International Journal of Foundations of Computer Science
© World Scientific Publishing Company

OPTIMAL CONSTRUCTION OF SENSE OF DIRECTION IN A TORUS BY A MOBILE AGENT

HANANE BECHA
University of Ottawa, Canada. [*]

and

PAOLA FLOCCHINI
University of Ottawa, Canada. [†]

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Sense of direction is a property of the edge-labeling of a network whose availability facilitates computations and often decreases their complexity. In this paper we consider the problem of providing a sense of direction to an anonymous, unoriented torus. The edges of the torus are initially labeled arbitrarily, and they have to be relabeled with a classical compass sense of direction.
We first describe an algorithm where the relabeling of the edges is performed by a mobile agent that carries a token. The algorithm is *optimal* both in terms of number of tokens (with no tokens the task cannot be performed), and in terms of number of moves. We then show that the same technique can be used to orient the torus in the classical message-passing environment, thus obtaining an orientation algorithm that improves all the existing ones in this setting.

1. Introduction

1.1. The Problem

Sense of direction is a particular way of labeling the edges of a network that has been defined in [17], and has been extensively studied since then from different perspectives (graph theory, communication networks, theoretical computer science) (e.g., [9, 8, 4, 18, 19]). Among other results, it has been shown that having a sense of direction in a network is extremely important both from computational and complexity points of view; for example, it has been shown that when a network

[*]School of Information Technology and Engineering, University of Ottawa, 800 King Edward, Ottawa, Canada. hbecha@site.uottawa.ca.

[†]School of Information Technology and Engineering, University of Ottawa, 800 King Edward, Ottawa, Canada. flocchin@site.uottawa.ca.

has sense of direction, several problems can be solved more efficiently, while others, which would be unsolvable otherwise, become solvable (e.g., see [16, 18, 26]).

Given the importance of sense of direction, an obviously interesting problem is its construction in a network that is initially labeled arbitrarily (this problem is sometimes referred to as the orientation problem). In classical message-passing distributed settings the orientation problem has been extensively studied (e.g, see [3, 11, 27, 22, 23, 29, 28, 30]). Typically, the works in the literature concentrate on specific topologies (e.g., the ring), whose edges are initially labeled arbitrarily, and determine distributed algorithms for relabeling the edges so to obtain an orientation (or a sense of direction): for example, a left/right labeling in the case of the ring. Depending on the assumptions on the system (anonymity versus distinct Ids for the nodes, presence of a leader, knowledge of the size of the network, etc.) various algorithms have been proposed and compared.

In mobile agents' environment, the orientation problem has never been directly addressed; it is however closely related to the widely studied *Map Construction* problem (e.g., see [2, 5, 12, 13, 14, 20, 24]). In the map-construction problem an agent has to traverse the network and to reproduce, in its local memory, an edge-labeled map isomorphic to the graph it is moving on. Once the map is constructed, the agent could easily relabel the edges according to a sense of direction.

In this paper we are interested in constructing a compass labeling of an anonymous Torus. The torus has anonymous nodes and its links are labeled according to an arbitrary local orientation (e.g., every node has four distinct labels for its four incident links); at the end of the computation the torus must be labeled with a compass sense of direction. The relabeling must be performed by a single mobile agent that moves from node to node. As in the classical works on Map Construction, we assume the agent has a token that can be placed on a node or removed from it and no other means to mark the nodes of the network while moving around (*Token Model*). From our technique we then derive a distributed algorithm for the message-passing setting that has a better bit complexity than the existing solutions.

1.2. Related Work

Studies on Map Construction of edge-labeled graphs (or digraphs) by an agent, have emphasized minimizing the cost of exploration in terms of either the number of moves (edge traversals) or the amount of memory used by the agent (e.g., see [2, 12, 13, 24]). Map construction of *anonymous* graphs is possible only if the agent is allowed to mark the nodes in some way; except when the graph has no cycles (i.e. the graph is a tree [14, 20]). For exploring arbitrary anonymous graphs, various methods for marking nodes have been used by different authors. Bender *et al.* [5] proposed the method of dropping a token on a node to mark it, and showed that any strongly connected directed graph can be explored using just one token if the size of the graph is known, and using $O(\log \log n)$ tokens, otherwise. Dudek *et al.* [15] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [6] was to employ two cooperating agents, one of which would stand on a node, thus marking it, while the other explores new edges. The whiteboard model (i.e., nodes have locally available a whiteboard where information can be written and read by the agents) has been used by Afek at al. in [1] and more recently by Fraigniaud et al.[21] for exploring directed graphs and by Fraigniaud et al. [20] for exploring trees.

The Orientation problem has been extensively studied in message-passing distributed settings (not supporting mobile agents) (e.g, see [11, 3, 27, 22, 23, 29, 28, 30]). In particular, the orientation for torus has been studied in [23, 27, 29, 30]. In [27, 29] the problem was examined in the different context of self-stabilization. In the case of [30] a $m \times n$ torus is oriented exchanging $O(N \log N)$ messages (where $N = m \times n$) when the nodes have distinct Ids and there is no Leader, and $O(N)$ when there is a Leader; in the case of [23] the orientation is achieved exchanging $O(N)$ messages regardless of the presence of the leader, as long as the nodes have distinct Ids. All algorithms in [23] and [30] exchange messages of $O(\log N)$ bits, as a consequence the linear message algorithms actually have a bit complexity of $O(N \log N)$.

1.3. Our Results

As mentioned above, a problem related to ours is the Map Construction problem. The two problems are not exactly the same, since in our case the network topology is known, and to be determined are its size and its initial labeling. Once the mobile agent determines dimensions and labeling of the torus, it can relabel it according to a compass sense of direction.

As observed above, in the token model the map of an arbitrary anonymous graph can be constructed using a constant number of tokens *only if the size of the graph is known*; otherwise more tokens are necessary [5]. An interesting question is for what classes of graphs a single token is sufficient to construct the edge labeling when the size of the graph is not known. In this paper we address this question by showing that, in the case of the torus, one token is indeed sufficient; we then design an algorithm that is also optimal in terms of number of moves. More precisely, we consider a $n \times m$ *Torus*. The torus is *anonymous*, i.e., the nodes are all identical and cannot be distinguished. Furthermore, the torus is not oriented, i.e., the links incident to each node are labeled with four distinct labels, but not with a consistent compass. The agent is located in an arbitrary node of the torus (the *homebase*) and can move *asynchronously* from node to node (i.e., the time it takes for the agent to move on a link is finite, but otherwise unpredictable); the agent knows the network is a torus but does not know its size. Moreover, the agent can carry one or two *tokens* that can be dropped on and picked up from the nodes of the network. In this model the agent has locally available enough memory to store information about the network while moving around. The efficiency of our solutions is measured in terms of number of moves performed by the agent. We first describe a simpler algorithm for constructing the edge-labeling of the torus when the agent has available two tokens. Essentially one token is used for detecting termination, while the other is used to actually construct the map. The algorithm is optimal in terms of number of moves, which are $O(N)$. We then modify this solution to adapt it to the situation when *only one token* is available. In this case, we do use the single token both to construct the labeling and to check for termination. This solution is both move-optimal and token-optimal since without a token the map cannot be constructed. Once the agent has a labeled map of the torus, it can relabel it according to a compass labeling in O(N) additional moves.

A technique that derives from the previous result can be applied in the message-passing model where we can then design a distributed algorithm for orienting the torus that exchanges $O(N)$ messages of constant size. Notice that $O(N)$ solutions already exist (see [23, 30]). However, in both of them, the size of the message is $O(\log N)$ bits, resulting in an overall bit complexity of $O(N \log N)$. Moreover, in the algorithms of [23, 30] either the nodes must have distinct Ids to start with, or these Ids have to be pre-computed. This implies that the nodes of the torus must have at least a space of $\log N$ bits to store this information. With our solution we do not require the pre-computation of distinct names for the nodes, which can remain anonymous until the end of the algorithm. Thus, with our algorithm the only space required at the nodes is the space to store the four labels associated to the incident edges, which is constant.

2. The Models and their Relationship

In this section we describe the computational models considered in the paper and related models. We also make some simple observations about the relationship between them.

Message-passing. A message-passing distributed environment consists of a finite collection of computational entities (the nodes) connected by communication links (the edges) that communicate by means of messages [25]. Each entity is endowed with local memory and has the following computational capabilities: it can access (storage and retrieval) the local memory, has local processing, and can send and receive messages.

At each time during the algorithm an entity x is in one state among a finite possible set of states and its behaviour is reactive: x only responds to external event (for example, to the arrival of a message) by executing a set of actions (and possibly changing state). A particular type of event is the *Spontaneous* event that triggers the initiator(s) of the computation. The main complexity measure for evaluating the performances of a distributed algorithm is message complexity: i.e., the total number of messages exchanged by the nodes during the computation. A more precise measure of communication is bit complexity, where the total amount of information exchanged is taken into account (number of messages and their size).

Mobile Agents. While the model of a message passing distributed environment has been widely studied and precisely defined, the mobile agents computational model has never been formalized precisely. In a mobile agent environment the nodes are passive entities, while the agents move from node to neighbouring node performing computations. Agents have some local memory they carry with them. In this setting two models that are often employed are the following:

- *Whiteboard* Model. The nodes have a finite amount of storage (typically of size $O(\log N)$) where agents can write and read. Whiteboards are accessed in mutual exclusion. When multiple agents are operating in the network, they can use the whiteboards to communicate to each other. If a single agent is present, it can use it to mark the nodes or write topological information. The whiteboard model has been used for exploration purposes, for example, by [1, 20, 21].

- *Token* Model. Each agent has one or more tokens that carries with itself. A tokens can be placed on a node or removed from it, and an agent can detect the presence of a token at a node. Each node has a small storage area (one bit only) to be able to contain one token. When multiple agents are operating in the network, they can use the tokens for communication or synchronization purposes. If there is a single agent, tokens allow it to mark nodes. When there are k tokens available for each agent, we call this model k-Token Model. Typically the effort is to design solutions for the 1-token model, when possible. The token model is a "classical" model that has been extensively studied and employed especially for exploration purposes (e.g., see [5, 13, 24]). The main reasons for the attention that has been given to this model is that it allows to see what problem can be solved with minimal marking capabilities to the agents and minimal memory at the nodes.

As mentioned above, typically the whiteboards have size of at least $O(\log N)$ (where N is the number of nodes of the network); this implies that any algorithm in the token model that uses less than $\log N$ tokens can be easily simulated in the whiteboard model (where the whiteboard is obviously used to store tokens). Thus we have:

Theorem 1 *Any algorithm in the Token Model that uses less than* $\log N$ *tokens per agent can be simulated in the Whiteboard Model, with whiteboards of size* $O(\log N)$.

The reverse is not true, in fact:

Theorem 2 *The* $O(1)$*-Token model is strictly weaker than the Whiteboard Model with whiteboards of size* $O(\log N)$.

Proof. We show that here exists problems that cannot be solved in the $O(1)$-token model, but can be solved with whiteboards of size $O(\log N)$. This is the case, for example, of the Map Construction problem which, in certain graphs, has been shown to be unsolvable using a constant number of tokens (see [5]). In the same paper it is shown that $O(\log \log n)$ tokens are sufficient. By Theorem 1 we have that the problem can be solved in a whiteboard model with whiteboards of size $O(\log N)$. □

It is interesting to observe the relationship between the computability power of the message-passing model and the one of the mobile agents model. It is quite obvious to see that:

Theorem 3 *[4] Any mobile agent algorithm in the Whiteboard Model can be simulated by a distributed algorithm in a message-passing system system.*

More involved is the reverse:

Theorem 4 *[10] Any distributed algorithm in a message-passing system can be simulated by a mobile agents' algorithm in the Whiteboard Model, with whiteboards of unbounded size.*

Essentially the messages exchanged in the distributed algorithm are carried by the mobile agents and the actions they would trigger in the message-passing system, are executed by the agents in the simulation. For the simulation to be possible, the whiteboards are heavily used to contain messages to be delivered and actions to be executed. In conclusion, the message-passing model and the whiteboard model are computationally equivalent (when the whiteboards are big enough). The typical complexity measure for the message passing setting (number of messages) would correspond to the number of moves performed by the agents in the mobile agents setting.

Since the $O(1)$-token model is weaker than the whiteboard one, the equivalence with the message passing setting obviously does not hold.

3. Orientation by the Mobile Agent

Let us first summarize the model's assumptions.

A single agent is placed on a node of the torus. The agent knows it is on a torus, the size of the torus is however unknown. The agent has a local memory where it will draw a labeled map of the torus; in order for the agent to perform the task, we assume the local memory is large enough to contain the final map. Notice that this assumption gives the agent an upper bound on the size of the torus; such an information is however not useful to compute the map more efficiently. The agent has also one (or, in some cases, two) tokens that can be placed on nodes or removed from them. The network is anonymous (i.e., all nodes are identical), and links adjacent to a node are identified with distinct port numbers, which however do not give any particular orientation.

3.1. Optimal Algorithm Using Two tokens

In this subsection we consider the case when the agent has available two identical tokens. In this case, we design an optimal algorithm that constructs the labeled map in $O(N)$ moves (where $N = n \times m$ is the size of the torus). In the next subsection we will modify the technique to obtain the same result reducing the number of tokens.

The Algorithm. The idea of the algorithm is to first construct the labelings of a column and of a row intersecting at the home base, and then to complete the map by incrementally constructing all the other rows and the other columns.

During the algorithm, one of the tokens is always kept at the homebase, while the second token is used by the agent to move around the torus. More precisely, while constructing a column (or a row), the agent uses the second token to move in a straight direction to construct a column (row).

Before describing in details the idea of our solution, we introduce some terminology. We call *expansion* of node x the action of the agent of visiting all the nodes at distance smaller then or equal to two from x (i.e., visiting the neighbourhood at distance two). We call *d-expansion* of node x ($1 \leq d < 4$) the action of visiting the nodes at distance smaller than or equal to two from x passing only through d of the four neigbhours. When we talk about a d-expansion of a node, we will specify through which of the four neighbours it is performed. The high level description of the algorithm is given below in Figure 1.

```
Protocol CONSTRUCT MAP  from $X_0$ (the homebase)
Release a token at $X_0$, the homebase.
CONSTRUCT COLUMN($X_0$)
 (* the column is composed by nodes $X_0, X_1, \dots, X_{n-1}$ *)
back at $X_0$
CONSTRUCT ROW($X_0$)
 (* the row is composed by nodes $X_0, Y_1, \dots, Y_{m-1}$ *)
go to $X_1$, i:=1
While not back at homebase do
        SELECT ROW DIRECTION
        CONSTRUCT ROW($X_i$)
        i:=i+1
end-while
go to $Y_1$, i:=1
While not back at homebase do
        SELECT COLUMN DIRECTION
        CONSTRUCT COLUMN($Y_i$)
        i:=i+1
end-while
```

Figure 1: Protocol CONSTRUCT MAP.

First Column and Row. The idea is as follows: from the homebase (let us call it X_0), the agent releases its first token marking its starting node; it chooses an arbitrary direction which determines the direction of movement, it then moves on that link reaching a node (call it X_1). Node X_1 is the next node in the column that the agent is trying to traverse, the agent starts drawing the column in its local memory. At this point, the agent has to find the next node in the column.

Notice that, the next node to be included in the column is the one, out of the three "candidate" nodes that are adjacent to X_1, that would *not* be visited in the 3-expansion of X_0 (i.e., the expansion that does not pass through X_1). In order to find this node, the agent checks, one by one, the three candidates. The checking procedure for candidate v works as follows: the agent drops its second token in v, it

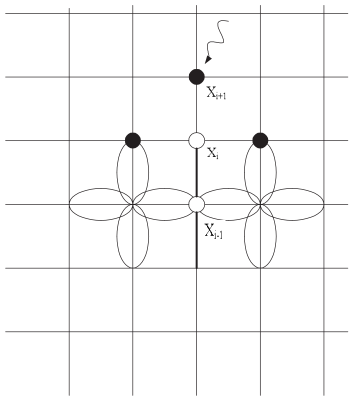

Figure 2: A 2-expansion from X_{i-1}. The black nodes are the candidate nodes.

then performs a 3-expansion of the homebase (not passing through X_1). If, during that expansion, the agent finds its token, then v *is not* the node to add to the column and therefore the agent has to preform the 3-expansion again for the next candidate. On the other hand if, during the 3-expansion for candidate v, the agent does not pass by v (i.e., by the node with the token) then v is the right node to add to the column. As we will show, the selected node (let us call it X_2) is unique, and the agent can move there to pick up its token and to continue the exploration for the next node to be added to the column.

Notice that, when looking for X_{i+1}, with $i > 1$ the agent needs to perform the expansions from X_{i-1} only in the two directions different from the ones leading to X_{i-2}, and X_i (*2-expansion*). In fact, let $X_0, X_1, \ldots, X_i$ be the first $i+1$ nodes included in the column with $i \geq 1$: then, to find the next node X_{i+1}, the agent can perform a 2-expansion from X_{i-1} selecting as X_{i+1} the unique node that is neighbour of X_i and has not been visited in the expansion (see Figure 2). The high level description of the column construction algorithm is given in Figure 3.

The procedure to construct a column terminates when the agent drops its second token in a node that does already contain a token (i.e., it is back at the homebase). Notice that, at this point the agent knows one dimension of the torus.

Once the column is constructed, the agent proceeds to construct the row (Algorithm CONSTRUCT ROW not reported here) following exactly the same procedure in the other direction and finding, as a byproduct, also the second dimension of the torus. The agent now knows the size of the torus; in order to construct the full map, it has to discover the correct labeling.

Other rows and columns. At this point the agent constructs, one by one, all the other rows following the same procedure described above, this time preceded by a procedure SELECT DIRECTION to select the correct orientation of each row (column).

Let $X_0, \ldots X_{n-1}$ be the nodes of the first column. The agent starts from the

```
Protocol CONSTRUCT COLUMN($X_0$)
Release a token at $X_0$, the homebase.
choose an arbitrary direction and move to
  an arbitrary neighbour $X_1$
draw $(X_0, X_1)$ with its labels in the local map
Repeat until back at the homebase
   /* Let $X_0 \dots X_i$ $(i \geq 1)$ be already constructed */
   consider as candidates the three neighbours of $X_i$
     different from $X_i$
   for each candidate
        release token at candidate
        If $i = 1$ then
           3-EXPAND($X_0$)
           /* do not expand through $X_{i-1}$ */
        If $i > 1$ then
           2-EXPAND($X_{i-1}$)
           /* do not expand through $X_{i-2}$ and $X_i$ */
        If no token is found
           this candidate is $X_{i+1}$: the next in the column.
           move to $X_{i+1}$ and
           mark $(X_i, X_{i+1})$ in local map. $i = i + 1$
```

Figure 3: Protocol CONSTRUCT COLUMN.

row corresponding to X_1 and then proceeds to the other rows. Before constructing the row corresponding to X_i, the agent has to correctly choose the orientation of the row between the two possible departing directions. To do so, it places the token in one of the two directions, it then goes to X_{i-1} and performs a 2-expansion. The 2-expansion will reveal the direction to be followed in the construction of the row (see Figure 4 and the algorithm in Figure 5).

Since the agent now knows the dimensions of the torus, the construction of each row terminates when the correct number of nodes has been included. The procedure to construct the columns is similar.

Correctness and Complexity

Lemma 1 *Using Algorithm* CONSTRUCT COLUMN*, the agent walks in a straight direction and correctly constructs a column.*

Proof. We prove by induction that nodes $X_0, X_1, \dots$ selected by Algorithm CONSTRUCT COLUMN are indeed consecutive nodes in a column.

Basis. Node X_1 is chosen arbitrary and is giving the direction of the column. An expansion from X_0 is then performed for each of the three candidate neighbours of X_1. By definition of candidate node and by the torus topology, the only node at distance one from X_1 which is not at distance two from X_0 is the next in the column. Thus, the only candidate node that is not visited during an expansion from

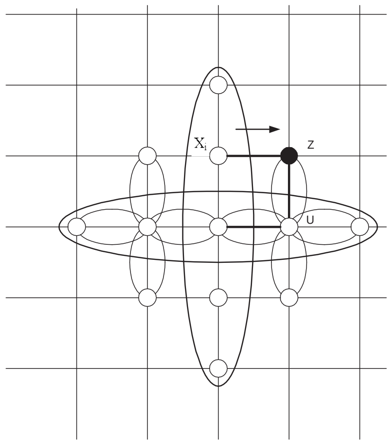

Figure 4: Determining the direction of the row starting from X_i.

SELECT ROW DIRECTION from X_i

```
- Place the token in an arbitrary neighbour $z$ of $X_i$
(different from $X_{(i-1)modn}$ and $X_{(i+1)modn}$)
- move to $X_{i-1}$
- perform a 2-expansion from $X_{i-1}$
(* not passing through $X_i$ and $X_{i-2}$ *)
- let $(X_{i-1}, u)$ be the edge through which the expansion
finds the token
- construct in your map $(X_{i-1}, u)$, $(X_i, z)$, and $(u, z)$
- continue the construction in the direction of $z$
```

Figure 5: Protocol SELECT ROW DIRECTION.

X_0 (because the expansion does not pass by X_1) is the next node in the column. This node is then correctly considered by the agent as X_2 when the token is placed on it and the expansion does not find it.

Induction. Let us assume that the agent has correctly moved in the same direction for the first $i-2$ steps (i.e., up to the expansion from the node X_{i-2}), thus finding that $X_0, X_1, \ldots X_i$ are consecutive nodes in a column. Consider now the next step of the algorithm (i.e., the expansion from the node X_{i-1} to determine node X_{i+1}). The algorithm performs a 2-expansion from X_{i-1} (i.e., not expanding X_{i-2} and X_i, see Figure 1) for each of the three candidate neighbours. By definition of torus, the only node at distance one from X_i which is not at distance two from X_{i-1} is the next in the column. This means that the only candidate that is not visited during an expansion from X_{i-1} (because the expansion does not pass by X_i) is precisely that node. Thus, when the expansion does not meet any token, it means that the token is currently located on the correct candidate, which is then considered X_{i+2}.

Finally notice that the construction of the column terminates when a token is placed on a candidate node that already contains a token (i.e., the agent is back at the homebase). □

Analogously we have that:

Lemma 2 *Using Algorithm* CONSTRUCT ROW*, the agent walks in a straight direction and constructs a row.*

Theorem 5 *Algorithm* CONSTRUCT MAP *is correct.*

Proof. From Lemmas 1 and 2, it follows that the first column and the first row are correctly constructed. At this point the construction of each new row and of each new column is preceded by the selection of the direction (algorithm SELECT DIRECTION). Consider the construction of the rows (the construction of the columns is analogous). Before constructing the row corresponding to X_i, the agent places the token on a neighbour z of X_i that does not lie on the column. It then performs a 2-expansion from X_{i-1}. Let us denote by u and w the two neighbours through which the expansion is performed (see Figure 4); obviously, the 2-expansion will find the token only in correspondence of one neighbour (in the figure, neighbour u). Thus, the agent can correctly draw in its map the edges (X_i, z), (X_{i-1}, u), and (u, z). The construction of the row correctly proceeds in the direction of z. □
We now compute the number of moves involved.

Theorem 6 *The number of moves performed by the agent to construct the map is* $\Theta(N)$*, which is optimal.*

Proof. Each expansion requires $O(1)$ moves and the agent performs $O(n)$ expansions for selecting the n nodes of the first column. Thus, each procedure CONSTRUCT COLUMN requires $O(n)$ moves, each CONSTRUCT ROW requires $O(m)$ moves. The algorithm performs procedure CONSTRUCT COLUMN m times, and procedure CONSTRUCT ROW n times, for a total of: $O(m \times n)$ moves. This complexity is clearly optimal, since to construct a torus of size $m \times n$, the agent has to visit at least $m \times n$ nodes. □

3.2. Reducing the Number of Tokens: Optimal Algorithm Using One token

We now discuss what happens if the agent has only one token available. We first very briefly show a quadratic algorithm that is a slight modification of the previous. We then show that an optimal algorithm with linear number of moves can be obtained also in this setting.

Using the algorithm of the previous section, the agent is still able to walk in a straight direction; it is not however able to detect the termination of the column (row) since there is no token available to mark the homebase.

Checking for termination: an idea. We could solve the termination problem by checking for termination each time a new node is added to the first column and to the first row as follows: when a node X_i is included in the column the agent has to check whether $X_i = X_0$ or not. The agent goes back to the homebase with its token, releases it there, then travels on the portion of the column just constructed

$[X_0, X_1, \ldots, X_i]$ and, if the token is found on X_i it decides that the column has been entirely constructed (i.e., $X_i = X_0$) and starts the construction of the row. It follows the same procedure during the construction of the row. After the first column and row (intersecting on the homebase) are constructed, the algorithm proceeds exactly like in the previous section since at this point the number of nodes in a column (row) is known and there is no need to check for termination. We call the algorithm for constructing the first column using this termination procedure: CONSTRUCT FIRST COLUMN WITH ONE TOKEN. As we see below, this idea requires $O(N^2)$ moves.

Lemma 3 *Algorithm* CONSTRUCT FIRST COLUMN WITH ONE TOKEN *is correct.*

Proof. From Lemmas 1 and 2, we know that the selection of the next node to be included in the column is correct. We have only to show that the algorithm can correctly terminate the construction of the column (row). Since after each new node X_i, the portion of the ring $X_0, \ldots X_i$ is known, the agent can correctly move back to the homebase to release the token. If $X_i = X_0$, then $X_0, \ldots X_i$ is a ring; thus, in this case the agent will find its token there. Since we are checking after each new node is added, if the token is not found in X_i, it means that $X_0, \ldots X_i$ is only a portion of the ring and the column has not been constructed yet. □

Analogous proof holds for the construction of the first row. At this point the algorithm is identical to the one of the previous section.

Theorem 7 *The number of moves performed by the agent to construct the map with one token is $O(N^2)$.*

Proof. Consider the construction of the first column. Each expansion requires $O(1)$ moves and the agent performs $O(n)$ expansions for selecting the n nodes of the first column. Furthermore, for each portion $X_0, \ldots, X_i$ the agent performs $O(i)$ moves to check for termination, for a total of $O(n^2)$ moves for checking termination. Thus, $O(n^2)$ moves are required in total for the construction of the column. Analogously, $O(m^2)$ moves are required for the construction of the first row. At this point, for each of the other $n-1$ rows, $O(m)$ moves are performed, for each of the other $m-1$ columns, $O(n)$ moves are performed. The total number of moves is then: $O(n^2 + m^2 + 2nm)$. The worst case occurs when one of the dimensions is $O(N)$; in this case, in fact the number of moves would be $O(N^2)$. □

Token-Optimal and Move-Optimal Solution. In order to obtain a linear solution, we use the same idea; however, instead of checking for termination every time a new node is included in the column (row), we proceed at successive steps, starting from step 0. The agent checks for termination every time that the column is composed by 2^i nodes. In other words, at step 0 the agent adds one node to the column and then checks for termination; at step i the agent adds new nodes to the column until the current column is composed by 2^i nodes, it then comes back to the homebase releasing the token to check the termination condition; if the termination condition is not met, the agent moves to step $i+1$ and continue the construction.

The high level description of the algorithm for constructing the first column with one token is given in Figure 6.

```
Protocol LINEAR CONSTRUCT FIRST COLUMN
WITH ONE TOKEN from $X_0$ (the homebase)
Select $X_1$; $i := 2$; $j := 0$; $done = false$
While not done do
    Repeat $2^{j-1}$ times when $j > 0$ (once when $j = 0$)
        Expand $X_{i-1}$ to determine $X_i$
        Draw new labeled edge in local map
    End Repeat
    Move to $X_0$ with token; release the token
    Traverse portion already constructed
    If find token $done = true$
end-while
```

Figure 6: Optimal Protocol.

Lemma 4 *Algorithm* LINEAR CONSTRUCT FIRST COLUMN WITH ONE TOKEN *is correct. The number of moves performed by the agent to construct the map with one token is* $\Theta(N)$.

Proof. The only difference between this solution and the quadratic one is that the termination condition is checked only once for each step; i.e., only when the column is composed by 2^i nodes ($i > 0$). Clearly, when 2^i is smaller than n, the termination condition is not met and the algorithm continues; on the other hand, when $2^i \geq n$ during the checking procedure the token will be found and the algorithm terminates.

To calculate the number of movements, consider first the construction of the first column. Each expansion requires $O(1)$ moves and the agent performs $O(n)$ expansions for selecting the n nodes of the first column. Furthermore, at step i the agent performs $2 \cdot 2^i$ moves to check for termination, which is detected when $2^i \geq n$; i.e., when i is $\lceil \log(n) - 1 \rceil$. Thus, the total number of moves for checking the termination of the column is $\sum_{i=0}^{\lceil \log(n)-1 \rceil} 2^{i+1}$, which is $O(n)$. In total $O(n)$ moves are required for the construction of the column. Analogously, $O(m)$ moves are required for the construction of the first row. At this point, for each of the other $n-1$ rows, $O(m)$ moves are performed, for each of the other $m-1$ rows, $O(n)$ moves are performed. The total number of moves is then: $O(n + m + 2nm)$, which is $O(N)$. This complexity is clearly optimal, since the agent has to visit all nodes. □

4. Orientation by Messages

As mentioned in the introduction, the orientation problem of the Torus has been studied in the message passing system in [30] and [23]. They distinguish between a model where the nodes have distinct Ids (Named Torus) and one where there exists a Leader (Leader Torus). In [30] it is shown an algorithm employing $O(N \log N)$ messages in a Named Torus, and $O(N)$ in a Leader Torus . In [23] it is shown that also in the Named Torus the linear $O(N)$ bound can be achieved;

the results derived from a new leader election algorithm for the unoriented torus, which is also described in [23]. Moreover, an algorithm for the orientation of a Leader Torus that achieves the complexity of $O(N)$ messages is also described. The algorithm, however, assumes that the nodes have distinct Ids. Alternatively, it could be preceded by a pre-naming of the nodes. In both [23] and [30] , messages have size $O(\log N)$ since they contain nodes' Ids; so the total bit complexity of their algorithm is $O(N \log N)$.

Notice that the solutions in [23, 30] could be easily "translated" in a *Whiteboard* mobile agents setting. They cannot, however, be transformed into a mobile agent solution in the weaker 1-Token Model. The main reason is that messages that arrive at a node in the distributed algorithms contain information of size $O(logN)$ which need to be stored at the nodes. While this could be easily achieved by having an agent carrying the information and writing it on the whiteboard, it cannot be performed when the whiteboard is not available. On the other hand, the distributed algorithm that we describe below is a "translation" of the mobile agents' algorithm described in Section 3.

Our Algorithm is described for the Leader Torus, and achieves a bit complexity of $O(N)$ without using any assumption on the presence of distinct Ids, neither by requiring a labeling of the nodes. In our case, in fact, all nodes (except for the leader) are (and will stay) anonymous. As a consequence, we just require a constant space at the node's memory and messages are of constant size.

The Algorithm. The orientation algorithm follows the idea of the algorithm of the previous section: first a column and a row are relabeled (Protocols ORIENTCOLUMN and ORIENTROW) and then all other rows and columns are relabeled proceeding as in Section 3.

We now describe the algorithm for orienting a column. Initially a node is in the state *Leader* and all the other nodes are *Idle.* The leader starts the computation by labeling an arbitrary link with NORTH and by sending a NORTH message through that port. Upon receiving a NORTH message, an idle node labels the corresponding port with SOUTH, sends a *candidate* message to its other three neighbours, and change state becoming *Column.* After receiving an acknowledgement from the three neighbours (which are now *Candidate* nodes), a *Column* node sends to SOUTH a request (*do-check* message) to start the check procedure which will designate the next node to be added to the column. A node receiving such a request, sends a message that will reach the nodes at distance two (analogously to the 2-expansion of the previous section) avoiding to send them in the already constructed NORTH and SOUTH directions. The candidate nodes who receive this expansion message reply to the last node added to the column saying that they are not the right candidates (see Figure 7). After receiving two such messages, the Column node knows the third neighbour is the node to be added to the column and continue the labeling procedure. Algorithm ORIENTCOLUMN is reported in Figure 8. At the end of the algorithm ORIENTCOLUMN the leader enters a special state (state COLUMN-DONE) and can start an analogous algorithm ORIENTROW to orient a row. The overall orienting algorithm ORIENT then consists of the execution of ORIENTCOLUMN and ORIENTROW for all the other rows and columns.

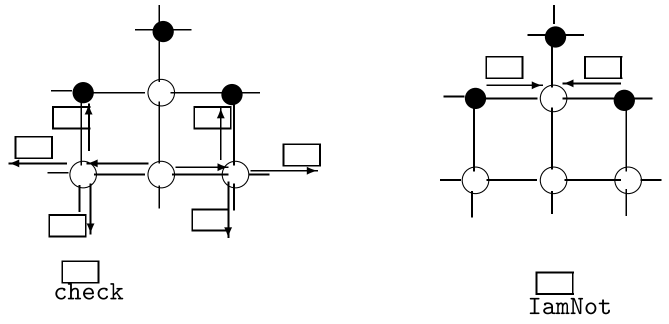

Figure 7: The candidates reached by the *check* message reply to the column that they are not the right candidates.

Correctness and Complexity.

Theorem 8 *Protocol* ORIENTCOLUMN *is correct.*

Proof. The proof really follows from the previous section since this is an exact simulation of algorithm CONSTRUCTCOLUMN performed with an agent. We can observe that after an Idle node becomes *Column* exactly three nodes will become *Candidate* and one of them will be selected to become *Column*. The one selected is the one that has not received the expansion message, by the same reasoning of Theorem 1, this is the north node. When the leader receives a NORTH message, the column has been fully labeled. □

Lemma 5 *Protocol* ORIENTCOLUMN *exchanges* $O(n)$ *messages, where* n *is the number of nodes in the column.*

Proof. Every node in the column of the leader receives a NORTH message and every node receives a *do-check* message. Every node of the column starts an expansion (check messages), which results in 8 messages (12 for the first expansion), for a total of 8(n-1)+12=8n+4 messages. Each node of the column (except for the leader) sends three *candidate* messages, for a total of $3(n-1)$ messages and to each of them it corresponds an *Ack* message. So, in total there are $16n+6$ messages. □

Analogously,

Lemma 6 *Protocol* ORIENTROW *exchanges* $O(m)$ *messages, where* m *is the number of nodes in the row.*

Since the orientation of a column is repeated m times, while the orientation of a row is repeated n times, we have:

Theorem 9 *Protocol* ORIENT *exchanges* $O(N)$ *messages of constant size in a torus of size* $N = m \times n$*. So the total bit complexity is* $O(N)$*.*

PROTOCOL ORIENTCOLUMN.

- Status Values: $\mathcal{S} = \{$LEADER,IDLE,COLUMN, COLUMN-DONE$\}$;
  Initially: $\{$LEADER IDLE$\}$;
  At the end: a column is relabeled, the leader is now in the state COLUMN-DONE, the nodes of the column are in the state COLUMN, all the others are IDLE.

```
LEADER
      Spontaneously
           send(NORTH) to $l$;

      Receiving(do-check) from NORTH
           send(check,2) to $N(x) - \{\textbf{sender}\}$; /* start the expansion */

      Receiving(NORTH) from $l$ /* the column is done */
           relabel $l$ with SOUTH;
           become(COLUMN-DONE) /* The leader is now ready to start the orientation of a row */
IDLE
    Receiving(NORTH) from $l$
             relabel $l$ with SOUTH;
             send(candidate) to $N(x) - \{\textbf{sender}\}$;
             become(WAITING) /* I need an ack before performing the check */

    Receiving(check,2) from $l$
             send(check,1) to $N(x) - \{\textbf{sender}\}$; /* I continue the expansion */

    Receiving(candidate) from $l$
             send(Ack) to $l$;
             set $link-to-column := l$;
             become(CANDIDATE) /* I could be the next node in the column */
WAITING
       Receiving(Ack) from three neighbours $l_1$, $l_2$, $l_3$
               send(do-check) to SOUTH;
               become(COLUMN)
COLUMN
      Receiving(IamNot) from two of the three neighbours $l_1$, $l_2$, $l_3$
              send(NORTH) to the third; /* the next in the column */

      Receiving(do-check) from NORTH
              send(check,2) to $N(x) - \{$ NORTH, SOUTH $\}$; /* start the expansion */

CANDIDATE
         Receiving(check,1))
                 send(IamNot) to $link-to-column$;
                 become(IDLE)
```

Figure 8: The Algorithm in the message-passing environment.

5. Conclusion

In this paper we have described an algorithm for orienting an anonymous unoriented torus by a mobile agent. If the agent has no tokens available the problem is unsolvable [7]; with our algorithm we have shown that one token is indeed necessary and sufficient, and that the orientation can be performed with an optimal number of moves.

The same technique can be used in a message-passing environment giving raise to the first $O(N)$-bit algorithm for constructing an orientation in a Torus with a Leader in the classical message-passing environment.

Since the construction of sense of direction is at least as difficult as the map construction problem, it is known that one token is not always sufficient for relabeling a graph with a sense of direction. An interesting problem would be to study for what other classes of graphs, besides tori, a single mobile agent can construct a sense of direction using one token only.

Acknowledgements

This research has been partially supported by NSERC. and by Dr. Flocchini's University Research Chair.

References

1. Y. Afek and E. Gafni. Distributed algorithms for unidirectional networks. *SIAM Journal on Computing*, 23(6):1152–1178, 1994.
2. S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.
3. H. Attiya, H. Shachnai, and T.Tamir. Local labeling and resource allocation using preprocessing. *SIAM Journal on Computing*, 28(4):1397–414, 1999.
4. L. Barrière, P. Flocchini, P. Fraigniaud, and N.Santoro. Rendezvous and election of mobile agents: impact of sense of direction. *Theory of Computing Systems*, 2005.
5. M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Information and Computation*, 176(1):1–21, 2002.
6. M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *35th Symp. on Foundations of Computer Science (FOCS)*, pages 75–85, 1994.
7. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs. In *18th Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978.
8. P. Boldi and S. Vigna. Complexity of deciding sense of direction. *SIAM Journal on Computing*, 29(3):779–789, 2000.
9. P. Boldi and S. Vigna. Lower bounds for sense of direction in regular graphs. *Distributed Computing*, 16(4):279–286, 2003.
10. J. Chalopin, E. Godard, and R. Ossamy Y. Métivier. Mobile agent algorithms versus distributed algorithms. In *10th Int. Conference on Principles of Distributed Systems (OPODIS)*, 2006.

11. A. K. Datta, S. Gurumurthy, F. Petit, and V. Villain. Self-stabilizing network orientation algorithms in arbitrary rooted networks. In *Int. Conference on Distributed Computing Systems*, pages 576–583, 2000.
12. X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.
13. A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *10th European Symposium on Algorithms (ESA'02)*, pages 374–386, 2002.
14. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.
15. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.
16. P. Flocchini, B. Mans, and N. Santoro. On the impact of sense of direction on message complexity. *Information Processing Letters*, 63(1):23–31, 1997.
17. P. Flocchini, B. Mans, and N. Santoro. Sense of direction: definition, properties and classes. *Networks*, 32(3):165–180, 1998.
18. P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, (291):29–53, 2003.
19. P. Flocchini, A. Roncato, and N. Santoro. Backward consistency and sense of direction in advanced distributed systems. *SIAM J. Computing*, 32(2):281–306, 2003.
20. P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symp. (LATIN)*, pages 141–151, 2004.
21. P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 246–257, 2004.
22. A. Israeli and M. Jalfon. Uniform self-stabilizing ring orientation. *Information and Computation*, 104(2):175–196, 1993.
23. B. Mans. Optimal distributed algorithms in unlabeled tori and chordal rings. *Journal on Parallel and Distributed Computing*, 46(1):80–90, 1997.
24. P. Panaite and A. Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33:281–295, 1999.
25. N. Santoro. *Design and Analysis of Distributed Algorithms*. John Wiley, 2006.
26. N. Santoro, J. Urrutia, and S. Zaks. Sense of direction and communication complexity in distributed networks. In *1st Int. Workshop on Distributed Algorithms (WDAG)*, pages 123–132, 1985.
27. S.Kekkonen-Moneta. Torus orientation. *Distrib. Comput.*, 15(1):39–48, 2002.
28. V. Syrotiuk and J. Pachl. A distributed ring orientation algorithm. In *2nd Int. Workshop on Distributed Algorithms (WDAG)*, volume 312, pages 332–336, 1987.
29. V. R. Syrotiuk, C. J. Colbourn, and J. K. Pachl. Wang tilings and distributed verification on anonymous torus networks. *Theory Comput. Syst.*, 30(2):145–163, 1997.
30. G. Tel. Network orientation. *Int. Journal of Foundations of Computer Science*, 5(1):1–41, 1994.