Mobile Agent Rendezvous in a Synchronous Torus*

Evangelos Kranakis[1,**], Danny Krizanc[2], and Euripides Markou[3,***]

[1] School of Computer Science, Carleton University, Ottawa, Ontario, Canada.
kranakis@scs.carleton.ca
[2] Department of Mathematics, Wesleyan University, Middletown, Connecticut 06459, USA.
dkrizanc@caucus.cs.wesleyan.edu
[3] School of Computational Engineering & Science, McMaster University, Hamilton, Ontario, Canada.
markoue@mcmaster.ca

Abstract. We consider the rendezvous problem for identical mobile agents (i.e., running the same deterministic algorithm) with tokens in a synchronous torus with a sense of direction and show that there is a striking *computational difference* between one and more tokens. More specifically, we show that 1) two agents with a constant number of unmovable tokens, or with one movable token, each cannot rendezvous if they have $o(\log n)$ memory, while they can perform rendezvous with detection as long as they have one unmovable token and $O(\log n)$ memory; in contrast, 2) when two agents have two movable tokens each then rendezvous (respectively, rendezvous with detection) is possible with constant memory in an arbitrary $n \times m$ (respectively, $n \times n$) torus; and finally, 3) two agents with three movable tokens each and constant memory can perform rendezvous with detection in a $n \times m$ torus. This is the first publication in the literature that studies tradeoffs between the number of tokens, memory and knowledge the agents need in order to meet in such a network.

Keywords: Mobile agent, rendezvous, rendezvous with detection, tokens, torus, synchronous.

---

* A preliminary version of this paper appeared in the proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN' 06), March 2006, Valdivia, Chile, LNCS 3887, pp. 653-664.

** Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

*** Work done partly while visiting Carleton University (2005-2006). Research supported in part by NSERC grant and by PYTHAGORAS project 70/3/7392 under the EPEAEK program of the Greek Ministry of Educational and Religious Affairs.

1 Introduction

We study the following problem: how should two mobile agents move along the nodes of a network so as to ensure that they meet or rendezvous?

The problem is well studied for several settings. When the nodes of the network are uniquely numbered, solving the rendezvous problem is easy (the two agents can move to a node with a specific label). However even in that case the agents need enough memory in order to remember and distinguish node labels. Symmetry in the rendezvous problem is usually broken by using randomized algorithms or by having the mobile agents use different deterministic algorithms. (See the surveys by Alpern [1] and [2], as well as the recent book by Alpern and Gal [4]). Yu and Yung [15] prove that the rendezvous problem cannot be solved on a general graph as long as the mobile agents use the same deterministic algorithm. While Baston and Gal [7] mark the starting points of the agents, they still rely on randomized algorithms or different deterministic algorithms to solve the rendezvous problem. Anderson and Fekete [5] and Alpern and Baston [3] study the problem in two-dimensional lattices having again the mobile agents use different strategies.

Research has focused on the power, memory and knowledge the agents need, to rendezvous in a network. In particular what is the 'weakest' possible condition which makes rendezvous possible? For example Yu and Yung [15] have considered attaching unique identifiers to the agents while Dessmark, Fraigniaud and Pelc [8] added unbounded memory; note that having different identities allows each agent to execute a different algorithm. Other researchers (Barriere et al [6] and Dobrev et al [9]) have given the agents the ability to leave notes in each node they visit. In another approach each agent has a stationary token placed at the initial position of the agent. This model is much less powerful than distinct identities or than the ability to write in every node. Assuming that the agents have enough memory, the tokens can be used to break symmetries. This is the approach introduced in [13] and studied in Kranakis et al [12] and Flocchini et al [10] for the ring topology. In particular the authors proved in [12] that two agents with one unmovable token each in a synchronous, n-node oriented ring need at least $\Omega(\log \log n)$ memory in order to do rendezvous with detection. They also proved that if the token is movable then rendezvous without detection is possible with constant memory.

We are interested here in the following scenario: there are two identical agents running the same deterministic algorithm in an anonymous oriented torus. In particular we are interested in answering the following questions. What memory do the agents need to solve rendezvous using unmovable tokens? What is the situation if they can move the tokens? What is the tradeoff between memory and the number of tokens?

1.1 Model and terminology

Our model consists of two identical mobile agents that are placed in an anonymous, synchronous and oriented torus. The torus consists of n rings and each of these rings consists of m nodes. Since the torus is oriented we can say that it consists of n vertical rings. A horizontal ring of the torus consists of n nodes while a vertical ring consists of m nodes. We call such a torus a $n \times m$ torus. The mobile agents share a common orientation of the torus, i.e., they agree on any direction (clockwise vertical or horizontal). Each mobile agent owns a number of identical tokens, i.e., the tokens are indistinguishable. A token or an agent at a given node is visible to all agents on the same node, but is not visible to any other agents. The agents follow the **same** deterministic algorithm and begin execution at the same time.

At any single time unit, the mobile agent occupies a node of the torus and may 1) stay there or move to an adjacent node, 2) detect the presence of one or more tokens at the node it is occupying and 3) release/take one or more tokens to/from the node it is occupying. We call a token *movable* if it can be moved by any mobile agent to any node of the network, otherwise we call the token *unmovable* in the sense that it can occupy only the node in which it has been released.

More formally we consider a mobile agent as a finite Moore automaton[4] $\mathcal{A} = (X, Y, \mathcal{S}, \delta, \lambda, S_0)$, where $X \subseteq \mathcal{D} \times \mathcal{C}_v \times \mathcal{C}_{MA}$, $Y \subseteq \mathcal{D} \times \{\texttt{drop}, \texttt{take}\}$, $\mathcal{S}$ is a set of σ states among which there is a specified state S_0 called the *initial* state, $\delta : \mathcal{S} \times X \rightarrow \mathcal{S}$, and $\lambda : \mathcal{S} \rightarrow Y$. $\mathcal{D}$ is the set of possible directions that an agent could follow in the torus. Since the torus is oriented, the direction port labels are globally consistent. We assume labels *up, down, left, right.* Therefore $\mathcal{D} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}, \texttt{stay}\}$ ($\texttt{stay}$ represents the situation where the agent does not move). $\mathcal{C}_v = \{\texttt{agent}, \texttt{token}, \texttt{empty}\}$ is the set of possible configurations of a node (if there is an agent and a token in a node then its configuration is $\texttt{agent}$). Finally, $\mathcal{C}_{MA} = \{\texttt{token}, \texttt{no} - \texttt{token}\}$ is the set of possible configurations of the agent according to whether it carries a token or not.

Initially the agent is at some node u_0 in the initial state $S_0 \in \mathcal{S}$. S_0 determines an action (drop token or nothing) and a direction by which the agent leaves u_0, $\lambda(S_0) \in Y$. When incoming to a node v, the behavior of the agent is as follows. It reads the direction i of the port through which it entered v, the configuration $c_v \in \mathcal{C}_v$ of node v (i.e., whether there is a token or an agent in v) and of course the configuration $c_{MA} \in \mathcal{C}_{MA}$ of the agent itself (i.e., whether the agent carries a token or not). The triple $(i, c_v, c_{MA}) \in X$ is an input symbol that causes the transition from state S to state $S' = \delta(S, (i, c_v, c_{MA}))$. S' determines an action (such as release or take a token or nothing) and a port direction $\lambda(S')$, by which the agent leaves v. The agent continues moving in this way, possibly infinitely.

We assume that the memory required by an agent is at least proportional to the number of bits required to encode its states which we take to be $\Theta(\log(|\mathcal{S}|))$ bits. Memory permitting, an agent can count the number of nodes between tokens, or the total number of nodes of the torus, etc. In addition, an agent might already know the number of nodes of the torus, or some other network parameter such as a relation between n and m. Since the

[4] The first known algorithm designed for graph exploration by a mobile agent, modeled as a finite automaton, was introduced by Shannon [14] in 1951.

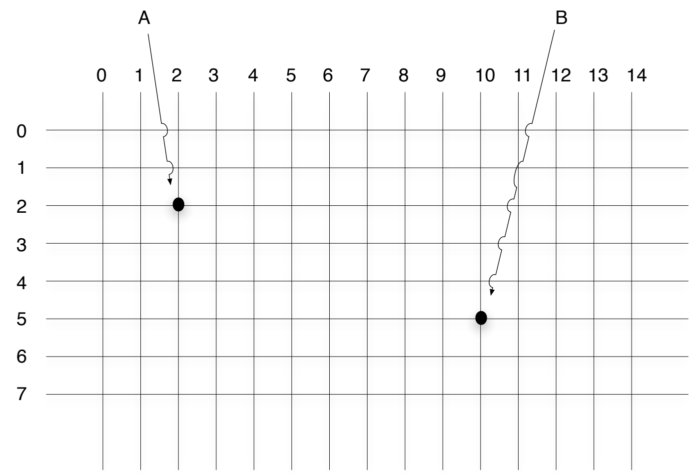

Fig. 1. Two agents in a 15×8 (2−dimensional) torus. Agent A has coordinates $(2, 2)$. Agent B has coordinates $(10, 5)$. Their distance is $d(A, B) = (\min\{|A_1 - B_1|, (n - |A_1 - B_1|)\}, \min\{|A_2 - B_2|, (m - |A_2 - B_2|)\}) = (\min\{|2 - 10|, (15 - |2 - 10|)\}, \min\{|2 - 5|, (8 - |2 - 5|)\}) = (\min\{8, 7\}, \min\{3, 5\}) = (7, 3)$.

agents are identical they face the same limitations on their knowledge of the network. In what follows, we assume that, unless explicitly stated, the agents have no knowledge about the number of nodes of the torus or any other parameter of the network, apart from its dimension.

The *distance* between two nodes on a d-dimensional torus $n_1 \times n_2 \times \cdots \times n_d$, is a d-vector whose ith element is $\min\{|x_i - y_i|, (n_i - |x_i - y_i|)\}$ where x_i, y_i are the ith co-ordinates of the nodes. An example of two agents in a torus is shown in Figure 1.

Let $U = \{(n_1/2, 0, ..., 0), (0, n_2/2, ..., 0), ..., (0, 0, ..., n_d/2)\}$, where each n_i is even, be a set consisting of d vectors in d-dimension.

Theorem 1. *Consider two agents placed in a d-dimensional oriented torus ($n_1 \times n_2 \times \cdots \times n_d$) so that their distance is the sum of vectors contained in any nonempty subset S of U. Assume that for any non-zero element of the distance, the number of nodes of that dimension of the torus is even. Then, no matter how many tokens or how much memory the agents have, it is impossible for the agents to rendezvous.*

Proof. Let $D = (x_1, x_2, ..., x_d)$, where $x_i = 0$ or $x_i = n_i/2$, $1 \leq i \leq d$ be the initial distance of the agents. As long as they do not release a token, the agents are in the same configuration. Since the configuration of each node they occupy is also the same (empty), the agents are in the same state and maintain their distance. They release tokens simultaneously and since they maintain their distance they meet tokens at the same time. In other words they are always in the same state and configuration while the configuration of the node they occupy is always the same. Therefore they maintain their distance forever. □

Corollary 1. *Two agents placed in a $n \times m$ torus are incapable of meeting each other (no matter how many movable or unmovable tokens they have) if their initial distance is either $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$.*

Theorem 1 is a generalization of Theorem 1 in [12] which states that it is impossible for two agents equipped with one unmovable token each, to rendezvous in a ring with n nodes if their initial distance is $n/2$, where n is even.

Definition 1. *We call rendezvous with detection (RVD) the problem in which the agents meet each other if their distance is not the sum of vectors contained in any nonempty subset S of U, otherwise they stop moving and declare that is impossible to meet each other.*

We say that an algorithm $\mathcal{A}$ solves RVD (or $\mathcal{A}$ is an RVD algorithm) if: a) $\mathcal{A}$ leads the agents to rendezvous, when their initial distance is not the sum of vectors contained in any nonempty subset S of U and b) $\mathcal{A}$ halts after a finite number of steps and the agents declare that rendezvous is impossible, when the distance is indeed the sum of vectors contained in a subset S of U.

Definition 2. *We call rendezvous without detection (RV) the problem in which the agents meet each other if their distance is not the sum of vectors contained in any nonempty subset S of U.*

Therefore we say that an algorithm $\mathcal{A}$ solves RV (or $\mathcal{A}$ is an RV algorithm) if the agents rendezvous when their initial distance is not the sum of vectors contained in any nonempty subset S of U. If, however, the distance is indeed the sum of vectors contained in a subset S of U then $\mathcal{A}$ may run forever.

We assume that at any single time unit an agent can traverse one edge of the network or wait at a node (we assume that taking or leaving a token can be done instantly). For a given torus G and starting positions s, s' of the agents we define as cost $\mathcal{CT}_{RVD}(A, G, s, s')$ of an RVD algorithm A, the maximum time (number of steps plus waiting time of an agent) needed either to rendezvous or to decide that rendezvous is impossible. The cost $\mathcal{CT}_{RV}(A', G, s, s')$ of an RV algorithm A', is the time needed to rendezvous (when it is possible of course). Finally, the time complexity of an RVD or RV algorithm is its maximum cost overall possible starting positions of the agents.

1.2 Our results

In the study of the rendezvous problem this paper shows that there is a striking *computational difference* between one and more tokens. More specifically, we show that:

1. Two agents with a constant number of unmovable tokens each cannot rendezvous if they have $o(\log n)$ memory.

2. Two agents with one movable token each cannot rendezvous if they have $o(\log n)$ memory.
3. Two agents with one unmovable token each can perform rendezvous with detection as long as they have $O(\log n)$ memory.
4. When two agents have two movable tokens each then rendezvous (respectively, rendezvous with detection) is possible with constant memory in an arbitrary $n \times m$ (respectively, $n \times n$) torus.
5. Two agents with three movable tokens each and constant memory can perform rendezvous with detection in an arbitrary $n \times m$ torus.

This is the first publication in the literature that studies tradeoffs between the number of tokens, memory and knowledge the agents need in order to meet in such a network.

1.3 Outline of the paper

In Section 2 we first give some preliminary results concerning possible ways that an agent can move in a torus using either no tokens or a constant number of unmovable tokens. Then we prove that rendezvous without detection in a torus cannot be solved by two agents with one movable token each, or with a constant number of unmovable tokens unless their memory is $\Omega(\log n)$ bits.

In Section 3 we give an algorithm for rendezvous with detection in a $n \times n$ torus using one unmovable token and $O(\log n)$ memory. We also give an algorithm for rendezvous with detection in a $n \times n$ torus using two movable tokens and constant memory. Next we give an algorithm for rendezvous without detection in an arbitrary $n \times m$ torus using two movable tokens and constant memory, stating the relation that m and n must have in order to do rendezvous with detection following that algorithm. Finally we give an algorithm for rendezvous with detection in a $n \times m$ torus using three movable tokens and constant memory.

In Section 4 we discuss the results and state some open problems.

2 Memory lower bounds of rendezvous

2.1 Preliminary results

Lemma 1. *Consider one mobile agent with σ states and no tokens. We can always (for any configuration of the automaton, i.e. states and transition function) select an $n \times n$ oriented torus, where $n = \Omega(\sigma)$ so that no matter what is the starting position of the agent, it cannot visit all nodes of the torus. In fact, the agent will visit at most $(\sigma + 1)n$ nodes.*

Proof. If we select an oriented $n \times n$ torus, where $n > \sigma$ then the agent has to repeat a state at some point (before visiting all nodes). Let S be the first state repeated. Let v be the node where the agent is placed when S is encountered for the first time and v' be the node where the agent is located when S is repeated for the first time. We call p_x, p_y the horizontal and vertical distance respectively of v' from v. Since S is the first state repeated, the total number of nodes visited by the agent until it encounters S again is at most $\sigma + 1$.

Once the agent is again at state S it has to repeat the same trajectory (p_x, p_y). Label the nodes of the torus $0, \ldots, n-1$ horizontally and vertically. If v_x, v_y are the coordinates of node v, then after n repetitions of state S, the position of the agent is:

$$(v_x + np_x) \bmod n = v_x$$

$$(v_y + np_y) \bmod n = v_y$$

This means that the agent is again at node v and state S. The agent has to continue moving visiting exactly the same nodes. Hence the agent will visit at most $(\sigma+1)n$ nodes. □

The above lemma can be generalized for the case of a $n \times m$ torus. It suffices to select $n = ap_x$ and $m = ap_y$, where a is such that $ap_x > \sigma$ and $ap_y > \sigma$. After a repetitions of the first repeated state the agent will be again located at the same node. Therefore it will visit at most $(\sigma + 1)a = O((\sigma + 1)(m + n))$ nodes.

Lemma 2. *Consider one mobile agent with σ states and one unmovable token. We can always (for any configuration of the automaton, i.e. states and transition function) select an oriented $n \times n$ torus, where $n = \Omega(\sigma^2)$ so that no matter what is the starting position of the agent, it cannot visit all nodes of the torus. In fact, the agent will visit at most $(\sigma + 1)(1 + \sigma n) = O(\sigma^2 n)$ nodes.*

Proof. As long as the agent does not release the token lemma 1 holds.

Suppose that the agent releases the token at some point. This point has to be before repeating a state (otherwise it will never take this decision since after repeating a state, everything is being repeated). Hence up to that point it has visited up to $\sigma + 1$ nodes in a $n \times n$ torus, where $n > \sigma$. After releasing the token the agent moves just like in Lemma 1 (without having any tokens). Take the first state S which is encountered and repeated after dropping the token.

Suppose that after at most n repetitions of S, the agent does not meet its token. Then it visits at most $(\sigma + 1)n$ nodes (see the proof of Lemma 1). Hence in that case the agent visits a total number of at most $(\sigma + 1)(n + 1)$ nodes.

Suppose now that at some point t, the agent sees again its token before finds himself located at a node twice having the same state. In view of Lemma 1, up to that point t, it has visited at most $(\sigma+1)n$ nodes. When it meets its token it could change its orbit visiting another $(\sigma+1)n$ nodes. After at most σ times it sees its token being in a state twice. In other words, it could enter at most σ different states (thus changing its orbit) when it meets its token. Therefore it will visit a total of at most $\sigma+1+(\sigma+1)\sigma n$ nodes and after that it visits exactly the same nodes. Hence if we select the size of the torus to be $n = \Omega(\sigma^2)$, then the agent will visit at most $(\sigma+1)(1+\sigma n) = O(\sigma^2 n)$ nodes. □

Again Lemma 2 can be easily generalized for the case of a $n \times m$ torus. In that case the agent will visit at most $O(\sigma^2(m+n))$ nodes.

For the case of more than one unmovable tokens, we can apply again the arguments used in Lemmas 1, 2. Observe that in this case, after the first token has been released, the agent cannot release a new token at a distance more than $\sigma+1$ nodes away from another token. Therefore we get:

Theorem 2. *Consider one mobile agent with σ states and a constant number k of identical unmovable tokens. We can always (for any configuration of the automaton, i.e. states and transition function) select a $n \times n$ oriented torus, where $n = \Omega(\sigma^2)$ so that no matter what is the starting position of the agent, it cannot visit all nodes of the torus. In fact, the agent will visit at most $O(\sigma^2 n)$ nodes.*

We will also need the following technical lemma:

Lemma 3. *Let A be an agent with σ states and one unmovable token in a $n \times n$ torus, where $n = \Omega(\sigma^2)$ and let v be a node in that torus. There are at most $(\sigma+1)(1+\sigma n) = O(\sigma^2 n)$ different starting nodes that we could have placed A so that node v is always being visited by A.*

Proof. By Lemma 2, agent A can visit at most $(\sigma+1)(1+\sigma n)$ different nodes. This means that starting from a node s, there are at most $(\sigma+1)(1+\sigma n)$ different nodes $u_i = (x_i, y_i)$ (where x_i, y_i are oriented horizontal and vertical respectively distances of node u_i from s) that could be visited. We prove that a fixed node v can be reached only starting from at most $(\sigma+1)(1+\sigma n)$ different nodes:

Take a node s as a starting node. Suppose that the given node v has relative distance (x, y) from s (e.g. x nodes horizontally clockwise and y nodes vertically clockwise) and can be reached by A. Change the starting node to s'. Now from the new starting node s', agent A can reach the node at distance (x, y) (which of course is different than v). Suppose that A can still reach node v which now is at a different distance (x', y') from s'.

By repeating this procedure you can have at most $(\sigma+1)(1+\sigma n) = O(\sigma^2 n)$ different starting nodes for which an agent starting there visits node v, otherwise it would mean that there exists a starting node such that once the agent started there, it could pass from more than $(\sigma+1)(1+\sigma n)$ nodes which in view of Lemma 2 is impossible. □

2.2 An $\Omega(\log n)$ memory lower bound for rendezvous using one movable token

Lemma 4. *Consider two mobile agents with σ states. They each have a token (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find an oriented $n \times n$ torus, where $n = \Omega(\sigma^2)$ and place the agents so that if they can not move tokens then they cannot rendezvous.*

Proof. If we place the agents at any distance, as long as they do not release their token they maintain their distance since they move in exactly the same way.

Suppose that at some point they release their token and they move.

a) Consider the case that they see a token before they repeat a state. The total number of nodes visited before a state is repeated is at most $\sigma + 1$. Therefore we can initially place the agents at such a distance (greater than $\sigma + 1$ in any dimension) so that if they see a token before repeating a state then this token is their own token. Hence they see their tokens at the same time and since they continue moving in exactly the same way they maintain their distance.

b) They repeat a state without having seen a token. Take the first state S that they repeat. Suppose that when first are at state S, at that moment they are at nodes v_1, v_2.

If n is the size of the torus, consider what happens after at most n repetitions of S:

i) either both of the agents do not see a token, or

ii) at least one of the agents sees a token

In case i) Lemma 1 holds and hence they will eventually be located again at nodes v_1, v_2 having state S, therefore maintaining their distance.

Suppose now case ii), i.e. at least one of the agents sees a token before the n repetitions. We prove that we can initially place the agents so that they never meet the other's token.

We place the first agent A in a node. If A can meet only its token, then by Lemma 2, the agent would visit at most $(\sigma + 1)(1 + \sigma n)$ nodes before it repeats everything. We prove that we can initially choose a node to place the other agent B so that anyone's token is out of reach of the other:

We need to place the second agent B so that

- it releases its token t_B at a node different from at most $(\sigma + 1)(1 + \sigma n)$ nodes visited by the first agent A and
- to avoid to visit the node where the first agent A released its token t_A

We can place the second agent B at a starting node out of at least $n^2 - (\sigma + 1)(1 + \sigma n)$ (taking $n = \Omega(\sigma^2)$) so that B's token is out of reach of A. Moreover, by Lemma 3 only

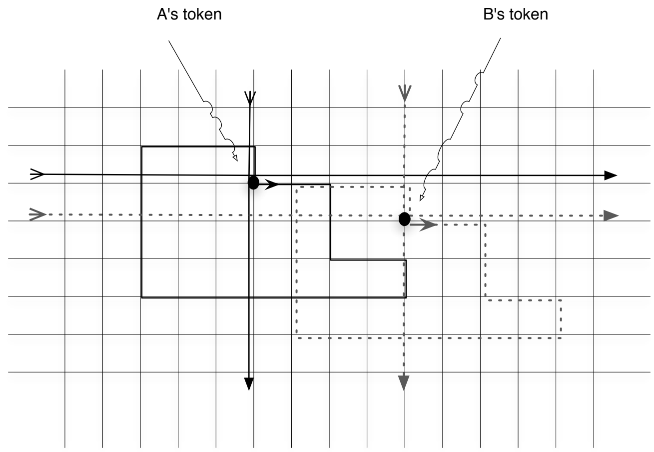

Fig. 2. Two agents with 1 unmovable token each cannot see each other's token.

$(\sigma + 1)(1 + \sigma n)$ starting nodes could lead agent B to meet A's token. Thus there are at least $n^2 - 2(\sigma + 1)(1 + \sigma n)$ starting nodes that satisfy the above property.

Therefore if we choose the size of the torus $n = \Omega(\sigma^2)$, then we can place the agents so that if they meet a token it is always their token. Hence they do this at the same time and their distance do not change. The situation has been illustrated in Figure 2 (orbits of agent A have been drawn with solid lines while orbits of agent B have been drawn with dashed lines). $\square$

Notice that in the previous scenario, where the two agents cannot move the tokens, there are still unvisited nodes (from the same agent) in the torus. In fact we proved Lemma 4 by describing a way to 'hide' token t_A in a node not visited by agent B and token t_B in a node not visited by agent A.

Definition 3. *If there are two starting nodes s, s' for the agents A and B so that agent A drops its token t_A in a node not visited by agent B and agent B drops its token t_B in a node not visited by agent A then we say that s, s' satisfy property π.*

If the agents could move the tokens, then it is easy to think of an algorithm where all nodes of any torus are visited by the same agent. For example consider the following algorithm:

- 1: go right until you meet the second token;

- 2: move the token down;

- 3: repeat from step 1;

Nevertheless the goal is again to place the agents in a way that they could meet only their own token. To achieve this we place the agents so that in a phase which starts when the agents move their tokens, up to the moment when they move their tokens again they do not meet each other's token.

Lemma 5. *Consider two mobile agents with σ states. They each have a token (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find an oriented $n \times n$ torus, where $n > 8\sigma^3 = \Omega(\sigma^3)$ and place the agents so that even if they can move tokens they cannot rendezvous.*

Proof. As long as they do not move tokens Lemma 4 holds. We can initially place the agents so that if they see a token, it is their own token (up to the moment that they decide to move it). Suppose that at some point they decide to move their token.

Then each agent moves with a probably different orbit than before. Every time they see a token they can change their orbit. Nevertheless they can change their orbits for at most σ times.

We proved in Lemma 4 that we could place the agents so that in the first part (i.e. until they decide to move the tokens for the first time), they do not meet and they do not see each other's token. Suppose that agent A has been placed at a node s.

In view of Lemma 4 there are at most $2(\sigma+1)(1+\sigma n)$ starting nodes so that each one paired with s do not satisfy property π up to the moment that the agents move the tokens for the first time. Every time they move the tokens we can have at most $2(\sigma+1)(1+\sigma n)$ new pairs (with different distances than the previous pairs). There are at least $n^2/4$ pairs with different distances in a $n \times n$ torus. Since there is a limited number ($\leq \sigma$) of different orbits that the agents could choose after moving the tokens, there will be left at least $\Omega(n^2/4 - 2\sigma(\sigma+1)(1+\sigma n))$ legal nodes (each one of them paired with s, satisfy property π).

This means that if we select the size of the torus $n > 8\sigma^3$ then there are some nodes that are legal for all orbits (i.e. if we place agents there then anyone's token is out of reach of the other).

To find a pair of such nodes, one may execute the following procedure:

- place agent A in a node

- place agent B in one of the $n^2 - 2(\sigma+1)(1+\sigma n)$ nodes as in Lemma 4

- after they have moved the tokens find the new different 'bad' nodes (which do not satisfy property π) (at most $2(\sigma+1)(1+\sigma n)$)

- If the so far selected pair s, s' does not belong to any of the previous sets of 'bad' nodes, repeat the previous step

- otherwise choose a different pair of starting nodes and repeat the whole procedure □

This implies the following theorem:

Theorem 3. *Two agents in a $n \times n$ torus with one movable token need at least $\Omega(\log n)$ memory to solve the RV problem.*

Proof. Suppose that the agents have a memory of r bits. Hence they can have at most 2^r states. By Lemma 5 as long as $n > 8\sigma^3$ the agents cannot perform rendezvous. Hence, the agents need at least $r = \Omega(\log n)$ memory in order to perform rendezvous. □

2.3 An $\Omega(\log n)$ memory lower bound for rendezvous using $O(1)$ unmovable tokens

Lemma 6. *Consider two mobile agents with σ states. They each have two tokens (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find a $n \times n$ oriented torus, where $n = \Omega(\sigma^2)$ and place the agents so that if they cannot move tokens they cannot rendezvous.*

Proof. In view of Lemmas 2, 4 we can select the torus and the starting positions so that an agent will visit at most $(\sigma + 1)(1 + \sigma n)$ nodes until it decides to release its second token and up to that point does not meet the other's token. Its second token will have to be released at a 'short' distance from the first one since an agent cannot count more than σ. Using similar arguments as in the proof of Lemma 4 one can show that there are at least $n^2 - 5(\sigma + 1)(1 + \sigma n)$ pairs of starting nodes that satisfy property π. □

This implies the following theorem:

Theorem 4. *Two agents in a $n \times n$ torus with two identical unmovable tokens each, need at least $\Omega(\log n)$ memory to solve the RV problem.*

Applying similar arguments we can extend the result to a constant number of unmovable tokens:

Lemma 7. *Consider two mobile agents with σ states. They each have a constant number of k identical tokens. Then we can always (for any configuration of the automatons, i.e. states and transition function) find an oriented $n \times n$ torus, where $n = \Omega(\sigma^2)$ and place the agents so that if they cannot move tokens they cannot rendezvous.*

Proof. Using similar arguments as in the proof of Lemma 4 one can show that there are at least $n^2 - \frac{k(k+2)}{2}(\sigma + 1)(1 + \sigma n)$ pairs of starting nodes that satisfy property π. □

Theorem 5. *Two agents in a $n \times n$ torus with a constant number of unmovable tokens need at least $\Omega(\log n)$ memory to solve RV problem.*

3 Rendezvous

3.1 Rendezvous with Detection (RVD) in a $n \times n$ torus using one token and $O(\log n)$ memory

We describe an algorithm which solves the RVD problem of two agents in a $n \times n$ torus, equipped with one unmovable token and $O(\log n)$ memory each. Below is a high level description of the algorithm (Algorithm 1).

First the agent (both agents run the same algorithm) moves in the initial horizontal ring; it releases its token and it counts steps until it meets a token twice. If its counters differ, then it can meet the other agent. Otherwise it does the same in the initial vertical ring. If it does not meet the other and does not decide that rendezvous is impossible (which means that the agents must have started in different rings), then it searches one by one the horizontal rings of the torus counting its steps. If at least one of its counters (representing horizontal or vertical distances) is different than $n/2$ then it can meet the other agent. Otherwise it stops and declares rendezvous impossible.

Algorithm 1 Algorithm for RVD with 1 token and $O(\log n)$ memory

1: SameRing
2: DifRing

As Algorithm 1 suggests, the agents first execute Procedure `SameRing`. If they do not meet each other and they do not decide that rendezvous is impossible, then they must have started in different rings ($c_1 = c_3$). In that case they execute Procedure `DifRing`. Their exploration finishes after at most $O(n^2)$ steps, while they need $O(\log n)$ memory for counting.

Lemma 8. *If the agents are located on the same ring of a $n \times n$ torus then Procedure* `SameRing` *is a RVD algorithm. Furthermore the agents are located in the same ring if and only if counters c_1 and c_3 differ.*

Proof. After $c_1 + c_2$ steps the agents see their token. So they are again located at their starting positions.

If $c_1 \neq c_2$, this means that the agents started in the same horizontal ring. They execute Procedure `Rendezvous` on the horizontal ring and since they have counted different they can break symmetries and perform rendezvous.

If $c_2 = c_1$ then the agents started either on the same horizontal ring at distance $n/2$ or in different horizontal rings. In that case ($c_2 = c_1$) they try the vertical ring. They count c_3 steps until they meet a token on the vertical ring. It is easy to see that $c_1 = c_3$ if and only if they have started on different rings. Hence since (by hypothesis) they have started in the same ring it holds $c_1 \neq c_3$.

Procedure SameRing

leave your token down
go right and count steps until you see a token
$c_1 \leftarrow$this number of steps
go right and count steps until you see a token
$c_2 \leftarrow$this number of steps
if $c_2 \neq c_1$ **then**
    Rendezvous(horizontal, c_1, c_2)
else
    go down and count steps until you see a token
    $c_3 \leftarrow$this number of steps
    if $c_1 = c_3/2$ or $c_3 = c_1/2$ **then**
        stop and declare rendezvous impossible
    end if
    if $c_1 \neq c_3$ and $c_1 \neq c_3/2$ and $c_3 \neq c_1/2$ **then**
        go down and count steps until you see a token
        $c_4 \leftarrow$this number of steps.
    end if
    if $c_4 \neq c_3$ **then**
        Rendezvous(vertical, c_3, c_4)
    end if
end if

Procedure Rendezvous(ring, c_1, c_2)

if ring = horizontal **then**
    if $c_2 > c_1$ **then**
        go right
    else
        go left
    end if
end if
if ring = vertical **then**
    if $c_2 > c_1$ **then**
        go down
    else
        go up
    end if
end if

- If $c_1 = c_3/2$ or $c_3 = c_1/2$ then they have started at distance $n/2$ in the same horizontal or vertical ring respectively. Hence (in view of Theorem 1) rendezvous is impossible.

- If $c_1 \neq c_3/2$ and $c_3 \neq c_1/2$, since $c_1 = c_2 \neq c_3$ the agents must have started at distance different than $n/2$ in the same vertical ring. After c_4 steps in the vertical ring they see their token. Since $c_4 \neq c_3$ the agents can break symmetries and perform rendezvous by executing Procedure `Rendezvous` on the vertical ring. □

Procedure DifRing

```
repeat
    go down to the next horizontal ring
    repeat
        go right
        c5 ←the number of steps right
    until (c5 = c1) OR (you meet a token)
until you meet a token
c6 ←the number of rings down
if c5 = c6 = c1/2 then
    stop and declare rendezvous impossible
else
    if c6 ≠ c1/2 then
        Rendezvous2(c6)
    else
        Rendezvous2(c5)
    end if
end if
```

Procedure Rendezvous2(ct)

```
if ct < c1/2 then
    reverse horizontal direction and go c5 horizontally and then vertically until you meet your token and wait
end if
if ct > c1/2 then
    wait
end if
```

Lemma 9. *If the agents are located on different rings of a $n \times n$ torus then Procedure* `DifRing` *is a RVD algorithm. Furthermore the agents are located in different rings if and only if counters c_1 and c_3 (from Procedure* `SameRing`*) are equal.*

Proof. in view of the previous lemma, the agents are in different rings initially if and only if it holds $c_1 = c_3 = n$. They explore the other horizontal rings. If they do not find a token after c_1 steps then of course they can be sure that there is no token at that ring. After exploring c_6 horizontal rings, they see a token located c_5 steps to the right of their starting position. If $c_5 = c_6 = c_1/2$ then (in view of Theorem 1) rendezvous is impossible. Otherwise:

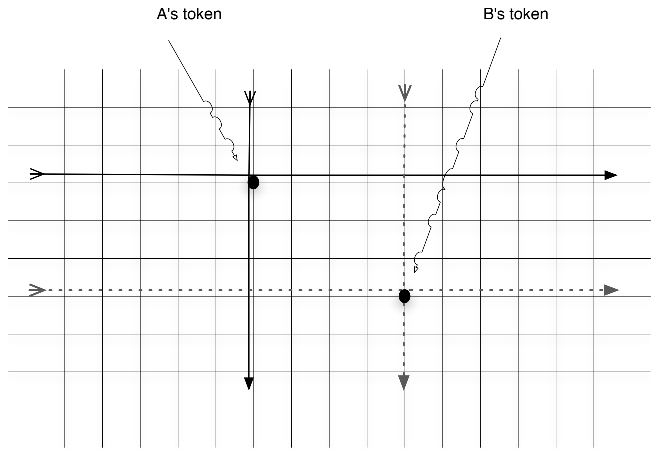

Fig. 3. Two agents with $O(\log n)$ memory and one unmovable token each.

- If $c_6 \neq c_1/2$ then they can break symmetries as follows: The agent that counted $c_6 < c_1/2$ reverses its horizontal direction and goes c_5 steps. Then reverses the vertical direction and moves until it meets a token. It waits there. The other agent just waits. They will rendezvous there.

If $c_6 = c_1/2$ then the agent who counts $c_5 < c_1/2$ reverses directions as above and moves. The other agent waits. □

An example has been illustrated in Figure 3. First the agents execute Procedure `SameRing`. They fail to meet each other because they are in different rings. Then they execute Procedure `DifRing` and they rendezvous.

Algorithm 1 together with Lemmas 8, 9 imply the following theorem.

Theorem 6. *The Rendezvous with Detection problem on a $n \times n$ torus can be solved by two agents using one unmovable token and $O(\log n)$ memory each, in time $O(n^2)$.*

The above result can be extended for the case of an arbitrary $n \times m$ torus. The main difference in that case is that the relation of counters c_1 and c_3 can no longer serve us as before (to distinguish between the two cases: the agents started in the same ring or not). But still, the agents are able to decide if they have started on the same ring or not as follows: they explore one by one the horizontal rings travelling c_1 steps horizontally. They will meet a token while going down (passing from one horizontal ring to the next) if and only if they have started on the same ring. Otherwise, they will meet a token while going right (before finishing the exploration of a horizontal ring). They can again do Rendezvous with detection in $O(nm)$ steps as long as they have $O(\log n + \log m)$ memory each. Hence the following theorem holds:

Theorem 7. *The Rendezvous with Detection problem on a $n \times m$ torus can be solved by two agents using one unmovable token and $O(\log n + \log m)$ memory each, in time $O(nm)$.*

3.2 Rendezvous with Detection in a $n \times n$ torus using two movable tokens and constant memory

We define Procedures `HorScan` and `VerScan` which will be used in our algorithms.

Procedure HorScan

```
repeat
    go down, right, up
until you meet a token
```

Procedure VerScan

```
repeat
    go right, down, left
until you meet a token
```

In these procedures the agent stops immediately after it meets a token. So for example, if it executes Procedure `HorScan` and then, after it goes right, it meets a token then it stops immediately; it does not go up.

We also use Procedure `FindTokenHor`:

Procedure FindTokenHor

```
1: repeat
2:     HorScan
3:     if you meet token up then
4:         HorScan
5:         go one step down and drop (or move) the second token
6:     end if
7: until you meet a token down or right
8: if you meet a token down then
9:     SameRing:=1
10: else
11:     SameRing:=0
12: end if
```

An agent following Procedure `FindTokenHor`, scans one by one the horizontal rings of the torus until it meets a token while moving down or right. Below we explain Procedure `FindTokenHor` and prove some of its properties.

Let the agents release their first token and execute Procedure `FindTokenHor`. During execution of `HorScan` (step 2 of Procedure `FindTokenHor`), the agent has to meet a token for the first time either after it moved down in the first step, or up or right (he can not meet a token while going down at a later step of `HorScan` since it would have met the token while going right earlier).

If it meets a token after it moved up, then this can be any token: its or the other's first token (or its or the other's second token when it scans a later horizontal ring). However, if it executes Horscan again (step 4 of Procedure `FindTokenHor`), then no matter what was the case, it is easy to see that the first token it meets now is its token (first or second) and it meets it after it moved up[5]. Furthermore in this case it is sure that the down ring had no tokens.

If it meets a token right then it is clear that it is the other's first token and that the two agents have started in different rings.

If it meets a token while it goes down then either it is its first token or the other's first token. In both cases this means that they have started in the same ring: if it is its first token it means that it has searched the whole torus and did not meet any other token while it was moving right.

Therefore the agent exits Procedure `FindTokenHor` knowing that it has started either in the same ring with the other agent (if it met a token after it moved down) or in different rings (if it met a token after it moved right).

We also use Procedure `FindTokenVer` which scans one by one the vertical rings of the torus using Procedure `VerScan`. Procedure `FindTokenVer` has exactly the same properties with Procedure `FindTokenHor` if we replace direction down with right, right with down and up with left. If we had a guarantee that the agents started in different rings then an agent executing Procedure `FindTokenVer` it will exit the procedure, meeting a token while it moves right. The movements of the agents following these two procedures are shown in Figure 4. Both procedures `FindTokenHor` and `FindTokenVer` need $O(n^2)$ time units.

Procedure FindTokenVer

```
repeat
    VerScan
    if you meet token left then
        VerScan
        go one step right and drop (or move) the second token
    end if
until you meet a token right
```

We also use Procedure `RVDRing` which appeared in [11] for rendezvous with detection in a ring using two tokens and constant memory. Suppose that the agents start in the same ring and they release their first token. If they execute the Procedure `RVDRing` then they perform rendezvous with detection (see Figure 5).

[5] Supposing that there are at most two tokens in the same horizontal ring.

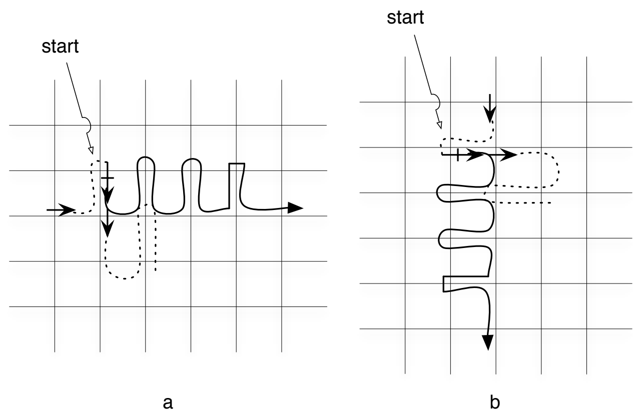

Fig. 4. a) An agent executing Procedure `FindTokenHor` and b) an agent executing Procedure `FindTokenVer`.

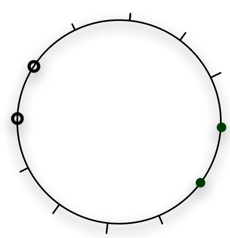

Fig. 5. In a ring: two agents with constant memory and two movable tokens each.

Lemma 10. *Consider two mobile agents with constant memory on an oriented ring consisting of n nodes. They each have two tokens (identical to each other). If they release their first tokens and execute Procedure* `RVDRing` *then they perform rendezvous with detection.*

Proof. Since after step 1 there are 4 tokens in the ring, each agent meets (at step 3) always the token it has dropped second. Moreover since the agents started at the same time and do complete cycles, they always meet their forth token at the same time. Consider the first moment at which after moving forward a token, it touches another token. The other agent moved forward the token at exactly the same time.

i) suppose that both agents see that their tokens touch other tokens. Since they have travelled exactly the same distance, this means that the distance between an agent's tokens is $n/2$. They discover this at the same time according to the procedure.

ii) suppose that only one of the agents (say A) sees its token touching another token. Since the agents travelled the same distance, it means that their initial distances were different. Now A moves to the next token just to see that it is not touching other tokens. It will be there in less than n steps, while in exactly n steps the other agent will also be there. □

Procedure RVDRing

move one step right and drop the second token
repeat
  move right until you meet the forth token t
  move token t one step to the right
until there is another token next to t
go right until you meet a token
if there is another token next to it **then**
  stop and declare rendezvous impossible
else
  wait there
end if

Procedure `RVDRing` takes $O(n^2)$ time. We also use in our algorithms a similar procedure with `RVDRing` for rendezvous with detection on a vertical ring of a torus. We only need to replace direction right with down.

Combining those procedures we now give the main Procedure `SearchTorus` that will be used in the Algorithm `RVD2n` which is a RVD algorithm for two agents with constant memory in a $n \times n$ torus. A high level description of the Procedure `SearchTorus` is the following:

The two agents search one by one the horizontal rings of the torus (using Procedure `FindTokenHor`) to discover whether they have started in the same ring. If so, then they execute Procedure `RVDRing`. Otherwise they try to 'catch' each other on the torus using a path, marked by their tokens. If they do not rendezvous then they search one by one the vertical rings of the torus (using Procedure `FindTokenVer`). They again try to 'catch' each other on the torus. If they do not meet this time they declare rendezvous impossible. Algorithm `RVD2n` takes $O(n^2)$ time.

Theorem 8. *The Rendezvous with Detection problem on a $n \times n$ torus can be solved by two agents using two movable tokens and constant memory each, in time $O(n^2)$.*

Proof. The agents execute Algorithm `RVD2n`. They release their first token and follow Procedure `FindTokenHor`.

If they find a token while going down this means that they have started in the same ring (horizontal or vertical). If this is the case, then they get synchronized by continuing moving in the same way (it is like executing Procedure `FindTokeHor` once more, ignoring the token that they have just found). Now they are at the same time back at their starting points knowing that they have started in the same ring. They execute Procedure `RVDRing` in the horizontal ring and then (if no rendezvous occurs) in the vertical ring. After that either they rendezvous or stop, declaring rendezvous impossible (which in view of Theorem 1 is true).

If they find a token while moving right then they know that they have started in different rings. In that case, after they meet a token right they go up until they meet a token.

Procedure SearchTorus

```
release the first token
FindTokenHor
if SameRing then
   Synchronize
   RVDRing on the horizontal ring
   RVDRing on the vertical ring
   if not rendezvous then
      stop and declare rendezvous impossible
   end if
else
   go up until you meet a token
   go one step down
   repeat
      go left, wait 1 time step
   until (rendezvous) OR (you meet a token for the second time)
   if not rendezvous then
      Synchronize2
      FindTokenVer
      go left until you meet a token
      go one step right
      repeat
         go up, wait 1 time step
      until (rendezvous) OR (you meet a token for the second time)
   end if
end if
```

Algorithm 2 RVD2n

```
SearchTorus
if not rendezvous then
   stop and declare rendezvous impossible
end if
```

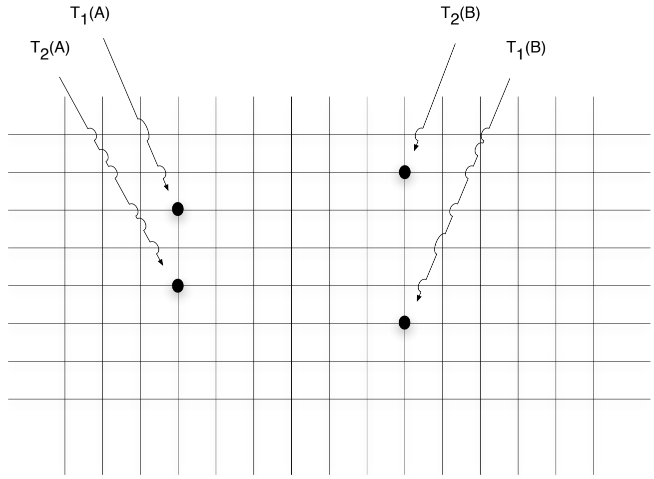

Fig. 6. A possible situation in the torus.

Consider agent A with initial vertical distance d_y from agent B. We suppose wlog that agent B is $d_y \leq n/2$ down of agent A and d_x to the right (notice that d_x can have any value lower than n). When A goes up (after it met a token right) it will meet either (again) B's first token (when $d_y = 1$ and d_x small enough) or B's second token since every agent moves its second token in the vertical ring where its first token lies.

Suppose that agent B still executes Procedure `FindTokenHor` (not having found yet a token right). We claim that if A goes one step down after it finds B's (first or second) token and then left-wait-1-step repeatedly, then either it will meet agent B, or it will find a token. Let's see why:

Agent A went down $d_y \leq n/2$ and then d_x until it met the other's first token. After that, agent A goes up and meets the other's second token $t_2(B)$[6] since: agent B moves its second token always in the vertical ring where its first token lies until it meets the other's first token. After that the first agent that will possibly move $t_2(B)$ is A. Therefore, token $t_2(B)$ has to be somewhere in a ring above the ring where A's first token $t_1(A)$ lies.

Suppose that token $t_2(B)$ is in a ring not adjacent to the ring that $t_1(A)$ lies. This means that the other agent B is still searching that ring coming from left to right. Therefore they will rendezvous.

If token $t_2(B)$ is just one horizontal ring above the ring where token $t_1(A)$ lies then the situation is shown in Figure 6. In that case the agents may not rendezvous if agent B reaches first $t_1(A)$. Suppose that they do not rendezvous. We prove now some properties about d_y and d_x.

[6] If agent A met token $t_1(B)$ again while going up, then the agents will rendezvous when A will start to go left, unless they are in a 2x2 torus

Agent A moves as follows: After $3n$ steps it meets its own token (going up) for the first time. it takes him another $3n$ time units to meet its token for the second time plus one for going down. This is repeated $d_y - 1$ times (until it is one ring above the ring where the other's first token lies). It takes another $3(d_x - 1) + 2$ to meet the other's first token. So far it has spent $(1 + 6n)(d_y - 1) + 3(d_x - 1) + 2$. Then it goes up $d_y + 1$ steps until it meets the other's second token. It goes one step down and another $2(d_x - 1) + 1$ steps until it meets its own first token for the first time. Totally it took him to be there $(1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + d_y + 2 + 2(d_x - 1) + 1 = 6nd_y + 2d_y - 6n + 5d_x - 1$.

Agent B needs $(1 + 6n)(n - d_y - 1) + 3(n - d_x - 1) + 2 = 6n^2 - 6nd_y - d_y - 3d_x - 2n - 2$ time units to be there.

If the agents do not rendezvous until A meets the token for the second time then this means that agent B reached that token ($t_1(A)$) first. Thus

$$6nd_y + 2d_y - 6n + 5d_x - 1 > 6n^2 - 6nd_y - d_y - 3d_x - 2n - 2$$

$$12nd_y + 3d_y + 8d_x + 1 > 6n^2 + 4n$$

Let $d_y = \frac{n-k}{2}$, $k \geq 0$. The above inequality implies that $d_x > \frac{3nk}{4} + \frac{5n}{16} + \frac{3k}{16} - \frac{1}{8}$. Since $d_x < n$, the previous inequality holds only when $k = 0$. This means that $d_y = n/2$. Observe that if n is an odd number, then they will always rendezvous.

Then they get synchronized by moving for example left-up-down until they meet the other's second token again. They pick it up (they will use it from now on as their second token) and they go one step down and then repeatedly right-wait-1 until they meet their first token. It is easy to see that they will reach their first token at the same time.

Now they repeat the same procedure by replacing down with right, right with down, up with left and left with up (the situation will be like the one shown in Figure 6 rotated by 90 degrees). Using the same arguments as before, if they do not rendezvous, then $d_x = n/2$ and in view of Theorem 1, they can safely decide that rendezvous is impossible. □

Another algorithm for this case is the following: if the agents discover that they have started on different rings then they **first** search whether they are at distance $(n/2, n/2)$ and if not then they search one by one the horizontal rings of the torus. We have chosen to present here the first approach since it is expandable to a $n \times m$ torus.

3.3 Rendezvous without Detection in a $n \times m$ torus using two movable tokens and constant memory

We now give Algorithm `RV2mn` which is a RV algorithm for two agents with constant memory in a $n \times m$ torus. Algorithm `RV2mn`, at first, copies Algorithm `RVD2n`. If no rendezvous occurs and no decision is made about its impossibility (i.e., the agents have started in different rings), the algorithm instructs the agents to mark a rectangle with their tokens on the torus. If $n > 2$ (the agents can identify such a configuration by executing an horizontal movement going for example right until they meet the second token; if they have met at least one node without a token then they conclude that $n > 2$) then they execute Procedure `Pendulum`: they try to shrink the rectangle and eventually meet which will happen unless they had started at distance $(n/2, m/2)$ (in that case the algorithm runs forever). We give below Procedure `Pendulum`.

Procedure Pendulum

```
(* if you hit another token when you move a token at any time in the procedure then stop and wait *)
repeat
  move token to the right
  repeat
    go down, left, right
  until you meet the third token
  move token to the right
  repeat
    go right
  until you meet the third token
  move token to the left
  repeat
    go up, right, left
  until you meet a token
  if you met a token while going up then
    wait
  else
    (* must have met a token while going right *)
    move token to the left
  end if
  repeat
    go left
  until you meet the third token
until rendezvous
```

We first prove that if the configuration is as in Figure 7 and either the horizontal distance between the agents is $n/2$ or the vertical distance between the agents is $m/2$ then Procedure `Pendulum` solves the Rendezvous without Detection problem for any $n > 2$.

Lemma 11. *Suppose that there is a rectangle marked by four tokens in a $n \times m$ torus, where $n > 2$. Suppose also that either the horizontal distance is $n/2$ or the vertical distance is $m/2$. There are two agents situated on the upper left and bottom right corners of the rectangle (see Figure 7). Then Procedure* `Pendulum` *is an RV algorithm. That is will lead the agents to rendezvous in $O(n^2 + m^2)$ time, unless their distance is $(n/2, m/2)$.*

Proof. Suppose that $d_x = n/2$ and $d_y < m/2$. Let agent A be the agent which has to go down distance d_y until he meets a token. The other agent B will have to go a distance $m - d_y > d_y$ until it meets a token. After the two agents move the tokens right, they go down until they meet the third token (notice that they will surely meet the token since they go down-left-right). When agent A meets the second token $T_1(A)$, agent B meets also

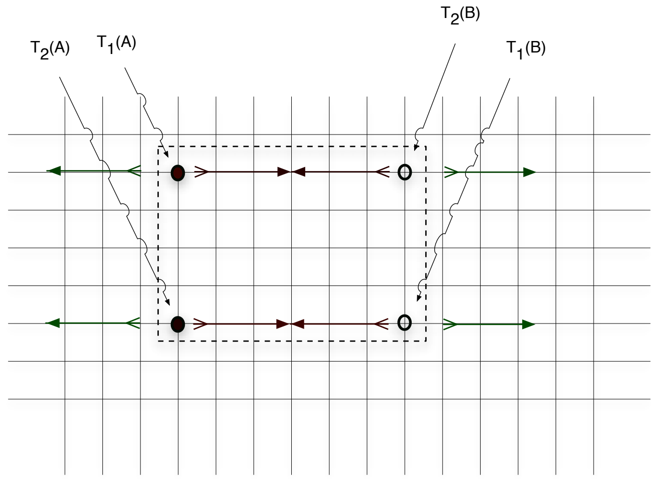

Fig. 7. Two agents with constant memory and two movable tokens each in a $n \times m$ torus.

the second token $T_1(B)$, since both agents have travelled the same distance. Therefore when agent A reaches $T_1(B)$, agent B is either on its way to meet the third token $T_2(B)$, or has already met it.

- If agent A reaches $T_2(B)$ before agent B, then it must meet it while going up, since agent B has not yet moved the token. In that case agent A waits and eventually agent B will be there.
- If agent A reaches $T_2(B)$ after agent B has left but B has not met again $T_2(B)$ during its horizontal movement then agent A will meet agent B.
- If agent A reaches $T_2(B)$ after agent B has left and B has already met $T_2(B)$ as the second token during its horizontal movement then agent B will reach, move and leave token $T_1(A)$ before agent A gets there. After that, agent B will reach token $T_2(A)$ before agent A meets token $T_2(A)$ as its third token, since agent B travels a vertical distance $m - d_y$ and agent A travels a vertical distance $m + d_y$ (notice also that agent B will not meet token $T_2(A)$ while going up). Therefore either the agents will meet, or agent B will reach, move and leave first token $T_1(B)$, which was its starting position. However now agent A is closer than in the beginning. For exactly the same reasons, the agents maintain their relative positions, i.e., agent B reaches, moves and leaves a token before agent A reaches, moves and leaves the same token. Since their distance decreases, they will rendezvous.

If $d_x < n/2$ and $d_y = m/2$. Let agent A be the agent which has to go right distance d_x until he meets a token. The other agent B will have to go a distance $n - d_x > d_x$ until it meets a token. After the two agents move the tokens right they go down and they meet at the same time their third token (notice that they will meet it for sure since they go down-left-right). Hence agent A is at $T_2(A)$ exactly when agent B is at $T_2(B)$. After $n+d_x$

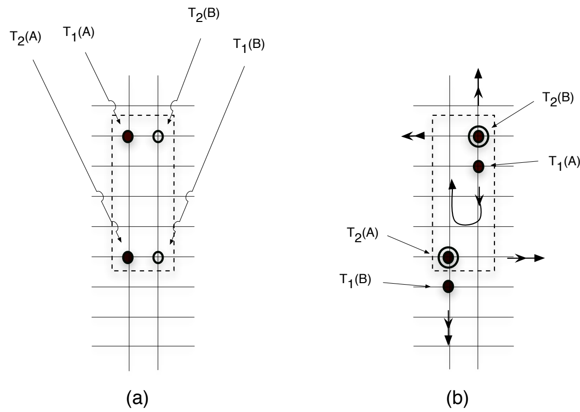

Fig. 8. a) A special configuration where $n = 2$. b) One agent A (upper left corner) moves the token right and then is moving in cycles in the inner area moving down (one step at each cycle) token $T_1(A)$ until it is next to token $T_2(A)$. The other agent B (down right corner) moves the token right and then is moving in cycles in the outside area moving down (one step at each cycle) token $T_1(B)$ until it is next to token $T_2(B)$. The agents do not move at all tokens $T_2(A)$ and $T_2(B)$.

steps, agent A reaches, moves and leaves $T_1(B)$ and agent B has already moved and left $T_2(B)$. Because of this agent A will meet token $T_2(B)$ not while going up.

- If agent A reaches $T_1(A)$ before agent B moved and left it then the two agents will meet.
- If agent A reaches $T_1(A)$ after agent B has left then agent B will reach, move and leave first token $T_2(A)$, since $d_y = m/2$. Hence during the horizontal movement from $T_2(A)$ to $T_1(B)$, either the agents will meet or agent B will first move and leave token $T_1(B)$, which was its starting position. However now agent A is closer than in the beginning. For exactly the same reasons, the agents maintain their relative positions, i.e., agent B reaches, moves and leaves a token before agent A reaches, moves and leaves the same token. Since their distance decreases, they will rendezvous.

□

For the remaining case, if $n = 2$, the configuration is as in Figure 8(a). In that case the agents execute Procedure `RVDRing2` which basically simulates the idea of the Procedure `RVDRing`. One agent A (upper left corner) moves the token right and then is moving in cycles in the inner area moving down (one step at each cycle) token $T_1(A)$ until it is next to token $T_2(A)$. The other agent B (down right corner) moves the token right and then is moving in cycles in the outside area moving down (one step at each cycle) token $T_1(B)$ until it is next to token $T_2(B)$. The agents do not move at all tokens $T_2(A)$ and $T_2(B)$. The distance covered by agent A until it meets and move again token $T_1(A)$ is different than the distance covered by agent B until it meets and move again token $T_1(B)$. Agent A walks from the inside, while agent B walks from the outside. This is illustrated in Figure 8(b).

Procedure RVDRing2

```
move token one step right
move one token, one step down
move left,right
repeat
    move down, left, right until you meet a token
    if you met a token while going down then
        wait there
    end if
    move up, right, left until you meet the second token right
    if you met a token while going up then
        wait there
    end if
    move down until you meet a token
    move one token, one step down
    move left,right
until there is another token to the left
go up until you meet a token
if there is another token next to it then
    stop and declare rendezvous impossible
else
    wait there
end if
```

The following Algorithm `RV2mn` is a RV algorithm for 2 agents having constant memory in a $n \times m$ torus. In fact one of the following things could happen: either the agents rendezvous, or they detect that they are in the same ring in symmetrical positions or the algorithm runs forever (in that case they are at horizontal distance $n/2$ and vertical distance $m/2$).

Algorithm 3 RV2mn

```
SearchTorus
if not rendezvous then
    Synchronize3
    BuildRectangle
    Synchronize4
    if n > 2 then
        Pendulum
    else
        RVDRing2
    end if
end if
```

Theorem 9. *The Rendezvous without Detection problem on an arbitrary $n \times m$ torus can be solved by two agents using two movable tokens and constant memory each, in time $O(n^2 + m^2)$.*

Proof. The agents follow Algorithm `RV2mn`. They first copy Algorithm `RVD2n` (as in the case of a $n \times n$ torus). We first prove that if they do not rendezvous after horizontal and

Procedure BuildRectangle

```
1: go right until you meet a token
2: move that token one step right
3: repeat
4:    go up, wait 1 time step
5: until you meet a token
6: go left until you meet a token
7: go down until you meet a token
```

vertical scanning then either their horizontal distance $d_x = n/2$ or their vertical distance $d_y = m/2$ (or both).

Let's see what happens during the horizontal scanning. Consider agent A whose vertical movement until it reaches agent B's ring is $d_y = \frac{m-k}{2}$, where $k \geq 0$ is an integer. Agent A travelled $6nd_y + 2d_y - 6n + 5d_x - 1$ while agent B travelled $6nm - 6nd_y - 3n - 3d_x + m - d_y - 2$ (see the proof of Theorem 8). Suppose that they do not rendezvous. This means that:

$$6nd_y + 2d_y - 6n + 5d_x - 1 > 6nm - 6nd_y - 3n - 3d_x + m - d_y - 2$$

$$12nd_y + 3d_y + 8d_x + 1 > 6nm + 3n + m$$

Since $d_y = \frac{m-k}{2}$, the above inequality implies:

$$d_x > \frac{3}{4}nk + \frac{3}{16}k + \frac{3n}{8} - \frac{m}{16} - \frac{1}{8}$$

The horizontal movement of agent A to reach B's token is either $d_x = \frac{n-\delta}{2}$ or $d_x = \frac{n+\delta}{2}$, where $\delta \geq 0$ is an integer. For these two cases we get from the previous inequality:

$$d_x = \frac{n-\delta}{2} : 8n - 8\delta > 12nk + 3k + 6n - m - 2 \tag{1}$$

$$d_x = \frac{n+\delta}{2} : 8n + 8\delta > 12nk + 3k + 6n - m - 2 \tag{2}$$

After picking up their second token and synchronizing (in a different way that suggested in the proof of Theorem 8 but still easy to implement) they do the vertical scanning. Consider the agent whose horizontal movement until it reaches the other's agent ring is $d'_x = \frac{n-\delta}{2}$, where $\delta \geq 0$. This agent travelled $6md'_x + 2d'_x - 6m + 5d'_y - 1$ while the other agent travelled $6nm - 6md'_x - 3m - 3d'_y + n - d'_x - 2$. Suppose that they do not rendezvous. This means that:

$$6md'_x + 2d'_x - 6m + 5d'_y - 1 > 6nm - 6md'_x - 3m - 3d'_y + n - d'_x - 2$$

$$12md'_x + 3d'_x + 8d'_y + 1 > 6nm + 3m + n$$

Since $d'_x = \frac{n-\delta}{2}$,

$$d'_y > \frac{3}{4}m\delta + \frac{3}{16}\delta + \frac{3m}{8} - \frac{n}{16} - \frac{1}{8}$$

As before we have:

$$d'_y = \frac{m-k}{2} : 8m - 8k > 12m\delta + 3\delta + 6m - n - 2 \quad (3)$$

$$d'_y = \frac{m+k}{2} : 8m + 8k > 12m\delta + 3\delta + 6m - n - 2 \quad (4)$$

Now observe the following:

– if $d_x = \frac{n-\delta}{2}$ then $d'_x = d_x$ and $d'_y = d_y = \frac{m-k}{2}$ and the relations (1), (3) hold,
– if $d_x = \frac{n+\delta}{2}$ then $d'_y = \frac{m+k}{2}$ and the relations (2), (4) hold.

In the first case, if we add the relations (1), (3) we get:

$$(3n - \frac{5}{4})(1-4k) + (3m - \frac{5}{4})(1-4\delta) + \frac{26}{4} > 0 \quad (5)$$

In the second case, if we add the relations (2), (4) we get:

$$(3n + \frac{11}{4})(1-4k) + (3m + \frac{11}{4})(1-4\delta) - \frac{3}{2} > 0 \quad (6)$$

It is easy to see now that if $k > 0$ and $\delta > 0$ both relations (5), (6) are false (assuming $n, m > 1$). Therefore either $k = 0$ or $\delta = 0$ (or both). Hence either $d_y = m/2$ or $d_x = n/2$ (or both).

Now the two agents get synchronized, build the rectangle and run the Procedure `Pendulum` which according to Lemma 11 ends up to rendezvous unless $d_x = n/2$ **and** $d_y = m/2$. □

An interesting question which naturally follows is: what is the relation of n and m for which Algorithm `RV2mn` is indeed a RVD algorithm? The answer is given by the following lemma.

Lemma 12. *If after the horizontal and vertical scanning of Algorithm* `RV2mn` *the agents do not rendezvous and* $\frac{n-1}{10} \leq m \leq 2n+17$ *then their distance is* $(n/2, m/2)$ *and therefore rendezvous is impossible.*

Proof. The agents execute Algorithm `RV2mn`. Recall that if the agents do not rendezvous after the horizontal scanning, then it must hold $d_x > \frac{3}{4}nk + \frac{3}{16}k + \frac{3n}{8} - \frac{m}{16} - \frac{1}{8}$.

Suppose that $k \geq 1$. Then $d_x > \frac{3}{4}nk + \frac{3}{16}k + \frac{3n}{8} - \frac{m}{16} - \frac{1}{8} \geq \frac{9n}{8} + \frac{1}{16} - \frac{m}{16}$. It holds that $d_x < n$, since the agents did not start on the same ring. Therefore it must hold that $\frac{9n}{8} + \frac{1}{16} - \frac{m}{16} < n - 1 \rightarrow m > 2n + 17$. The contraposition implies that if $m \leq 2n + 17$ then $k = 0$. This means that $d_y = m/2$.

After the vertical scanning it holds $d'_y = \frac{m}{2} > \frac{3}{4}m\delta + \frac{3}{16}\delta + \frac{3m}{8} - \frac{n}{16} - \frac{1}{8}$.

Suppose that $\delta \geq 1$. Then $\frac{3}{4}m\delta + \frac{3}{16}\delta + \frac{3m}{8} - \frac{n}{16} - \frac{1}{8} \geq \frac{9m}{8} + \frac{1}{16} - \frac{n}{16}$. Therefore it must hold $\frac{9m}{8} + \frac{1}{16} - \frac{n}{16} < \frac{m}{2} \rightarrow m < \frac{n-1}{10}$. The contraposition of this concludes the lemma. □

Therefore by Lemma 12 if we knew that $\frac{n-1}{10} \leq m \leq 2n+17$ then Algorithm RV2mn would be a RVD algorithm for the $n \times m$ torus.

3.4 Rendezvous with Detection in a $n \times m$ torus using three movable tokens and constant memory

If the agents have 3 tokens then we can extend the Algorithm RVD2n to get a RVD algorithm for a $n \times m$ torus. The idea is the following: If the agents do not meet while they copy Algorithm RVD2n then they mark a rectangle on the torus using their two tokens each. Next they release their third token to the right of their starting position. They travel on this rectangle (one agent from inside and the other from outside), each time moving one step the fifth token they meet: first they move it to the right until it hits another token and then down until it touches a token. Next they go left until they meet a token and then up until they meet a token. If at that point they see two tokens adjacent then they declare rendezvous impossible. Otherwise they wait until rendezvous which will occur in less than $n+m$ time. Algorithm RVD3mn takes $O(n^2+m^2)$ time. The Synchronize procedures can be implemented easily.

Theorem 10. *The Rendezvous with Detection problem on an arbitrary $n \times m$ torus can be solved by two agents using three movable tokens and constant memory each, in time $O(n^2+m^2)$.*

Proof. The agents follow Algorithm RVD3mn. Suppose that after executing Procedure SearchTorus they do not rendezvous neither decide that it is impossible. This means that either their horizontal distance $d_x = n/2$ or their vertical distance $d_y = m/2$ (or both). The agents get synchronized and form a rectangle in the torus. One of the agents (say A) travels $(2d_x+2d_y)(d_x+d_y-1)+d_x+d_y$ until it sees its third token touches for the second time a token. The other agent B travels $(2(n-d_x)+2(m-d_y))(n-d_x+m-d_y-1)+n-d_x+m-d_y$ until it sees its third token touches for the second time a token.

Suppose wlog that $d_x = n/2$ and $d_y \leq m/2$. If $d_y = m/2$ then the two agents always move at the same time their tokens and at the end (when their third token is adjacent for the second time to a token) they find out that the other's token is adjacent to another token as well. Therefore they declare rendezvous impossible.

If $d_y < m/2$ then by the time agent A sees its third token touching for the second time a token, agent B has traveled at most $d_x+d_y-1 < n/2+m/2-1$ times the rectangle. Thus B needs at least one more round to move its third token close to another token. This means another $2(n-d_x)+2(m-d_y) \geq n+m$ steps for agent B. But in $d_x+d_y \leq n/2+m/2$ steps agent A will meet B's third token. Therefore if A waits there, B will eventually come. □

Algorithm 4 RVD3mn

```
SearchTorus
if not rendezvous then
    Synchronize3
    BuildRectangle
    Synchronize4
    go right one step and drop the third token
    repeat
        go right until you meet a token
        go down until you meet a token
        go left until you meet a token
        go up until you meet a token
        go right until you meet a token
        move that token one step to the right
    until the token hits another token
    repeat
        go down until you meet a token
        go left until you meet a token
        go up until you meet a token
        go right until you meet a token
        go down until you meet a token
        move that token one step down
    until the token is adjacent to another token
    go left until you meet a token
    go up until you meet a token
    if there are two tokens adjacent then
        stop and declare rendezvous impossible
    else
        wait
    end if
end if
```

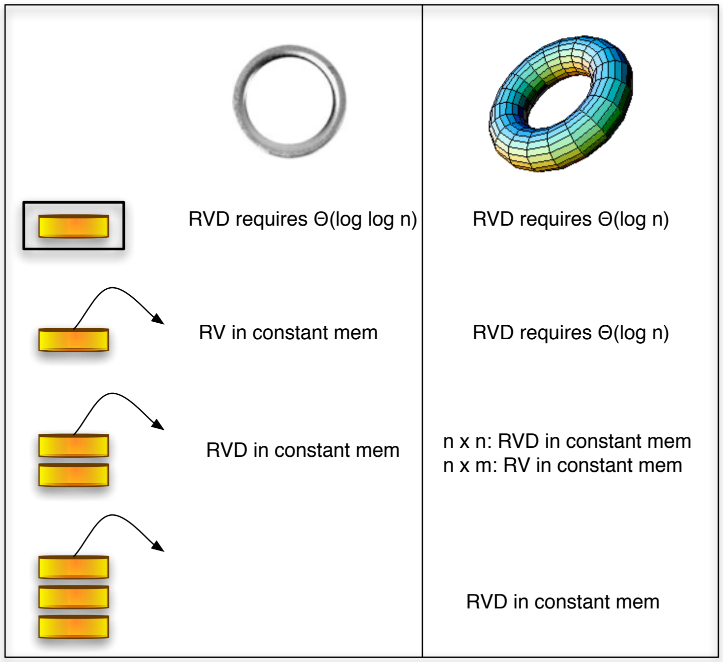

Fig. 9. Ring vs Torus.

4 Conclusion

In this paper we investigated on the number of tokens and memory, two agents need in order to rendezvous in an anonymous oriented torus.

It appears that there is a strict hierarchy on the power of tokens and memory with respect to rendezvous: a constant number of unmovable tokens are less powerful than two movable tokens. While the hierarchy collapses on three tokens (we gave an algorithm for rendezvous with detection in a $n \times m$ torus when the agents have constant memory each), it remains an open question if three tokens are strictly more powerful than two with respect to rendezvous with detection. It is also interesting that although a movable token is more powerful than an unmovable one (we showed that an agent with one unmovable token cannot visit all the nodes of a torus with a properly selected size unless it has $\Omega(\log n)$ memory, while it could do it with a constant memory if it could move its token) it appears that this power is not enough with respect to rendezvous; the agents with one movable token each, still require $\Omega(\log n)$ memory to rendezvous in the torus. The results for ring and torus topologies are shown in Figure 9.

As this is the first publication in the literature that studies tradeoffs between the number of tokens, memory, knowledge and power the agents need in order to meet on a torus network, a lot of interesting questions remain open:

- Can we improve the time complexity for rendezvous without detection on a $n \times m$ torus using constant memory? Can we improve the time complexity for rendezvous with detection on a $n \times n$ torus using constant memory?

- What is the lower memory bound for two agents with two movable tokens each in order to do rendezvous with detection in a $n \times m$ torus? In particular, can they do it with constant memory?

- What is the situation in a d-dimensional torus? Is it the case that with $d-1$ movable tokens, rendezvous needs $\Omega(\log n)$ memory while with d movable tokens and constant memory rendezvous with detection can be done? How does this change if the size of the torus is not the same in every dimension?

- What are the results if the torus is not oriented? If the torus is asynchronous?

- Finally, an interesting problem is that of many agents trying to rendezvous (or gathering) in a torus network.

References

1. S. Alpern, The Rendezvous Search Problem, SIAM Journal of Control and Optimization, 33, pp. 673-683, 1995. (Earlier version: LSE CDAM Research Report, 53, 1993.)
2. S. Alpern, Rendezvous Search: A Personal Perspective, Operations Research, 50, No. 5, pp. 772-795, 2002.
3. S. Alpern and V. Baston, Rendezvous on a Planar Lattice, Operations Research, 53, No. 6, pp. 996-1006, 2005.
4. S. Alpern and S. Gal, The Theory of Search Games and Rendezvous, Kluwer Academic Publishers, Norwell, Massachusetts, 2003.
5. E.J. Anderson and S. Fekete, Two-dimensional Rendezvous Search, Operations Research, 49, No. 1, pp. 107-188, 2001.
6. L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, Election and Rendezvous of Anonymous Mobile Agents in Anonymous Networks with Sense of Direction, Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO), pp. 17-32, 2003.
7. V. Baston and S. Gal, Rendezvous Search When Marks Are Left at the Starting Points, Naval Research Logistics, 47, No. 6, pp. 722-731, 2001.
8. A. Dessmark, P. Fraigniaud, and A. Pelc, Deterministic Rendezvous in Graphs, 11th Annual European Symposium on Algorithms (ESA), pp. 184-195, 2003.
9. S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro, Multiple agents rendezvous in a ring in spite of a black hole, Symposium on Principles of Distributed Systems (OPODIS '03), LNCS 3144, pp. 34-46, 2004.
10. P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, Multiple Mobile Agent Rendezvous in the Ring, LATIN 2004, LNCS 2976, pp. 599-608, 2004.
11. L. Gasieniec, E. Kranakis, D. Krizanc, X. Zhang, Optimal Memory Rendezvous of Anonymous Mobile Agents in a Uni-directional Ring. In proceedings of SOFSEM 2006, 32nd International Conference on Current Trends in Theory and Practice of Computer Science January 21 - 27, 2006 Merin, Czech Republic, SVLNCS, 2006, to appear.
12. E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, Mobile Agent Rendezvous Search Problem in the Ring, International Conference on Distributed Computing Systems (ICDCS), pp. 592-599, 2003.
13. C. Sawchuk, Mobile Agent Rendezvous in the Ring, PhD thesis, Carleton University, School of Computer Science, Ottawa, Canada, 2004.
14. CL. E. Shannon, Presentation of a Maze-Solving Machine, in 8th Conf. of the Josiah Macy Jr. Found. (Cybernetics), pp. 173-180, 1951.
15. X. Yu and M. Yung, Agent Rendezvous: A Dynamic Symmetry-Breaking Problem, in Proceedings of ICALP '96, LNCS 1099, pp. 610-621, 1996.