ON-LINE MATCHING ROUTING ON TREES

BY ALAN ROBERTS AND ANTONIOS SYMVONIS

{alanr, symvonis}@cs.su.oz.au

TECHNICAL REPORT 514
27 May 1997
On-line Matching Routing on Trees

Alan Roberts and Antonios Symvonis

Department of Computer Science, University of Sydney, NSW 2006, Australia.
{alanr,symvonis}@cs.su.oz.au

Abstract

In this paper we examine on-line heap construction and on-line permutation routing on trees under the matching model. Let $T$ be an $n$-node tree of maximum degree $d$. By providing on-line algorithms we prove that:

(i) For a rooted tree of height $h$, on-line heap construction can be completed within $(2d - 1)h$ routing steps.

(ii) For an arbitrary tree, on-line permutation routing can be completed within $4dn$ routing steps.

(iii) For a complete $d$-ary tree, on-line permutation routing can be completed within $2(d - 1)n + 2d \log^2 n$ routing steps.

Technical Report 514
Basser Department of Computer Science
University of Sydney
Original: 27 May 1997

1 The work of Dr Symvonis was supported by an ARC Institutional Grant.
1 Introduction

In packet routing problems we are given a network (usually represented by a connected, undirected graph) and a set of packets distributed over the nodes of the network. Each packet has an origin node and a destination node and our aim is to route the packets to their destinations as fast as possible. The movement of the packets is subject to a set of routing rules which define the routing model. Typical routing rules found on frequently used models include: restrictions on the number of packets that can reside on a node at any given time instance, restrictions on the number of packets that can be transmitted/received by a node at any single step, restrictions on whether two packets can simultaneously traverse an edge in opposite directions, restrictions on the amount of information that can be used in making routing decisions, etc.

The distribution of packet origins and destinations specifies the routing pattern. There are four main categories of routing patterns. A pattern may be many-to-one, one-to-many, many-to-many or one-to-one. In many-to-one patterns, each node is the origin of at most one packet but may be the destination of more than one packet. In one-to-many patterns, each node may be the origin of many packets but cannot be the destination of more than one packet. In many-to-many patterns, each node may be the origin and destination of many packets. Finally, in one-to-one patterns, each node is the origin and destination of at most one packet. One-to-one patterns are often referred to as partial permutations. If every node starts with exactly one packet (rather than at most one packet) then the pattern is called a permutation.

Packet routing algorithms fall into two main categories, namely off-line and on-line algorithms. In off-line routing, a routing schedule which dictates how each packet moves during each step of the routing is precomputed. Then, when the routing is actually carried out, all packets move according to the routing schedule. We can think of the routing schedule to consist of a collection of paths, each path corresponding to a particular packet and describing the route that the packet follows from its origin to its destination node. The paths are computed in a “centralized manner”, that is, information regarding the origin/destination of all packets participating in the routing is used in determining the route of any individual packet.

In contrast with off-line routing, in on-line routing, routing decisions are made in a “distributed manner” by the nodes of the network. At each routing step, every node examines the packets that reside in it (some of them have just arrived) and decides, whether to advance them to neighbouring nodes or to store them in local queues. The decision made by each node depends on local information, usually consisting of the origin/destination nodes of the packets residing in it, and knowledge regarding the topology of the network. (More complicated on-line schemes where “local knowledge” incorporates information accumulated in the node since the beginning of the routing can be also defined.)

In this paper we examine permutation routing on trees under the matching model of routing. The model was originally introduced by Alon, Chung and Graham in their study of permutation routing [3,2]. The only routing operation allowed is the exchange of the packets at the end-points of an edge. Many exchanges can occur simultaneously on a given
step, however, these exchanges must occur over a disjoint set of edges, i.e., a matching, of the network. Another feature of the matching model is that packets are not necessarily consumed when they reach their destinations for the first time. Instead they continue moving until all packets reach their destinations simultaneously, at which time they are all consumed and the routing is over.

Most of the work available on the matching model concentrates to off-line routing. Alon, Graham and Chung [3,2] gave an off-line algorithm for routing permutations on arbitrary trees under the matching model. Their algorithm route any permutation on a tree of $n$ nodes within $3n$ steps. They also gave a lower bound of $3n/2$ steps for off-line matching routing on trees. Roberts, Symvonis and Zhang [10] established a new upper bound of $2.3n$ steps for off-line matching routing. They also proved that for the case of bounded degree trees and complete d-ary trees, routing can always be completed within $2n+o(n)$ steps and $n+o(n)$ steps, respectively. Zhang [11] further reduced the bound for permutation routing on arbitrary trees to $3n/2+o(n)$ steps. For product networks, including hypercubes and meshes, Annexstein and Baumslag [4] derived off-line algorithms that can be considered to follow the matching model. More specifically, based on their work, any permutation on a hypercube of $2^n$ nodes can be off-line routed within $2n-1$ steps, while any permutation on an $n \times n$ mesh can be off-line routed within $3n-3$ steps.

The odd-even transposition algorithm [5] (see also [1,7]) can be considered as the first work related to on-line matching routing. The odd-even transposition algorithm sorts a permutation on a linear array of $n$ nodes within $n$ steps by performing at each step comparisons/exchanges over a set of disjoint array edges. We are not aware of any other on-line algorithms for packet routing under the matching model.

More recently, Pantziou. Roberts and Symvonis [8,9] gave an algorithm for on-line routing on trees under a variation of the matching model called the matching model with consumptions. In this variation of the model, packets are consumed as soon as they reach their destinations, thus making it possible to route any kind of routing pattern. Their algorithm was able to complete the routing of any $k$-packet pattern on any $n$-node tree $T$ within $d(k-1)+d\cdot dist$ routing steps, where $d$ is the maximum degree of $T$ and $dist$ is the maximum origin to destination distance of any packet. They also gave an optimal off-line algorithm which can complete the many-to-many routing of $k$ packets on any tree within $2(k-1)+dist$ routing steps. For the same model (matching model with consumptions) Zhang and Krizanc [6] have also given an off-line algorithm for many-to-many routing on trees. However, until now no bounds were known for on-line routing on trees under the original matching model of [3,2].

In our attempt to derive on-line algorithms for routing on trees under the matching routing model, we run into a problem of independent interest. This is the problem of heap construction. Consider a rooted tree $T$ and let each of its nodes have a key-value associated with it. We say that $T$ is heap ordered if each non-leaf node satisfies the heap invariant: “the key-value of the node is not larger than the key-values of its children”. When the key-value at each node is carried by (or associated with) the packet currently in the node, the problem of heap construction is simply to route the packets on the tree in a way that guarantees that at the end of the routing the packets are heap-ordered based on the key-values they carry. Needless to say, we are interested in forming the heap on-line
and in the smallest number of parallel routing steps where routing is performed according to the matching routing model.

In this paper, we provide and analyse on-line matching routing algorithms that support the following results:

(i) For an arbitrary $n$-node rooted tree of maximum degree $d$ and height $h$, heap construction can be completed within $2dh$ routing steps.
(ii) For an arbitrary $n$-node tree of maximum degree $d$, permutation routing can be completed within $4dn$ steps.
(iii) For a complete $n$-node $d$-ary tree, permutation routing can be completed within $2(d - 1)n + 2d \log^2 n$ steps.

These are the first results concerning on-line routing of permutations under the matching model of routing. Note that, our results also provide on-line algorithms for routing on general undirected graphs since, in the case of general graphs routing can be performed over a spanning tree of the graph.

The rest of the paper is organised as follows: In Section 2, we give necessary definitions and explain the notation used in the rest of the paper. In Section 3, we present and analyse our on-line algorithm on heap construction. In Section 4, we describe a recursive on-line algorithm for matching routing and we bound its routing time. We conclude in Section 5 by discussing further research directions.

2 Preliminaries

A tree $T = (V, E)$ is a connected undirected acyclic graph with node set $V$ and edge set $E$. A rooted tree is a tree in which one of its nodes, say $r$, is designated as its root. The depth of a node $v$ of a rooted tree is equal to the distance (i.e., the length of the shortest path) between node $v$ and the root $r$ of the tree. The height of a rooted tree is defined to be the largest depth over all nodes of the tree.

Consider an arbitrary node $v$ of a tree rooted at node $r$. All nodes that appear in the simple path from $v$ to $r$, including $v$ and $r$, are called ancestors of $v$. All nodes that have $v$ as their ancestor, form the set of the descendents of $v$. Note that $v$ is an ancestor and dependent of itself. The subtree rooted at $v$, denoted by $T_v$, consists of all the descendents nodes of $v$ and the edges that connect them.

Throughout this paper we analyse our algorithms in terms of rooted trees. It is thus necessary to refer to the relative position of nodes, edges and packets within a tree or sub-tree. To do this we use the notions of node levels and edge levels. All nodes of depth $k$ are referred as level-$k$ nodes while all edges connecting a $k$-level node with a $(k + 1)$-level node are referred as level-$k$ edges. The idea is illustrated in Figure 1. We also frequently use the notion of up and down, above and below. These directions are naturally defined by considering the usual recursive tree layout where the root is drawn at the top of the diagram and the rest of the tree is hung below the root, as shown in Figure 1. Finally, by
In the course of analysing the algorithms, it is necessary to refer to paths within the tree during routing. Consider a packet $p$ that is routed on a rooted tree. We define $M(p,t)$ to be the path from the node containing $p$ to the root node at the end of step $t$ of the algorithm. Note that $M(p,t)$ includes the node containing $p$. $M(p,0)$ denotes the initial path from $p$ to the root of the tree.

In this paper, we assumed that the tree topology is known in advance and this knowledge is used in preprocessing. The following two lemmata (considered to be part of the folklore) are used in our algorithms:

**Lemma 1** A tree $T$ of maximum degree $d$ can be edge-coloured with exactly $d$ colours. Moreover, a valid edge colouring can be computed in linear time.

**Lemma 2** Consider an $n$-node tree $T$. There exists a node $r$ of $T$ such that each tree of $T\{r\}$ has at most $\lceil n/2 \rceil$ nodes. Moreover, node $r$ can be identified in linear time.

In an on-line algorithm the decision about whether or not to exchange the packets at the endpoints of a given edge usually involves comparing the packet. The edges over which comparisons occur on a given step are said to be active on that step.

The decision on whether to exchange or not the packets at the endpoints of an active edge as well as the actual exchange can be both implemented with one routing step as follows: both nodes at the endpoints of the active edge send the packets they hold to the node at the other side of the edge while, at the same time they also keep a copy of their own packet. Then, the two packets are available for comparison (according to some ordering criterion) at both nodes and thus a consistent decision can be made regarding which packet is kept at each node and which packet is disregarded.

### 3 On Line Heap Construction

We present an algorithm that heap orders a rooted tree $T$ of height $h$ and maximum degree $d$. The algorithm completes the heap construction in $2dh$ routing steps and its
correctness is proved based on potential function arguments. A fine-tuned version of the algorithm saves \( h \) additional steps, thus completing the routing in \((2d - 1)h\) routing steps.

3.1 The Heap Construction Algorithm

The algorithm proceeds in cycles of length \( d \). At alternative cycles, alternative-level edges become \textit{potentially active}. During any cycle, each of the potentially active edges that are incident to any node become active exactly once, each at a different step. So, at each time step, the set of active edges forms a matching and exchanges that bring the smallest keys closer to the root take place over these active edges. The algorithm in detail is as follows:

\begin{algorithm}
\textbf{Heap}(T)
/* \( T \) is a rooted tree of height \( h \) and maximum degree \( d \). */

(i) For each node \( v \) of \( T \). label the \( ch(v) \) edges incident to \( v \) which connect it to its children with \textbf{distinct} labels in \( \{0, \ldots, ch(v) - 1\} \).

(ii) \textbf{cycle} = 1
\hspace{1cm} \textbf{t} = 1

(iii) While \textbf{cycle} \leq 2h do
\hspace{1cm} (a) During odd cycles, let the set of \textit{potentially active} edges consists of all level-\((h-i)\)
\hspace{1cm} edges, for all odd integers \( i \) in \( \{1 \cdots h\} \), while, during even cycles, let the set of
\hspace{1cm} \textit{potentially active} edges consists of all level-\((h-i)\) edges, for all even integers \( i \)
\hspace{1cm} in \( \{1 \cdots h\} \).
\hspace{1cm} /* Note that during the first cycle level-\((h-1)\) edges become active. */
\hspace{1cm} (b) While \( t \leq d \cdot \textbf{cycle} \) do
\hspace{2cm} - Out of all potentially active edges, let the edges with label congruent to \( t \ mod \ d \) be \textbf{active}.
\hspace{2cm} - For all active edges, if the exchange of the keys at the endpoints of the edge results to getting the lower key closer to the root, perform the exchange; otherwise, do nothing.
\hspace{2cm} \hspace{1cm} \textbf{t} = \textbf{t} + 1
\hspace{1cm} (c) \textbf{cycle} = \textbf{cycle} + 1

3.2 Analysis of Algorithm Heap

In [10] an off-line heap construction algorithm was given. Algorithm \textit{Heap} is similar to the off-line algorithm of [10] except that it is on-line. However, it is possible that the heap ordering given by Algorithm \textit{Heap} will be different to that given by the algorithm of [10] when they are run on the same problem instance.

From the statement of the algorithm, it is obvious that it terminates after exactly \( 2dh \) routing steps. Thus, we need to show that within \( 2dh \) routing steps the tree has been heap ordered. By using the techniques of [10] we develop an analysis of Algorithm \textit{Heap}.
Fig. 2. The path $M(p,t)$ for an arbitrary packet $p$. The packets with keys greater than the key of $p$ are coloured black. All other packets are coloured white. Block $W^t_0$ can be empty.

The analysis relies on the combinations of colouring and potential function arguments. We choose an arbitrary packet $p$ and colour each packet of $T$ white or black using a simple scheme that depends on $p$. Each packet of $T$ that has a larger key than the key of $p$ is coloured black. All other packets (including $p$) are coloured white. By proving statements about the movement of these white and black packets a bound can be placed on the time that it takes for $p$ to reach its final position.

Consider the path $M(p,t)$ from the root to $p$ for any step $t$. It consists of alternating blocks of black and white packets. We shall refer to these as black blocks and white blocks. In any block we refer to the packet which is closest to the root as the first packet of the block. In any block the packet which is furthest from the root is referred to as the last packet of the block.

Let the blocks be labelled as shown in Figure 2. As can be seen the blocks are labelled in order, starting at the root and moving down $M(p,t)$. $g(t)$ denotes the number of black blocks in $M(p,t)$. The white blocks are labelled $W^t_i$ starting with block $W^t_0$ (which can be empty) which is closest to the root and continuing down to $W^t_{g(t)}$ which is furthest from the root and contains packet $p$. The black blocks are labelled $B^t_i$ starting with block $B^t_1$ which is closest to the root and continuing down to $B^t_{g(t)}$ which is furthest from the root.

In the analysis we use the notation $|X|$ to refer to the number of packets in block $X$.

As mentioned previously, the algorithm proceeds in cycles of $d$ steps. Consider the path $M(p,t)$ from the node of $p$ to the root at the end of step $t$. As time proceeds black packets move down the path or off the path until eventually at some time $t'$, $(t' \geq t)$ the path from the node of $p$ to the root contains only white packets. If all packets on $M(p,t')$ are white then all packets on $M(p,t')$ (including $p$) are in the block $W^t_{g(t')}$. The aim of the analysis is to bound the time that it takes for this to happen and use this to bound the running time of Algorithm $Heap(T)$.

**Lemma 3** Consider the path $M(p,t)$. Suppose that $M(p,t+1)$ contains a black packet $x$ that was not in $M(p,t)$. Then during step $t+1$, $x$ swapped with a black packet that was on $M(p,t)$.

**Proof.** Packets move by exchanges. No black packet can move towards the root by exchanging with a white packet since black packets by definition have keys larger than white packets. $\Box$

**Lemma 4** Let $p$ be an arbitrary packet in $T$ routed by algorithm Algorithm $Heap(T)$. Consider time step $t = kd$, $(k \geq 1)$ (the end of the last step of cycle $k$) and an arbitrary
black block $B_i^t$, $(1 \leq i \leq g(t))$. Let $L$ denote the node that contains the last packet of block $B_i^t$ ($L$ does not vary when we refer to different time steps). It holds that at the end of cycle $k + 1$ (i.e., at the end of step $t + d$), node $L$ contains a white packet.

**Proof.** Let $L'$ refer to the node that is immediately below node $L$ on the path $M(p, t)$. $L$ and $L'$ do not vary when we refer to different time steps.

On the first cycle of Algorithm $Heap(T)$ (i.e., for $0 \leq t \leq d$) it might happen that none of the edges below $L$ become active (The algorithm might have started with the child edges of this level inactive). This is why the lemma is stated so that it only applies to cycles beyond the first.

Consider what happens during cycle $k + 1$, $(k \geq 1)$ of the algorithm. Consider the nodes $L$ and $L'$ at the end of step $t = kd$ (the last step of cycle $k$). By definition, at the end of step $t$, node $L$ has a black packet in it and node $L'$ has a white packet in it. Accordingly, if the child edges of node $L$ are allowed to be active during cycle $k + 1$ then a swap occurs.

Assume for the sake of contradiction that the child edges of $L$ are not allowed to be active during cycle $k + 1$. Since the algorithm makes the child edges of alternate levels active on alternate cycles it holds that the child edges of $L$ were active during cycle $k$. We know that at the end of cycle $k$, node $L$ contains a black packet and node $L'$ contains a white packet. Lemma 3 implies that there was no exchange across the edge between $L$ and $L'$ during cycle $k$, since a black packet cannot move towards the root by swapping with a white packet. Hence at the beginning of cycle $k$, node $L$ contains a black packet and node $L'$ contains a white packet. However, the edge between $L$ and $L'$ is active during cycle $k$ and an exchange has to occur. This is a contradiction since the exchange did not take place. $\blacksquare$

**Lemma 5** Let $p$ be an arbitrary packet in $T$. Consider time step $t = kd$, $(k \geq 1)$ (the end of the last step of cycle $k$). Assume that the number of black blocks at the beginning of cycle $k + 1$ is the same as the number at the end $(g(t) = g(t + d))$. Then at least one of the following holds:

- The number of black packets in $M(p, t)$ is greater than the number of black packets in $M(p, t + d)$, or
- $|W_d^{t+d}| \geq |W_d^t| + 1$

**Proof.** The proof is by contradiction. Assume for the sake of contradiction that the number of black packets in $M(p, t + d)$ is the same as the number of black packets in $M(p, t)$ and that $|W_d^{t+d}| = |W_d^t|$.

Consider the blocks $B_i^t$, $(1 \leq i \leq g(t))$ on $M(p, t)$. Let $L_i$ denote the node that contains the last packet of block $B_i^t$ and let $L'_i$ denote the node that is immediately below node $L_i$ on the path $M(p, t)$. $L_i$ and $L'_i$ do not vary when we refer to different time steps.

By definition, each of the nodes $L_i$, $(1 \leq i \leq g(t))$ contains a black packet at the end of step $t$ and each of the nodes $L'_i$ contains a white packet at the end of step $t$. Accordingly,
by Lemma 4 each of the nodes \( L_i \) contains a white packet at the end of step \( i + d \). We assumed that the number of black packets on \( M(p, t) \) was the same as the number of black packets on \( M(p, t + d) \). Let \( q \) be any black packet that is on \( M(p, t) \). Suppose that \( q \) swaps with a white packet on some step \( t' \), \( t \leq t' \leq t + d \) during cycle \( k + 1 \). By Lemma 3, once a black packet has left a node there is no way that it can return except by swapping with a black packet. Therefore \( q \), or the black packet that was swapped with it, must be on \( M(p, t') \) as otherwise the number of black packets on \( M(p, t + d) \) would be less than the number of black packets on \( M(p, t) \). This implies that \( W^t_{g(t)} \) contains at least two packets. Also, we assumed that \( |W^{t+d}_0| = |W^t_0| \). If \( B'_1 \) contains only one packet then by Lemma 4 it would hold that \( |W^{t+d}_0| > |W^t_0| \). Therefore \( B'_1 \) contains at least two packets.

Since both \( B'_1 \) and \( W^t_{g(t)} \) contain at least two packets it holds that \( g(t + d) > g(t) \). This contradicts the assumption that \( g(t) = g(t + d) \) which was made in lemma.

Lemma 6  Let \( p \) be an arbitrary packet in \( T \). Consider time step \( t = kd \), \( k \geq 1 \) (the end of the last step of cycle \( k \)). Assume that \( g(t + d) = g(t) - \lambda \), \( \lambda > 0 \). Then at least one of the following holds true.

- There are at least \( \lambda + 1 \) more black packets present in \( M(p, t) \) than in \( M(p, t + d) \), or
- There are \( \lambda \) more black packets present in \( M(p, t) \) than in \( M(p, t + d) \) and \( |W^{t+d}_0| \geq |W^t_0| + 1 \)

Proof. Assume for the sake of contradiction that one of the following statements hold.

(i) There are at most \( \lambda - 1 \) more black packets present in \( M(p, t) \) than in \( M(p, t + d) \), or

(ii) There are at most \( \lambda \) more black packets present in \( M(p, t) \) than in \( M(p, t + d) \) and \( |W^{t+d}_0| \leq |W^t_0| \).

Notice that the disjunction of the above two statements is the complement of the disjunction of the two statements in the lemma.

Consider the blocks \( B'_i \), \( (1 \leq i \leq g(t)) \) on \( M(p, t) \). Let \( L_i \) denote the node that contains the last packet of block \( B'_i \) and let \( L'_i \) denote the node that is immediately below node \( L_i \) on the path \( M(p, t) \). \( L_i \) and \( L'_i \) do not vary when we refer to different time steps.

By definition each of the nodes \( L_i \) \( (1 \leq i \leq g(t)) \) contains a black packet at the end of step \( t \) and each of the nodes \( L'_i \) contains a white packet at the end of step \( t \). Accordingly, by Lemma 4 each of the nodes \( L_i \) contains a white packet at the end of step \( t + d \). Taking this into account, we consider the following two cases.

(i) \( B'_i \) is a single packet block. Then \( |W^{t+d}_0| \geq |W^t_0| + 1 \) and so to contradict the lemma we must assume that there are at most \( \lambda - 1 \) more black packets present in \( M(p, t) \) than in \( M(p, t + d) \). This assumption and Lemma 4 imply that \( g(t + d) \geq g(t) - (\lambda - 1) = g(t) - \lambda + 1 \). This contradicts the assumption that \( g(t + d) = g(t) - \lambda \) which was made in the lemma.
(ii) $B_1^t$ contains at least 2 black packets. In this case, if a white packet exchanges with the first packet of $B_1^t$ at some time $t'$, $(t < t' \leq t + d)$ then $|W_0^{t+d}| \geq |W_0^t| + 1$ and the argument is the same as for the previous case. Accordingly, to contradict the lemma we must assume that no white packet swaps with with the first packet of $B_1^t$ during cycle $k + 1$. We also know that in any case, to contradict the lemma we must assume that there can be at most $\lambda$ more black packets present in $M(p, t)$ than in $M(p, t + d)$. However, because $B_1^t$ contains at least 2 black packets, our assumptions and Lemma 4 imply that $g(t + d) \geq (g(t) + 1) - \lambda$. This contradicts the assumption that $g(t + d) = g(t) - \lambda$ which was made in the lemma.

Theorem 7 Consider an $n$-node tree $T$ with maximum degree $d$ and height $h$. Algorithm Heap($T$) heap orders $T$ within $2th$ steps.

Proof. The proof is achieved using a potential function argument. Let $p$ be an arbitrary packet and let all packets be coloured relative to $p$. We define a potential function $\Phi(p, t)$ which gives the potential of a packet $p$ at the end of step $t$ of the routing. The magnitude of $\Phi(p, t)$ corresponds to how well sorted the path $M(p, t)$ is, relative to packet $p$. By proving that $\Phi(p, t)$ is strictly decreasing we are able to place a bound on the running time of Algorithm Heap($T$).

We begin by defining a quantity $m_p$ which is used to define $\Phi(p, t)$. Consider the first time step $t'$ for which $M(p, t')$ consists only of white packets. $m_p$ is defined as the number of white packets (including $p$) on $M(p, t')$. Note that $m_p = |W_0^{t'}|$. Given this definition of $m_p$, $\Phi(p, t)$ is defined by the following equation:

$$\Phi(p, t) = \max(m_p, |W_0^t|) - |W_0^t| + \sum_{i=1}^{g(t)} |B_i^t| - g(t)$$

Note that if $g(t) = 0$ then there are no black blocks and so the term $\sum_{i=1}^{g(t)} |B_i^t|$ is zero.

We prove that each of the following claims hold for any packet $p$.

1. $\Phi(p, t) \geq 0$ for $t \geq 0$
2. If $\Phi(p, t) = 0$, $(t \geq 0)$ then $\Phi(p, t + 1) = 0$
3. If $t = kd$ and $\Phi(p, t) > 0$, $(k \geq 1)$ then $\Phi(p, t + d) \leq \Phi(p, t) - 1$
4. $\Phi(p, d) \leq 2h - 1$

These claims allow us to make the following deductions. By Claim 4 each packet has a potential of at most $2h - 1$ at the end of the first cycle. By Claim 3 the potential of each packet drops by at least 1 for every cycle of routing beyond the first. By Claim 2 the potential remains at zero once it has reached zero for the first time. By Claim 1 the potential cannot drop below zero. Hence, using all of these claims, it holds that the potential of all packets will be zero after at most $2h$ cycles, and will remain at zero. The proof of the theorem is completed by showing that when the potential of all packets is 0 the tree is heap ordered.
We proceed with the proofs of the above claims:

**Claim 1** \( \Phi(p, t) \geq 0 \) for \( t \geq 0 \)

**Proof.** For any \( t \geq 0 \) it holds that \( \max(m_p, |W_0'|) - |W_0'| \geq 0 \). Also \( \sum_{i=1}^{g(t)} |B_i'| - g(t) \geq 0 \) since there cannot be more black blocks than there are black packets. \( \blacksquare \)

**Claim 2** If \( \Phi(p, t) = 0 \), \((t \geq 0)\) then \( \Phi(p, t + 1) = 0 \)

**Proof.** If \( \Phi(p, t) = 0 \) then \( \max(m_p, |W_0'|) = |W_0'| \). Therefore all packets on the path \( M(p, t) \) are white. Accordingly, there are no black blocks and \( |B_i'| = g(t) = 0 \). By Lemma 3 black packets cannot move upwards by swapping with white packets. Therefore all packets on the path \( M(p, t + 1) \) are white, \( \max(m_p, |W_0'|) - |W_0'| \) is non increasing, therefore \( \Phi(p, t + 1) = 0 \). \( \blacksquare \)

**Claim 3** If \( t = kd \) and \( \Phi(p, t) > 0 \), \((k \geq 1)\) then \( \Phi(p, t + d) \leq \Phi(p, t) - 1 \)

**Proof.** If \( g(t + d) > g(t) \) then the claim holds as it is not possible for the sum of the remaining terms to increase. If \( g(t + d) = g(t) \) then by Lemma 5 either \( \sum_{i=1}^{g(t+d)} |B_i'| \) is less than \( \sum_{i=1}^{g(t)} |B_i'| \) or \( \max(m_p, |W_0'| + |W_d'|) \) is less than \( \max(m_p, |W_0'| - |W_d'|) \) or both. Therefore, if \( g(t + d) = g(t) \) then \( \Phi(p, t + d) \leq \Phi(p, t) - 1 \). In the same way, if \( g(t + d) = g(t) - \lambda \) then by Lemma 6, \( \Phi(p, t + d) \leq \Phi(p, t) - 1 \). Hence in every case the claim holds. \( \blacksquare \)

**Claim 4** \( \Phi(p, d) \leq 2h - 1 \)

**Proof.** Recall that the heap construction algorithm starts so that the level-\((h - 1)\) edges are active during the first cycle. This is done so that packets in level-\(h\) nodes do not have to wait until the second cycle to get a chance to rise. This ensures a better time bound.

Consider the potential at the end of the first cycle (time \( t = d \)) for any packet \( p \) that is at a level-\(i\) node where \( 0 \leq i < h \) (we deal with the case of packets that are at level-\(h\) nodes later). We observe the following facts about packets that are at level-\(i\) nodes at the end of step \( d \):

(i) \( \max(m_p, |W_0'|) \leq h + 1 \)
(ii) \( |W_0'| \geq 0 \)
(iii) \( \sum_{i=1}^{g(d)} |B_i'| \leq h - 1 \)
(iv) \( g(d) \geq 1 \) if \( \sum_{i=1}^{g(d)} |B_i'| \geq 1 \)

By these facts \( \Phi(p, d) \leq 2h - 1 \) for any such packet \( p \). Thus, it only remains to prove the claim for any packet \( p \) that is initially at a level-\(h\) node. The algorithm starts so that the level-\((h - 1)\) edges are active during the first cycle. Since \( p \) is at a level-\(h\) node at the end of step \( d \) we conclude that during the first cycle the packet above it was white. Therefore \( \sum_{i=1}^{g(d)} |B_i'| \leq h - 1 \). Further, we observe that all of the facts given above are still true. Hence it holds that for any packet \( p \), \( \Phi(p, d) \leq 2h - 1 \). \( \blacksquare \)
In conclusion, using all of the above claims it follows that for any packet $p$, $\Phi(p, 2dh) = 0$. Hence, in the colouring induced by packet $p$, all packets above $p$ on $M(p, 2dh)$ must be white. This means that for any packet $p$, all packets above $p$ on $M(p, 2dh)$ have keys less than or equal to the key of $p$. We conclude that Algorithm $\text{Heap}(T)$ heap orders $T$ within $2dh$ steps.

Observe that in a rooted tree of maximum degree $d$, the only node which can have $d$ children is the root of the tree. All other nodes can have at most $d - 1$ children. Thus, in our algorithm, only the cycles where the edges incident to the root, i.e., level-0 edges, are active need to last for $d$ steps. All other cycles need to last at most $d - 1$ steps. Thus, we can save one step for every alternative cycle, for a total savings of $h$ steps. Thus, Algorithm $\text{Heap}$ can be fine tuned such that it heap orders any tree within $2dh - h = (2d - 1)h$ steps. Thus, we have:

**Theorem 8** Any rooted tree of height $h$ and maximum degree $d$ can be heap ordered in $(2d - 1)h$ routing steps by a fine tuned version of Algorithm $\text{Heap}$.

Even though Algorithm $\text{Heap}$ terminates after exactly $2dh$ steps, for several distributions of the keys, the tree is heap ordered several steps before the termination of the algorithm. Thus, it is tempting to stop the execution of the algorithm earlier. The next theorem shows that this is not the case. More specifically, we present an instance of the heap ordering problem for which Algorithm $\text{Heap}$ completes the heap construction after $2d(h - 1)$ routing steps.

In our construction, we assume that the algorithm always selects the same edge labelling for the same tree “shape”. Thus, we assume a fixed labelling and we only have to provide an assignment of keys to nodes that forces the algorithm to heap order the tree slowly. For simplicity, we assume the original version of Algorithm $\text{Heap}$ (without the fine tuning that lead to Theorem 8.)

**Theorem 9** There exist trees $T$ and initial distributions of keys such that Algorithm $\text{Heap}(T)$ requires at least $2d(h - 1)$ steps to complete the heap ordering.

**Proof.** We prove the theorem by generalising the lower bound of [10] to on-line heap construction. The aim is to give a lower bound construction for which the algorithm will take at least $2d(h - 1)$ steps to complete the heap ordering.

Figure 3 shows an instance of the lower bound construction of height $h$. The construction consists of a tree in which the root node has degree $d$ and all other internal nodes have degree 3. Each node of $T$ starts with a packet. The packets are given keys as follows. Packets in the level $h$ leaves of $T$ are given keys $(h - 1), (h - 2), \ldots, 0$ in decreasing order from left to right. All other packets in $T$ are given a key of $h$ (these are the black packets that we initially have in the internal nodes). The edges of the all nodes that lead to their children are labelled from right to left in increasing order starting with 0. Note that this instance can be extended so that each internal node of the tree has degree $d$. Simply add
Fig. 3. An instance of the lower bound construction of height $h$. The key of each packet is printed in the node. All packets with key $h$ are coloured black. All other packets are coloured white.

$d - 3$ children to the nodes of degree 3 in Figure 3 and assign a key of $h$ to each of them. We avoided using this construction for the sake of simplicity.

The algorithm activates alternate-level edges on alternate cycles and each cycle takes $d$ steps. For any active edge a packet can move towards the root only by exchanging with a packet of larger key. Accordingly, the white packets move up towards the root but because of the way that the edges are numbered each white packet is delayed by other white packets that have smaller keys.

Let $t_i^l$ be the last step on which the packet with key $i$ is at level $l + 1$ on the backbone of $T$ (the path that consists of edges labelled “1”). Let $u$ be a node of $T$. Let $T_u$ denote the subtree of $u$. By examining the construction we observe that the first white packet to arrive at $u$ is the white packet of smallest key out of those white packets initially in $T_u$. We also observe that for any $i$, $(0 < i \leq h - 1)$ the packet with key $i$ will stay at level $i + 1$ until the packet with key $i - 1$ at level $i$ moves closer to the root. Since alternate levels are active on alternate cycles it follows that

$$t_i^l = t_{i-1}^{l-1} + d$$

Also, since each edge is active at most once in any cycle, $t_0^0 = d(h - 1)$. Hence we have:

$$t_{h-1}^{h-1} = t_{h-2}^{h-2} + d$$
$$= t_0^0 + d(h - 1) \quad (by \ repeated \ substitution)$$
$$= 2d(h - 1)$$
Heap construction cannot be completed until the packet with key $h - 1$ reaches level $h - 1$ for the first time. We conclude that since $t_{h-1}^{h-1} = d(2h - 1)$ the theorem holds. \hfill \Box

4 On Line Tree Routing

4.1 The Routing Algorithm

Routing on an arbitrary tree $T$ is performed in a recursive way. We designate a node $r$ of $T$ to be its root, and thus turn $T$ into a rooted tree. Then, we move all packets to their destination subtrees rooted at children of $r$, but not necessarily to their correct destination node. Lemma 2 guarantees that no subtree rooted at a child of $r$ has more than $\lceil n/2 \rceil$ nodes. Then, we complete the routing recursively by routing one permutation in each subtree. Note that the routing within all subtrees can proceed in parallel. The detailed algorithm follows:

**Algorithm** TreeRoute($T$)

/* $T$ is an arbitrary tree of maximum degree $d$ */

(i) Turn $T$ into a rooted tree by choosing a node $r$ to be its root such that each subtree rooted at each child of $r$ has at most $\lceil n/2 \rceil$ nodes (Lemma 2).
   Denote by $T_j$, $0 \leq j < ch(r)$ the subtree rooted at the $j$-th child of $r$.
(ii) For all subtrees $T_j$, $0 \leq j < ch(r)$ in parallel, do
   (a) Set the keys of those packets in $T_j$ that have destinations outside of $T_j$ to 0.
   (b) Run Algorithm Heap($T_j$)
(iii) Set the key of the packet at $r$ to 0
(iv) Run Algorithm TreeRoute($T$)
(v) For each $T_j$, $0 \leq j < ch(r)$ in parallel, run Algorithm TreeRoute($T_j$)

The following algorithm moves packets across the root into their destination sub-trees. It is used by Algorithm TreeRoute($T$) to achieve the routing for each level of recursion. It assumes that each subtree is heap ordered with respect to the keys assigned to the packets by step ii.a of Algorithm TreeRoute.

**Algorithm** Transfer($T$)

/* $T$ is a rooted tree with root $r$ and maximum degree $d$ */

(i) Label the edges of $T$ with labels in $0, \ldots, d - 1$ so that no two adjacent edges have the same label (Lemma 1).
(ii) $t = 1$
(iii) Mark the packet that is destined for the root $r$ with a *
(iv) While $t \leq (d - 1)|T| + 1$ do
    /* step $t$ of Algorithm Transfer($T$) */
    (a) Make all edges labelled with numbers congruent to $t \mod d$ active.
(b) For each active edge in parallel, do:
- Let \( e \) denote the active edge incident on the root \( r \). Let \( p \) denote the packet at \( r \) and let \( q \) denote the packet at the other end-point of \( e \). If \( p \) and \( q \) must both cross \( e \) to advance towards their destinations then set the key of \( p \) to 1 and exchange \( p \) with \( q \). If \( p \) is marked with a * (i.e. destined for \( r \)) and \( q \) must pass through \( r \) to reach its destination, then exchange \( p \) with \( q \). Otherwise, do nothing.
- For all other active edges, if a packet that has key 0 can be moved towards the root by exchanging with a packet that has key 1, or with the packet that is marked with a *, then do it. If the packet that is marked with a * can be moved towards the root by exchanging with a packet that has key 1, then do it. Otherwise, do nothing.

(c) \( t = t + 1 \)

4.2 Analysis of Algorithm TreeRoute

We first analyse Algorithm Transfer. We then bound the number of routing steps required by Algorithm TreeRoute(\( T \)).

4.2.1 Analysis of Algorithm Transfer(\( T \))

From the description of Algorithm Transfer(\( T \)) it follows that the algorithm terminates after exactly \( (d - 1)|T| + 1 \) steps. In this section we prove that Algorithm Transfer(\( T \)) transfers all packets to their correct destination subtrees within \( (d - 1)|T| + 1 \) steps.

For the purposes of the analysis we use a colouring argument. Let all packets with key 1 be coloured black and let all packets with key 0 be coloured white. Let the packet that is marked with a * (i.e. the packet that is destined for the root of \( T \)) be coloured red.

Consider an arbitrary white packet \( p \). As before, we use \( M(p, t) \) to denote the path from the root of \( T \) to the node that contains \( p \) at the end of step \( t \) of Algorithm Transfer(\( T \)) (\( M(p, 0) \) denotes the initial path). We also use the notion of above, below, up and down with respect to a given path or sub-tree. The following observations are based on the algorithm.

**Observation 1** A packet that was not black at the start of routing can only become black when it is in the root (step in.b of the algorithm). Black packets can only enter a node from above. White packets cannot move downwards, however, a white packet can become black and then move downwards (step in.b of the algorithm).

**Observation 2** The only way that the red packet can move upwards is by swapping with a black packet. However, a white packet can become black at the root and then move downwards by swapping with the red packet. The only way that the red packet can move downwards is by swapping with a white packet.

**Lemma 10** Consider any step \( t \geq 0 \) of Algorithm Transfer(\( T \)). Let \( p \) be an arbitrary
white packet in $T$. Consider the path $M(p, t_i)$. Let $x$ be a non-white packet that is on $M(p, t_i)$. Suppose that on step $t_i + 1$, $x$ is compared with a packet $x'$ that is immediately below it on $M(p, t_i)$. Then it holds that $x'$ is white.

**Proof.** For the purposes of this proof we define the notion of a *non-white comparison*. Consider the path $M(p, t)$ for some step $t \geq 0$. Suppose that during step $t + 1$ there is a comparison between a non-white packet that is on $M(p, t)$ and another packet that is also on $M(p, t)$. Then we say that a non-white comparison occurred on step $t + 1$ on the path $M(p, t)$. Many such non-white comparisons may occur simultaneously.

When Algorithm Transfer($T$) begins, all of the sub-trees rooted at children of $r$ are in heap-order and hence all packets on $M(p, 0)$ are white, with the possible exception of the red packet. As the algorithm proceeds, the contents of the path $M(p, t)$ change. To facilitate the proof argument we define the increasing sequence of time steps $S(p) = \langle t_1, \ldots, t_l \rangle$ which includes all time steps $t_i$ for which there are non-white comparisons at time $t_i + 1$ on the path $M(p, t_i)$, up to the step where $p$ enters the root of $T$. If the lemma applies for some packets $p$ and $x$ at some time $t$, then that means that there will be a non-white comparison on the path $M(p, t)$ on step $t_i + 1$. Accordingly, the proof is completed by induction on the terms of the sequence $S(p)$.

To prove the basis of the induction, we consider the first time step $t_1$ of $S(p)$. By Observations 1 and 2, black packets can only enter a node from above and the red packet can enter from below only by exchanging with a black packet. Accordingly, $M(p, t_1)$ only contains one non-white packet as otherwise $t_1$ would not be the first time step of the sequence $S(p)$. Therefore, when step $t_1 + 1$ is carried out this non-white packet will be compared with a white packet. Hence the lemma holds for step $t_1$.

![Diagram](image)

**Fig. 4.** The nodes $X$ and $X'$ on the path $M(p, t)$

Assume now that the lemma holds for each time step $t_i$, $(1 \leq i < l)$. We complete the induction by proving that it will hold for $t_{i+1}$. Consider the statement of the lemma at time $t = t_{i+1}$. Suppose that at the end of step $t_{i+1}$, packet $x$ is in node $X$ and packet $x'$ is in a node $X'$, immediately below $X$ on the path $M(p, t_{i+1})$. ($X$ and $X'$ do not vary when we refer to different time steps). Referring to Figure 4. let $b$ be labels of the edge
connecting nodes \( X \) and \( X' \). Now consider the step \( t_{i+1} + 1 \). By our assumption the edge with label \( b \) is active on step \( t_{i+1} + 1 \). There are two cases:

(i) The edge labelled \( b \) has not been active at any time before step \( t_{i+1} + 1 \). By Observations 1 and 2, a black packet can only enter a node from above and the red packet can enter from below only by exchanging with a black packet. Accordingly, it is not possible for node \( X' \) to contain a non-white packet at the beginning of step \( t_{i+1} + 1 \) as this would imply that edge \( b \) has been active before step \( t_{i+1} + 1 \). Therefore \( x' \) is white and the lemma holds for step \( t_{i+1} + 1 \).

(ii) The edge labelled \( b \) has been active before. The algorithm makes each edge active exactly once every \( d \) steps. So the last time that the edge labelled \( b \) was active was on step \((t_{i+1} + 1) - d\). Now assume for the sake of contradiction that \( x' \) is non-white. By Observations 1 and 2 it is not possible for a non-white packet move upwards by swapping places with a white packet. The edge that is labelled \( b \) is not active between step \((t_{i+1} + 1) - d\) and step \( t_{i+1} + 1 \). Therefore, if \( x' \) is in node \( X' \) at the beginning of step \( t_{i+1} + 1 \), then there was a non-white packet in node \( X' \) at the end of every step from \((t_{i+1} + 1) - d\) to \( t_{i+1} \) inclusive. Each child edge of node \( X' \) is active between step \((t_{i+1} + 1) - d\) and \((t_{i+1} + 1) + 1\). Since at least one child node of \( X' \) contains a white packet the induction hypothesis immediately implies that \( X' \) contains a white packet at the end of step \( t_{i+1} \). This is a contradiction of our assumption that \( x' \) is non-white. Therefore \( x' \) is white and the lemma holds for step \( t_{i+1} + 1 \).

**Corollary 11** Consider the root \( r \) of \( T \). Suppose that on step \( t \), \((t \geq 0)\), the edge leading to sub-tree \( T_i \), \((0 \leq i < k)\) is active. Then if the root of \( T_i \) contains a non-white packet at the beginning of step \( t \) it follows that \( T_i \) does not contain any white packets.

**Proof.** Let \( r_i \) denote the root of sub-tree \( T_i \). Suppose that \( r_i \) contains a non-white packet at the beginning of step \( t \). Assume for the sake of contradiction that \( T_i \) contains at least one white packet at the beginning of step \( t \). When Algorithm Transfer\((T)\) begins, \( T_i \) is in heap-order, hence if \( t \leq d \) and \( T_i \) contains at least one white packet then \( r_i \) contains a white packet at the beginning of step \( t \). This is a contradiction of our assumption that a non-white packet is at \( r_i \) at the beginning of step \( t \). Therefore, the corollary holds for \( t \leq d \). Accordingly, for the remainder of the proof we consider the case where \( t > d \).

By Observations 1 and 2 it is not possible for a non-white packet to move upwards by swapping places with a white packet. The edge between \( r_i \) and the root of \( T \) is not active between step \( t - d \) and step \( t \). Therefore, if \( r_i \) contains a non-white packet at the beginning of step \( t \), it must also contain a non-white packet at the end of every step from \( t - d \) to \( t - 1 \) inclusive. Each child edge of node \( r_i \) is active between step \( t - d \) and step \( t \). Since we assumed that \( T_i \) contains at least one white packet at the beginning of step \( t \), Lemma 10 implies that a white packet will enter node \( r_i \) at some time between step \( t - d \) and step \( t \). As a result the packet that is in \( r_i \) at the beginning of step \( t \) is white. This contradicts the premise that \( r_i \) contains a non-white packet at the beginning of step \( t \). We conclude that the corollary also holds for \( t > d \).

**Lemma 12** Algorithm Transfer\((T)\) routes all packets to their destination sub-trees within \((d - 1)|T|\) steps.
Proof. Consider the root $r$ of $T$ and its sub-trees $T_i$, $0 \leq i < ch(r)).$ When the red packet leaves a sub-tree $T_i$, the conditions of Corollary 11 are satisfied and there are no white packets in the sub-tree of $T_i$. Accordingly, the red packet cannot enter a sub-tree more than once. Therefore the red packet cannot enter $r$ more than $d - 1$ times since the routing pattern is a permutation.

We can make another deduction about the red packet. Suppose that the red packet is at $r$ at some step $t$ and the edge leading to sub-tree $T_i$ is active on step $t$. If the red packet does not enter $T_i$ on step $t$, then by Corollary 11 and the fact that the routing pattern is a permutation, it holds that routing to $T_i$ is complete. This implies that the number of steps that the red packet spends at $r$ without exchanging is at most $d$. Further, if the red packet spends $k \leq d$ steps at $r$ without exchanging then it can enter $r$ at most $d - k - 1$ times.

Consider a packet $x$ that is in $r$, $x$ remains in $r$ until the edge leading to its destination sub-tree $T_i$ is active. Because the routing pattern is a permutation there are white packets in sub-tree $T_i$. Therefore, by Corollary 11, $x$ will exchange with a white packet when the edge leading to $T_i$ is active. As a result we can make the following inferences:

(i) At least once in every cycle of $d$ steps, a packet enters $r$.
(ii) Any packet (apart from the red packet) that enters $r$ during a step $t$ will enter its destination sub-tree before step $t + d$.
(iii) No packet apart from the red packet can ever enter $r$ more than once.

By these inferences each packet (apart from the red packet) spends at most $d - 1$ steps in $r$, counting from the step that it enters $r$ up to but not including the step that it leaves $r$. Using the deductions that we made about the red packet earlier, we conclude that after $(|T| - 1)(d - 1) + (d - k - 1) + k = (d - 1)|T|$ routing steps, Algorithm Transfer has routed all packets to their destination subtrees. $lacksquare$

4.2.2 An upper bound for Algorithm TreeRoute($T$)

Theorem 13 Let $T$ be an $n$-node tree with maximum degree $d$. Algorithm TreeRoute($T$) can route any permutation on $T$ within $4dn$ steps.

Proof. The algorithm is recursive. When it is run on a sub-tree $T_i'$ it partitions $T_i'$ into sub-trees of size at most $|T_i'|/2$. Let $T_i^0$ denote the largest sub-tree that the algorithm runs on in the $i$-th level of recursion ($T_i^0 = T$). We know that $|T_i^0| \leq n/2^i$ and so there cannot be more than $\log n$ levels of recursion. Accordingly,

$$\sum_{i=0}^{\log n} h(T_i^0) \leq n/2 + \cdots + 1 < n$$

By Theorem 7, Algorithm Heap($T_i^0$) takes $2d \cdot h(T_i^0)$ steps. Also, by Lemma 12 we know
that Algorithm \( \text{Transfer}(T'_i) \) takes \((d - 1)|T'_i|\) steps to route all packets to their destination sub-trees. The cost of the \(i\)-th level of recursion is therefore bounded by
\[
2d \cdot h(T'_i) + (d - 1)|T'_i|
\]

We conclude that the total cost of the algorithm is bounded above by
\[
\sum_{i=0}^{\log n} [2d \cdot h(T'_i) + (d - 1)|T'_i|]
= 2d \sum_{i=0}^{\log n} h(T'_i) + (d - 1) \sum_{i=0}^{\log n} |T'_i|
= 2d \sum_{i=0}^{\log n} h(T'_i) + (d - 1)(n + n/2 + \cdots + 1)
< 4dn
\]

\[\text{Theorem 14 } \text{Let } T \text{ be a complete } d\text{-ary tree of } n \text{ nodes. Algorithm TreeRoute}(T) \text{ can route any permutation on } T \text{ within } 2(d - 1)n + 2d \log^2 n \text{ steps.}\]

\[\text{Proof. } \text{The proof is the same as that of Theorem 13 except that in the case of } d\text{-ary trees the sum of the heights of the sub-trees over all of the levels of recursion is bounded by } 2d \cdot \log^2 n \text{ rather than } 2dn:\]
\[
\sum_{i=0}^{\log n} h(T'_i) \leq \log n + \cdots + 1
< \log^2 n
\]
This gives a bound of
\[
\sum_{i=0}^{\log n} [2d \cdot h(T'_i) + (d - 1)|T'_i|] \leq 2d \log^2 n + (d - 1)(n + n/2 + \cdots + 1)
< 2(d - 1)n + 2d \log^2 n
\]

In [8,9] a lower bound for on-line matching routing was given. The result is repeated here for completeness. It relates to all on-line algorithms that repeatedly cycle through a fixed sequence of matchings.
Theorem 15 There exists an $n$-node tree of maximum degree $d$ and a permutation on it that requires $\Omega(dn)$ steps for its routing by any on-line algorithm that activates matchings in a fixed order.

5 Conclusion

In this paper, we presented the first on-line algorithms for heap construction and permutation routing on trees under the matching model. The algorithm for permutation routing is a recursive one and it is desirable to derive a non-recursive counterpart of it. The analysis of such algorithms appears to much more challenging than their design and deserves attention. One such algorithm that we were not successful in analysing consists of collapsing Algorithm Heap and Algorithm Transfer into a continuous flow of packets towards the leaves of the tree.
References


