University of Ljubljana
Institute of Mathematics, Physics and Mechanics
Department of Mathematics
Jadranska 19, 1 111 Ljubljana, Slovenia

Preprint series, Vol. 44 (2006), 1017

AN OPTIMAL PERMUTATION ROUTING ALGORITHM FOR FULL-DUPLEX HEXAGONAL MESH NETWORKS

Ignasi Sau Janez Žerovnik

ISSN 1318-4865

October 25, 2006

Ljubljana, October 25, 2006

An optimal permutation routing algorithm for full-duplex hexagonal mesh networks

Ignasi Sau* and Janez Žerovnik†

Abstract

In the permutation routing problem, each processor is the origin of at most one packet and each processor is the destination of no more than one packet. We study this problem in an hexagonal network (that is, a finite convex subgraph of a triangular grid), a widely used network in practical applications. We use the addressing scheme described by F.G. Nocetti, I. Stojmenovic and J. Zhang (2002, IEEE Trans. on Parallel and Distrib. Systems).
In this paper, a distributed optimal routing algorithm for full-duplex hexagonal mesh networks is presented. 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 each processor is the destination of no more than one packet. This problem has been studied in a wide diversity of scenarios, such Mobile Ad Hoc Networks [9], Cube-Connected Cycle (CCC) Networks [8], Wireless and Radio Networks [3], All-Optical Networks [12] and Reconfigurable Meshes [2], among others.

Recently, an optimal algorithm for permutation routing has been found in 2-circulant graphs [7] for full-duplex model. Routing algorithms for 2-circulants with not full-duplex links are studied in [5]. In this paper we find an optimal algorithm for permutation routing in full-duplex hexagonal networks.

Tesselation of the plane with hexagons maybe considered most natural because the cells have optimal diameter to area ratio. If centers of neighboring cells are connected, we obtain a triangular grid. The hexagonal networks we study here are finite subgraphs of the triangular grid, more formally, they are convex subgraphs of the triangular grid. The triangular grid can also be obtained

*Email: ignasi.sau@gmail.com, Address: Mascotte Project, CNRS/I3S/INRIA 2004 route des Lucioles - B.P. 93 F-06902 Sophia-Antipolis Cedex, France.

†Email: janez.zerovnik@imfm.uni-lj.si, Address: IMFM, Jadranska 19, SI-1000 Ljubljana, Slovenia and University of Maribor, Smetanova 17, SI-2000 Maribor, Slovenia.

from the basic 4-mesh by adding NE to SW edges, which is called a 6-mesh in [17]. Two-dimensional meshes are among the most studied topologies for computer networks.

Another possible application for our problem can be easily found on a radiocommunication wireless environment. Let the base stations be placed on the centers the hexagonal tesselation of the plane. Thus, the interconnection network among base stations constitutes an hexagonal network (that is, a triangular grid, see Figure 1).

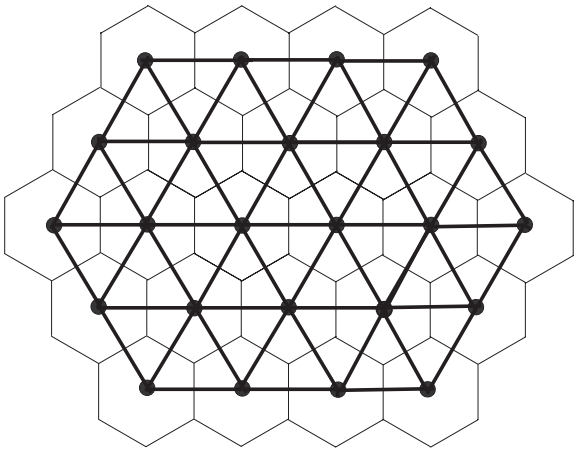

FIG. 1 – Hexagonal network (△) and hexagonal tesselation (⬡)

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 we have pairs of users that want to communicate. Each user can only establish one call at a given moment. If we assume 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. Our algorithm 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 preliminary concepts that will be used in later results are defined. The description of our algorithm is provided in Section 3. Finally, Section 4 concludes this work.

2 Preliminaries

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

2.1 Network topology

Nodes in an 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. The most known application may be to model cellular networks with hexagonal networks were nodes are base stations. But these networks have been also applied in chemistry to model benzenoid hydrocarbons (see, for example [16, 10]) in image processing, computer graphics [11], as well as wireless and interconnection networks.

We will deal with a hexagonal mesh network with full-duplex links, that is, an edge of the network can be crossed by two simultaneous messages, one in each direction. We can think of such a bidirectional link in the way that it is shown in Figure 2. I.e., each edge between 2 nodes u and v is made of 2 independent arcs, $\{uv\}$ and $\{vu\}$.

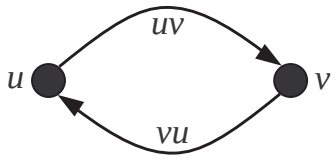

FIG. 2 – Each edges consists of 2 independent links

Remark 2.1 *If the network is not full-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 [5].*

2.2 Routing problem

We suppose that 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 crossed by the packets.

Remark 2.2 *The special case that 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 [14] and hexagonal networks [13].*

Under the multiport model, at each step a packet can either stay put or move to an adjacent node by crossing a link, but no link (recall that in the full-duplex case, there are 2 links between 2 adjacent nodes) can be crossed by two packets 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 an hexagonal network is 6, 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 that will be described in Section 3 is implemented independently at each node, without assuming any global knowledge about the network. I.e., it is a 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

In this work we will strongly use some fundamental results of [13]. In this paper the authors solve the problem of routing a *single* message through an hexagonal communication network. The first idea that will be of our interest (in fact, this was firstly introduced in [15]) is the representation of any address $D - S$ on a basis consisting of three unitary vectors i, j, k on the directions of three axis x, y, z with a 120 degree angle among them (see Figure 3).

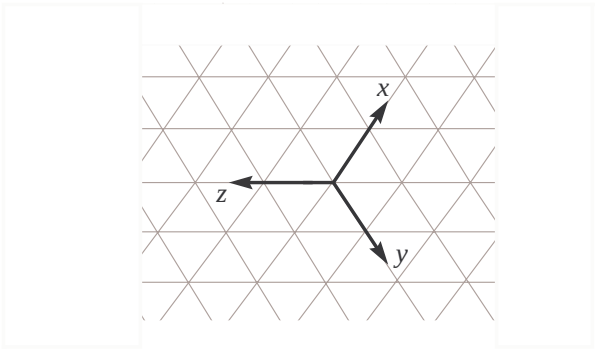

FIG. 3 – Axis used in an hexagonal network

Thus, we assume that each node is labeled with an address expressed on this basis (i, j, k). At the beginning, each node S knows the address of its destination node D, and computes the relative address $D - S$. As we will discuss later, this information is added in a heading to the message to be transmitted, constituting in this way the packet that will be sent through the network.

2.4 Previous results

Remarking that $i + j + k = 0$, the key observation [13] is that, being (a, b, c) and (a', b', c') the addresses of two $D-S$ pairs, then $(a, b, c) = (a', b', c')$ iff $\exists d \in \mathbb{Z}$ such that $a' = a+d$, $b' = b+d$ and $c' = c + d$. Now, we shall give a formal definition about what we intuitively mean by the *shortest path form.*

Definition 2.1 *An address $D - S = (a, b, c)$ 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.*

The next result simplifies extraordinarily the routing in hexagonal networks, as we will see later.

Theorem 2.1 ([13]) *An address (a, b, c) is of the shortest path form if and only if the following conditions are satisfied :*

1. *At least one component is zero (that is, $abc = 0$).*
2. *Any two components can not have the same sign (that is, $ab \leq 0,\ ac \leq 0,$ and $bc \leq 0$).*

Corollary 2.1 ([13]) *Any address has a unique shortest path form.*

Thus, each address $D - S$ written in the shortest path form has at most 2 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 ([13]) *If $D - S = ai + bk + ck$, then*

$$|D - S| = \min(|a - c| + |b - c|, |a - b| + |b - c|, |a - b| + |a - c|).$$

3 An optimal routing algorithm

In Section 3.1 we will describe our algorithm for permutation routing in full-duplex hexagonal networks, and in Section 3.2 we will prove that this algorithm is optimal. Informally, the idea of this algorithm is to find 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 p has to wait plus the length of its path (l_p) do not exceed the maximum length, l_{max}.

3.1 Formal description

First of all, some definitions are required in order to simplify further proofs.

Definition 3.1 *Given a packet p and its address (a, b, c) in the shortest path form, we denote by l_p the length of this shortest path :*

$$l_p := |a| + |b| + |c|$$

Definition 3.2 *Denote by l_{max} the maximum length over all packets :*

$$l_{max} := \max_p(l_p)$$

Observe that, since $|V| < \infty$, this maximum is indeed achieved (possibly by several messages). We have that 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}.*

Proof: By definition of l_{max} and taking into account that any algorithm can move a packet only one position at each step. □

Definition 3.3 *We will say that 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, only one of them will be able to move on the next step of the algorithm.

Definition 3.4 *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 will call w_p^i the* waiting time of p *or, simply, the* delay of p.

Definition 3.5 *Given a packet p and its delay w_p^i, we will say that w_p^i is an* allowed delay *if $l_p + w_p^i \leq l_{max}$. Similarly, we will say that w_p^i is an* additional *(or* forbidden*)* delay *if $l_p + w_p^i > l_{max}$.*

Definition 3.6 *Finally, we will say that a packet p is* saturated *at the end of step i if $w_p^i = l_{max} - l_p$.*

The idea is that if a packet p is saturated, it must not wait anymore unless the algorithm becomes not optimal.

In Figure 4 a possible packet model is represented. In this model, each packet has 2 "boxes" with capacities l_p and $l_{max} - l_p$, respectively. At each step of the algorithm, two things can happen. If packet p has moved during step i, then an item of the box on the left (1st 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 (2nd box) is filled. If the 1st box is full, it means that packet p has reached its destination. Conversely, if the 2nd 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 greater than l_{max}).

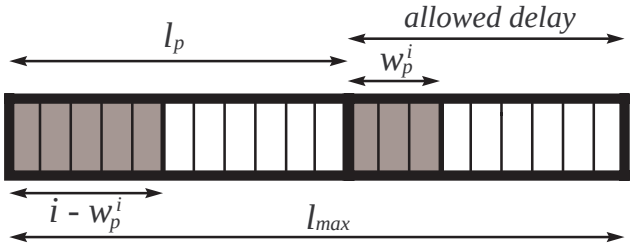

FIG. 4 – Packet model used on the permutation routing algorithm

Remark 3.1 *The model displayed in Figure 4 is only used for analysis, but in fact the only information that each packet p has to carry is l_p and w_p^i (i can be stored on the nodes). That is, only an integer and a counter. 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 [1], this would introduce additional time complexity of the preprocessing phase.*

We are ready now to give an exact description. An optimal message routing algorithm for each node u is described in Table 1. Note that the main loop for is infinite. We will see 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 are assuming 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.*

```
      At each node u of the network :

 1 :  begin
 2 :      preprocessing;
 3 :      for (i = 0, , i++) {
 4 :          Reception phase :
 5 :              for each packet in node u
 6 :                  update_packet;
 7 :          Transmission phase :
 8 :              for each packet in node u
 9 :                  decide_outgoing_edge;
10 :              for each queue
11 :                  order_queue;
12 :                  send the 1st packet of the queue;
13 :      }
14 :  end
```

TAB. 1 – **Algorithm $\mathcal{A}$**

In the next sections a detailed explanation of each of the procedures that appear on this general scheme will be given. Later, unless we say it explicitly, we are always assuming that the routing is executed under the rules defined by Algorithm $\mathcal{A}$.

3.1.1 Description of the procedure "preprocessing"

Before packets begin to be sent through the network, one packet is placed each node. Each packet has only the address of its destination node, namely D. With this information, each source node S can **compute the address of the packet** : $D - S$. If $D - S = (a, b, c)$, then 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, or only one non-zero component (recall Theorem 2.1).

After this, each node **computes the length of each packet's address**. If $D - S = (a, b, c)$ in the shortest path form, then $l_p = |a| + |b| + |c|$.

Finally, for each message a heading containing $D-S$, l_p and the counter w_p^i is added (recall that a packet is made of a message and a heading). Remark that this information is enough considering the packet model of Figure 4, as we will prove in Lemma 3.4.

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

3.1.2 Description of the procedure "update_packet"

This is just an updating of the packet's address. Another option could be to update when the node sends the packet, but updating the address in reception offers more robustness against link failures. After updating the address, we must check if the packet has reached its destination node (and then, the message is stored there).

Without ambiguity, we can label the 6 incoming edges of a node as it is shown in Figure 5. Being the previous packet address (a, b, c), we can describe this simple procedure as it is shown in Table 2. A higher level simplification is shown in Table 3.

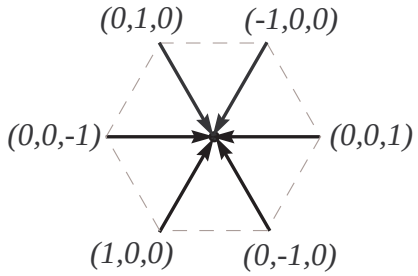

FIG. 5 – Possible incoming edges

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 we can assure that when packets meet, no additional delay is introduced, as we will prove in Proposition 3.1. When we have the address of the shortest path of a packet with 2 non-zero components, we can choose one of both directions to begin with, as we can see with an example in Figure 6. The procedure *decide_outgoing_edge* makes the choice between both alternatives in each case.

We can partition all the $|V|$ addresses 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. Recall that in an hexagonal mesh each node has 6 outgoing edges, and thus 6 queues. Without ambiguity, we can label the 6 outgoing edges of a node as it is shown in Figure 7.

```
1 :  begin
2 :      case  incoming_edge :
3 :          if (1, 0, 0)
4 :              then   (a, b, c) ← (a − 1, b, c);
5 :          if (−1, 0, 0)
6 :              then   (a, b, c) ← (a + 1, b, c);
7 :          if (0, 1, 0)
8 :              then   (a, b, c) ← (a, b − 1, c);
9 :          if (0, −1, 0)
10 :             then   (a, b, c) ← (a, b + 1, c);
11 :         if (0, 0, 1)
12 :             then   (a, b, c) ← (a, b, c − 1);
13 :         if (0, 0, −1)
14 :             then   (a, b, c) ← (a, b, c + 1);
15 :     end case ;
16 :     if (a, b, c) == (0, 0, 0)
17 :         then   this is the destination node ;
18 : end
```

TAB. 2 – Procedure **update_packet**

For the updated packet address (a, b, c), we can describe this routing procedure as it is shown in Table 4. Again, a higher level simplification is shown in Table 5.

For instance, if the packet address is of the type $(-, 0, +)$ then, according to Table 4, this packet goes first in the direction $-x$, and after in the direction $+z$. We symbolize this rule by the arrow $\leftarrow\!\!\checkmark$.

Giving a last example, the routing of the address $(+, -, 0)$ is represented by $\nwarrow$. In Figure 8 these routing rules are summed up.

Definition 3.7 *Given a packet p and its address (a, b, c) with 2 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 $D - S$ has only one non-zero component, we talk about the direction of *p.*

3.1.4 Description of the procedure "order_queue"

At each of the 6 queues of each node, the same priority policy applies : order the outgoing packets according to *decreasing remaining steps*. That is, the packet that has more remaining steps has priority 1, the next one has priority 2, and so on. Let's see in Lemma 3.2 and Lemma 3.3 that there cannot be equality in the number of remaining steps, and thus this is a correct policy.

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

Proof: According to the routing rules of *decide_outgoing_edge* procedure (see Table 4 or Figure 8), one can easily check that two shortest paths with two non-zero components can only intersect

1 :	begin
2 :	$(a, b, c) \leftarrow (a, b, c) - incoming\_edge$;
3 :	if $(a, b, c) == (0, 0, 0)$
4 :	then *this is the destination node ;*
5 :	end

TAB. 3 – Shorter description of the procedure **update_packet**

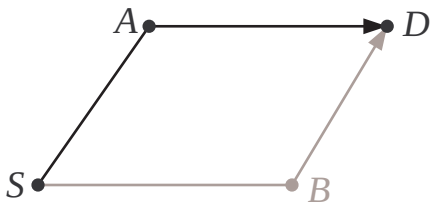

FIG. 6 – Given the $D - S$ address, there are 2 possible routings for the shortest path

when their first direction is already finished.
Since packets with only one direction can be in conflict obviously only on that direction, we can generalize this result by saying that a packet can wait only during its *last* direction. □

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

Proof: Because of Lemma 3.2, both packets 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, which is not possible in a permutation routing. □

3.2 Proof of optimality

The next result will be useful in the proof of Proposition 3.1, which is the main result that will allow 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: We have that for a packet p (see Figure 4) the number of remaining steps is $l_p - (i - w_p^i) = l_p - i + w_p^i$, and the remaining allowed delay is $l_{max} - l_p - w_p^i$. Thus, besides of the sign, both magni-

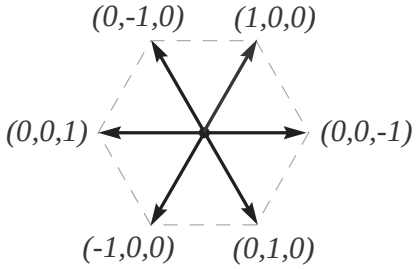

FIG. 7 – Possible outgoing edges

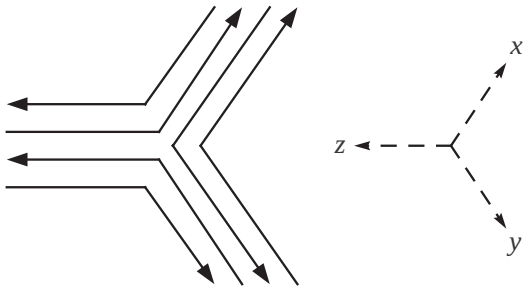

FIG. 8 – Routing of the packets according to Algorithm $\mathcal{A}$

tudes differ only on i and l_{max}, that 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 will prove by induction on the number of steps that no additional delay is introduced at any queue of the network after a generic step i.
First of all, after the first step ($i = 1$) the only packets that could have additional delay are those with 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 remaining steps policy, and thus none has waited.
Now suppose that no additional delay has been introduced after step $i-1$ ($i \geq 2$) and let's see 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 will have maximum priority because of Proposition 3.4, and thus this packet will not wait. Now, suppose that there are 2 saturated packets p and p' in the same queue. By definition of saturated packet, we have that $l_p + w_p^i = l_{p'} + w_{p'}^i = l_{max}$. The number of remaining steps of p is $l_p - (i - w_p^i) = l_{max} - i$, which is the same as the number of remaining steps of $p' : l_{p'} - (i - w_{p'}^i) = l_{max} - i$. Thus, both pa-

```
1 :  begin
2 :      case  packet_address :
3 :          if (+, 0, 0)
4 :              then   outgoing_edge = (1, 0, 0);
5 :          if (−, 0, 0)
6 :              then   outgoing_edge = (−1, 0, 0);
7 :          if (0, +, 0)
8 :              then   outgoing_edge = (0, 1, 0);
9 :          if (0, −, 0)
10 :             then   outgoing_edge = (0, −1, 0);
11 :         if (0, 0, +)
12 :             then   outgoing_edge = (0, 0, 1);
13 :         if (0, 0, −)
14 :             then   outgoing_edge = (0, 0, −1);
15 :         if (+, −, 0)
16 :             then   outgoing_edge = (0, −1, 0);
17 :         if (−, +, 0)
18 :             then   outgoing_edge = (−1, 0, 0);
19 :         if (+, 0, −)
20 :             then   outgoing_edge = (0, 0, −1);
21 :         if (−, 0, +)
22 :             then   outgoing_edge = (−1, 0, 0);
23 :         if (0, +, −)
24 :             then   outgoing_edge = (0, 0, −1);
25 :         if (0, −, +)
26 :             then   outgoing_edge = (0, −1, 0);
27 :     end case;
28 :  end
```

TAB. 4 – Procedure **decide_outgoing_edge**

ckets must 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 mesh networks.*

Proof: Because of Proposition 3.1, all packets reach their destination nodes in a number of steps that is lower or equal 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* (see, for instance, [7, 6]). Let's give a precise definition.

```
1 :  begin
2 :      if  packet_address has only 1 non-zero component
3 :              outgoing_edge = the edge corresponding to the non-zero component;
4 :      else
5 :              outgoing_edge = the edge corresponding to the negative component;
6 :  end
```

TAB. 5 – Shorter description of the procedure **decide_outgoing_edge**

Definition 3.8 *A routing algorithm is called* oblivious *if the path of v for each node v depends only in v and its destination, although the waiting time at an intermediate node may depend on other paths.*

In other words, for an oblivious algorithm, the routing of a packet is determined only by its origin and destination.

Definition 3.9 *Being $\mathcal{P}_{uv}$ the path between u and v, and oblivious algorithm $\{\mathcal{P}_{uv} : u, v \in V\}$ is called* translation invariant *if $\mathcal{P}_{(u+w)(v+w)} = \mathcal{P}_{uv} + w$, $\forall\ u, v, w \in V$.*

Thus, a translation invariant oblivious algorithm is completely determined by paths $\mathcal{P}_{uv}$, for all $v \in V$ and $u = \vec{0}$.

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

Proof: Optimality (that is, 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 each packet. Finally, it is clear that to route a packet only the difference $D - S$ between the source and destination node is necessary, and thus we have proved the invariance. □

3.3 About the queue policy

We shall remark here that one of the most important points on the description of the algorithm is what refers to *queue policy*. 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 must be different. Although both policies seem to be similar, the total length policy combined by the routing issues shown in Figure 8 yields a non-optimal algorithm, while the remaining steps policy is optimal (as we have proved in Theorem 3.1). Indeed, let's see with a counterexample that the total length policy is not optimal. In Figure 9 the origin nodes for each packet are labeled with small letters, while their corresponding destination nodes are labeled with capital letters. Thus, we have that $l_{A-a} = 6$, $l_{B-b} = 5$, $l_{C-c} = 5$ and $l_{P-p} = 4$. One can check that the total numbers 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 $7 > 6 = l_{max}$.

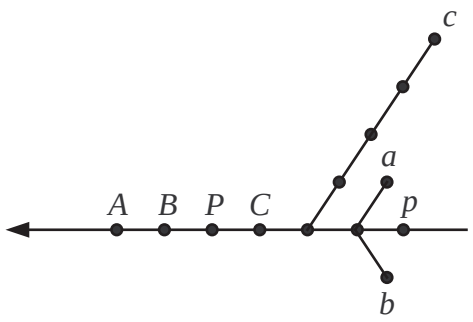

FIG. 9 – Counterexample to the *total length* queue policy

4 Conclusions

In this paper, we have presented a distributed optimal routing algorithm for full-duplex hexagonal networks. Furthermore, we have proved that this algorithm is oblivious and translation invariant. The next natural step is considering not full-duplex hexagonal 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 an obvious generalization in the three-dimensional hexagonal network [4], where the "basic" pieces that "tesselate" the space are tetrahedrons. The first thing one can think of is finding 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.

Acknowledgement. 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 project COST 293 (GRAAL).

References

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

[2] J. C. Cogolludo and S. Rajasekaran. Permutation Routing on Reconfigurable Meshes. *Algorithmica*, 31 :44–57, 2001.

[3] A. Datta. A Fault-Tolerant Protocol for Energy-Efficient Permutation Routing in Wireless Networks. *IEEE Transactions on Computers*, 54(11) :1409–1421, November 2005.

[4] 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, September 2005.

[5] 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.

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

[7] F. K. Hwang, T. S. Lin, and R. H. Jan. A Permutation Routing Algorithm for Double Loop Network. *Parallel Processing Letters*, 7(3) :259–265, 1997.

[8] G. E. Jan and M.-B. Lin. Concentration, load balancing, partial permutation routing, and superconcentration on cube-connected cycles parallel computers. *J. Parallel Distrib. Comput.*, 65 :1471–1482, 2005.

[9] 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.

[10] S. Klavžar, A. Vesel, and P. Žigert. On resonance graphs of catacondensed hexagonal graphs : structure, coding, and hamiltonian path algorithm. *MATCH Communications in Mathematical and in Computer Chemistry*, 49(49) :99–116, 2003.

[11] E. Kranakis, H. Sing, and J. Urrutia. Compas Routing in Geometric Graphs. In *Proc. 11th Canadian Conf. Computational Geometry*, pages 51–54, 1999.

[12] W. Liang and X. Shen. Permutation Routing in All-Optical Product Networks. *IEEE Transactions on Circuits and Systems*, 49(4) :533–538, April 2002.

[13] 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, September 2002.

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

[15] I. Stojmenović. Honeycomb Networks : Topological Properties and Communication Algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 8(10) :1036–1042, October 1997.

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

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