Optimal Map Construction of an Unknown Torus

Hanane Becha, Paola Flocchini
School of Information Technology and Engineering
University of Ottawa, Canada. {hbecha,flocchin}@site.uottawa.ca.

Abstract

In this paper we consider the map construction problem in the case of an anonymous, unoriented *torus of unknown size. An agent that can move from node to neighbouring node in the torus is initially placed in an arbitrary node and has to construct an edge-labeled map. In other words, it has to draw, in its local memory, an edge-labeled torus isomorphic to the one it is moving on. The agent has enough local memory to represent the torus and one or two tokens that can be dropped on and picked up from nodes. Efficiency is measured in terms of number of moves performed by the agent.*

When the agent has no token available, the problem is clearly unsolvable. In the paper we show that, when the agent has one token available there exists an optimal algorithm for constructing the map of the torus; the agent, in fact, performs $\Theta(N)$ moves (where N is the number of nodes of the torus). Before showing the optimal solution with the optimal number of tokens, we describe a simpler solution that works when two tokens are available, we then modify it to obtain the same bound when the agent has only one token available.

Keywords: Map construction, network exploration, mobile agents, tokens, distributed algorithms, topological information, torus, anonymity.

1 Introduction

In this paper we are interested in the *Map Construction* problem, widely studied in the literature (e.g., see [1, 2, 4, 5, 6, 8, 11]). 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. In the literature, this problem is sometimes also referred to as *Exploration*; in some instances, however, exploration refers to the simpler problem of traversing the whole network without constructing its topology.

Studies on map construction of edge-labeled graphs (or digraphs), 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 [1, 4, 5, 11]). Map construction of *anonymous* graphs is possible only if the agents are allowed to mark the nodes in some way; except when the graph has no cycles (i.e. the graph is a tree [6, 8]). For exploring arbitrary anonymous graphs, various methods for marking nodes have been used by different authors. Bender *et al.* [2] 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. In the following we will refer to this model as the *token model*. Dudek *et al.* [7] used a set of distinct markers to explore unlabeled undirected graphs. Yet another approach, used by Bender and Slonim [3] 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 Fraigniaud and Ilcinkas [9] for exploring directed graphs and by Fraigniaud et al. [8] for exploring trees. In [6, 9] the authors focus on minimizing the amount of memory used by the agents for exploration (they however do not require the agents to construct a map of the graph, the agents must only visit all the nodes). Finally, a problem that is somehow related to ours, but in a totally different model, is the problem of orienting a torus in the classical message passing system (e.g., see [10, 12, 13]). In this case, the torus is unlabeled and a message-passing algorithm has to be designed to construct a compass labeling of the edges. Among these works, the closer to ours is [10], where a linear orientation algorithm is designed for a non-anonymous torus.

1-4244-0054-6/06/$20.00

As mentioned before, in the token model, the map of an arbitrary anonymous graph can be constructed using a single token only if the size of the graph is known; otherwise more tokens are necessary [2]. An interesting open question is for what classes of graphs a single token is sufficient to perform the map construction when the size of the graph is not known.

In this paper we address this question by showing a class of graphs where one token is indeed sufficient and where the algorithm is optimal in terms of number of moves. We consider a highly symmetric environment: a network whose topology is a $n \times m$ *torus*. The network 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 will be measured in terms of number of moves performed by the agent.

We first describe a simpler algorithm for constructing the map of the torus when the agent has available two tokens. Essentially one token is used for detecting termination, while the other token is used to actually construct the map. The algorithm is optimal in terms of number of moves, which are $\Theta(n \times m)$. 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 map and to check for termination. This solution is both move-optimal and token-optimal since without a token the map cannot be constructed.

2 Optimal Algorithm Using Two tokens

In this section 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 section we will modify the technique to obtain the same result reducing the number of tokens.

2.1 The Algorithm

The idea of the algorithm is to first construct a column and a row intersecting at the home base, and then 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 $X_0, \ldots, X_{n-1}$ *)
back at X_0
CONSTRUCT ROW(X_0)
(* the row is composed by nodes $X_0, \ldots, Y_{m-1}$ *)
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 start drawing the column in its local memory. At this point, the agent has to find the next node in the column.

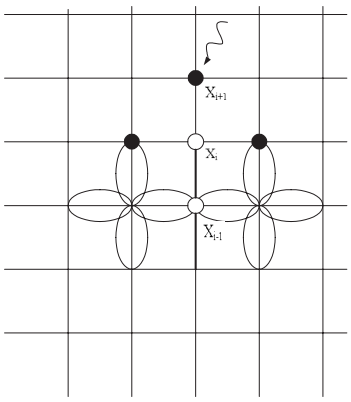

Figure 2. A 2-expansion from X_i. The black nodes are the candidate nodes.

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 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 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.

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 $X_0 \ldots X_i$ $(i \geq 1)$ 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_i)
      /* do not expand through X_{i-1} */
    If $i > 1$ then
      2-EXPAND(X_i)
      /* do not expand through X_{i-1} and X_{i+1} */
    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.

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 only 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 row corresponding to X_1 and then proceeds to the other rows. Before constructing row 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).

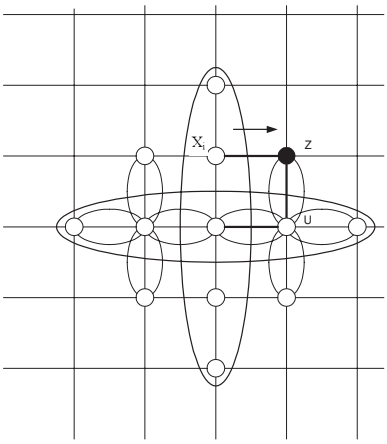

Figure 4. Determining the direction of the row starting from X_i.

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.

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.

2.2 Correctness and Complexity

Lemma 2.1 *Using Algorithm* CONSTRUCT COLUMN*, the agent walks in a straight direction and correctly constructs a column.*

Proof. We prove by induction that nodes $X_0, X_1, \ldots$ 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 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 2.2 *Using Algorithm* CONSTRUCT ROW*, the agent walks in a straight direction and construct a row.*

Theorem 2.1 *Algorithm* CONSTRUCT MAP *is correct.*

Proof. From Lemmas 2.1 and 2.2, 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 proceed in the direction of z. ∎

We now compute the number of moves involved.

Theorem 2.2 *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 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 3.1 *Algorithm* CONSTRUCT FIRST COLUMN WITH ONE TOKEN *is correct.*

Proof. From Lemmas 2.1 and 2.2, 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 3.1 *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 home base 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 3.2 *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. ∎

References

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[2] M. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *Proc. 30th ACM Symp. on Theory of Computing (STOC'98)*, pages 269–287, 1998.

[3] M. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. 35th Symp. on Foundations of Computer Science (FOCS'94)*, pages 75–85, 1994.

[4] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

[5] A. Dessmark and A. Pelc. Optimal graph exploration without good maps. In *Proc. 10th European Symposium on Algorithms (ESA'02)*, pages 374–386, 2002.

[6] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51:38–63, 2004.

[7] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[8] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symp. (LATIN'04)*, pages 141–151, 2004.

[9] P. Fraigniaud and D. Ilcinkas. Digraph exploration with little memory. In *21st Symp. on Theoretical Aspects of Computer Science (STACS'04)*, pages 246–257, 2004.

[10] B. Mans. Optimal distributed algorithms in unlabeled tori and chordal rings. *Journal on Parallel and Distributed Computing*, 46(1):80–90, 1997.

[11] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33:281–295, 1999.

[12] S.Kekkonen-Moneta. Torus orientation. *Distrib. Comput.*, 15(1):39–48, 2002.

[13] 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.