can talk about the position of the packets at time t of the routing. At time $t=0$ the packets are in their initial position in the graph. We assume that routing steps occur instantly.

In the analysis of our algorithms for the matching model we are going to use the "*charging argument*" formulated by Borodin et al. [8] 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. (In Section 4, the definition of a *deflection* is extended).

Borodin et al. [8] 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 et al. [8]). *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). The analysis of our algorithm is based on the simulation of the matching routing on tree T by hot-potato routing on a graph G_T which is derived from T.

3.1. A lower bound

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

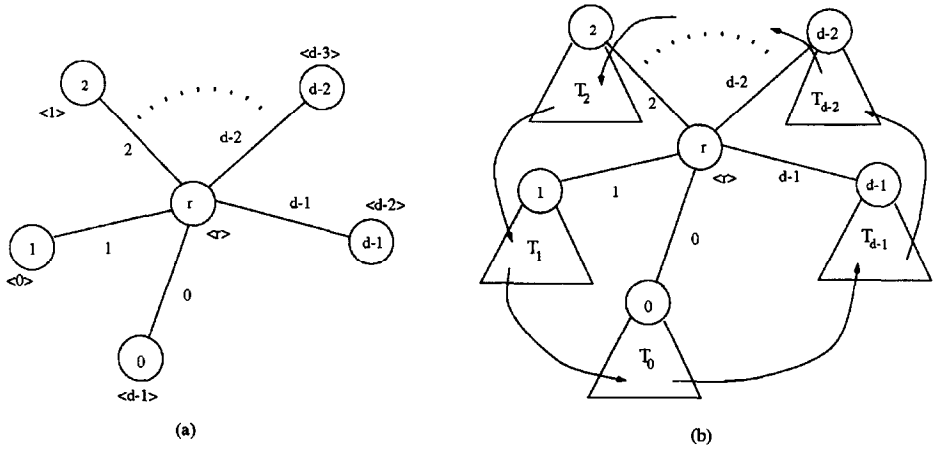

Fig. 1. Worst case permutations for (a) an n-node star of degree $d = n - 1$, and (b) a tree of maximum degree d, for some constant $d \geqslant 2$. The numbers in the nodes are node labels, the numbers attached to edges denote the order in which edges are activated and the numbers between angle brackets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

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. Observe that this is a natural class of algorithms for on-line routing. This is because it is not enough for a node in the graph to simply select the edge which will be active during the next step based on some criteria. The selections made by each node should also be compatible with the selections of other nodes, i.e., the set of active edges should form a matching.

Consider the permutation shown in Fig. 1(a) for a *star* of degree d (the number of nodes is $n = d + 1$). 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 Fig. 1(b). Each subtree T_i, $0 \leqslant i \leqslant d - 1$, has $(n - 1)/d$ nodes and the packets in subtree T_i have destinations in subtree $T_{(i-1) \bmod d}$, $0 \leqslant i \leqslant d - 1$. By using exactly the same argument,

we conclude that $\Omega(dn)$ steps are required for the completion of the routing. Thus, there exists a tree of maximum degree d, and a permutation on its nodes that requires $\Omega(dn)$ steps for its routing by any on-line algorithm (from the class studied).

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 (i.e., it reached its destination) it is consumed. Following, the consumption of the packet, if any, each node might inject a new packet into the routing. The packet injection has to be done after the consumption of packets that arrived at their destination, since we do not want a packet that will be consumed at the node it currently is to prevent the injection of a new packet at that step. In order to keep the description of our algorithm short and clear, we omit the code that deals with the consumption and injection of packets.

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 $T=(V,E)$ */
(i) [Preprocessing] For each node $v\in V$ label the edges incident on v with labels in $\{0,\ldots,d-1\}$, so that no two edges incident on v have the same label.
(ii) $t=0$
(iii) For each node $v\in V$ select a packet $p\in M$ (if any) with $orig(p)=v$ and inject it into the routing.
(iv) 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.[3] 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.

[3] The swap of the packets at the endpoints of an edge is executed in an on-line fashion. Each endpoint of the edge sends the packet it holds to the other endpoint while it keeps a copy of its packet. Then, both nodes execute the test in procedure *Update*, they both make consistent decisions on which packet to hold, and they discard the other.

Procedure *Update*(u,v)
(i) $u' = dest_T(packet(u))$
(ii) $v' = dest_T(packet(v))$
(iii) **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 a general charging scheme that is used for the analysis of hot-potato routing algorithms. Consider the routing problem $\mathscr{R} = (T,M)$ which is routed by algorithm *On-Line-Tree-Routing*. Based on $\mathscr{R} = (T,M)$ and algorithm *On-Line-Tree-Routing*, we define a routing problem $\mathscr{R}' = (G_T,H)$ and the hot-potato Algorithm *On-Line-Simulation* such that, the number of steps required for the routing of problem $\mathscr{R} = (T,M)$ by algorithm *On-Line-Tree-Routing* is a function of the number of steps required for the routing of problem $\mathscr{R}' = (G_T,H)$ by Algorithm *On-Line-Simulation*.

Consider a tree T of maximum degree d and let each edge in T be labelled with an integer $i \in \{0,\ldots,d-1\}$, 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, $j \in \{0,\ldots,d-1\}$, 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.[4] 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 $\{0,\ldots,d-1\}$ 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 Fig. 2.

Lemma 2. *Let x and y be two nodes of graph G_T. Then there exists a unique shortest path in G_T which connects nodes x and y.*

Proof. Follows from the construction of graph G_T (based on tree T) and the fact that there exists a unique shortest path between any pair of nodes of an undirected tree. □

[4] Since tree T is undirected, $\{u,v\}$ and $\{v,u\}$ denote the same edge. So, in G_T only one new node is introduced, that is, in G_T $\{u,v\}^i$ and $\{v,u\}^i$ denote the same node.

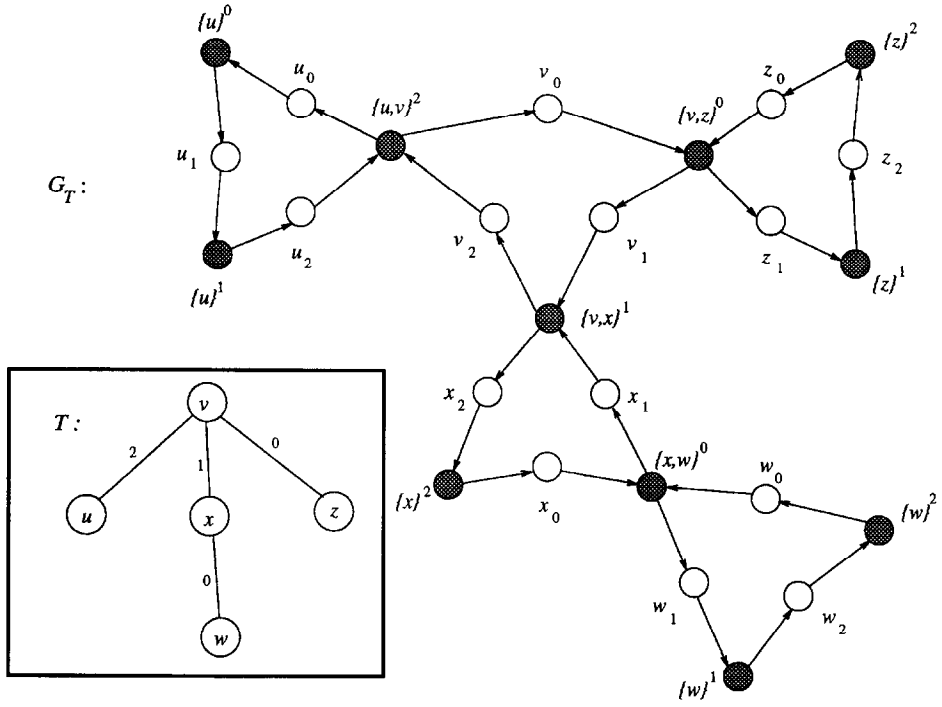

Fig. 2. Tree T and the corresponding graph G_T used in the analysis of Algorithm *On-Line-Tree-Routing*.

Lemma 3. *Let u and v be two neighbouring nodes in T. Then $dist_{G_T}(u_i, v_i) = 2d$ for all $i \in \{0, \ldots, d-1\}$.*

Proof. Let l be the label of edge $\{u,v\}$ of T. For an arbitrary $i \in \{0,\ldots,d-1\}$, the shortest path from u_i to v_i in G_T is made up of the paths $\Pi_1 = \langle u_i \to \cdots \to u_l \to \{u,v\}^l \rangle$ and $\Pi_2 = \langle \{u,v\}^l \to v_{(l+1) \bmod d} \to \cdots \to v_i \rangle$. Every second node of Π_1 (starting from u_i) corresponds to node u of T, while every second node of Π_2 (starting from $v_{(l+1) \bmod d}$) corresponds to node v of T. To see that the length of the combined path is equal to $2d$, observe that the subscripts of every second node of the combined path form the sequence $\langle i, (i+1) \bmod d, (i+2) \bmod d, \ldots, (i-1) \bmod d, i \rangle$. □

Lemma 4. *Let u, v be two distinct nodes in T. Then $dist_{G_T}(u_i, v_i) \leqslant 2d \cdot dist_T(u,v)$ for all $i \in \{0,\ldots,d-1\}$.*

Proof. We prove the lemma by induction on $dist_T(u,v)$. Lemma 3 forms the basis of the induction. Let $\delta > 1$ be the diameter of tree T (the case where $\delta = 1$ is covered by the basis of the induction). For the induction hypothesis assume that the lemma is true for any pair u, v of nodes which satisfy $dist_T(u,v) < \mu$, where $1 \leqslant \mu \leqslant \delta$. Consider any pair u, w of nodes with $dist_T(u,w) = \mu$ and let the shortest path from u to w in T be $\langle u \to \cdots \xrightarrow{l} v \xrightarrow{m} w \rangle$ (l and m are the labels of the edges incident to v in the path). Then,

for all $i \in \{0,\ldots,d-1\}$, we have that $dist_{G_T}(u_i,v_i) \leqslant 2d \cdot dist_T(u,v)$ (by the induction hypothesis) and $dist_{G_T}(v_i,w_i) = 2d$ (by Lemma 3). Thus, for all $i \in \{0,\ldots,d-1\}$, we conclude that $dist_{G_T}(u_i,w_i) \leqslant 2d(dist_T(u,v)+1) = 2d \cdot dist_T(u,w)$.

Note that the path $\langle u_i \to \cdots \to v_i \to \cdots \to w_i \rangle$ on which we based the proof of this lemma might not be a simple path. The path $\langle v_{(l+1) \bmod d} \to \cdots \to v_m \to \{v,w\}^m \rangle$ is a sub-path of the shortest path from u_i to w_i in G_T. If this sub-path does not go through v_i, then $dist_{G_T}(u_i,w_i) < 2d \cdot dist_T(u,v) + 2d$. This is how the "$<$" operator was introduced in the lemma. □

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 $\mathscr{R} = (T,M)$ is a many-to-one routing problem, that is, $|M| \leqslant n$ 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 $\mathscr{R}' = (G_T,H)$ 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 (recall 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 $\mathscr{R}' = (G_T,H)$. 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

Note that all the nodes in this class are of the form u_i (where $i \in \{0 \cdots d-1\}$ and u is a node of T) and have in-degree and out-degree equal to 1.

[*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

Note that all the nodes in this class are of the form u^i (where $i \in \{0 \cdots d-1\}$ is not the label of any edge incident to node u of T) and have in-degree and out-degree equal to 1. Moreover, no packet in H is destined for a node in this class.

[*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

Note that all the nodes in this class are of the form $\{u,v\}^i$ (where $i \in \{0 \cdots d-1\}$ is the label of edge $\{u,v\}$ of T) and have in-degree and out-degree equal to 2.

[*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 $v_{(i+1) \bmod d}$. (Note that the test on which node $\{u,v\}^i$ based its decision is identical to the test performed by Procedure *Update*. Even the distances are taken on tree T instead on G_T.)

Lemma 5. *Let the many-to-one routing problem $\mathscr{R} = (T,M)$ be routed by Algorithm "On-Line-Tree-Routing" and the many-to-one routing problem $\mathscr{R}' = (G_T,H)$ 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 if and only if packet p_h is consumed at time $2c$, and*

(ii) *at time t packet p_m is at node u if and only if at time $2t$ packet p_h is at node $u_{t \bmod d}$, $t \leqslant c$.*

Proof. We prove statement (ii) by induction on t. Statement (i) then follows from statement (ii).

The basis ($t = 0$) of statement (ii) is obviously correct. To see that, realise that at time $t = 2t = 0$ both packets p_m and p_h are located at their origin nodes in T and G_T, respectively. Let $u = orig(p_m)$. Then, by the construction of routing problem $\mathscr{R}'$, packet p_h originates at node $u_0 = u_{t \bmod d}$.

For the inductive step of statement (ii), we consider the routing steps before the consumption of packets p_m (for the "only if" part) and p_h (for the "if" part). For one step of Algorithm *On-Line-Tree-Routing* we follow two steps of Algorithm *On-Line-Simulation*.

By the induction hypothesis, at time $t < c$ (packet p_m is consumed at time c) of the matching routing, packet p_m is at node u if and only if at time $2t$ of the hot-potato routing, packet p_h is at node $u_{t \bmod d}$.

($\Rightarrow$) First consider the case where there does not exist an edge incident to node u of T with label t mod d. Then, at time $t+1$ packet p_m is still at node u. Thus, in this case, we have to show that at time $2(t+1) = 2t+2$ packet p_h is at node $u_{(t+1) \bmod d}$ of G_T. By the induction hypothesis, at time $2t$ packet p_h is at node $u_{t \bmod d}$. Node $u_{t \bmod d}$ has out-degree 1, and by rule *On-line-node-1*, at step $2t+1$ packet p_h is located at node $\{u\}^{t \bmod d}$. This is also a node of out-degree 1 and (by the construction of graph G_T) the out-going edge leads to node $u_{(t+1) \bmod d}$. Thus, by rule *On-line-label-1*, at step $2t+2$ packet p_h is located at node $u_{(t+1) \bmod d}$ of G_T.

Consider now the case where there exists an edge incident to node u with label (t mod d), say edge $\{u,v\}$. Let $q_m \in M$ be the packet in node v of T at time t and let $q_h \in H$ be its corresponding packet which participates in the hot-potato routing (the case where at time t no packet is at node v is simpler and can be handled similarly).

By the induction hypothesis, at time t packets p_h and q_h are in nodes $u_{t \bmod d}$ and $v_{t \bmod d}$, respectively. Both of these nodes have out-degree 1 and their out-going edges lead to node $\{u,v\}^{t \bmod d}$. According to rule *On-line-node-1*, both packets advance to that node and thus, at time $2t+1$ packets p_h and q_h are at node $\{u,v\}^{t \bmod d}$. This node has out-degree 2 and thus, the nodes at which packets p_h and q_h reside at time $2t+2$ are determined based on rule *On-line-edge-2*.

Let us return now to the matching routing of packets p_m and q_m which, at time t, are at nodes u and v, respectively. Let $u' = dest(p_m)$ and $v' = dest(q_m)$. Algorithm *On-Line-Tree-Routing* calls procedure *Update*(u,v). There are two cases to consider based on whether procedure *Update*(u,v) swapped packet p_m with packet q_m.

Case 1: A swap took place. As a result of the swap, at time $t+1$ packet p_m is at node v. Since the swap took place, it must hold that $dist_T(u,v') + dist_T(v,u') < dist_T(u,u') + dist_T(v,v')$ (see procedure *Update*). Consider now the hot-potato routing. Node $\{u,v\}^{t \bmod d}$ decides which node to forward packet p_h to based on the result of exactly the same comparison. In the case where $dist_T(u,v') + dist_T(v,u') < dist_T(u,u') + dist_T(v,v')$ packet p_h is forwarded to node $v_{(t+1) \bmod d}$. Thus, if at time $t+1$ packet p_m is at node v then, at time $2t+2 = 2(t+1)$ packet p_h is at node $v_{(t+1) \bmod d}$.

Case 2: A swap did not take place. As a result, at time $t+1$ packet p_m is still at node u. Because the swap did not take place, it must hold that $dist_T(u,v') + dist_T(v,u') \geqslant dist_T(u,u') + dist_T(v,v')$. In this case, node $\{u,v\}^{t \bmod d}$ (based on rule *On-line-edge-1*) forwards packet p_h to node $u_{(t+1) \bmod d}$. Thus, if at time $t+1$ packet p_m is at node u then, at time $2t+2 = 2(t+1)$ packet p_h is at node $u_{(t+1) \bmod d}$.

($\Leftarrow$) The "if" part of the proof is also based on the fact that rule *On-line-edge-2* performs exactly the same comparison with that performed by procedure *Update*. Since it is symmetric to the "only if" part, it is omitted.

Statement (i), that is, packet p_m is consumed at time c if and only if packet p_h is consumed at time $2c$, now follows from Statement (ii) and the fact that both Algorithm *On-Line-Tree-Routing* and Algorithm *On-Line-Simulation* consume packets as soon as they enter their destination node. □

Theorem 6. *Algorithm "On-Line-Tree-Routing" routes any many-to-one routing problem $\mathscr{R} = (T,M)$ 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. We consider the routing of problem $\mathscr{R}' = (G_T, H)$ by Algorithm *On-Line-Simulation*. From the construction of G_T (Lemma 4), it follows that the maximum distance that some packet of H has to travel in G_T is at most $2d \cdot dist$. Concentrate

on an arbitrary packet $p_m \in M$ and its corresponding packet $p_h \in H$. We prove the theorem by showing that p_h is consumed by time $2(d(k-1)+d \cdot dist)$. This implies (by Lemma 5) that packet p_m is consumed by time $d(k-1)+d \cdot dist$.

The number of packets in the constructed hot-potato routing problem $\mathscr{R}'$ is k, that is, equal to the number of packets of the matching routing problem $\mathscr{R}$. We show that each packet that participates in the hot-potato routing might be deflected at most $k-1$ times. The proof is directly based on Lemma 1, developed by Borodin et al. [8]. Thus, we have to specify how to construct the deflection sequence for each deflection of some packet $p_h \in H$.

Firstly observe that a deflection can take place only at nodes of G_T which correspond to edges of T, that is, nodes of the form $\{u,v\}^i$, where $\{u,v\}$ is an edge of T with label i. This is because only these nodes have in-degree (and out-degree) equal to 2.

Consider the deflection of packet p_h which was deflected from node $\{u,v\}^i$ to node $v_{(i+1) \bmod d}$ at time t by packet p_h^1. For the construction of the deflection sequence, we follow the packet that caused the deflection, i.e., packet p_h^1, until it reaches its destination or, until the first time that it is deflected by another packet, say p_h^2. We then follow p_h^2, and so on. The deflection sequence ends with a packet p_h^l that reaches its destination, i.e, it reaches node $dest(p_h^l)$.

In order to be able to apply Lemma 1, we have to show that the shortest path from node $v_{(i+1) \bmod d}$ to $dest(p_h^l)$ is at least as long as the deflection path.

This can be established by observing that the unique shortest path in G_T (Lemma 2) between nodes $v_{(i+1) \bmod d}$ and $dest(p_h^l)$ passes through node $\{u,v\}^i$. More specifically, the unique shortest path in G_T from $v_{(i+1) \bmod d}$ to $dest(p_h^l)$ consists of the path $\Pi_1 = \langle v_{(i+1) \bmod d} \rightarrow \cdots \rightarrow v_{i \bmod d} \rightarrow \{u,v\}^i \rangle$ (in which every second node, stating from $v_{(i+1) \bmod d}$, corresponds to node v of T) followed by the unique shortest path $\Pi_2 = \langle \{u,v\}^i \rightarrow \cdots \rightarrow dest(p_h^l) \rangle$. Since path Π_1 is non-empty (by construction, it has length $2d-1>0$) and path Π_2 is the deflection path, the shortest path from node $v_{(i+1) \bmod d}$ to node $dest(p_h^l)$ is actually longer than the deflection path.

Now, based on Lemma 1 we conclude that each packet of H is deflected at most $k-1$ times, i.e., as many times as the number of all other packets participating in the routing. Note that the cost of a deflection is $2d$, that is, a packet which is at node $\{u,v\}^i$ and is deflected to node $v_{(i+1) \bmod d}$ has to traverse all the edges of the circuit $\Pi_1 = \langle \{u,v\}^i \rightarrow v_{(i+1) \bmod d} \rightarrow \cdots \rightarrow v_{i \bmod d} \rightarrow \{u,v\}^i \rangle$ in order to return to node $\{u,v\}^i$ (the circuit has length $2d$).

Since the cost of each deflection is $2d$ steps and the maximum distance that some packet has to travel on G_T is $2d \cdot dist$ (Lemma 4), we conclude that Algorithm *On-Line-Simulation* finishes the routing of problem $\mathscr{R}' = (G_T, H)$ after at most $2(d(k-1)+d \cdot dist)$ steps. Then, the theorem follows from Lemma 5. □

3.3.2. *Many-to-many routing*

Consider a many-to-many routing problem $\mathscr{R} = (T, M)$ which is routed by algorithm *On-Line-Tree-Routing*. Based on the routing of $\mathscr{R} = (T, M)$ by Algorithm *On-Line-*

Tree-Routing, we define a dynamic routing problem $\mathscr{R}' = (G_T, H)$ which is again routed by Algorithm *On-Line-Simulation*.

The construction is similar to the one used for the analysis of the many-to-one routing. The only difference is that we do have to specify for each packet in H the time at which it is generated. We first route problem $\mathscr{R} = (T, M)$ 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 $\mathscr{R} = (T, M)$ terminates, we are ready to fully specify problem $\mathscr{R}' = (G_T, H)$. 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.

The following lemma establishes that each packet of H can be injected into the hot-potato routing at the time it is generated.

Lemma 7. *Consider the many-to-many routing problem $\mathscr{R} = (T, M)$ which is routed by Algorithm "On-Line-Tree-Routing" and the constructed dynamic routing problem $\mathscr{R}' = (G_T, H)$ 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 $2t$.*

Proof. We prove the lemma by induction on t. The lemma is trivially true for $t = 0$. This case corresponds to the many-to-one static routing problem.

Assume that the lemma holds for every time instance $t' < t$. Lemma 5 which was proved for the case of many-to-one routing, can be extended to cover all packets of the matching routing generated at time instances smaller or equal to t'. (The proof is identical and thus omitted.) We now consider the only two cases in which, at time t, Algorithm *On-Line-Tree-Routing* injects a new packet at node u:

Case 1: At time t, no packet which was generated at a time instance smaller that t is present at node u.

In this case, at time $2t$ no packet which participates in the hot-potato routing is at node $u_{t \bmod d}$ and thus, the creation of a new packet is possible.

Case 2: At time t, a packet p_m, which was generated at a time $t' < t$ is consumed at node u.

By the induction hypothesis, the corresponding packet p_h which participates in the hot-potato routing was generated at time $2t' < 2t$. In this case, packet p_h is consumed at time $2t$ at node $u_{t \bmod d}$. This is because Lemma 5 (in its extended version) holds. Thus, after packet p_h is consumed, the generation of a new packet is possible. □

Theorem 8. *Algorithm "On-Line-Tree-Routing" routes any many-to-many routing problem $\mathscr{R} = (T, M)$ 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 the extension of Lemma 5 to many-to-many routing and on Lemma 7 it is possible to handle the consumption and, more importantly the generation of new packets. We then apply the same argument (based on the deflection sequences) as in the proof of Theorem 6. The only difference is that now k is the number of packets that participated in the many-to-many routing. □

4. Off-line Routing

In the case where each node of the tree is the origin of at most one packet, i.e., in many-to-one problems, the output of an off-line algorithm can be considered to be a sequence of matchings (not necessarily maximal) on tree T. Each matching corresponds to the set of edges which swap the packets at their endpoints during the corresponding routing step.

In the case of many-to-many routing, each node of the tree might contain initially more than one packet. In our off-line algorithm, we assume that one of the packets at each node (if there are any) initially participates in the routing and the remaining packets are injected whenever possible. We assume no particular order when injecting the packets of the same node, however, it is trivial to generate the packets (of the same node) according to some ordering criteria. The output of the off-line algorithm is augmented to contain for each packet the time that it is injected into the routing.

4.1. The off-line algorithm

Consider an edge that has one packet at each endpoint and assume that both packets have to cross the edge in order to reach their destinations. The algorithm identifies all the edges of the tree that belong in this category and swaps the packets at their endpoints. Consider also a node of the tree that does not hold a packet and has some neighbours holding packets that have to enter it in order to reach their destinations. The algorithm also identifies all the nodes in this category and for each of them chooses arbitrarily one of the neighbours to forward the packet to it through the common edge.

In the formal description of 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 Fig. 3 consists of two in-trees rooted at nodes

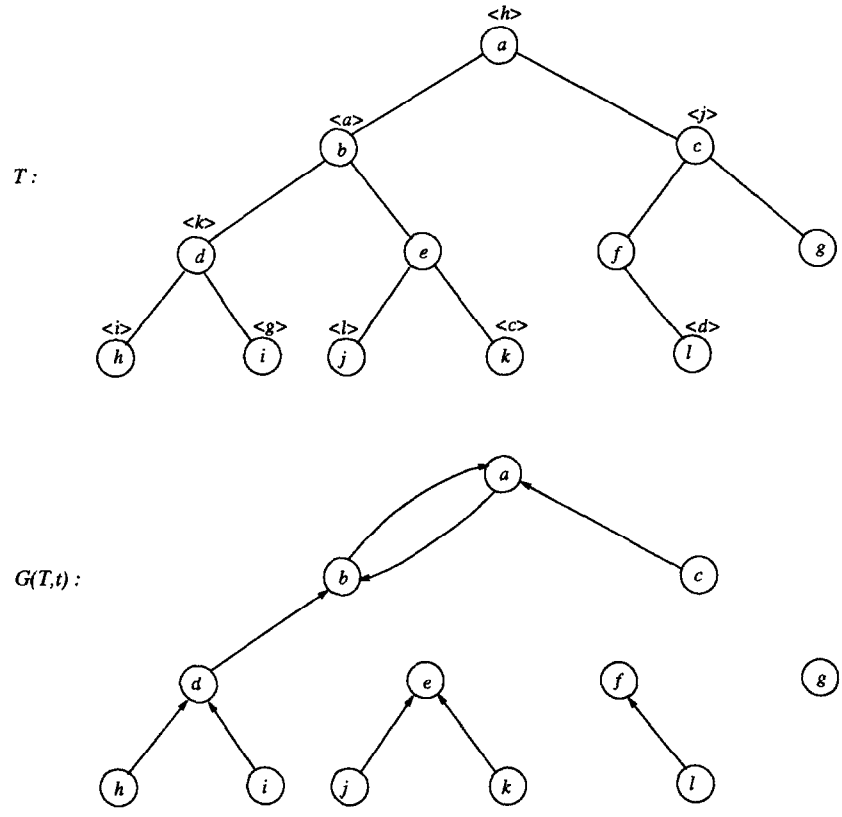

Fig. 3. Tree T at step t and the corresponding auxiliary graph $G(T,t)$.

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). Fig. 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 $T=(V,E)$ */
(i) $t=0$

(ii) For each node $v \in V$ select a packet $p \in M$ (if any) with $orig(p) = v$ and inject it into the routing.

(iii) While *there are packets that haven't reached their destination* do
  (a) Construct the auxiliary graph $G(T,t)$.
  (b) Denote by $\mathscr{S}$ be the set of tree edges which swap the packets at their endpoints during the next routing step. Insert into set $\mathscr{S}$:
    – 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 $\mathscr{S}$ is the one which corresponds to the selected edge of the in-tree.
  (c) Swap the packets at the endpoints of edges in $\mathscr{S}$.
  (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 Fig. 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 $\{f,l\}$.

Note that Algorithm *Off-Line-Tree-Routing* builds the routing schedules for the packets in M step by step. It is classified as an off-line algorithm because the decision on how to move a given packet at time t is based on information collected from all nodes of tree T. This information is represented by graph $G(T,t)$.

4.2. *Analysis of algorithm "Off-Line-Tree-Routing"*

For the analysis of Algorithm *Off-Line-Tree-Routing* we again employ elements of hot-potato routing. Consider the routing problem $\mathscr{R} = (T,M)$ which is routed by algorithm *Off-Line-Tree-Routing*. Based on $\mathscr{R} = (T,M)$ and algorithm *Off-Line-Tree-Routing*, we define a routing problem $\mathscr{R}' = (G_T,H)$ and a hot-potato Algorithm *Off-Line-Simulation* such that, the number of steps required for the routing of problem $\mathscr{R} = (T,M)$ by algorithm *Off-Line-Tree-Routing* is a function of the number of steps required for the routing of problem $\mathscr{R}' = (G_T,H)$ by Algorithm *Off-Line-Simulation*.

Given tree $T = (V,E)$, we construct the bipartite graph $G_T = (A,B,E_T)$ as follows:

$$A = \{v_A \mid v \in V\}, \qquad B = \{v_B \mid v \in V\}$$

$$E_T = \{(v_A, v_B),\ (v_B, v_A) \mid v \in V\} \cup \{(v_A, w_B),\ (w_B, v_A) \mid \{v,w\} \in E\}$$

Fig. 4 shows a tree T and its corresponding graph G_T. For clarity, a single edge with direction arrows at both of its endpoints is used instead of a pair of anti-parallel edges.

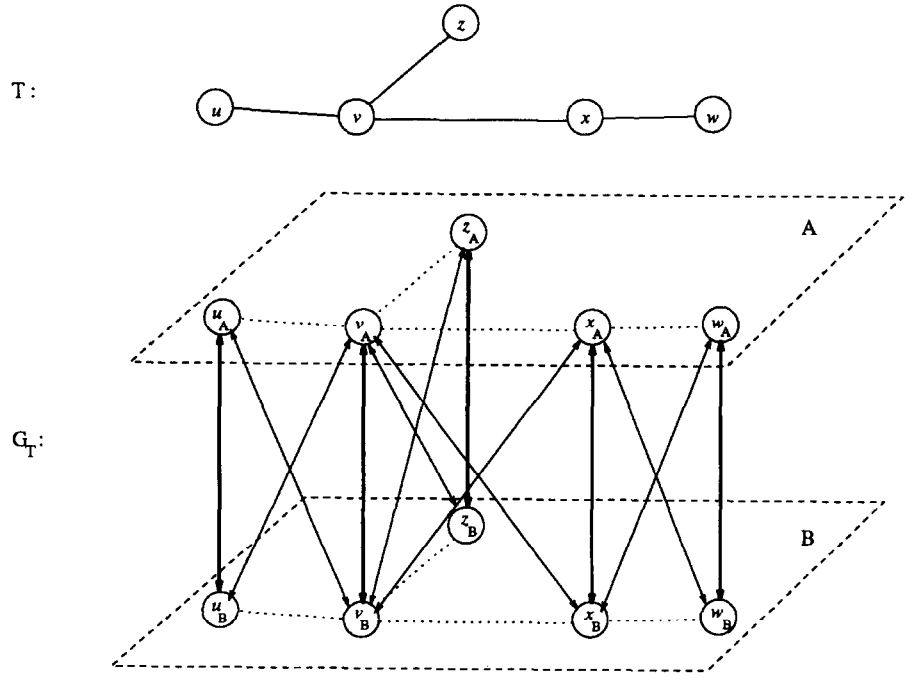

Fig. 4. Tree T and the corresponding graph G_T used in the analysis of Algorithm *Off-Line-Tree-Routing*.

Let u and w be two distinct nodes of tree T and let $\Pi_T = \langle u = v^0 \rightarrow \cdots \rightarrow v^i \rightarrow \cdots \rightarrow v^\mu = w \rangle$, $\mu \geq 1$, be the unique shortest path from node u to node w (i.e., edges $\{v^i, v^{i+1}\}$, $0 \leq i < \mu$ belong in T). By *primitive*(u,w) we denote the path $\Pi_{G_T} = \langle u_A = v_A^0 \rightarrow v_B^1 \rightarrow \cdots \rightarrow v_B^i \rightarrow v_A^i \rightarrow \cdots \rightarrow v_A^\mu = w_A \rangle$ of G_T.

A path Π of G_T is called a *primitive path* if and only if there exist two distinct nodes u and w of T such that $\Pi = primitive(u,w)$.

For example, consider graph G_T in Fig. 4. The paths $\Pi_1 = \langle v_A \rightarrow x_B \rightarrow x_A \rangle$ and $\Pi_2 = \langle u_A \rightarrow v_B \rightarrow v_A \rightarrow z_B \rightarrow z_A \rangle$ are primitive paths. The paths $\Pi_3 = \langle v_B \rightarrow v_A \rightarrow x_B \rightarrow x_A \rangle$, $\Pi_4 = \langle u_A \rightarrow v_B \rightarrow x_A \rangle$, and $\Pi_5 = \langle u_A \rightarrow v_B \rightarrow v_A \rightarrow x_B \rightarrow x_A \rightarrow v_B \rightarrow v_A \rangle$ are not primitive paths. This is because Π_3 starts from a node in B, in Π_4 node v_B is not followed by node v_A, and Π_5 is not a simple path.

Lemma 9. *Let u_A, $w_A \in A$ be two distinct nodes of graph G_T. Then, there is a unique primitive path from node u_A to node w_A. Moreover, the primitive path from node u_A to node w_A is of length $2dist_T(u,w)$*

Proof. Follows from the construction of graph G_T (based on tree T), the definition of a primitive path, and the fact that there exists a unique shortest path between any pair of nodes of an undirected tree. □

4.2.1. Many-to-one routing

For simplicity, we first analyse Algorithm *Off-Line-Tree-Routing* for many-to-one routing problems. In the next section, we extend the analysis to many-to-many routing. Assume that problem $\mathscr{R}=(T,M)$ is a many-to-one routing problem, that is, $|M|\leqslant n$ 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 $\mathscr{R}'=(G_T,H)$ by describing how to construct the set of packets H based on the packets of set M.

For each packet $p\in M$ with $u=orig(p)$ and $v=dest(p)$ we create two packets p^1 and p^2 in H (these two packets *correspond to* p). Packet p^1 is referred as the *twin packet* of packet p^2, and vice versa. For p^1 and p^2 we set: $orig(p^1)=orig(p^2)=u_A$ and $dest(p^1)=dest(p^2)=v_A$. So, all packets that participate in the hot-potato routing are generated in, and are destined for, nodes in the set A. For simplicity, we refer to all packets of the form p^1 and p^2 as packets of *type*-1 and *type*-2, respectively, where p is their corresponding packet in the tree routing.

Algorithm *Off-Line-Simulation* is the hot-potato algorithm which we use in our simulation. It specifies the rules that the nodes of graph G_T use when they decide which packet to forward (if any) to each of their outgoing edges.

Algorithm *Off-Line-Simulation*

Rules for nodes in A

[*Off-line-A-1*] A packet of type-1, currently at node u_A and destined for node w_A, is always sent to node v_B where node v is the first node in the shortest path from u to w in tree T (Fig. 5).

[*Off-line-A-2*] A packet of type-2, currently at node u_A, is always sent to node u_B (Fig. 5).

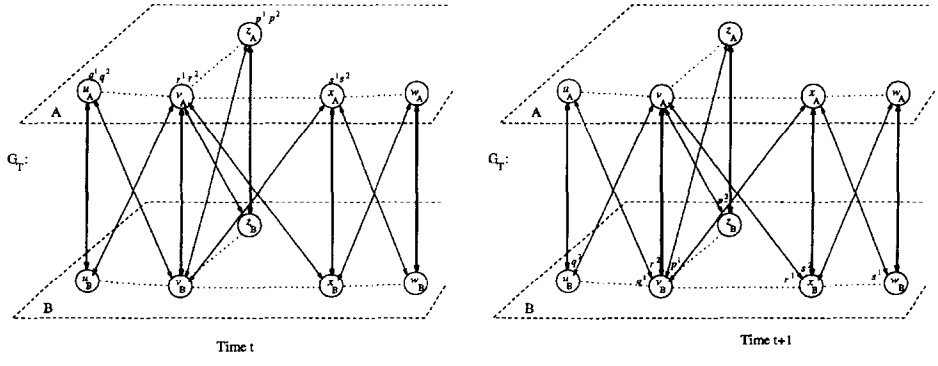

Fig. 5. Forwarding packets according to rules *Off-line-A-1* and *Off-line-A-2*. Packets p, q, r, and s of the matching routing are all destined for node w of T.

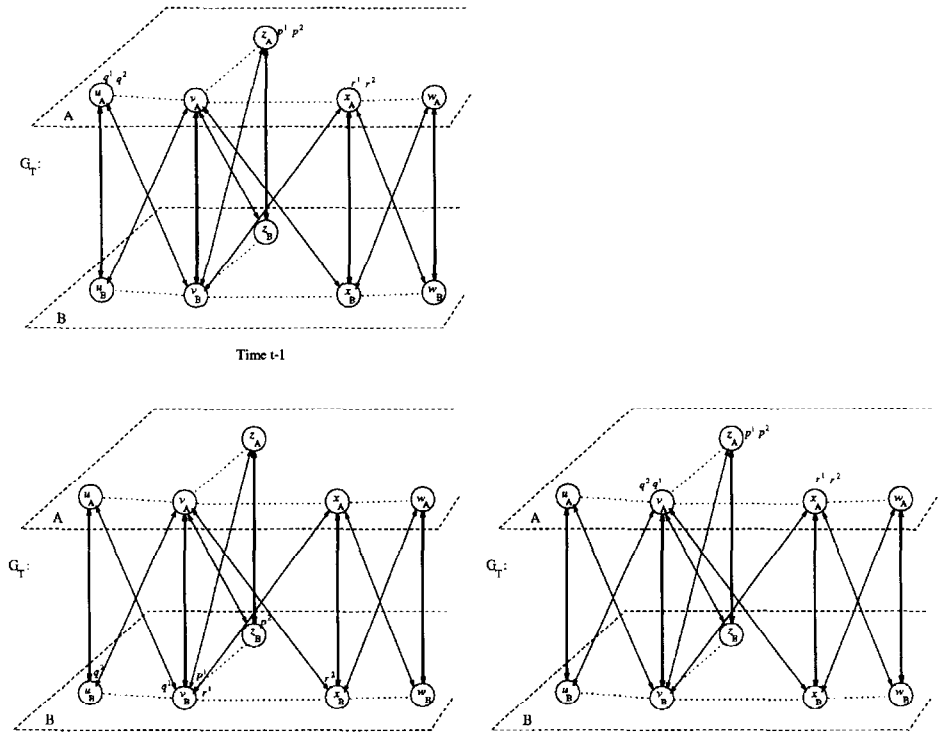

Fig. 6. At time t packets p^1, q^1 and r^1 are assigned to outgoing edges of node v_B according to rule *Off-line-B-1*. Assuming that node u_A is of lower order than x_A and z_A, packet q^1 was forwarded to node v_A while packets p^1 and r^1 were assigned to edges which lead to the nodes they arrived from, that is nodes z_A and x_A, respectively. Packets q^2, p^2, and r^2 are assigned to edges according to rule *Off-line-B-3*.

Rules for nodes in B

We first assign outgoing edges based on rules *Off-line-B-1* and *Off-line-B-2* to all packets for which it is possible to do so. We then assign outgoing edges to packets by applying repeatedly rules *Off-line-B-3* and *Off-line-B-4* in that order (recall that this is an off-line algorithm).

[*Off-line-B-1*] If node v_B contains only packets of type-1, then it forwards to node v_A the packet which arrived from the node of lowest order, and it returns each of the remaining packets to the node it arrived from (Fig. 6).

[*Off-line-B-2*] Consider a node u_B containing a packet p^2 which arrived from node u_A and a packet q^1 which arrived from node v_A. If node v_B contains packets p^1 and q^2 (the twin packets of p^2 and q^1, respectively) then q^1 and q^2 are forwarded to node u_A while packets p^1 and p^2 are forwarded to node v_A. Any other packet at nodes u_B and v_B is forwarded to the node it arrived from (Fig. 7).

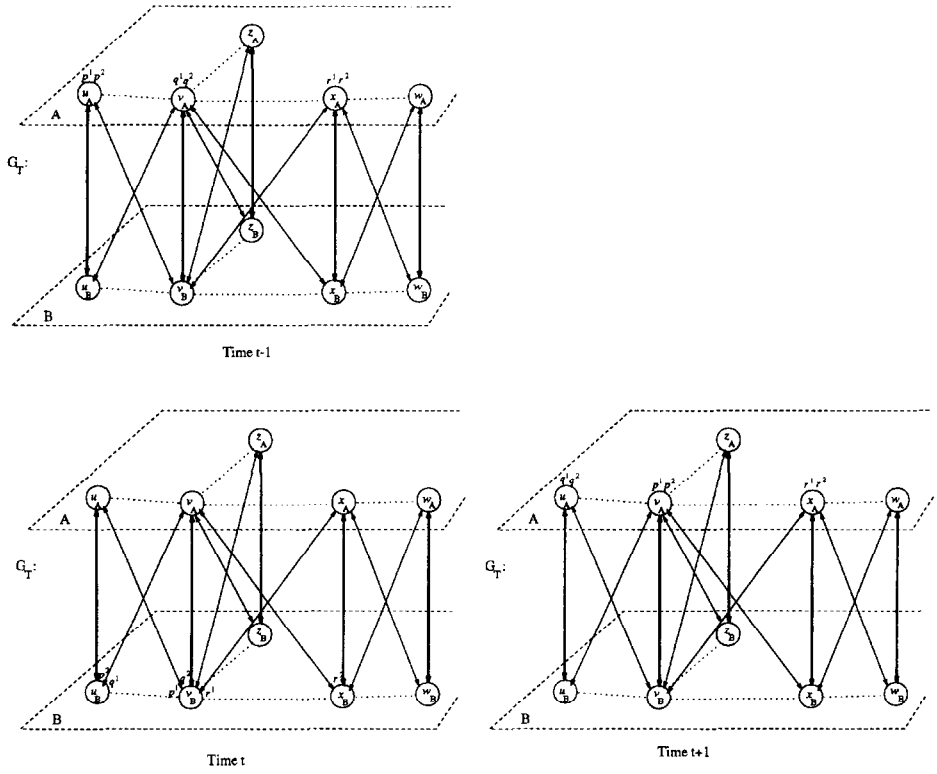

Fig. 7. At time t packets p^1, p^2, q^1, and q^2 are assigned to outgoing edges of nodes v_B and u_B according to rule *Off-line-B-2*. Packet r^1 is also assign to an outgoing edge according to rule *Off-line-B-2*. Packet r^2 is assigned to an outgoing edge according to rule *Off-line-B-3*.

[*Off-line-B-3*] A packet p^2 currently at node v_B with its twin packet p^1 at node u_B, is forwarded to node u_A if packet p^1 moves to node u_A, or to node v_A if packet p^1 moves to node v_A.

[*Off-line-B-4*] A packet p^1 currently at node v_B, which arrived at v_B from node u_A, returns to node u_A (note that rules *Off-line-B-1* or *Off-line-B-2* did not apply).

The idea behind this simulation of the matching routing is the following: Consider packet $p \in M$ which is at node u of T and wants to move to node v in order to reach its destination along the unique shortest path on T. In the matching routing p has to move through a swap from u to v. However, if this is achieved, any other packet that wants to enter u from an edge different than $\{u, v\}$ is not allowed to do so since only one edge incident to a node can be active at the same step. The simulation uses packet p^1 to indicate the desire of packet p to move from u to v. It uses packet p^2 in order to block other packets from advancing to node u. The following lemma formalises the properties of the simulation.

Lemma 10. *Let the many-to-one routing problem $\mathscr{R}=(T,M)$ be routed by Algorithm "Off-Line-Tree-Routing" and the routing problem $\mathscr{R}'=(G_T,H)$ by Algorithm "Off-Line-Simulation". Consider an arbitrary packet $p\in M$ and let $p^1,p^2\in H$ be the packets which correspond to it. Then,*

(i) *packet p is consumed at time c if and only if packets p^1 and p^2 are consumed at time $2c$, and*

(ii) *at time t packet p is at node u if and only if at time $2t$ packets p^1 and p^2 are at node u_A.*

Proof. We prove statement (ii) by induction on t. Statement (i) then follows from statement (ii).

The basis ($t=0$) of statement (ii) follows from the construction of routing problem $\mathscr{R}'=(G_T,H)$. For the inductive step of statement (ii) we consider the routing steps before the consumption of packet p (for the "only if" part) and of packets p^1 and p^2 (for the "if" part). We follow one step of Algorithm *Off-Line-Tree-Routing* for every two steps of Algorithm *Off-Line-Simulation*.

($\Rightarrow$) By the induction hypothesis, at time $t<c$ packet p is at node u of T if and only if at time $2t$ packets p^1 and p^2 are at node u_A. Let $\langle u\rightarrow v\rightarrow\cdots\rightarrow dest(p)\rangle$ be the unique shortest path from node u to node $dest(p)$. Edge (u,v) belongs in $G(T,t)$ (by the construction of $G(T,t)$) and thus, nodes u and v belong in the same in-tree or 1-loop in-tree.

Case 1: Nodes u and v belong to an in-tree of $G(T,t)$, say $T^{u,v}$. Note that u cannot be the root of the in-tree. We consider two sub-cases based on whether v is the root of the in-tree $T^{u,v}$.

Case 1.1: Node v is the root of the in-tree $T^{u,v}$. By the induction hypothesis, at time $2t$ packets p^1 and p^2 are at node u_A. According to rules *Off-line-A-1* and *Off-line-A-2*, at time $2t+1$ packets p^1 and p^2 are located at nodes v_B and u_B, respectively. Since at time t of the matching routing on T no packet was located at node v (v is the root of an in-tree of $G(T,t)$), at time $2t$ no packet is located at node v_A and thus, by rule *Off-line-A-2*, at time $2t+1$ no type-2 packet is located at node v_B. Thus, at time $2t+1$ node v_B contains only packets of type-1, among which is packet p^1. Let $x^{\min}=\min\{x\,|\,(x,v)\in$ in-tree $T^{u,v}\}$ (according to the order of nodes of T). We consider two cases:

Case 1.1.1: $x^{\min}=u$. In this case, the edge $\{u,v\}$ of T is selected to swap the packets at its endpoints, and thus, at time $t+1$ of the matching routing, packet p is at node v.

Consider the packets at node v_B at time $2t+1$. These packets are assigned to outgoing edges according to rule *Off-line-B-1*. Thus, at time $2t+2$ packet p^1 is forwarded to v_A while each of the other packets is forwarded to the node it arrived from. By rule *Off-line-B-3*, packet p^2 is forwarded from node u_B to node v_A. Thus, at time $2t+2=2(t+1)$ both packets p^1 and p^2 are at node v_A.

Case 1.1.2: $x^{\min}\neq u$. In this case, the edge $\{u,v\}$ of T is not selected to swap the packets at its endpoints and thus, at time $t+1$ packet p is at node u of T. Consider the packets at node v_B at time $2t+1$. These packets are assigned to outgoing edges

according to rule *Off-line-B-1*. Packet p^1 is not selected to be forwarded to node v_A. Thus, by the same rule, it is forwarded to the node it arrived from, that is, node u_A. By rule *Off-line-B-3*, packet p^2 which is located at time $2t+1$ at node u_B, is forwarded to node u_A. Thus, at time $2t+2=2(t+1)$ both packets p^1 and p^2 are at node u_A.

Case 1.2: Node v is not the root of the in-tree $T^{u,v}$. In this case, according to Algorithm *Off-Line-Tree-Routing*, edge $\{u,v\}$ is not selected to swap the packets at its endpoints and thus, at time $t+1$ packet p is at node u. Moreover, since node v is a node of the in-tree $T^{u,v}$, at time t it does hold a packet, say packet q.

Assume that node v is adjacent to the root, say node r, of the in-tree $T^{u,v}$, that is, $\langle u \to v \to r\rangle$ is a path from u to the root r of $T^{u,v}$. By the induction hypothesis, at time $2t$ packets p^1 and p^2 are at node u_A while packets q^1 and q^2 are at node v_A. According to rules *Off-line-A-1* and *Off-line-A-2*, at time $2t+1$ packet p^2 is at node u_B, packets p^1 and q^2 are at node v_B, and packet q_1 is at node r_B (possibly together with other packets of type-1).

In the case where node v is the node of lowest order which is adjacent to the root r of $T^{u,v}$, packets q^1 and q^2 are forwarded to node r_A (by rules *Off-line-B-1* and *Off-line-B-3*, respectively). By rule *Off-line-B-4*, packet p^1 is forwarded to u_A and then, due to rule *Off-line-B-3*, packet p^2 is also forwarded to node u_A. Thus, at time $2t+2$ both packets p^1 and p^2 are located at node u_A.

In the case where v is not the node of lowest order which is adjacent to the root r of $T^{u,v}$, the application of the same rules results in having packets p^1 and p^2 located at node u_A at time $2t+2$ (and packets q^1 and q^2 at node v_A). Thus, in any case, at time $2t+2=2(t+1)$ packets p^1 and p^2 are at node u_A.

In the proof of Case 1.2, we assumed that node v is adjacent to the root, say node r, of the in-tree $T^{u,v}$. If this is not the case, an induction on the distance of node v from r in $T^{u,v}$ is required to establish that at time $2t+2$ both packets p^1 and p^2 are located at node u_A.

Case 2: Nodes u and v belong to an 1-loop in-tree of $G(T,t)$, say $T^{u,v}$. The proof of this case is similar to that of the case where nodes u and v belong to an in-tree of $G(T,t)$. The only difference is that rule *Off-line-B-2* is applied (instead of rule *Off-line-B-1*) to assign the packets at the nodes of the 1-loop to outgoing edges. In order to avoid repetition, we omit the proof of this case. The reader who is interested in constructing a detailed proof, should consider the following two sub-cases:

Case 2.1: Edge (u,v) is one of the edges forming the 1-loop of the 1-loop in-tree $T^{u,v}$.

Case 2.2: Edge (u,v) is not one of the edges forming the 1-loop of the 1-loop in-tree $T^{u,v}$.

($\Leftarrow$) The "if" part of the proof is also based on the fact the rules *Off-line-B-1* and *Off-line-B-2* of Algorithm *Off-Line-Simulation* assign packets to outgoing edges in a way which mirrors the selection of the matching on T by Algorithm *Off-Line-Tree-Routing*. Since it is symmetric to the "only if" part, it is omitted.

Statement (i), that is, packet p is consumed at time c if and only if packets p^1 and p^2 are consumed at time $2c$, follows from Statement (ii) and the fact that both

Algorithms *Off-Line-Tree-Routing* and *Off-Line-Simulation* consume packets as soon as they enter their destination node. □

Corollary 11. *Let the routing problem* $\mathscr{R}=(T,M)$ *be routed by Algorithm "Off-Line-Tree-Routing" and the routing problem* $\mathscr{R}'=(G_T,H)$ *by Algorithm "Off-Line-Simulation" and consider the case where* $|M|=1$. *Let* p *be the only packet in* M *and let* $p^1, p^2 \in H$ *be the packets which correspond to it. Then, packet* p^1 *is routed on* G_T *along path primitive*(u,v) *where* $u=orig(p)$ *and* $v=dest(p)$.

We analyse Algorithm *Off-Line-Simulation* by considering "deflections" of type-1 packets. However, the notion of deflection which we use is slightly different from the one which is traditionally used in the literature (and which was also introduced in Section 2). Traditionally, a deflection is considered to be a move of a packet which does not take it closer to its destination. A deflection is caused because all edges which lead closer to the destination of the packet are assigned to other packets. This definition assumes that each packet always tries to move along a shortest path from the node it resides in to its destination. The definition is accurate for all greedy hot-potato algorithms but it fails to serve algorithms which might try to route packets along non-minimal paths.

Consider a hot-potato algorithm which is used to do the routing on a graph G. Assume that packet p is at node v of G at time t and let $\Lambda(p,v,t)$ be the set of paths from node v to $dest(p)$ which packet p is allowed to follow in order to reach its destination. Set $\Lambda(p,v,t)$ might be a non-finite set since the paths are not restricted to be simple paths. Let $\Lambda'(p,v,t) \subseteq \Lambda(p,v,t)$ be the set of paths of minimum length among the paths of $\Lambda(p,v,t)$. Then, a *deflection of packet* p *at node* v *at time* t is defined to be the event where packet p fails to move along a path in set $\Lambda'(p,v,t)$ due to the rules of the routing. In this paper, we assume that another packet also present in node v at time t (which influences the routing decisions with its presence) causes the deflection of packet p. In the rest of this section, we use this broader definition for deflections.

For example, consider Algorithm *Off-Line-Simulation* and the routing of problem $\mathscr{R}'=(G_T,H)$. Assume that packet $p^1 \in H$ which is destined for node w_A is, at time t, located at node u_A of Fig. 4. Then, the set $\Lambda(p^1,u_A,t)$ includes the primitive path $\langle u_A \to v_B \to v_A \to x_B \to x_A \to w_B \to w_A \rangle$ and, among others, the non-simple path $\langle u_A \to v_B \to u_A \to v_B \to v_A \to x_B \to x_A \to w_B \to w_A \rangle$. The paths $\langle u_A \to v_B \to x_A \to w_B \to w_A \rangle$ and $\langle u_A \to v_B \to v_A \to x_B \to w_A \rangle$ do not belong in $\Lambda(p^1,u_A,t)$ since no packets which is routed by Algorithm *Off-Line-Simulation* moves along them. Set $\Lambda'(p^1,u_A,t)$ includes only the primitive path $\langle u_A \to v_B \to v_A \to x_B \to x_A \to w_B \to w_A \rangle$.

We analyse the performance of Algorithm *Off-Line-Simulation* when it is used to route problem $\mathscr{R}'=(G_T,H)$ which was obtained from problem $\mathscr{R}=(T,M)$. We concentrate only on deflections of packets of type-1. Lemma 10 allows us to ignore in our analysis packets of type-2 since they are consumed at their destination at the same time that their twin packets are.

Consider packet $p^1 \in H$ which at time $t-1$ is located at node u_A and it corresponds to $p \in M$, and let $w = dest(p)$. Let (u_A, v_B) be the first edge of the unique primitive path from u_A to w_A. According to Algorithm *Off-Line-Simulation*, packet p_1 can move only along paths which emanate from u_A and all have edge (u_A, v_B) as their first edge. Thus, packet p^1 is never deflected at a node of set A. At time t, packet p^1 is located at node v_B. It might only move along paths (to its destination) which start with edge (v_B, u_A) or edge (v_B, v_A). Among the paths of these two classes, the path of minimum length starts with edge (v_B, v_A). It is possible that the routing rules dictate that packet p^1 is forwarded along edge (v_B, u_A) due to the presence of some other packet $q^\star$ (the notation $q^\star$ is used to denote either the type-1 packet q^1 or the type-2 packet q^2) in node v_B at time t. In this case, we say that the packet is deflected at time t by packet $q^\star$. Note that packets of type-1 can be only deflected at nodes of B.

Consider the deflection of packet p^1 from node v_B to node u_A at time t which was caused by packet $q_1^\star$ ($q_1^\star$ is either q_1^1 or q_1^2). Packet $q_1^\star$ is forwarded along edge (v_B, v_A) to node v_A or along edge (v_B, x_A) to node x_A, where x is the first node in the path from v to $dest(q_1)$ in T (the choice depends on whether packet q_1 was selected for a swap in the tree routing). At time $t+1$ packet $q_1^\star$ and its twin packet are both at the same node (Lemma 10). We then follow packet q_1^1 along its primitive path until it reaches its destination or until it is deflected at some node w_B, whichever happens first. If it was deflected first, let packet $q_2^\star$ be the packet which caused the deflection. In a similar way, we follow packets $q_2^\star$ for one step to a node in A and then we follow packet q_2^1 along its primitive path until it reaches its destination or until it is deflected, whichever happens first, and so on. This process finishes when the packet which we follow, say $q_l^\star$, reaches its destination. The *deflection path which corresponds to the deflection of packet* p^1 *at time* t, denoted $dp(p^1, t)$, is the path which we followed from node v_B to the destination of packet $q_l^\star$, that is node $dest(q_l^\star)$. The length of $dp(p^1, t)$ might be as large as $2n-2$. At time t', where $t < t' \leqslant t + 2n - 2$, the deflection path might be still "growing". We use the notation $endpoint(p^1, t, t')$ to denote the endpoint node of the deflection path developed up to time t'. (In the event where the deflection path was completely formed before time t', $endpoint(p^1, t, t')$ is undefined.)

Lemma 12. *Consider the routing of problem* $\mathscr{R}' = (G_T, H)$ *by Algorithm* Off-Line-Simulation *and let* $p^1 \in H$ *be an arbitrary type-1 packet. Then, at time* t' *and for any node* u_A *(of* A*) it holds*:

(i) *At most 2 deflection paths for* p^1 *have* $endpoint(p^1, t_1, t') = endpoint(p^1, t_2, t') = u_A$, *and*

(ii) *if there exist two deflection paths for* p^1 *with* $endpoint(p^1, t_1, t') = endpoint(p^1, t_2, t') = u_A$, *then there does not exist a deflection path* $dp(p^1, t)$ *with* $endpoint(p^1, t, t') = v_A$ *where* u *and* v *are neighbours in tree* T.

Proof (*sketch*). The lemma is formally proven by induction on the number of routing steps. Only part (i) of the lemma is used in the rest of the paper. Part (ii) is an invariant from which part (i) follows.

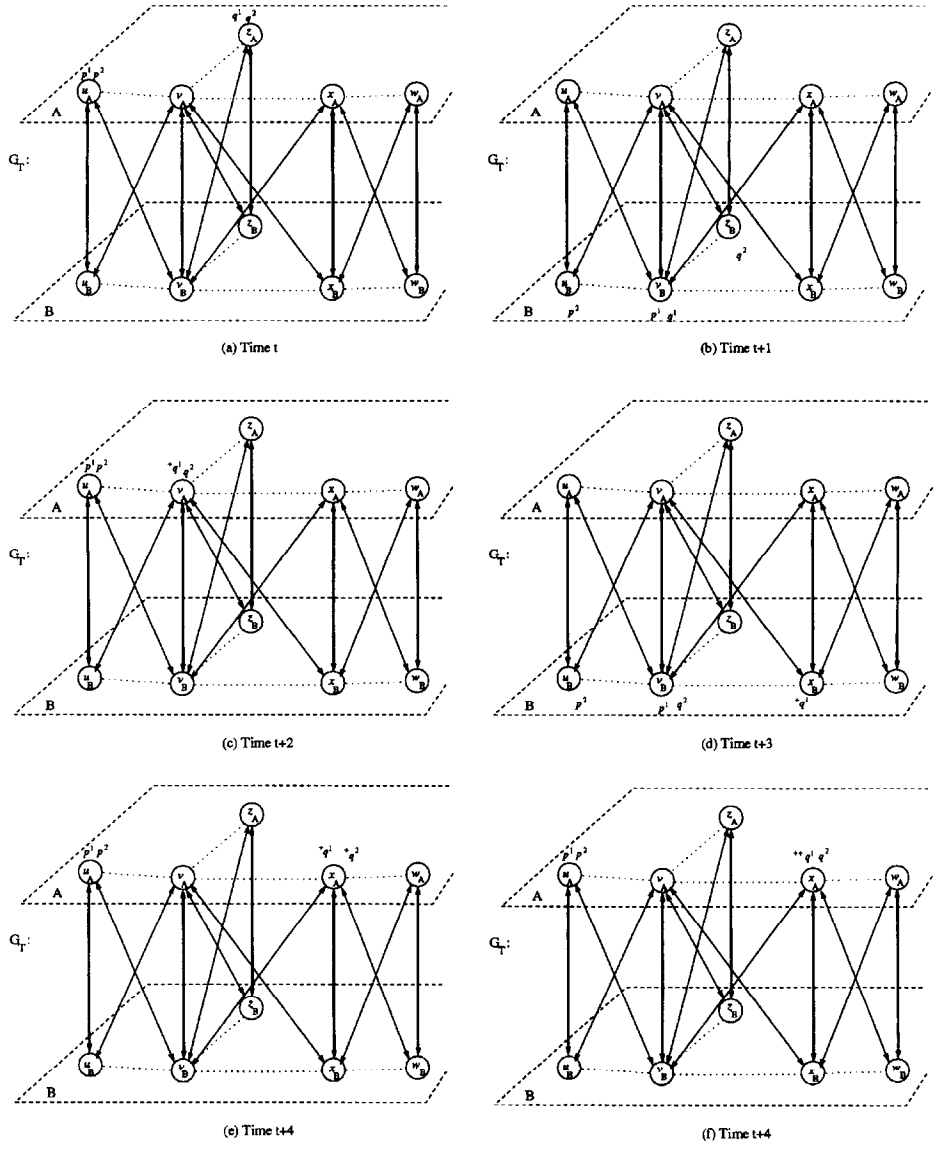

Fig. 8. The situation where two deflection paths due to deflections of the same packet have common endpoint nodes.

We first demonstrate how it is possible to have two deflection paths with the same endpoint. This is illustrated in Fig. 8. Assume that the destination of all packets which appear in Fig. 8(a) is node w_A. At time $t+1$ packets p^1 and q^1 meet at node v_B and one of them is deflected by the other, say p^1 is deflected by q^1. This is illustrated in Fig. 8(c) with a $^+$-sign to the left of q^1, i.e., $^+q^1$, denoting that the node in which packet q^1 resides is the endpoint of the deflection path $dp(p^1, t+1)$. At time $t+3$, p^1

and q^2 meet at v_B (Fig. 8(d)) and, according to the routing rules, p^1 is again deflected, this time by q^2. This is illustrated in Fig. 8(e) where, at time $t+4$, packet q^2 (in node x_A) appears with a $^+$-sign to its left, i.e., $^+q^2$, denoting that node x_A in which packet q^2 resides is the endpoint of the deflection path $dp(p^1, t+3)$. Recall that the deflection paths are defined in such a way that when they leave a node in A they always follow the route of a type-1 packet. This is depicted in Fig. 8(f) where, again at time $t+4$, both deflection paths have as their endpoints the node occupied by packet q^1, denoted by $^{++}q^1$.

Observe now that advancing packets move slower than the endpoints of the deflection paths. For this reason, and due to the special structure of the bipartite graph, they cannot catch up with them after their distance at the underlying tree structure is greater than 1. So, we only have to consider the case where the endpoints of two deflection paths catch up with each other. This appears to be possible since deflection paths are not necessarily primitive paths. More specifically, at places where deflections take place, the path might reach a node in A not from its corresponding node in B. Such an example is the deflection path $dp(p^1, t+3)$ for the scenario described in Fig. 8. The deflection path $\langle v_B \to x_A \to w_B \to w_A \rangle$ starts at v_B and moves to x_A (instead of v_A which would be the case if it was a primitive path).

The basis of the induction is easy to establish since at the first four steps of the hot-potato routing each packet might be deflected at most twice. The induction step is then proved by considering cases based on the next move of the packet at the endpoint of the deflection path. By using the routing rules it is easy (but tedious and thus omitted) to show that both parts of the lemma are correct. □

Theorem 13. *Algorithm "Off-Line-Tree-Routing" routes any many-to-one routing problem $\mathscr{R}=(T,M)$ 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.*

Proof. We consider the routing of problem $\mathscr{R}'=(G_T,H)$ by Algorithm *Off-Line-Simulation*. Concentrate on an arbitrary packet $p \in M$ and its corresponding packet $p^1 \in H$. We prove the theorem by showing that p^1 is consumed by time $2(2(k-1)+dist)$. This implies (by Lemma 10) that packet p is consumed by time $2(k-1)+dist$.

Packet p^1 can be deflected at most $2(k-1)$ times. This is because, the number of deflection paths for which p^1 is responsible can be at most that large. To see that realize that by Lemma 12 each type-1 packet at the time of its consumption is the endpoint of at most two deflection paths due to p^1.

The cost of each deflection is 2 steps, that is, if packet p^1 is deflected from node u_B at time t, it returns to it at time $t+2$ (rule *Off-line-A-1*). Given that the distance that packet p^1 has to travel is $2dist_T(orig(p), dest(p)) \leqslant 2dist$, we conclude that the hot potato routing finishes by time $2(2(k-1)+dist)$. By Lemma 10 we conclude that the matching routing on T finishes after at most $2(k-1)+dist$ routing steps. □

Krizanc and Zhang [15] 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.

4.2.2. Many-to-many routing

Consider the many-to-many routing problem $\mathscr{R}=(T,M)$ which is routed by algorithm *Off-Line-Tree-Routing*. Based on the routing of $\mathscr{R}=(T,M)$ by Algorithm *Off-Line-Tree-Routing*, we define a dynamic routing problem $\mathscr{R}'=(G_T,H)$ which is again routed by Algorithm *Off-Line-Simulation*.

We first route problem $\mathscr{R}=(T,M)$ by Algorithm *Off-Line-Tree-Routing* and we observe for each individual packet the time at which it is injected into the routing. When the routing of $\mathscr{R}=(T,M)$ terminates, we are ready to fully specify problem $\mathscr{R}'=(G_T,H)$. For each packet $p\in M$ which was injected into the matching routing at time t, we create a pair of packets p^1 and p^2 in H with $birth(p^1)=birth(p^2)=2t$. The origin and the destination nodes of p^1 and p^2 are set as in the analysis of the many-to-one routing.

The following lemma establishes that each packet of H can be injected into the hot-potato routing at the time it is generated.

Lemma 14. *Consider the many-to-many routing problem $\mathscr{R}=(T,M)$ which is routed by Algorithm "Off-Line-Tree-Routing" and the constructed dynamic routing problem $\mathscr{R}'=(G_T,H)$ which is routed by Algorithm "Off-Line-Simulation". Let p be an arbitrary packet in M and let p^1 and p^2 be its corresponding packets in H. If Algorithm "Off-Line-Tree-Routing" injects packet p at time t then Algorithm "Off-Line-Simulation" can inject packets p^1 and p^2 at time $2t$.*

Proof. Follows from Lemma 10 and a simple induction on time t. □

Lemma 14 makes it possible to simulate the injection of packet of the matching routing with the dynamic creation (even in an off-line fashion) of their corresponding packets into the hot-potato routing. The next theorem follows immediately.

Theorem 15. *Algorithm "Off-Line-Tree-Routing" routes any many-to-many routing problem $\mathscr{R}=(T,M)$ 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.*

5. Conclusions

In this paper, we presented an extensive study of the many-to-many routing problem on n-node trees under the matching routing model. We developed optimal on-line and

off-line algorithms which were analysed with the help of tools originally developed for hot-potato routing. The study of matching routing is a relatively new topic and several problems remain open. The following list states some of them.

(i) What is the time complexity of the matching routing problem? Given a routing problem $\mathscr{R}=(T,M)$ and an integer t, can we answer the question "is it possible to route problem $\mathscr{R}=(T,M)$ in less than t steps?" in polynomial time?

(ii) Derive an on-line algorithm which routes permutations on trees of maximum degree d under the original matching model (i.e., no consumptions are allowed) in $\Theta(dn)$ steps. During the course of our research, we have considered several on-line algorithms which perform well in simulations. However, we have not succeeded to get an analysis for any of them which matches the desired bound.

(iii) There is a permutation on an n-node tree which requires $\lfloor 3(n-1)/2 \rfloor$ steps for its routing under the original routing model. Is it possible to route any permutation in at most $3n/2$ steps? [3, 2].

References

[1] S. Akl, Parallel Sorting Algorithms, Academic Press, New York, 1985.

[2] N. Alon, F.R.K. Chung, R.L. Graham, Routing permutations on graphs via matchings (extended abstract), in: Proc. 25th Annual ACM Symp. on Theory of Computing, San Diego, CA, 16–18 May 1993, pp. 583–591, ACM SIGACT, ACM Press, New York, 1993.

[3] Alon, Chung, Graham, Routing permutations on graphs via matchings, SIAM J. Discrete Math. 7 (1994) 513–530.

[4] P. Baran, On distributed communication networks, IEEE Trans. on Comm. Systems, CS-12 (1964) 1–9.

[5] A. Bar-Noy, B. Schieber, P. Raghavan, H. Tamaki, Fast deflection routing for packets and worms, in: Proc. 12th Annual ACM Symp. on Principles of Distributed Computing (PODC 93), Ithaca, NY, pp. 75–86, August 1993.

[6] Baumslag, Annexstein, A unified framework for off-line permutation routing in parallel networks, Math. Systems Theory 24 (1991) 233–251. Appeared also in 2nd Annual ACM Symp. on Parallel Algorithms Architectures (SPAA 90),

[7] I. Ben-Aroya, A. Schuster, Greedy hot-potato Routing on the two-dimensional Mesh, in: Jan van Leeuwen (Ed.), Proc. 2nd European Symp. on Algorithms ESA '94, Utrecht, September 1994, Lecture Notes in Computer Science, vol. 855, Springer, Berlin, 1994, pp. 365–376.

[8] A. Borodin, Y. Rabani, B. Schieber, Deterministic many-to-many hot potato routing, Tech. Report RC 20107 (6/19/95), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, June 1995.

[9] J.T. Brassil, R.L. Cruz, Bounds on maximum delay in networks with deflection routing, in: Proc. 29th Allerton Conf. on Comm. Control Comput., 1991, pp. 571–580.

[10] U. Feige, P. Raghavan, Exact analysis of hot-potato routing, in: Proc. 33rd Annual Symp. on Found. Comput. Sci., Pittsburgh, PA, 24–27 October 1992, IEEE Computer Society Press, Los Alamitos, 1992, pp. 553–562.

[11] N. Haberman, Parallel neighbor-sort (or the glory of the induction principle), Tech. Report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Port Royal Road, Springfieldn VA 22151, 1972.

[12] B. Hajek, Bounds on evacuation time for deflection routing, Distributed Comput. 5 (1991) 1–6.

[13] C. Kaklamanis, D. Krizanc, S. Rao, Hot-potato routing on processor arrays, in: Proc. 5th Annual ACM Symp. on Parallel Algorithms Architectures, SPAA'93, Velen, Germany, June 30–July 2 1993, New York, ACM SIGACT, ACM SIGARCH, ACM Press, pp. 273–282.

[14] M. Kaufmann, H. Lauer, H. Schroder, Fast deterministic hot-potato routing on processor arrays, in: D.Z. Du, X.S. Zhang (Eds.), Proc. 5th Internat. Symp. on Algorithms Computation ISAAC '94,

Beijing, P.R. China, August 1994, Lecture Notes in Computer Science, vol. 834, Springer, Berlin, 1994, pp. 333–341.

[15] D. Krizanc, L. Zhang, Packet routing via matchings, Unpublished manuscript, 1996.

[16] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays – Trees – Hypercubes, Morgan Kaufmann, San Mateo, CA 94403, 1991.

[17] Newman, Schuster, Hot-potato algorithms for permutation routing, IEEE Trans. Parallel Distri. Systems 6(11) (November 1995) 1168–1176.

[18] G. Pantziou, A. Roberts, A. Symvonis, Dynamic tree routing under the "matching with consumption" model. In: T. Asano, Y. Igarashi, H. Nagamochi, S. Miyano, S. Suri, (Eds.), Proc. 7th Internat. Symp. on Algorithms and Computation (ISAAC '96), Osaka, Japan, December 1996, Lecture Notes in Computer Science, vol. 1178, Springer, Berlin, 1996, pp. 275–284.

[19] M. Ramras, Routing permutations on a graph, Networks 23 (1993) 391–398.

[20] A. Roberts, A. Symvonis, L. Zhang, Routing on trees via matchings, in: Proc. 4th Workshop on Algorithms and Data Structures (WADS'95), Kingston, Ontario, Canada, Lecture Notes in Computer Science, vol. 955, Springer, Berlin, 1995, pp. 251–262. Also Tech. Report 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.

[21] L. Zhang, personal communication, 1996..