An Optimal Permutation Routing Algorithm on Full-Duplex Hexagonal Networks[†]

Ignasi Sau[1‡] and Janez Žerovnik[2§]

[1] *Mascotte joint Project- INRIA/CNRS-I3S/UNSA- 2004, route des Lucioles - Sophia-Antipolis, France; and Graph Theory and Combinatorics group at Applied Mathematics IV Department of UPC - Barcelona, Spain.*
`email: ignasi.sau@gmail.com`
[2] *University of Maribor, FME, Smetanova 17, SI-2000 Maribor, Slovenia; and IMFM, Jadranska 19, SI-1000 Ljubljana, Slovenia.*
`email: janez.zerovnik@imfm.uni-lj.si`

received 13 March 2008, *accepted 10 July 2008*.

In the permutation routing problem, each processor is the origin of at most one packet and the destination of no more than one packet. The goal is to minimize the number of time steps required to route all packets to their respective destinations, under the constraint that each link can be crossed simultaneously by no more than one packet. We study this problem in a hexagonal network, i.e. a finite subgraph of a triangular grid, which is a widely used network in practical applications.
We present an optimal distributed permutation routing algorithm on full-duplex hexagonal networks, using the addressing scheme described by F.G. Nocetti, I. Stojmenović and J. Zhang (*IEEE TPDS 13(9): 962-971, 2002*). Furthermore, we prove that this algorithm is oblivious and translation invariant.

Keywords: hexagonal networks, permutation routing, shortest path, distributed algorithm, communication networks, oblivious algorithm.

1 Introduction

The packet-routing problem on any interconnection network is essentially important. This problem involves how to transfer the right data to the right place within a reasonable amount of time. To measure the routing capability of an interconnection network, the partial permutation routing (PPR) problem is usually used as the metric. In the PPR problem, each processor is the origin of at most one packet and the destination of no more than one packet. This problem has been studied in a wide diversity of scenarios,

[†]A short conference version of this article (with incomplete proofs) was presented in Sau and Žerovnik (2007).

[‡]Partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Educación y Ciencia of Spain, European Regional Development Fund under project TEC2005-03575, Catalan Research Council under project 2005SGR00256, and COST action 293 GRAAL, and CRC CORSO with France Telecom.

[§]Supported in part by the Slovenian research agency ARRS, grants L2-7207-0101 and P1-0285-0101.

such as Mobile Ad Hoc Networks Karimou and Myoupo (2005), Cube-Connected Cycle (CCC) Networks Jan and Lin (2005), Wireless and Radio Networks Datta (2005), All-Optical Networks Liang and Shen (2002), and Reconfigurable Meshes Cogolludo and Rajasekaran (2001).

Recently an optimal algorithm for permutation routing has been found on full-duplex 2-circulant graphs Hwang et al. (1997). Routing algorithms for 2-circulants with half-duplex links are studied in Dobravec et al. (2003). In this paper we give an optimal algorithm for permutation routing on full-duplex hexagonal networks.

Two-dimensional meshes are among the most studied topologies for computer networks. Tessellation of the plane with hexagons may be considered as the most natural because cells have optimal diameter to area ratio. If centers of neighboring cells are connected, one obtains a triangular grid. Hexagonal networks are finite subgraphs of the triangular grid, and in this article we study convex subgraphs (that is, that contain all shortest paths between any pair of nodes) of the triangular grid. The triangular grid can also be obtained from the basic 4-mesh by adding NE to SW edges, which is called a 6-mesh in Trobec (2000).

Another suitable application for this problem can be easily found on a radiocommunication wireless environment Nocetti et al. (2002). Let the base stations be placed at the centers of a hexagonal tessellation of the plane. The interconnection network among base stations constitutes a hexagonal network, as shown in Figure 1.

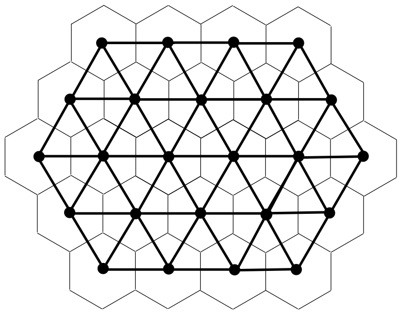


Figure 1: Hexagonal network (△) and hexagonal tessellation (⬡).

Coordinates must be defined to associate to each mobile user the base station which is the center of the hexagon where this user is. The problem of exchanging messages among mobile users corresponds to a routing problem, since there are pairs of users willing to communicate. Each user can only establish one call at a given moment. If one assumes that in each hexagonal cell there is at most one user that sends a message and at most one user that receives a message, the problem can be modeled as partial permutation routing. The algorithm that we propose can be used to route the messages between pairs of users that are in different hexagons. If there are more messages to be routed, i.e. there are more than one message originating from the same cell or a cell is the destination of more than one message, our algorithm still can be used, but the optimality may not be achieved. If the two communicating users belong to the same hexagon, some local delivery mechanism can be carried out.

This paper is structured as follows. In Section 2 we define preliminary concepts that will be used later. The description and analysis of our algorithm are provided in Section 3. Finally, Section 4 concludes this work.

2 Preliminaries

In this section we describe the network topology under study, we formally define the routing problem and we recall previous results on this field.

2.1 Network topology

Nodes in a hexagonal network are placed at the vertices of a regular triangular tessellation, so that each node has up to six neighbors. These networks have been studied in a variety of contexts, specially in wireless and interconnection networks Decayeux (2006); Decayeux and Seme (2005), but they have been also applied, for instance, in chemistry to model benzenoid hydrocarbons Tošić et al. (1995).

We focus on the hexagonal network with full-duplex links, that is, an edge of the network can be crossed simultaneously by two messages, one in each direction. Equivalently, each edge between two nodes u and v is made of two independent arcs $\{uv\}$ and $\{vu\}$, as illustrated in Figure 2a.

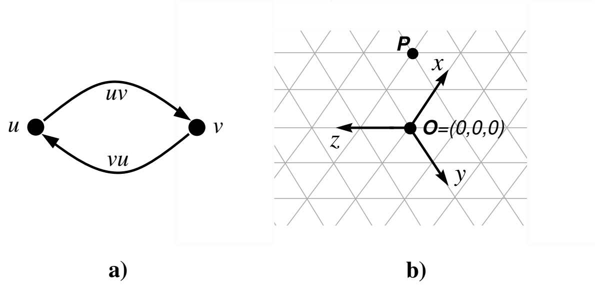


Figure 2: a) Each edge consists of two independent links. **b)** Axis used in a hexagonal network.

Remark 2.1 *If the network is half-duplex, it is easy to construct a 2-approximation algorithm from an optimal algorithm for the full-duplex case by introducing* odd-even *steps, as explained for example in Dobravec et al. (2003).*

2.2 Routing problem

We deal with a partial permutation routing problem. In an infinite triangular grid, we are given a subset V of nodes, and a permutation π that acts on this subset $\pi : V \to V$. Each node $u \in V$ wants to send a message to the node $\pi(u)$. Thus, we have $|V|$ pairs of communicating nodes and $|V|$ messages to be delivered simultaneously. All the edges and nodes of the *host* graph (the hexagonal graph) can be traversed by the packets.

Remark 2.2 *The special case where only one node has a packet to send is known as* 2-terminal routing *(*minimum *if a shortest path is used). Optimal algorithms for the 2-terminal routing problem have been found, for instance, in 2-jump circulant graphs Robič and Žerovnik (2000) and hexagonal networks Nocetti et al. (2002).*

Under the store-and-forward Δ-port model Fraigniaud and Lazard (1994), at each step a packet can either stay or move to an adjacent node by crossing a link, but no link can be crossed by two packets at the same step (recall that in the full-duplex case, there are two links between two adjacent nodes). A node can send or receive packets through all its incident edges at the same step. Cohabitation of multiple packets at the same node is allowed. Thus, a queue is required for each outgoing edge at each node. Since the outdegree in a hexagonal network is six, the same number of queues are required at each node. The goal is to minimize the number of time steps required to route all packets to their respective destinations.

The algorithm described in Section 3 is implemented independently at each node, without assuming any global knowledge about the network. I.e., it is a fully distributed algorithm.

Remark 2.3 *One could also have defined the permutation routing with a permutation π that acts on all the nodes of an infinite hexagonal network, defining the (infinite set of) non-communicating nodes as fixed points of π.*

2.3 Addressing model and previous results

In Nocetti et al. (2002) the authors solve the problem of routing a *single* message following a shortest path through a hexagonal communication network. The first idea that is of our interest (in fact, this was first introduced in Stojmenović (1997)) is the representation of any address on a generating system consisting of three unitary vectors ***i***, ***j***, ***k*** on the directions of three axis x, y, z with a 120 degree angle between each pair. These three axes intersect on an arbitrary (but fixed) node O labeled with the address $\boldsymbol{O} = (0, 0, 0)$, as it is depicted in Figure 2b. An example will be given later. Clearly any path on the triangular grid is a sequence of moves in directions **i**, **j**, **k**, or in the opposite directions. From commutativity of vector addition it follows that any path can be expressed as a combination $P_1\mathbf{i} + P_2\mathbf{j} + P_3\mathbf{k}$. Hence each node P of the hexagonal network can be labeled with an address $\boldsymbol{P} = (P_1, P_2, P_3)$ based on paths from the origin O. Using that $\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k} = 0$, the key observation Nocetti et al. (2002) is that, if (a, b, c) and (a', b', c') are the relative addresses of two packets, then $(a, b, c) = (a', b', c')$ if and only if there exists $d \in \mathbb{Z}$ such that $a' = a + d$, $b' = b + d$, and $c' = c + d$. For instance, consider the node P of Figure 2b. We can express its address $\boldsymbol{P}$ in different equivalent ways, for instance: $\boldsymbol{P} = (3, 1, 2) = (2, 0, 1) = (1, -1, 0) = (0, -2, -1)$. Among many possible paths and corresponding addresses we will later use the unique address called the address in the shortest path form. As we will see in Section 3, at the beginning of the routing algorithm each node S knows the address of the destination node D of the message placed initially at S, and computes the relative address $\overrightarrow{SD} = \boldsymbol{D} - \boldsymbol{S}$ of the message. Note that this relative address does not depend on the choice of the origin node O. As discussed later, this relative address is the only information that is added to the heading of the message to be transmitted, constituting in this way the packet to be sent through the network. The relative address $\overrightarrow{OP} = \boldsymbol{P} - \boldsymbol{O}$ can be expressed in different ways. Note that these addresses are related to paths of different lengths.

Definition 2.1 *A relative address $\overrightarrow{SD} = (a, b, c)$ is of the* shortest path form *if there is a path from node S to node D, consisting of a units of vector* **i**, *b units of vector* **j** *and c units of vector* **k**, *and this path has the shortest length among all paths going from S to D.*

In what follows, (a, b, c) denotes a relative address. We will later work only with the addresses in the shortest path form. The next result simplifies extraordinarily the routing in hexagonal networks, as we will see in Section 3.

Theorem 2.1 (Nocetti et al. (2002)) *An address (a, b, c) is of the shortest path form if and only if at least one component is zero, and any two components do not have the same sign.*

In the example above, one can check that the address $(1, -1, 0)$ has minimum length, and it is the only one that satisfies the conditions of Theorem 2.1.

Corollary 2.1 (Nocetti et al. (2002)) *Any address has a unique shortest path form.*

Thus, each relative address $\overrightarrow{SD}$ written in the shortest path form has at most two non-zero components, and they have different sign. In fact, it is easy to find the shortest path form (and thus, the length) using the next result.

Theorem 2.2 (Nocetti et al. (2002)) *If $\overrightarrow{SD} = a\mathbf{i}+b\mathbf{j}+c\mathbf{k}$, then the length of the path given by the relative address $\overrightarrow{SD}$ satisfies*

$$|\overrightarrow{SD}| = \min(|a-c|+|b-c|, |a-b|+|b-c|, |a-b|+|a-c|).$$

3 An optimal routing algorithm

In Section 3.1 we describe our algorithm for permutation routing in full-duplex hexagonal networks, and in Section 3.2 we prove that this algorithm is optimal. Informally, the idea of this algorithm is to define a routing for each type of shortest path and a suitable queue policy, in such a way that the number of steps that a packet has to wait plus the length of its shortest path (from its origin to its destination) do not exceed the bound given by the maximum length over the paths of all packets to be sent.

3.1 Formal description

We introduce some definitions in order to simplify further proofs.

Definition 3.1 *Given a packet p and its relative address (a, b, c) in the shortest path form, we denote by ℓ_p the length of this shortest path, and by ℓ_{max} the maximum length over all shortest paths:*

$$\ell_p := |a| + |b| + |c| \;, \quad \ell_{max} := \max_p(\ell_p)$$

Since $|V| < \infty$, this maximum is indeed achieved, possibly by several messages. By the definition of ℓ_{max} and taking into account that any algorithm can move a packet only one position at each step, ℓ_{max} is in fact a lower bound for all algorithms.

Lemma 3.1 (Lower Bound) *The number of steps of any permutation routing algorithm is at least ℓ_{max}.*

Definition 3.2 *Two packets p and p' are* in conflict *or, simply,* meet*, if they are simultaneously (i.e. on the same step of the algorithm) in the same outgoing queue at the same node of the network.*

Intuitively, if two (or more) packets meet, at most one of them moves on the next step of the algorithm.

Definition 3.3 *We denote by w_p^i the number of steps waited by packet p until the end of the i*th *step of the algorithm. We call w_p^i the* delay of p.

Given a packet p and its delay w_p^i, w_p^i is an allowed delay *if $\ell_p + w_p^i \leq \ell_{max}$. Similarly, w_p^i is an* additional *(or* forbidden*)* delay *if $\ell_p + w_p^i > \ell_{max}$.*

Finally, a packet p is saturated *at the end of step i if $w_p^i = \ell_{max} - \ell_p$.*

The idea is that if a packet p is saturated it must not wait anymore. Otherwise, the algorithm becomes not optimal. In Figure 3 the packet model that we use for the analysis is represented. In this model, each packet has two *boxes* with capacities ℓ_p and $\ell_{max} - \ell_p$, respectively. At each step of the algorithm, two things can happen. Namely, if packet p has moved during step i, then an item of the box on the left (first box) is filled. On the other hand, if packet p has waited in some queue during step i, then an item of the box on the right (second box) is filled. If the first box is full, it means that packet p has reached its destination. Conversely, if the second box is full, it means that packet p is *saturated*, and thus it cannot wait anymore if we want to guarantee the optimality of our algorithm (because the number of steps required by p would be strictly greater than ℓ_{max}).

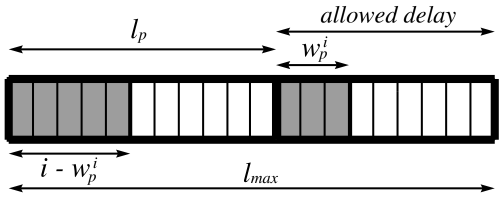


Figure 3: Packet model used on the permutation routing algorithm.

Remark 3.1 *The model displayed in Figure 3 is only used for analysis, but in fact the only information that each packet p has to carry is the relative address until its destination. That is, only a vector with at most two non-zero integers. It is important to clarify that it is not necessary that the global constant ℓ_{max} is available at every node, because some broadcasting protocol would have to be carried out to compute and share it. Although it is possible to broadcast on hexagonal mesh in polynomial time Chen et al. (1990), this would introduce additional time complexity in the preprocessing phase.*

Table 1 describes an optimal message routing algorithm for each node P. Note that the main loop for is infinite. We prove that in the case of permutation routing, the number of steps i at each node is at most ℓ_{max}, but written in this way the algorithm can be applied in a more general routing scenario.

Remark 3.2 *We assume that the network has a global clock, and that all nodes are synchronized. Packets can be sent only on discrete clock events, while the other tasks (**preprocessing**, **update_packet**, **decide_outgoing_edge and order_queue**) are done between two consecutive clock events.*

In next sections we provide a detailed explanation of each of the procedures that appear on this general scheme.

3.1.1 Description of the procedure **preprocessing**

Before packets begin to be sent through the network, at most one packet is placed at each node. Each packet has associated the address of its destination node, namely D. With this information, each source node S can compute the relative address (a, b, c) of the packet that will be sent from S to D. Because of Theorem 2.1, to write this address in the shortest path form it is enough to check which address among $(0, b-a, c-a)$, $(a-b, 0, c-a)$ and $(a-c, b-c, 0)$ has two components of different sign.

Then each node computes the length of each packet's address (a, b, c). As it is in the shortest path form, the path that the packet will have to follow has length $\ell_p = |a| + |b| + |c|$. Finally, for each message a heading containing the components of the relative address is added. Note that this information is enough considering the packet model of Figure 3, as we prove in Lemma 3.4.

Thus, since this task involves just integer addition and comparison, the time complexity of the preprocessing phase is $\mathcal{O}(1)$ assuming fixed integer size to codify all addresses, or $\mathcal{O}(\log(n))$, where n is the maximum size of the integer required to codify the addresses of the nodes.

```
At each node P of the network:

begin
    preprocessing;
    for i from 0 do
        Reception phase:
            for each packet in node P
                update_packet;
        Transmission phase:
            for each packet in node P
                decide_outgoing_edge;
            for each queue
                order_queue;
                send the 1st packet;
    increment i
    end for
end
```

Table 1: Algorithm $\mathcal{A}$.

3.1.2 Description of the procedure **update_packet**

Updating the address in reception offers more robustness against link failures. Of course, a node knows from which neighbor a packet comes from. Without ambiguity, we assign to each incoming edge a vector as shown in Figure 4a. (Formally, in Table 2 *incoming_edge* is the vector assigned to the edge used by the incoming packet. The same idea applies to *outgoing_edge* in Table 3.) Then to update the relative address (a, b, c) of the packet, one has just to decrease by one the component corresponding to the incoming edge, as described in Table 2. After updating the address, we check if the packet has reached its destination node.

```
begin
    (a, b, c) ← (a, b, c) − incoming_edge;
    if (a, b, c) == (0, 0, 0)
        then this is the destination node;
end
```

Table 2: Description of the procedure **update_packet**.

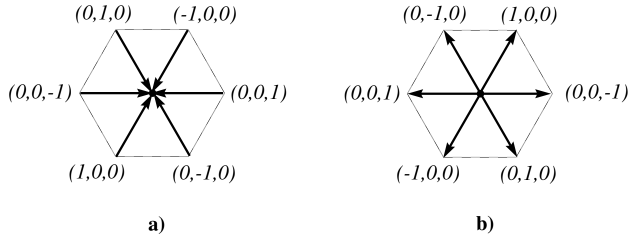


Figure 4: **a)** Possible incoming edges to (0,0,0). **b)** Possible outgoing edges from (0,0,0).

3.1.3 *Description of the procedure* **decide_outgoing_edge**

This is the phase where the main core of the routing is carried out. The idea of this routing is that one can assure that when packets meet, no additional delay is introduced, as we prove in Proposition 3.1. Given the address of the shortest path of a packet with two non-zero components, one can choose one of both directions to begin with, as we see with an example in Figure 5. This procedure makes the choice between both alternatives in each case.

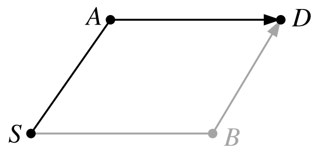


Figure 5: Given the relative address of the packet, there are two possible choices for assigning priorities to the components of the shortest path.

Because of Theorem 2.1, we can partition the $|V|$ relative addresses in the shortest path form into 12 disjoint classes, according to the sign of their non-zero components. These classes are, namely:

$$(+,0,0),(-,0,0),(0,+,0),(0,-,0),(0,0,+),(0,0,-),$$
$$(+,-,0),(-,+,0),(+,0,-),(-,0,+),(0,+,-),$$
$$\text{and } (0,-,+).$$

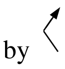
Note that the address of a packet may vary at each step, as we have seen in the **update_packet** procedure. It is also interesting to observe that the same triple can appear many times as more pairs can have the same relative addresses. However, at a given time step, all relative addresses of packets that appear at the same node are different.

Recall that in a hexagonal network each node has six outgoing edges, and thus six queues. Without ambiguity, we label the six outgoing edges of a node as shown in Figure 4b. For the updated packet address (a, b, c), Table 3 describes this routing procedure.

For instance, if the packet address is of the type $(-, 0, +)$ then, according to Table 3, this packet goes first in the direction $-x$, and after in the direction $+z$. (See Example 3.1.) We symbolize this rule by

```
begin
  if packet_address has only 1 non-zero component
    outgoing_edge = the edge corresponding
                     to the non-zero component;
  else
    outgoing_edge = the edge corresponding
                     to the negative component;
end
```

Table 3: Description of the procedure **decide_outgoing_edge**.

the arrow . Another example: the routing of the address $(+, -, 0)$ is represented by . That is, we always give priority to the negative component. In Figure 6 these routing rules are summed up.

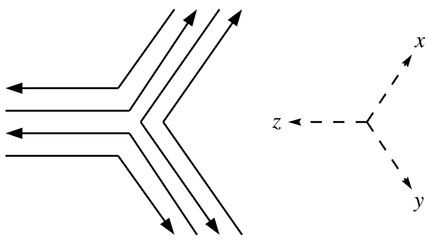


Figure 6: Routing of the packets according to Algorithm $\mathcal{A}$.

Definition 3.4 *Given a packet* p *and its relative address* (a, b, c) *with two non-zero components, we call the* first *(resp.* second*)* direction of p *to the first (resp. second) direction of the movement of the message according to the rules of Algorithm* $\mathcal{A}$*. If* $\overrightarrow{SD}$ *has only one non-zero component, we say that it is* the direction of p.

Example 3.1 *We illustrate the operation of procedures* ***decide_outgoing_edge*** *and* ***update_packet****. Assume a packet with relative address* $(-1, 0, 4)$ *is given to the procedure* ***decide_outgoing_edge****. As it has two nonzero components, it is added to the queue corresponding to the negative component, i.e.* $(-1, 0, 0)$*. After possibly waiting some communication rounds in the queue, the packet will be transmitted. After arrival to the next node, first its relative address will be altered by the procedure* ***update_packet*** *to* $(0, 0, 4)$*, which is* $(-1, 0, 4) - (-1, 0, 0)$*. Now the relative address has only one nonzero component, and therefore the procedure* ***decide_outgoing_edge*** *will send it to the queue of this direction, i.e.* $(0, 0, 1)$*.*

3.1.4 Description of the procedure **order_queue**

At each of the six queues of each node, the same priority policy applies: order the outgoing packets according to *decreasing number of remaining steps until the destination*. That is, the packet that has more remaining steps has priority 1, the next one has priority 2, and so on. Let us show in Lemma 3.2 and Lemma 3.3 that there cannot be equality in the number of remaining steps, hence this policy is correct.

Lemma 3.2 *Packets can only wait, possibly, during their last direction.*

Proof: Suppose that the claim is false, and consider the first (according to the running time of the algorithm) two packets having a relative address with two non-zero components that meet when their first direction is not yet finished. According to the routing rules of **decide_outgoing_edge** procedure (see Table 3 or Figure 6), the direction where packets met must be the same: either $-x$, $-y$, or $-z$. Since these packets are the first packets that meet, both must have the same origin node, a contradiction. □

Note that one can generalize this result by saying that a packet can wait only during its *last* direction, because packets with only one direction can be in conflict obviously only on this direction. We are now ready to prove that the queue policy that we have defined is correct:

Lemma 3.3 *Two packets in a given queue cannot have the same number of remaining steps.*

Proof: Let p and p' be two packets in the same queue. Because of Lemma 3.2, both p and p' must be in their last direction. If they had the same number of remaining steps, they would have the same destination node, a contradiction. □

3.2 Proof of optimality

The next observation is useful in the proof of Proposition 3.1, which is the main result that allows us to prove the optimality of Algorithm $\mathcal{A}$.

Lemma 3.4 *The following two queue policies are equivalent:*

1. *Order the outgoing packets according to decreasing remaining steps.*
2. *Order the outgoing packets according to increasing remaining allowed delay.*

Proof: For any packet p (see the model of Figure 3) the number of remaining steps is $\ell_p - (i - w_p^i) = \ell_p - i + w_p^i$, and the remaining allowed delay is $\ell_{max} - \ell_p - w_p^i$. Thus, without taking the sign into account, both magnitudes differ only on i and ℓ_{max}, which are constants for all packets at a given step i of the algorithm. □

Proposition 3.1 *Algorithm $\mathcal{A}$ introduces no additional delay to any packet p.*

Proof: We prove by induction on the number of steps that no additional delay is introduced at any queue of the network after step i.

First of all, after the first step ($i = 1$) the only packets that could have additional delay are those with length ℓ_{max}. But, since all those nodes must be placed into different nodes at the beginning of the algorithm, no two nodes can have been in the same queue, and thus all of them must have had maximum priority according to the queue policy, hence none has waited.

Now we suppose that no additional delay has been introduced after step $i-1$ ($i \geq 2$) and let us show that no additional delay is introduced after step i. It is enough to see that there can be at most one saturated packet at each queue. Indeed, if there is only one saturated packet, this packet has maximum priority because of Lemma 3.4, and thus this packet does not wait. Now, suppose that there are two saturated packets p and p' in the same queue. By definition of saturated packet, $\ell_p + w_p^i = \ell_{p'} + w_{p'}^i = \ell_{max}$. The number of remaining steps of p is $\ell_p - (i - w_p^i) = \ell_{max} - i$, which equals the number of remaining steps of p': $\ell_{p'} - (i - w_{p'}^i) = \ell_{max} - i$. Thus, both packets have the same destination node, contradicting the assumption of permutation routing. □

Theorem 3.1 *Algorithm $\mathcal{A}$ is an optimal permutation routing algorithm for full-duplex hexagonal networks.*

Proof: Because of Proposition 3.1, all packets reach their destination nodes in a number of steps not greater than ℓ_{max}, and thus the lower bound of Lemma 3.1 is attained. □

Besides minimizing the number of steps, a routing algorithm must also be easy to implement; namely, the routing at each step should be determined efficiently. This class of algorithms is called *oblivious* Hwang et al. (1997, 2002).

Definition 3.5 *A routing algorithm is called* oblivious *if the routing of a packet only depends on its origin and destination nodes, although the waiting time at any intermediate node may depend on other paths. An oblivious algorithm is* translation invariant *if the path between any two nodes u and v depends only on the relative address from u to v (i.e. the path does not depend on the absolute addresses of u and v).*

Thus, a translation invariant oblivious algorithm is completely determined by paths from any node to all other nodes.

Corollary 3.1 *Algorithm $\mathcal{A}$ is oblivious, translation invariant and minimum permutation routing.*

Proof: Optimality, i.e. minimum permutation routing, has been proved in Theorem 3.1. The obliviousness is straightforward since our algorithm only uses the origin and destination nodes for routing each packet. Finally, it is clear that to route a packet only the relative address $\overrightarrow{SD}$ between the source and destination node is necessary, and thus the invariance is also proved. □

Concerning the complexity of the algorithm, at each step each node has to carry out just constant time computations: integer addition and comparison. As stated in Theorem 3.1, the number of time steps is ℓ_{max}. Hence the time complexity of Algorithm $\mathcal{A}$ is $\mathcal{O}(\ell_{max})$.

3.3 About the queue policy

The queue policy plays an important role on the description of our routing algorithm. Another possibility could have been to order the packets according to their *total length* ℓ_p, and then, in case of equality, according the their remaining steps. Note that there is no ambiguity since all destination nodes are different. Although both policies seem to be similar, the total length policy combined by the routing rules shown in Figure 6 yield a non-optimal algorithm, while the remaining steps policy is optimal, as it has been proved in Theorem 3.1. Indeed, let us show with a counterexample that the total length policy is not optimal. In Figure 7 the origin nodes for each packet are labeled with small letters, while their corresponding destination nodes are labeled with capital letters. We have that $\ell_{A-a} = 6$, $\ell_{B-b} = 5$, $\ell_{C-c} = 5$ and $\ell_{P-p} = 4$. One can check that the total number of steps required by the packet that starts at p to reach P is 7 (because it waits 3 steps) and thus this algorithm is not optimal, since $7 > 6 = \ell_{max}$.

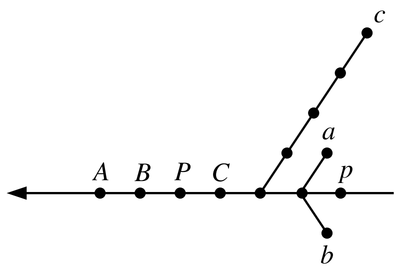


Figure 7: Counterexample to the *total length* queue policy.

4 Conclusions

In this article we have presented a distributed optimal permutation routing algorithm for full-duplex hexagonal networks. Furthermore, we have proved that this algorithm is oblivious and translation invariant. In this way, we have given an answer to an open question proposed in Nocetti et al. (2002). The time complexity of our algorithm is $\mathcal{O}(\ell_{max})$, where ℓ_{max} is the maximum length over the shortest paths of all the packets to be sent.

The next natural step is to consider half-duplex hexagonal networks and honeycomb (half and full-duplex) networks. Another avenue of further research could address generalization of the optimal permutation routing algorithms to more general networks. For example, generalization from 2-circulant graphs to 3-circulant graphs, k-circulants, or, more general, Cayley graphs. Hexagonal networks have a generalization in the three-dimensional hexagonal network García et al. (2003), where the *basic* pieces that *tessellate* the space are tetrahedrons. One could try to find a basis such that all vectors add up to $\vec{0}$, as it happens on the plane. But, unfortunately, one hits on the well known topological result about the non-orientability of the surface of a sphere (a tetrahedron is topologically equivalent to a sphere). Thus, new strategies should be devised for the 3D scenario.

Acknowledgements

We thank the anonymous referees for careful reading and helpful remarks. Most of this work was done while one of us (I.S.) was visiting Institute of Mathematics, Physics and Mechanics in Ljubljana, supported by STSM grant sponsored by European Action COST 293 (GRAAL).

References

M. Chen, K. Shin, and D. Kandlur. Addressing, routing and broadcasting in hexagonal mesh multiprocessors. *IEEE Transactions on Computers*, 1(C-39):10–18, 1990.

J. Cogolludo and S. Rajasekaran. Permutation routing on reconfigurable meshes. *Algorithmica*, 31:44–57, 2001.

A. Datta. A fault-tolerant protocol for energy-efficient permutation routing in wireless networks. *IEEE Transactions on Computers*, 54(11):1409–1421, 2005.

C. Decayeux. *Réseaux hexagonaux: Modélisation Géomètrique et Application à la Téléphonie Mobile*. PhD thesis, Université de Picardie Jules Verne, October 2006.

C. Decayeux and D. Seme. 3d hexagonal network: Modeling, topological properties, addressing scheme, and optimal routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 16(9):875–884, 2005.

T. Dobravec, J. Žerovnik, and B. Robič. Permutation routing in double-loop networks: design and empirical evaluation. *Journal of Systems Architecture*, 48:387–402, 2003.

P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53:79–134, 1994.

F. García, J. Solano, I. Stojmenović, and M. Stojmenović. Higher dimensional hexagonal networks. *Journal of Parallel and Distributed Computing*, 63(1164-1172), 2003.

F. Hwang, T. Lin, and R. Jan. A permutation routing algorithm for double loop network. *Parallel Processing Letters*, 7(3):259–265, 1997.

F. Hwang, Y. Yao, and B. Dasgupta. Some permutation routing algorithms for low-dimensional hypercubes. *Theoretical Computer Science*, 270:111–124, 2002.

G. E. Jan and M.-B. Lin. Concentration, load balancing, partial permutation routing, and superconcentration on cube-connected cycles parallel computers. *Journal of Parallel and Distributed Computing*, 65: 1471–1482, 2005.

D. Karimou and J. F. Myoupo. An application of an initialization protocol to permutation routing in a single-hop mobile ad hoc networks. *The Journal of Supercomputing*, 31:215–226, 2005.

W. Liang and X. Shen. Permutation routing in all-optical product networks. *IEEE Transactions on Circuits and Systems*, 49(4):533–538, 2002.

F. G. Nocetti, I. Stojmenović, and J. Zhang. Addressing and routing in hexagonal networks with applications for tracking mobile users and connection rerouting in cellular networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):963–971, 2002.

B. Robič and J. Žerovnik. Minimum 2-terminal routing in 2-jump circulant graphs. *Computers and Artificial Intelligence*, 19(1):37–46, 2000.

I. Sau and J. Žerovnik. Optimal permutation routing on mesh networks. In *International Network Optimization Conference*, Spa, Belgium, April 2007. 6 pages.

I. Stojmenović. Honeycomb networks: Topological properties and communication algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 8(10):1036–1042, 1997.

R. Tošić, D. Masulović, I. Stojmenović, J. Brunvoll, B. Cyvin, and S. Cyvin. Enumeration of polyhex hydrocarbons up to h=17. *Journal of Chemical Information and Computer Sciences*, 35:181–187, 1995.

R. Trobec. Two-dimensional regular d-meshes. *Parallel Computing*, 26:1945–1953, 2000.