Journal of Universal Computer Science, vol. 6, no. 2 (2000), 240-255
submitted: 15/7/00, accepted: 14/2/00, appeared: 28/2/00 *© Springer Pub. Co.*

Potential-function-based Analysis of an off-line Heap Construction Algorithm

Alan Roberts
(Department of Computer Science, University of Sydney, Australia,
alanr@cs.usyd.edu.au)

Antonios Symvonis
(Department of Computer Science, University of Sydney, Australia,
symvonis@cs.cs.usyd.edu.au)

Abstract: In this paper we examine the problem of heap construction on a rooted tree T from a packet routing perspective. Each node of T initially contains a packet which has a key-value associated with it. The aim of the heap construction algorithm is to route the packets along the edges of the tree so that, at the end of the routing, the tree is heap ordered with respect to the key values associated with the packets. We consider the case where the routing is performed according to the matching model and we present and analyse an off-line algorithm that heap orders the tree within $2h(T)$ routing steps, where $h(T)$ is the height of tree T. The main contribution of the paper is the novel analysis of the algorithm based on potential functions. It is our belief that potential functions will be the main vehicle in analysing fast non-recursive routing algorithms.

Key Words: Heap construction, matching routing model, off-line/on-line algorithms, packet routing, potential functions.

1 Introduction

The *packet routing problem* on a connected undirected graph $G = (V, E)$ can be defined as follows: We are given a set of packets, each packet p is initially located at node $origin(p) \in V$ and destined for node $dest(p) \in V$. Assuming synchronous communication, our task is to route all packet from their origin to their destination nodes in the smallest possible number of routing steps. Note that in this general definition of the routing problem, there is not any restriction on the number of packets that originate from, or, are destined for, a single node. In the case where each node of the graph is the origin and the destination of exactly one packet we have a *permutation routing problem.*

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 routine, 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 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.

During the routing, the movement of the packets follows a set of rules. These rules specify the *routing model*. Routing models might differ on the way edges are treated (unidirectional, bidirectional), the number of packets a node can receive/transmit 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.

In this paper, we assume the *matching routing model*, defined by Alon, Chung and Graham in [2]. When routing is performed under the matching routing model, the only operation allowed during the routing is the exchange of the packets at the endpoints of an edge of graph G. The exchange of the packets at the endpoints of a set of disjoint edges (a *matching* on G) can occur in a single routing step. The edges over which exchanges of packets take place at a given step, are said to be *active* during that step.

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 (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 with respect to the key-values they carry. Needless to say, we are interested in forming the heap in the smallest number of parallel routing steps when routing is performed according to the matching routing model.

The notion of the heap was introduced by Williams [15]. Floyd [7] described how to create a heap efficiently. More details can be found in Knuth's book [9]. Heaps are also discussed in the context of the Parallel Random Access Machine (PRAM) model [5, 13, 17]. Rao and W. Zhang [13], and W. Zhang and Korf [17] described how to construct a heap data structure (implemented as a complete binary tree) of n elements in $2\log_2 n$ steps.

From the description of the heap construction problem, it is evident that the problem is closely related to permutation routing on trees. Since initially each node has exactly one packet (key-value) and at the end exactly one packet (key-value) is located at each node, we actually route a permutation. The difference with the classical permutation routing problem is that we do not know at the beginning of the routing which is the permutation that we have to route! What we do know is that at the end of the routing the packets have to be located at nodes such that they satisfy the heap invariant property. Actually, for non-trivial trees, there definitely are several permutations of the packets that satisfy the heap invariant property. Even though at the beginning we do not know which of the permutations to route, we want to route the permutation that requires the smallest number of routing steps.

One general way to approach the heap construction problem (or, any other routing problem) is by routing packets in a "greedy" manner where progress is always made towards satisfying the heap invariant (or,in other words, towards the termination of the routing). This raises the question of how do we quantify the "progress" towards satisfying a property. One possible answer comes through potential functions. Let the *state* of routing at a given time-step be informally defined as the collection of the locations of all packets at that time step. Then, a *potential function* is an integer valued function of time, denoted $\Phi(t)$, which is defined based on the state of the routing at time step t. Assume that we have identified an integer valued function $\Phi(t)$ that satisfies the following properties:

1. $\Phi(0) \geq 0$
2. $\Phi(t+1) \leq \Phi(t) - 1, \ t \geq 0$
3. $\Phi(t) \leq 0$ implies that the routing has been completed.

The above relations imply that the routing finishes after at most $\Phi(0)$ routing steps. $\Phi(0)$ is referred as the *initial potential* and provides an upper bound on the routing performance of the algorithm. The challenge is to identify the potential function $\Phi()$ that satisfies the above relations and has as small as possible initial potential. Note that, the above descriptions of the potential function method is somehow simplified. The potential function can actually take real values and the potential might decrease by 1 in more that one routing steps (usually a constant number of steps).

In this paper, we present an off-line greedy algorithm which heap orders a tree T of height $h(T)$ in at most $2h(T)$ routing steps. We use an algorithm that can be considered to be an extension of the odd-even transposition sorting method [1, 8, 9]. The same algorithm was used for building a heap priority queue (i.e., a complete binary tree) in an EREW-PRAM environment [13, 17]. While the proof originally reported in [13] appears to generalise to arbitrary trees, we provide a proof based on potential functions. We consider this new proof to be of great importance. It is our belief that potential functions will be the main vehicle in analysing fast non-recursive routing algorithms and we treat the proof we present in this paper as the first step towards the analysis of greedy routing algorithms on trees.

We have just claimed that our proof based on potential functions will facilitate the analysis (through similar proofs) of greedy routing algorithms on trees. This claim raises two questions: Firstly, what is wrong with the existing algorithms for routing on trees (and their analysis) and, secondly, why is the problem of routing on trees so important to justify work on analysing algorithms using a special methodology (i.e., based on potential functions). These two questions are interrelated and we address them together in the rest of this introductory section. In the rest of the paper, unless otherwise specified, we assume that routing is performed according to the matching routing model.

It is known that there exist n-node trees for which there are permutations that require $\lfloor 3(n-1)/2 \rfloor$ routing steps under the matching model [2]. The algorithm presented by L. Zhang [16] routes any permutation in at most $3n/2 + \log n$ routing steps. Thus, an optimal algorithm (within a logarithmic additive term) is available and it can be claimed that the problem has been addressed in a satisfactory and conclusive way. This is certainly the case if "routing on trees" is considered in isolation and treated as a problem of academic only interest.

In practice, the problem of routing permutations on trees very rarely (if ever) arises, simply because real interconnection networks are more sophisticated and complicated than trees. Of course, we can argue that a routing algorithm on a tree is still useful since routing on an arbitrary interconnection network can be performed along the edges of one of its spanning trees. However, this approach is not viable since each edge of the tree becomes a potential bottleneck point which, in turn, results to the lower bound on the number of required routing steps dominated by the bisection of the tree rather than the diameter and the bisection of the real interconnection network.

Thus, realistic routing requires algorithms designed directly for the real interconnection network. Further practical considerations dictate that the algorithms should have a simple control structure, be able to handle dynamic situations (i.e., to allow for the creation of new packets while the routing takes place), be starvation free (i.e., guarantee delivery of each packet within a finite number of steps), etc. As a result of all of these requirements, several (usually greedy) routing schemes have been devised (under a variety of routing models) and it has been established through experimentation that they work well in practice. However, this observed behaviour is not usually accompanied by a matching analysis since the upper bounds that we are able to obtain are trivial and usually much worse than the observed behaviour. A classical example of this situation is routing under the hot-potato model where, even thought the basic routing model has been around since 1964 [3], the first non-trivial analysis of algorithms for that routing model was provided in 1992 by Feige and Raghavan [6].

Given that there is a gap to close between the theoretically predicted and the observed performance of several practical algorithms on real interconnection networks, the obvious question is how do we approach the problem of bridging that gap? Obviously, we should first study the routing problem on the simplest non-trivial form of network, i.e., a tree, and then benefit from our experience to extend the derived algorithms and analysis methods to general networks. Within this framework which treats the problem of routing on trees as a fundamental problem that will help to better understand and develop the techniques that will be used for the analysis of routing on realistic networks, it is easy to answer the question raised earlier in this introduction, i.e., "what is wrong with the existing algorithms for routing on trees?" The problem is that the existing algorithms are designed with a particular analysis in mind that explores properties of trees, and thus, are not extendable to more sophisticated networks.

It is revealing to study the literature of routing on n-node trees under the matching model. The first algorithm was provided by Alon, Chung and Graham in [2] and formed the basis of all algorithms that followed. The algorithm consists of three phases: Firstly, a special node is identified and it is designated as the root of the tree. This special node has the property that when it becomes the root of an n-node tree, all subtrees rooted at its children have at most $\lfloor n/2 \rfloor$ nodes. (The fact that such a node always exists is considered to be folklore in graph theory.) This first phase of the algorithm takes no routing steps, since it is assumed that each node of the tree (processor of the interconnection network) knows the whole tree topology. During the second phase, the algorithm routes all packets to a node in the subtree that contains their destination node (but not necessarily to their destination). At the end of this phase, all packets are in their correct subtree which consists of at most $\lfloor n/2 \rfloor$ nodes. As it is easy to guess, the third phase consists of recursively routing the packets to their destination node within the subtrees of size at most $\lfloor n/2 \rfloor$. An analysis based on recurrence relations results to an upper bound of $3n$ routing steps for permutation routing. The algorithm of Alon, Chung and Graham in [2] was improved by Roberts, Symvonis, and L. Zhang [14]. They managed to reduce the required number of routing steps to $2.3n$ by partially overlapping the two routing phases. Later on, L. Zhang [16] managed to route any permutation in at most $3n/2 + \log n$ routing steps by a clever partitioning of the tree (called caterpillar partition) and, again, by partially overlapping the routing phases. Krizanc and L. Zhang [10] used a similar approach to study the case where each node of the tree can be the destination of more that one packet and showed that the routing can be completed within $9n$ steps. Independently, Pantziou, Roberts and Symvonis [12] provided an algorithm that handles dynamic routing. A consequence of their analysis is that the same problem studied by Krizanc and L. Zhang can be solved in at most $3n$ routing steps. Note that all of the above algorithms are off-line routing algorithms.

The fact that all of the above algorithms, with the exception of the algorithm by Pantziou, Roberts and Symvonis [12], are based on a special partitioning of the tree makes it difficult to adapt them for other networks. However, they all possess another more disturbing characteristic, that is, they are all recursive. Recursive algorithms usually require a complex control structure when they are implemented in hardware and the situation becomes even worse when they have to perform a lot of non-routing related computations, such as computing the partition of the tree. The list of disadvantages gets longer if we also consider the fact that recursive algorithms are not particularly useful in situations where the routing is *dynamic*, that is, packets are generated and inserted into the routing, while the routing of other packets takes place.

We consider the problem of routing on trees as a fundamental problem, the study of which will facilitate the analysis of new and existing practical routing algorithms. We believe that one of the most promising methods is that of potential functions since that framework of analysis appears to successfully encompass the class of greedy routing algorithms. There are two more reasons that support our belief. Firstly, potential functions have already been used in the analysis of routing on trees. More specifically, all algorithms that follow the framework introduced by Alon, Chung and Graham [2] use potential functions to analyse the routing that takes place during the second phase, i.e., the routing of each packet to its destination subtree. The second reason for which we believe that potential functions will play an important role in the analysis of routing algorithms has to do with the heap construction algorithm that we analyse in this paper. When the algorithm is restricted to operate on a linear array, one of the simplest forms of a tree, it reduces to the odd-even transposition sorting method. Interestingly enough, if we adapt our analysis to take into account the fact that the routing takes place on a linear array, we match the upper bound obtained for the odd-even transposition sorting by totally different methods including induction [1, 8] and the zero-one principle [11].

A final point which deserves attention is the selection of the matching routing model. This routing model uses limited resources and at the same time is sophisticated enough (through the use of matchings) so that the resulting analysis of routing on trees should be non-trivial. The matching model is also attractive because of its relation with group theoretic problems. The problem of deciding what is the smallest number of routing steps required for the routing of a permutation is equivalent to determining the diameter of a *permutation graph* in which each state of the routing is represented by a vertex and (u, v) is a edge of the permutation graph if state v of the routing can be obtained in one routing step from state u (using the routing model under consideration).

Note also that it is likely that an analysis obtained for one routing model can be extended to the analysis under a different model through reductions. This was the approach used in [12] where basic techniques developed for the analysis of hot-potato routing by Borodin, Rabani and Schieber [4] were coupled with reductions to yield upper bounds on the performance of dynamic (off-line and on-line) routing algorithms on trees under a modification of the matching model.

In the preceeding paragraphs, we highlighted the importance of exploring the problem of routing on trees for the development and analysis of practical routing algorithms on arbitrary interconnection networks and we argued that potential functions can play a prominent role in the analysis of such algorithms. In this paper, we analyse, based on potential functions, an off-line algorithm for the heap ordering problem. The presesnted off-line algorithm can be transformed to an on-line one at the cost of a slowdown penalty that depends on the maximum degree of the tree. We consider our analysis to be a first step towards the fundamental task of analysis algorithms for routing on trees based on potential functions. Hopefully, the study and the extension of the obtained potential function will lead to similar potential functions that can be used, firstly, for the analysis of routing on trees and, subsequently, of routing on arbitrary networks.

The rest of the paper is organized as follows: In Section 2, we present the heap construction algorithm. In Section 3, based on potential function arguments, we show that the presented algorithm heap orders a tree T of height $h(T)$ in at most $2h(T)$ routing steps. In Section 4, we demonstrate an initial distribution of key-values on a tree that requires $2h(T) - 1$ steps for its heap ordering by our algorithm and thus, prove that our analysis is tight. We conclude in Section 5.

2 The Heap Construction Routing Algorithm

A *tree* $T = (V, E)$ is an undirected acyclic graph with node set V and edge set E. Throughout the paper we assume n-node trees, i.e., $|V| = n$. A tree T is *rooted* if one of its nodes, say r, is distinguished as its root. The *depth* $d_T(v)$ of node v is defined to be the distance (length of a shortest path) from the root r to v. The *height* of tree T, denoted by $h(T)$, is defined to be $h(T) = max_{v \in V} d_T(v)$. We say that a node v is a *level-i node* (or, at *depth-level i*) if $d_T(v) = i$. The root of the tree is a level-0 node. We say that an edge e is a *level-i edge* if it connects a level-i node with a level-$(i+1)$ node. All edges connected to the root r are level-0 edges. Figure 1 demonstrates the introduced terminology for nodes/edges on a complete ternary tree.

Consider a rooted tree T and let each node of T initially contain a packet. Moreover, let each packet have a *key-value* associated with it, with all key-values drawn from a totally ordered set (in this paper, the set of integers). Our objective is to route the packets (using the matching routing model) in such a way that, at the end of the routing, the tree is heap ordered with respect to the key-values of the packets at its nodes. It is not difficult to heap order T within $O((h(T))^2)$ routing steps. This is achieved by establishing the heap property in a bottom-up manner. (If we assume that all subtrees rooted at children of node x are heap ordered then the subtree rooted at x, i.e., T_x, can be heap ordered in exactly $h(T_x)$ routing steps.) However, we can heap order a tree substantially faster. We describe an algorithm that completes the task in at most $2h(T)$ routing steps.

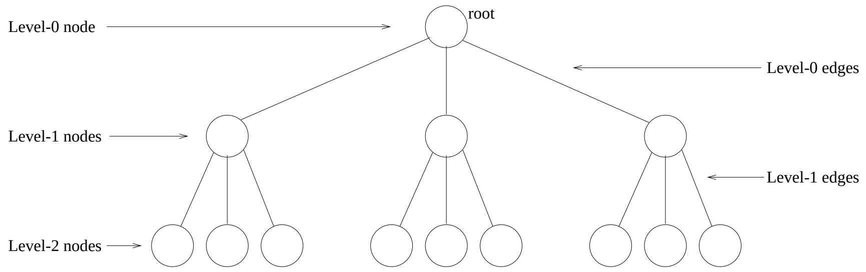

Figure 1: A complete ternary tree.

The algorithm we propose works for arbitrary rooted trees and it can be considered to be a generalisation of the odd-even transposition sorting method [1, 8, 9]. A similar algorithm was proposed in [13, 17] in the context of a EREW PRAM priority queue implemented by a complete binary tree.

Algorithm *Odd-Even_Heap_Construction*(T)
/* W.l.o.g., we assume that all key-values associated with the packets are distinct. */

1. Assign label $h(T) - i$ to each level-i edge, $0 \leq i < h(T)$.
2. $t = 1$
3. **While** $t \leq 2h(T)$ **do**
   (a) For any node u with edges connecting to its children labelled congruent to $t \bmod 2$, select out of the children of u the child, say v, that contains the packet of the smallest key-value. Order for a comparison between the key-values of the packets at u and v to take place at time t. If the key-value of the packet at v is smaller than the key value of the packet at u, a swap of the packets takes place.
   (b) $t = t + 1$

Algorithm *Odd-Even_Heap_Construction*() was classified as an off-line algorithm due to step 3(a) and the requirement that the edge connected to the node holding the minimum key value becomes active. A trivial on-line extension is possible but the number of routing steps increases by a factor of $2d_{\max} + 1$, where $d_{\max}$ is the maximum degree of the tree. With $2d_{\max}$ exchanges a node can inspect all the packets located at its children (at most $d_{\max}$) and decide which is the one with the smallest key-value. A final exchange, if necessary, brings the packet with the smallest key-value to the node. Note that this on-line extension, results to a final distribution of the packets at the nodes of the tree identical to that of the off-line algorithm.

The similarity of Algorithm *Odd-Even_Heap_Construction*() with the odd-even transposition method should be obvious. At odd (even) steps only odd (even) labelled edges can be *active*. However, because of the restrictions of the matching routing model, out of the potentially active edges that are connected to the same node, only one becomes active. One thing that remains to be clarified is why we labelled the edges in a bottom-up fashion instead of the traditional top-down. The reason becomes evident during the analysis of the algorithm.

3 Analysis of Algorithm *Odd-Even_Heap_Construction*()

Throughout this section, we assume that the movement of packets is according to algorithm *Odd-Even_Heap_Construction*(). Consider an arbitrary packet p. Let all packets in the tree which have larger key-value than p be coloured *black* and all other packets (including p) be coloured *white*. This colouring is done only for the purposes of the analysis. Note that it is specific to packet p.

Let $M(p,t)$ be the simple path which goes from the root of the tree to the node that holds packet p at the end of step t of the routing. Note that $M(p,0)$ denotes the path which goes from the root of the tree to the node that initially contains packet p.

Consider $M(p,t)$ for any $t \geq 0$. It consists of alternating blocks of black and white packets. We refer to them as *black blocks* and *white blocks*, respectively. We refer to the packet of a block that is closest to the root as the *first* packet of the block and to the packet of a block that is closest to p as the *last* packet of the block. The remaining packets of the block, if any, are referred as the *middle* packets of the block. Let the blocks be numbered as we walk away from the root as shown in Figure 2. Let W_i^t refer to the i-th white block away from the root, where counting begins at 0. Similarly, let B_i^t refer to the i-th black block away from the root, where counting begins at 1. By $g(t)$ we denote the number of black blocks in the path $M(p,t)$. Note that the first white block W_0^t may be empty. Note also that when all packets above p are white, all of the packets on the path, including p, are in the block W_0^t. By $|X|$ we denote the number of packets in block X.

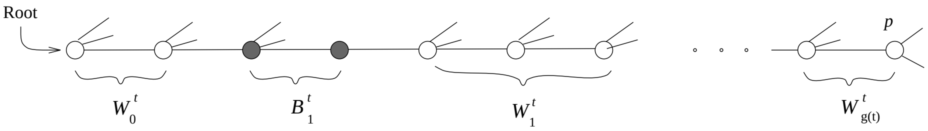

Figure 2: The figure shows a path $M(p,t)$ from the root of the tree to the node that contains packet p at the end of a particular step t. In general, W_0^t may be empty.

Lemma 1. *Consider the path $M(p,t)$, $t \geq 1$, and assume that it contains black packets. Consider any black block B_i^t, $1 \leq i \leq g(t)$. Then, one of the following occurs when step $t+1$ is carried out:*

i) The last packet of B_i^t moves towards p, or,

ii) the last packet of B_i^t is not present in $M(p,t+1)$.

Proof. We prove the lemma by induction on t. The only way that the conditions listed in the lemma do not occur for a given block B_i^t, $1 \leq i \leq g(t)$ when step $t+1$ is carried out, is if the last packet of that block is not compared with any packet immediately below it. As we show, the only time that this can happen is on the first step of the routing, i.e., $t = 0$. This is the reason that the lemma considers the case where $t \geq 1$.

Assume that the lemma is true for $t' = t-1$, $t' \geq 1$. Consider a black block B_j^t, $1 \leq j \leq g(t)$. There are two cases to consider:

1. B_j^t *is a single-packet block.* Denote the single packet in B_j^t by q. There are three sub-cases:
   (a) *q was the last packet of a multi-packet block $B_{j'}^{t-1}$, $1 \leq j' \leq g(t-1)$, at the end of step $t-1$.* By the induction hypothesis, q moved towards p and formed the 1 packet block. Since it moved towards p during step t, the edge immediately above the node currently holding q was active at time t. Thus, one of the edges immediately below that node is active during step $t+1$. So, q either moves closer to p or it moves off the path[1], i.e., it is not present in $M(p, t+1)$.
   (b) *q was the first packet of a 2-packet block $B_{j'}^{t-1}$, $1 \leq j' \leq g(t-1)$, at the end of step $t-1$.* One of the edges immediately below the node holding the last packet of the 2-packet block $B_{j'}^{t-1}$ was active during step t (by the induction hypothesis). This implies that is it now the turn of one of the edges immediately below the node holding q to be active. Thus, q either moves closer to p or it moves off the path, i.e., it is not present in $M(p, t+1)$.
   (c) *q was a middle packet of a multi-packet block $B_{j'}^{t-1}$, $1 \leq j' \leq g(t-1)$, at the end of step $t-1$.* For q to form a single-packet block at the end of step t, during step t the packet immediately above q in $M(p, t-1)$ must have moved off the path (i.e., it is not present in $M(p,t)$) and the packet immediately below q must have moved either closer to p or off the path (and thus, it is not present in $M(p,t)$). This implies that, at the beginning of step $t+1$, the packet immediately below q (in $M(p,t)$) is a white packet and also that one of the edges which connect the node holding q with the nodes immediately below it is active. Thus, q either moves closer to p or it moves off the path, i.e., it is not present in $M(p, t+1)$.
2. B_j^t *is a multi-packet block.* Because of the induction hypothesis, at the end of step $t-1$ the last packet of the block B_j^t, say q, was either the second-last packet of a block $B_{j'}^{t-1}$, $1 \leq j' \leq g(t-1)$, or it was a middle packet of the block and the packet immediately below it moved off the path (i.e., it is not present in $M(p,t)$). Since, in any case, the packet of block $B_{j'}^{t-1}$ immediately below q moved, the edges immediately below the node holding that packet were active at time t. This implies that the edges immediately below the node currently holding q are active during step $t+1$.

Because of the nature of our algorithm, there are blocks for which the lemma does not hold at $t = 0$. This is due to the fact that the last packets of these blocks are immediately above edges that are not active during the first step. However, we show that the lemma holds for the case of $t = 1$ (the basis of the induction).

Suppose that there was a block B_j^0, $1 \leq j \leq g(0)$, the last packet of which was not compared with any packet immediately below it during the first step. Suppose that this block becomes the j'-th block after the first step of routing has occurred. Then, because alternate edges are active on alternate steps, the last packet of block $B_{j'}^1$, $1 \leq j \leq g(1)$ is compared on step 2 of the routing with the packets that are immediately below it. These packets include at least one white

[1] Note that in the case that q was adjacent to p the two packets might be swapped. This is considered as moving off the path $M(p, t+1)$.

packet (the first packet of the white block $W^1_{j'}$, $1 \le j \le g(1)$). This concludes the proof.

Observation 2. *Consider the path $M(p,t)$, $t \ge 0$. No black packet joins the path except by swapping with a black packet. Therefore, the number of black packets that are on the path $M(p,t+1)$ is always less than or equal to the number of black packets that are on the path $M(p,t)$.*

Lemma 3. *Consider paths $M(p,t)$ and $M(p,t+1)$, $t \ge 1$, and assume that $g(t) = g(t+1) \ge 1$. Then, at least one of the following is true after step $t+1$ is carried out:*

i) There is at least one more black packet present in $M(p,t)$ than in $M(p,t+1)$.

ii) $|W^{t+1}_0| \ge |W^t_0| + 1$ (i.e., at least one white packet joins block W^t_0).

Proof. For the sake of contradiction, assume that the number of black packets in $M(p,t)$ is the same as the number of black packets in $M(p,t+1)$ and that no white packet joins block W^t_0 (i.e., $|W^{t+1}_0| = |W^t_0|$).

By Lemma 1, and since the number of black packets on the two paths remains unchanged, we know that when step $t+1$ is carried out, the last packet of each black block B^t_i, $1 \le i \le g(t)$, moves closer to packet p and that the white block $W^t_{g(t)}$ consists of at least two packets. Since no white packet joins block W^t_0, we also conclude that the first black block B^t_1 consists of at least two packets.

Consider the (possibly empty) sequence $[B^t_k, \ldots, B^t_{g(t)}]$, $k \le g(t)+1$, of consecutive single-packet black blocks[2]. The packets of these blocks move towards p and thus, they continue being single-packet blocks. Since $g(t) = g(t+1)$ and no black packet moves off the path, the last packet of each black block on the path at the end of step t becomes the first packet of a black block on the path when step $t+1$ is carried out. But the last packet of B^t_{k-1} is not able to join any (previously existing) black block. Note that block B^t_{k-1} must exist since we have established that block B^t_1 contains at least 2 black packets. This implies that $g(t+1) \ge g(t)+1$, a contradiction.

Lemma 4. *Consider paths $M(p,t)$ and $M(p,t+1)$, $t \ge 1$, and assume that $g(t+1) = g(t) - \lambda$, $\lambda > 0$. Then, at least one of the following is true after step $t+1$ is carried out:*

i) There are at least $\lambda+1$ more black packets present in $M(p,t)$ than in $M(p,t+1)$.

ii) There are at least λ more black packets present in $M(p,t)$ than in $M(p,t+1)$ and $|W^{t+1}_0| \ge |W^t_0| + 1$.

Proof. For the sake of contradiction, assume that there are at most $\lambda - 1$ more black packets present in $M(p,t)$ than in $M(p,t+1)$, or there are at most λ more black packets present in $M(p,t)$ than in $M(p,t+1)$ and $|W^{t+1}_0| = |W^t_0|$ (i.e., no white packet joined block W^t_0)[3]. Also recall that, by Lemma 1, when step $t+1$

[2] $[B^t_{g(t)+1}, B^t_{g(t)}]$ denotes the empty sequence.

[3] Note that this assumption is the logical negation of the statement of the lemma; as it is required in a proof by contradiction.

is carried out the last packet of each black block B_i^t, $1 \leq i \leq g(t)$, moves closer to p or it moves off the path. We consider two cases based on the number of packets of the first black block B_1^t.

1. B_1^t *is a single-packet block.* In this case, because the black packet in B_1^t moves closer to p or off the path, a white packet joins W_0^t. Thus, our assumption must be that at most $\lambda - 1$ more black packets are present in $M(p,t)$ than in $M(p,t+1)$. Then, the last packet of all but at most $\lambda - 1$ black blocks in path $M(p,t)$ becomes the first packet of a black block in path $M(p,t+1)$ when step $t+1$ is carried out. That implies that $g(t+1) \geq g(t) - (\lambda - 1) = g(t) - \lambda + 1$, a contradiction since we assumed that $g(t+1) = g(t) - \lambda$.
2. B_1^t *contains at least 2 black packets.* The situation where a white packet joining the path (replacing the first black packet of B_1^t) also joins W_0^t is covered by the proof given in the previous case. So, assume that the number of packets in W_0^t didn't change. Thus, our assumption must be that at most λ more black packets are present in $M(p,t)$ than in $M(p,t+1)$. In the case that the λ black packets which moved off the path were single-packet black blocks, consider the black block closest to p which had at least 2 packets, say B^t. Since all packets that moved off the path were single-packet blocks, the last packet of B^t forms a block of its own in $M(p,t+1)$. Thus, $g(t+1) \geq g(t) - \lambda + 1$, a contradiction since we assumed that $g(t+1) = g(t) - \lambda$. We also arrive at the same contradiction in the case where black packets from black blocks in $M(p,t)$ with at least 2 packets move off the path during step $t+1$.

This completes the proof.

Theorem 5. *Algorithm* Odd-Even_Heap_Construction() *heap orders any tree T in at most $2h(T)$ steps, where $h(T)$ is the height of the tree T.*

Proof. Consider an arbitrary packet p and the smallest t' such that path $M(p,t')$, $t' \geq 0$, consists only of white packets. Let m be the number of white packets (including p) in the path. (By definition, $m = |W_0^{t'}|$.)

Define the potential $\Phi(p,t)$ of packet p after t steps of routing, $t \geq 0$, as

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

For an arbitrary packet p we show that the following hold:

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

From the above conditions, it is easy to prove that the potential of all packets is equal to 0 after at most $2h(T)$ steps. We complete the proof by showing that when the potential of all packets drops to 0 the tree is heap ordered.

Claim 6. $\Phi(p,t) \geq 0, \ t \geq 0.$

Proof of Claim 6 Simply observe that for any $t \geq 0$ it holds that $\max(m, |W_0^t|) - |W_0^t| \geq 0$ and $\sum_{i=1}^{g(t)} |B_i^t| - g(t) \geq 0$ (there are at least as many black packets as the number of black blocks). ∎

Claim 7. *If $\Phi(p,t) = 0$ then $\Phi(p,t+1) = 0$, $t \geq 0$.*

Proof of Claim 7 Simply observe that if the potential is 0 at the end of step t, $t \geq 0$, then $|W_0^t| = \max(m, |W_0^t|)$ and $\sum_{i=1}^{g(t)} |B_i^t| = g(t) = 0$ (all packets in path $M(p,t)$ are white). Since the number of black packets in path $M(p,t)$ is non-increasing, we deduce that if $\Phi(p,t) = 0$ then $\Phi(p,t+1) = 0$. ∎

Claim 8. *If $t > 0$ and $\Phi(p,t) > 0$ then $\Phi(p,t+1) \leq \Phi(p,t) - 1$.*

Proof of Claim 8 If $g(t+1) > g(t)$ then $\Phi(p,t+1) \leq \Phi(p,t) - 1$ as it is not possible for the sum of the remaining terms to increase. If $g(t+1) = g(t)$ then, by Lemma 3, $|W_0^{t+1}| \geq |W_0^t| + 1$ and/or $\sum_{i=1}^{g(t)} |B_i^t| > \sum_{i=1}^{g(t+1)} |B_i^{t+1}|$. Therefore, if $g(t+1) = g(t)$ then $\Phi(p,t+1) \leq \Phi(p,t) - 1$. If $g(t+1) = g(t) - \lambda$, $\lambda > 0$, then by Lemma 4, $\sum_{i=1}^{g(t+1)} |B_i^{t+1}| - |W_0^{t+1}| \leq \sum_{i=1}^{g(t)} |B_i^t| - |W_0^t| - \lambda - 1$. We conclude that in every case, if $t > 0$ and $\Phi(p,t) > 0$ then $\Phi(p,t+1) \leq \Phi(p,t) - 1$. ∎

Claim 9. $\Phi(p,1) \leq 2h(T) - 1$.

Proof of Claim 9 First consider the potential at time $t = 1$ (after the first step of the routing) of all packets at level-i nodes, $0 \leq i < h(T)$. We have that: i) $\max(m, |W_0^1|) \leq h(T) + 1$, ii) $|W_0^1| \geq 0$, iii) $\sum_{i=1}^{g(1)} |B_i^1| \leq h(T) - 1$, and iv) $g(1) \geq 1$ if $\sum_{i=1}^{g(1)} |B_i^1| \geq 1$. By plugging the above inequalities to the potential function we get that $\Phi(p,1) \leq 2h(T) - 1$ for any packet p at a level-i node, $0 \leq i < h(T)$.

Consider now the potential at time $t = 1$ (after the first step of the routing) of any packet at a level-$h(T)$ node. Recall from the routing algorithm that the edge immediately above any level-$h(T)$ node is active during the first step of routing [4]. Therefore, since p is at a level-$h(T)$ node after the first routing step, we deduce that the packet immediately above it must be a white packet. This implies that in this case too, $\sum_{i=1}^{g(1)} |B_i^1| \leq h(T) - 1$. Since inequalities i), ii) and iv) still hold, we conclude again that $\Phi(p,1) \leq 2h(T) - 1$ for any packet p at a level-$h(T)$ node. This completes the proof. ∎

From the above facts it is easy to deduce that $\Phi(p, 2h(T)) = 0$ for any packet p. This implies that the tree is heap ordered. To see that, and for the sake of contradiction, assume that at time t the potential of all packets is 0 and that there are two adjacent packets p and q that violate the heap invariant, i.e., the key-value of p is larger than the key-value of q, and p is immediately above q. But p is a black packet in the path $M(q,t)$ and thus $\Phi(q,t) \geq 1$, a contradiction since we assumed that $\Phi(q,t) = 0$.

It is worth observing that the proof just completed provides an alternative correctness proof for the odd-even transposition method on linear arrays, as Corollary 10 indicates.

[4] This is why in Algorithm *Odd-Even_Heap_Construction*() the edges of the tree were labelled in a bottom-up fashion.

Corollary 10. *The odd-even transposition method sorts the elements of a linear array of n nodes within n steps.*

Proof. We first observe that the Algorithm *Odd-Even_Heap_Construction*() reduces to odd-even transposition sorting when carried out on a linear array. It therefore suffices to show that in the case of a linear array of n nodes, $\Phi(p,1) \leq n-1$ for any packet p. Once this has been done, the claims that were proven in Theorem 5 for a general tree, immediately imply the result.

Let the root be a node at one end of the array. Consider any packet p. We first observe that there are at most $n - \sum_{i=1}^{g(1)} |B_i^1|$ white packets. Therefore, $\max(m, |W_0^1|) - |W_0^1| \leq n - \sum_{i=1}^{g(1)} |B_i^1|$. We also observe that if $\sum_{i=1}^{g(1)} |B_i^1| = 0$ then $\max(m, |W_0^1|) = |W_0^1|$ and $g(1) = 0$. Thus, either $\Phi(p,1) = 0$ or, $\sum_{i=1}^{g(1)} |B_i^1| \geq 1$ in which case $g(1) \geq 1$. We conclude that in any case $\Phi(p,1) \leq n-1$.

The fact that step 3(a) of Algorithm *Odd-Even_Heap_Construction*() can be simulated in an on-line fashion with $2d_{\max}+1$ routing steps, allows us to state the following corollary:

Corollary 11. *A tree T can be heap ordered in an on-line fashion within $(4d_{\max}+2)h(T)$ routing steps where $d_{\max}$ is the maximum degree of tree T.* ∎

4 A Lower Bound for Algorithm *Odd-Even_Heap_Construction*()

In this section we show that the analysis provided for Algorithm *Odd-Even_Heap_Construction*() is tight. We provide an instance of the heap ordering problem that requires $2h(T)-1$ routing steps for its solution by our algorithm. Note that the provided lower bound applies to our algorithm only and is not a lower bound for the heap construction problem.

Theorem 12. *There exist trees and initial distributions of key-values on them for which Algorithm* Odd-Even_Heap_Construction() *needs $2h(T)-1$ routing steps, where $h(T)$ is the height of the tree that is to be heap ordered.*

Proof. We prove the theorem by demonstrating a tree T of height h and an assignment of key-values to the packets at its nodes for which Algorithm *Odd-Even_Heap_Construction*() terminates after $2h-1$ routing steps. Figure 3 shows such a tree and the assignment of key-values to its packets for the case that $h=4$. It is trivial to generalise the tree and the assignment for arbitrary $h>0$.

Let h denote the height of this tree structure in the general case. Let the packets at the leaves of the tree structure be assigned the distinct key-values $h-1,\ h-2,\ \ldots,\ 1,\ 0$ in decreasing order from left to right. Let all of the other packets in the tree (shown as black packets on the diagram) be given the same number k, where $k \geq h$. Let the levels of the tree be numbered as shown, from the root downwards in increasing order, beginning with 0.

Referring to Figure 3, let t_l^i denote the first step that a packet with key-value i reaches level l on the main backbone of the tree. We now observe that sorting cannot be completed until the packet with key-value i, $0 \leq i \leq h-1$ has reached

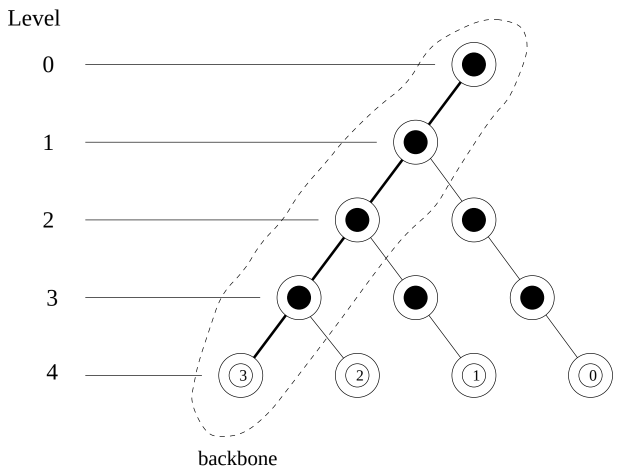

Figure 3: The tree structure and the assignment of key-values used in the proof of Theorem 12, for $h = 4$.

level i. We intend to prove the theorem by showing that with our algorithm, the packet with key-value $h-1$ reaches level $h-1$ for the first time on step $2h-1$.

Consider a node u of the tree during sorting. When deciding whether or not to advance a packet from one of the children of u into node u, the algorithm only considers the child that contains the packet of smallest key-value (referred as the "smallest first" rule). This implies that the packet with key-value $i-1$, $0 < i \leq h-1$, will always reach level i before the packet with key-value i does. It also implies that when the packet with key-value $i-1$ is at level i, the packet with key-value i is at level $i+1$. Furthermore, the "smallest first" rule implies that none of the packets with the h smallest key-values moves downwards in the tree on any step during the heap ordering.

Once the packet with key-value $i-1$ has reached level i, it does not move again until it moves "upwards" towards the root, because all packets that are below it in the tree have larger key-values. When it does move upwards from level i to level $i-1$, the packet with key-value i moves from level $i+1$ to level i of the backbone on the next step of the heap ordering. Therefore we have

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

as a direct consequence of the matching model and the "smallest first" rule. Also, from a distance bound we have $t_0^0 = h$. Combining these we get

$$\begin{aligned} t_{h-1}^{h-1} &= t_{h-2}^{h-2} + 1 \\ &= t_0^0 + h - 1 \\ &= 2h - 1 \end{aligned}$$

Sorting cannot be complete until the packet with key-value $h-1$ reaches level $h-1$ for the first time. We observe, however, that because none of the packets with the h smallest key-values move downwards in the tree on any step during the heap ordering, the algorithm terminates after step t_{h-1}^{h-1} has occurred. We conclude that the theorem holds.

5 Conclusions

In this paper we presented an off-line algorithm that heap orders a rooted tree T of height $h(T)$ in at most $2h(T)$ routing steps when the routing is performed under the matching routing model. The algorithm can be trivially extended to be on-line, for a slowdown of a factor of $2d_{\max} + 1$, where $d_{\max}$ is the maximum degree of the tree. As the most important contribution of the paper, we consider the analysis of the algorithms which is based on potential functions, since we believe that it will facilitate the analysis of greedy routing algorithms on arbitrary interconnection networks.

The long term goal of our research is the analysis of dynamic greedy routing algorithms and the development of an understanding of the quantities that influence the performance of the algorithms. The next step towards our goal is the development and the analysis of a greedy algorithm for routing permutations on trees under the matching model. The algorithms presented in [12] can be the starting point.

References

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

[2] Alon, Chung, and Graham. Routing permutations on graphs via matchings. *SIAM Journal on Discrete Mathematics*, 7:513–530, 1994.

[3] P. Baran. On distributed communication networks. *IEEE Trans. on Commun. Systems*, CS-12:1–9, 1964.

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

[5] N. Deo and S. Prasad. Parallel heap: An optimal parallel priority queue. *The Journal of Supercomputing*, 6(1):87–98, March 1992.

[6] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (Pittsburgh, Pennsylvania, October 24–27, 1992)*, pages 553–562, Los Alamitos-Washington-Brussels-Tokyo, 1992. IEEE Computer Society Press.

[7] R.W. Floyd. Algorithm 245: Treesort 3. *Communications of ACM*, 7:701, 1964.

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

[9] D.E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching.* Addison-Wesley, 1973.

[10] D. Krizanc and L. Zhang. Many-to-one packet routing via matchings. In *Proceedings of the Third Annual International Computing and Combinatorics Conference, Shanghai, China*, August 1997. To appear.

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

[12] G. Pantziou, A. Roberts, and A. Symvonis. Dynamic tree routing under the "matching with consumption" model. In T. Asano, Y. Igarashi, H. Nagamochi, S. Miyano, and S. Suri, editors, *Proceedings of the 7th International Symposium on Algorithms and Computation ISAAC '96 (Osaka, Japan, December 1996)*, LNCS 1178, pages 275–284. Springer-Verlag, 1996.

[13] N. Rao and W. Zhang. Building heaps in parallel. *Information Processing Letters*, 37:355–358, March 1991.

[14] A. Roberts, A. Symvonis, and L. Zhang. Routing on trees via matchings. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS'95), Kingston, Ontario, Canada*, pages 251–262. Springer-Verlag, LNCS 955, aug 1995. Also TR 494, January 1995, Basser Dept of Computer Science, University of Sydney.

[15] J.W.J Williams. Algorithm 232: Heapsort. *Communications of ACM*, 7:347–348, 1964.

[16] Louxin Zhang. Optimal bounds for matching routing on trees. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 445–453, New Orleans, Louisiana, 5–7 January 1997.

[17] W. Zhang and R.E. Korf. Parallel heap operations on an EREW PRAM. *Journal of Parallel and Distributed Computing*, 20(2):248–255, February 1994.