Nordic Journal of Computing

PARAMETRIC PERMUTATION ROUTING VIA MATCHINGS

PETER HØYER [*] AND KIM S. LARSEN [*]
Department of Mathematics and Computer Science
Odense University, Denmark
{u2pi,kslarsen}@imada.ou.dk

Abstract. The problem of routing permutations on graphs via matchings is considered, and we present a general algorithm which can be parameterized by different heuristics. This leads to a framework which makes the analysis simple and local.

CR Classification: F.1.2, F.2.2

Key words: permutation routing, matching, graph algorithm

1. Introduction

The routing problem we consider is the following: We are given an undirected connected graph with n nodes and a permutation π of the nodes. Each node u contains one packet which must be routed to $\pi(u)$. The routing is carried out in a sequence of steps. In one step, each packet can either remain at its current location, or it can be swapped with a neighbor. Thus, at all times each node has exactly one packet. We are interested in designing an algorithm for this problem with a low complexity measured in the number of steps necessary in the worst case to ensure that all packets are routed to their correct locations independent of the initial configuration.

This problem was first defined and investigated in Alon *et al.* [1993, 1994], and an upper bound of $3(n-1)$ for the problem was obtained. The improvement of this upper bound to $\frac{13}{5}n$ was obtained by a minor change in the algorithm combined with a more careful analysis of the algorithm Roberts *et al.* [1995]. Also in Roberts *et al.* [1995], upper bounds were derived for some special cases defined by the structure of the graphs: trees of bounded degree in $2n + o(n)$, trees of degree at most 3 in $2n$, and complete d–ary trees in $n + o(n)$.

In Høyer and Larsen [1996], the technical report version of the present paper, the upper bound for the general case was improved to $2n-3$ for $n > 1$. Independently, a $2n$ algorithm as well as a $\frac{3}{2}n + 9\log_3 n$ algorithm was developed in Zhang [1997]. Clearly, the latter bound is better than the one

[*] Supported in part by the ESPRIT Long Term Research Programme of the EU under project number 20244 (ALCOM-IT).

Received January, 1997; revised January 1998; accepted May 1998.

given in the present paper, but unfortunately it is based on complicated constructions and proofs.

The best lower bound reported is from Alon *et al.* [1993, 1994], where it is shown that the *star* consisting of n nodes requires $\lfloor \frac{3}{2}(n-1) \rfloor$ steps in the worst case.

We see the following as the two major problems remaining: First, the gap of an additive logarithmic term between the lower bound and the best upper bound from Zhang [1997] should be closed. Second, algorithms which can be parameterized with some kind of measures of the structure of a given graph should be developed. The special cases treated in Roberts *et al.* [1995], as described above, are examples of such measures of graph structure, but this could be generalized significantly. If there are cycles in the graph, for instance, this could be exploited to avoid the bottleneck problems present in all the general results developed so far.

In this paper, we do not solve any of these open problems, but we believe that our results can contribute to the solution of these. It is the hierarchical structure of our algorithm as well as the proofs that give this hope. In greater detail, our algorithm is parameterized by a heuristic. So, in fact, a number of algorithms can be designed, several of which can be shown to give an upper bound of $2n-3$. We discuss one such algorithm in the main part of the paper, and mention another in the conclusion. The algorithm is recursive and the recursive phases for subproblems start at different times. The analysis of the algorithm is local and is performed for each subproblem in isolation, despite of the fact that they may be of varying size and may be prevented from starting for a while because of other subproblems.

2. Preliminaries

The problem we set out to solve is the following. We are given an undirected connected graph with n nodes. We are also given a set of n *packets*. Each packet is associated with exactly one node in the graph, referred to as the packet's *destination node*. A *state* is a bijective map from the set of packets to the set of nodes. Given such a state, called the *initial state*, we must find a sequence of states beginning with the initial state and ending with the state, where each packet is mapped to its destination node. Any two consecutive states in the sequence must form a *step*. Two ordered states form a step if the latter can be obtained from the former by *applying a matching*. A *matching* for the graph is a set of node pairs such that any node is in at most one pair and any node pair in the matching are connected by an edge in the graph. The state s_2, which is the result of applying a matching to a state s_1, is defined as follows. For any packet p, if $s_1(p)$ is not in any of the pairs in the matching, then $s_2(p) = s_1(p)$, and if $s_1(p)$ is in the matching paired with $s_1(q)$, then $s_2(p) = s_1(q)$. The goal is to find a sequence of steps as short as possible, and this problem is normally referred to as *Permutation Routing via Matchings*.

Since we are only interested in the length of the sequence, and not in the complexity of deciding which step to apply when, the problem is an *off-line* problem. Traditionally, the number of steps (that is, the number of states minus one) is used as a measure of complexity, since this can be interpreted as the running time of a parallel machine or network carrying out one step at a time. Clearly, if the graph consists of at most one node, then the number of steps necessary is zero, so in the rest of this paper, we assume that $n \geq 2$.

Since there are no restrictions on which type of graphs we route on, all results in the literature are based on trees (since, if nothing else, routing can take place on a spanning forest of the graph). We assume without loss of generality that the graph is a rooted tree. The following result is folklore in graph theory.

PROPOSITION 1. *For any tree T with n nodes, there exists a node u in T such that each subtree T_i of size n_i formed from T by removing u (and all incident edges) satisfies $n_i \leq n/2$.*

In order to discuss and analyze these routing algorithms, some definitions are convenient. Suppose a node u has been chosen. Then a tree can be divided up into the node u and all the subtrees of u. Packets have destination nodes, and the one with u as its destination is called the *green packet*. If a packet has a destination node different from u, it belongs to a subtree, which is then referred to as the packet's *destination subtree*. Packets which are already in their destination subtree at a given point in time are called *home packets*. Other packets in that subtree, with the exception of the green packet, are called *foreign packets*. A subtree is *completed* if it contains all its home packets, and hence, no other packets.

Like other algorithms in the literature, the main idea of the algorithm is to select a node u, route packets to their destination subtrees in order to complete them, and then solve the problem recursively for the individual subtrees. One of the main elements of our approach, is that we can take advantage of the fact that recursion can be applied to a subtree as soon as it is completed, even though this may not hold for other subtrees.

One interesting question that arises is how fast the foreign packets can be moved out of a subtree, or how fast they can be made ready to move out. In Alon *et al.* [1993, 1994], subtrees were "heap-ordered" as the initial step in the algorithms. There, heap-ordering means placing all foreign packets in the top part of the subtree, i.e., for any foreign packet, there is no home or green packet between it and the root. We refer to this as *strong* heap-order. If the objective is merely to get foreign nodes out as fast as possible, this is not necessary. In fact, the general improvement in Roberts *et al.* [1995] was obtained by a relaxation of the heap-ordering requirements. We formalize this idea and refer to it as *weak* heap-order.

Let T_i be any subtree of u containing $\hat{f}_i$ foreign packets and $\hat{h}_i$ home packets. Then T is *weakly heap-ordered* if, for all $j \leq \hat{f}_i$, j foreign packets can be moved out of the subtree in $2j - 1$ steps, and all non-home packets can be moved out in at most $2\hat{f}_i + 1$ steps. The subtree T_i is thought of as being rooted at a neighbor of u. The following algorithm will weakly heap-order the subtree T_i.

ALGORITHM Weakly-Heap-Ordering-Algorithm (tree T_i)

(1) For all non-foreign packets p with a foreign packet among one of its children, match p with exactly one of these foreign packets.

(2) If the green packet is in the subtree, and if the parent of the node holding the green packet holds a home packet, and neither were matched in (1), then match that home packet with the green.

(3) Carry out the step defined by this matching.

(4) Repeat the above a total of $\hat{h}_i$ steps, adding an extra step if the green packet is in the subtree.

The following lemma appeared in Roberts *et al.* [1995].

LEMMA 1. *Let T be any tree on n nodes, and let u be any node satisfying proposition 1. Let $\hat{h}_i$ denote the number of home packets in subtree T_i. Then T_i can be weakly heap-ordered in at most $\hat{h}_i$ steps if T_i does not contain the green packet, and at most $\hat{h}_i + 1$ steps otherwise.*

PROOF. Omitted. A proof of the lemma can be found in Høyer and Larsen [1996] and in Roberts *et al.* [1995]. □

Weakly heap-ordering of subtrees is one of the important subroutines of the algorithm to be presented in the next section; the other is *cycles.* Cycles are intended to be carried out after all subtrees have been weakly heap-ordered. A cycle is a sequence of steps, the primary purpose of which is to route foreign packets in the roots of subtrees through u to their destination subtrees. However, since moving packets through u disturbs the weak heap-order of subtrees, subtrees are continuously being weakly heap-ordered in parallel with this. The purpose of keeping subtrees weakly heap-ordered is also to make cycles as long as possible, since this improves the overall complexity.

We first consider the actions involving u in the steps of the cycle. Afterwards, we define the rest of the matchings which ensure that subtrees are continuously being weakly heap-ordered. Assuming that the green packet is on u, the first step involves swapping the green packet with some foreign packet in the root of some subtree T_i. We say that T_i *initiates* the cycle. After this step, in each step, the packet p on u swaps with a packet in p's destination subtree. This continues until the green packet is again on u. If the first step is left out, we call it a *green cycle* (since, eventually, it places the green packet on u). The *length* of a cycle is the number of steps in the sequence.

We now define the remaining parts of the matchings in a cycle. In each step, the packet on u swaps with a packet in some subtree T_i with root v.

In such a step, the matching is defined as follows. Clearly, u is matched with v. For all subtrees formed by removing u and v (and all their incident edges), the matching is defined by rules 1 and 2 in WEAKLY-HEAP-ORDERING-ALGORITHM. This implies that after this step, all subtrees of u, with the exception of T_i, are again weakly heap-ordered. Carrying out this sequence of steps defined above is referred to as *executing* the cycle.

Among other things, we will prove that immediately after a cycle has been executed, all subtrees are again weakly heap-ordered. This result depends primarily on the fact that the tree which initiates the cycle gets the green packet. Due to the continuous weakly heap-ordering, the green packet will not get out of that subtree again until all foreign packets have come out. Thus, when the cycle ends, that subtree, the root of which is also the last node in the cycle to be matched with u, will be completed. This, as well as related results, is proven below, after we have presented the algorithm.

3. The Algorithm

In this section, we show that there exists a very simple algorithm that routes the packets in at most $2n - 3$ steps. The algorithm goes through four stages. For any fixed tree T, let u be any node in T satisfying proposition 1. In the first stage, for each subtree T_i formed by removing u, we route the packets in T_i in order to make them ready for the second and third stages. In these, we exchange packets between the subtrees according to some heuristic α such that the green packet is routed to u and all other packets are routed to their destination subtrees. Finally, in stage four, we recursively route the packets for each subtree.

Within this framework, a specific algorithm is obtained by specifying a set of rules for how to exchange the packets in stages two and three. We have tight upper and lower bounds for the running time of any such algorithm.

We allow such a set of rules to depend on the placements of the packets, i.e., in each step, the choice of a matching may depend on the instance of the permutation of the packets. We show that this does not improve the algorithm. The general algorithm is as follows.

```
ALGORITHM General-Routing-Algorithm (tree $T$)
(1) Let $u$ be any node in $T$ satisfying proposition 1. Weakly heap-order
    the subtrees of $u$.
(2) If the green packet is not on $u$, execute the green cycle.
(3) While there exists a subtree which is not completed, choose an incom-
    plete subtree $T_i$ according to the heuristic $\alpha$ and execute the cycle
    initiated by $T_i$.
(4) As soon as a subtree is completed, recursively route the packets in that
    subtree.
```

In order to analyze this algorithm, we need some basic facts on the cycles. A similar lemma is stated in Roberts *et al.* [1995] without a proof.

LEMMA 2. (THE CYCLE LEMMA) *Let u be any node in T satisfying proposition 1. If the subtrees of u are weakly heap-ordered, then the following properties hold:*

(1) By executing a non-green (green) cycle of length $k+1$ (k), k packets are routed to their destination subtrees.

(2) If the last cycle, if any, is not the green cycle, then it completes at least two subtrees.

(3) Suppose the green packet is on u. Then any cycle completes a subtree, and furthermore, for any incomplete subtree, there exists a cycle which completes it. In fact, for any incomplete subtree, there exists a cycle which completes that subtree and one more subtree.

(4) Immediately after any cycle, all incomplete subtrees are weakly heap-ordered.

PROOF. Property 1 clearly holds for a green cycle since in each step, the packet on u is routed to its destination subtree. For any non-green cycle, it is only in the first step that a packet is not routed to its destination subtree.

If the green packet is on u, there cannot be exactly one subtree containing foreign packets. Property 2 follows.

Suppose that the green packet is on u. Let T_i be any incomplete subtree. By letting T_i initiate a cycle, the green packet is moved from u into T_i and will not be moved out again before the step where T_i is completed. Note that, if this cycle has length $k+1$, then T_i is completed after exactly $k+1$ steps.

To prove the second part of property 3, let T_i be any incomplete subtree. Since T_i is not completed, it contains some foreign packet p which can be moved out of T_i as the last foreign packet in T_i. Assume p has destination subtree T_j, and consider a cycle initiated by T_j. Then T_j will be completed after this cycle, and therefore p will have been moved out of T_i, which implies that T_i has also been completed. Thus, T_i and one more subtree, T_j, are completed by the cycle. Note that, if the cycle has length $k+1$, then T_i is completed after at most k steps.

To prove property 4, let T_i be any incomplete subtree. Assume we swap the packet on the root of T_i and the packet on u. If T_i is completed by that step, then afterwards T_i is weakly heap-ordered by definition. Now, suppose T_i is not completed by the swap. Then T_i requires exactly one more step to restore its weak heap-ordering. By the proof of the first part of property 3, the cycle contains a next step, and that step will not involve the edge between u and the root of the subtree T_i. Hence T_i will, in that next step, restore its weak heap-ordering. □

3.1 Analysis of the general algorithm

In the rest of this section, let T be any fixed tree on n nodes, and let u be any node satisfying proposition 1. Immediately by lemma 2, for any heuristic α, the general routing algorithm runs in finitely many steps. Let S^j denote the number of steps needed to perform stage j ($j = 1, 2, 3$) in GENERAL-ROUTING-ALGORITHM. Let $S = S^1 + S^2 + S^3$ denote the number of steps needed to complete all of the subtrees of u.

We start by bounding the running time of the first two stages. Consider the permutation of the packets immediately after stage 2. For each subtree T_i, let n_i denote the size of T_i, and let h_i (f_i) denote its number of home (foreign) packets. Since the green packet is on u, we have $n_i = h_i + f_i$. Let $h = \sum_i h_i$ ($f = \sum_i f_i$) denote the total number of home (foreign) packets after stage 2. Observe that $n = h + f + 1$.

LEMMA 3. *Let r denote the degree of u. Let c denote the number of non-green cycles, i.e., the number of cycles executed in stage 3. Then*

$$\begin{aligned} S^1 + S^2 &\leq h + 1 \\ S^3 = f + c &\leq f + \min\{r-1, (n-1)/2\}. \end{aligned}$$

Furthermore, if for some subtree T_j of size n_j, $2n_j = n$ and T_j is not completed after stage 2, then $S^1 + S^2 \leq h$.

PROOF. The first inequality follows immediately from lemma 1 and property 1 in lemma 2. By property 2 and the first (and weak) part of property 3 in lemma 2, there can be at most $r - 1$ non-green cycles. Furthermore, since any non-green cycle routes at least two foreign packets to their destination subtrees, there can be at most $\lfloor (n-1)/2 \rfloor$ non-green cycles. The second inequality then follows from property 1 in lemma 2.

To prove the third and last inequality, assume T_j satisfies the conditions in the lemma. Since T_j is not completed after stage 2, T_j did not contain the green packet in the beginning of the algorithm. Therefore, since $n_j = n/2$, T_j did contain a home packet at the beginning.

Now, consider the running time of the first stage of the algorithm. Let $\hat{h_i}$ denote the number of home packets in subtree T_i at the beginning of the algorithm. Let $h^1 = h - S^2$ denote the total number of home packets after stage 1. Observe that $h^1 = \sum_i \hat{h_i}$ since the number of home packets does not change in stage 1. Since T_j did not contain the green packet, by lemma 1, T_j can be weakly heap-ordered in at most $\hat{h_j} \leq h^1$ steps. Furthermore, each other subtree T_i, $i \neq j$, can be weakly heap-ordered in at most $\hat{h_i} + 1 \leq (h^1 - 1) + 1 = h^1$ steps. Thus,

$$S^1 + S^2 \leq h^1 + S^2 = h,$$

proving the third and last inequality. □

We are now in a position to prove that for any heuristic α, all, except at most one, of the subtrees of u will be completed fast.

Lemma 4. *Let T_i be any of the r subtrees of u. For any heuristic α used in the general algorithm, if T_i satisfies any of the following two conditions,*

- *T_i is not the unique largest subtree of u, or*
- *$n_i \leq n/4$, where n_i is the size of T_i,*

then T_i is completed after at most $2(n - n_i)$ steps.

Proof. By lemma 3, the number of steps before all of the subtrees of u are completed is bounded from above by

$$\begin{aligned} S \;=\; S^1 + S^2 + S^3 \;&\leq\; (h+1) + f + \min\{r-1, (n-1)/2\} \\ &=\; n + \min\{r-1, (n-1)/2\}. \end{aligned}$$

Suppose that T_i is not the unique largest subtree, and let T_m be a largest subtree of u different from T_i. Since each of the subtrees of u contains at least one node,

$$n \;=\; \sum_j n_j + 1 \;\geq\; n_m + n_i + (r-2) + 1 \;\geq\; 2n_i + (r-1).$$

Thus, $r - 1 \leq n - 2n_i$, so $S \leq n + (r-1) \leq 2(n - n_i)$.

Now, suppose that T_i satisfies the second condition in the lemma, that is, T_i is of size of $n_i \leq n/4$. Then $S \leq n + n/2 \leq n + (n - 2n_i)$. □

3.2 Choosing a specific heuristic

Let T_j be a largest subtree of u in T. Suppose that T_j is the unique largest subtree, and that T_j is of size $n_j > n/4$. By lemma 4, any other subtree, T_i, will be completed in at most $2(n - n_i)$ steps. Therefore, to obtain an algorithm which completes every subtree T_i in time $2(n - n_i)$, we need only consider T_j when choosing a heuristic. One now sees that there is a natural choice to consider, namely the following *largest subtree heuristic* α_0:

- If one of the largest subtrees of u, say T_j, is not completed after stage 2, let T_j initiate the first non-green cycle.
- Thereafter, choose any incomplete subtree to initiate a cycle.

Lemma 5. *By applying the largest subtree heuristic, α_0, every subtree T_i is completed in at most $2(n - n_i)$ steps.*

Proof. By lemma 4, it only remains to be proven that the lemma holds for the unique largest subtree T_j of u, provided it exists. By properties 1 and 3 in the cycle lemma, subtree T_j is completed after at most $S' = S^1 + S^2 + (f+1)$ steps. Now, apply lemma 3. If $2n_j < n$, then S' is bounded from above by $h + 1 + (f+1) = n + 1 \leq n + (n - 2n_j)$. Otherwise $2n_j = n$, and then S' is bounded from above by $h + (f+1) = n = 2(n - n_j)$. □

We remark that by applying α_0, one can prove that *every* subtree is completed in at most $2(n - n_j)$ steps, where n_j is the size of a largest subtree. We shall, however, not use that fact in this paper.

THEOREM 1. *For all $n \geq 2$, the general algorithm using heuristic α_0 uses at most $2n - 3$ steps.*

PROOF. Clearly, the theorem holds for $n = 2$. If $n > 2$ and u has a subtree of size at least 2, it follows from lemma 5 that the theorem holds by induction. Otherwise, T is the star of n nodes, so by the second inequality in lemma 3, T will be routed in at most $(n-1) + \lfloor (n-1)/2 \rfloor \leq 2n - 3$ steps by the general algorithm. □

An interesting property of the heuristic α_0 is that it is independent of the actual instance of the permutation on the tree. Given only the tree T (and not the permutation of the packets), we can find a node u satisfying proposition 1 and a largest subtree T_j. Now, we start the algorithm and after stage 2, we simply check if the root node in T_j is a foreign packet or not.

With regards to upper and lower bounds, it can be proven that there exists an infinite family of trees on which the general algorithm using any heuristic uses at least $2n - 3$ steps for some permutation π. Thus, the upper bound given by the general algorithm using heuristic α_0 in theorem 1 is tight. We can also show that the general algorithm using any heuristic uses fewer than $3n$ steps, and that it will in fact use $3n$ steps, except for a logarithmically additive term, on some permutations if we just use the heuristic saying that any incomplete subtree can initiate any non-green cycle. All of these results can be found in Høyer and Larsen [1996].

4. Concluding remarks

The contribution of this paper lies in the generality and simplicity of the algorithm, and the characterization of $2n$ algorithms in terms of what conditions a heuristic must fulfill. Most importantly, the proof of lemma 4 offers a general technique for analyzing algorithms for routing problems with overlapping recursive phases, by considering the situation locally from the point of view of an arbitrary subtree.

Clearly, choosing an appropriate heuristic is critical to the running time of the algorithm. However, the one presented here is not the only option. Another heuristic for which we can also prove an upper bound of $2n$ is the following: "If there is a unique subtree which became weakly heap-ordered strictly later than all other subtrees, then let that subtree initiate the first cycle. Afterwards, keep choosing a subtree with a maximal number of foreign packets to initiate each cycle." More heuristics could be found, but this is not that interesting in itself.

Below, we discuss some further directions for future work. Clearly, we would like to see the gap between the lower bound of roughly $\frac{3}{2}n$ and the upper bound from Zhang [1997] closed. From the lower bounds mentioned at the end of the previous section, it is clear that improvements are not found by simply studying other heuristics. Letting recursive phases overlap even more than now may, however, be worth some considerations. Currently, we let recursive phases start at different times, but not until the packets belonging to a subgraph have all been moved to that subgraph. It might be possible to start the recursive phase as soon as “enough” packets have arrived, in the sense that it can be guaranteed that the remaining packets will arrive before we run out of packets from that subgraph around the new root. This could be an alternative or a contributing idea to the one from Zhang [1997] of dividing the graph into three areas (instead of at least two as in our paper).

A more general and very interesting problem is that of taking full advantage of the graph structure, i.e., design an algorithm which given a graph solves the problem as fast possible on that specific type of graph. Here, our guess is that the problem of solving this optimally is at least **NP**–hard. The problem could be generalized even further by making the initial permutation a parameter as well. We leave these as open problems.

Acknowledgements

The authors would like to thank Joan Boyar for many valuable discussions contributing to our understanding of the problem.

References

Alon, N., Chung, F. R. K., and Graham, R. L. 1993. Routing permutations on graphs via matchings (extended abstract). In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 583 – 591.

Alon, N., Chung, F. R. K., and Graham, R. L. 1994. Routing permutations on graphs via matchings. *SIAM Journal of Discrete Mathematics 7*, 3, 513 – 530.

Høyer, P. and Larsen, K. S. 1996. Permutation Routing via Matchings. Technical Report 1996 – 16, Department of Mathematics and Computer Science, Odense University.

Roberts, A., Symvonis, A., and Zhang, L. 1995. Routing on Trees via Matchings. Technical Report 494, Basser Dept. of Computer Science, University of Sydney.

Roberts, A., Symvonis, A., and Zhang, L. 1995. Routing on Trees via Matchings. In *Lecture Notes of Computer Science, Vol. 955: 4th Workshop on Algorithms and Data Structures*, 251 – 262.

Zhang, L. 1997. Optimal Bounds for Matching Routing on Trees. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 445 – 453.