Deterministic symmetric rendezvous with tokens in a synchronous torus

Evangelos Kranakis\textsuperscript{a}, Danny Krizanc\textsuperscript{b}, Euripides Markou\textsuperscript{c,∗}

\textsuperscript{a} School of Computer Science, Carleton University, Ottawa, Ontario, Canada
\textsuperscript{b} Department of Mathematics, Wesleyan University, Middletown, CT 06459, USA
\textsuperscript{c} Department of Computer Science & Biomedical Informatics, University of Central Greece, Lamia, Greece

\section*{A R T I C L E   I N F O}

Article history:
Received 18 September 2008
Received in revised form 7 August 2010
Accepted 27 January 2011
Available online 24 February 2011

Keywords:
Mobile agent
Rendezvous
Rendezvous with detection
Tokens
Torus
Synchronous

\section*{A B S T R A C T}

In the rendezvous problem, the goal for two mobile agents is to meet whenever this is possible. In the rendezvous with detection problem, an additional goal for the agents is to detect the impossibility of a rendezvous (e.g., due to symmetrical initial positions of the agents) and stop. We consider the rendezvous problem with and without detection for identical anonymous mobile agents (i.e., running the same deterministic algorithm) with tokens in an anonymous synchronous torus with a sense of direction, and show that there is a striking \textit{computational difference} between one and more tokens. Specifically, we show that (1) two agents with a constant number of unmovable tokens, or with one movable token each, cannot rendezvous in an \(n \times n\) torus if they have \(o(\log n)\) memory, while they can solve the rendezvous with detection problem in an \(n \times m\) torus as long as they have one unmovable token and \(O(\log n + \log m)\) memory; in contrast, (2) when two agents have two movable tokens each then the rendezvous problem (respectively, rendezvous with detection problem) is solvable 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 solve the rendezvous with detection problem in an \(n \times n\) 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 a torus.

\section*{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 at a node or an edge?

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\textsuperscript{[1,2]}, as well as the book by Alpern and Gal\textsuperscript{[4]}.) Yu and Yung\textsuperscript{[20]} 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\textsuperscript{[7]} mark the starting points of the agents, they still rely on randomized algorithms or different deterministic algorithms.

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 [20] have considered attaching unique identifiers to the agents, while Dessmark et al. [11] added unbounded memory; note that having different identities allows each agent to execute a different algorithm. Other researchers [6, 12] have given the agents the ability to leave notes in each node they visit. De Marco et al. [10] study the rendezvous problem in arbitrary asynchronous networks for two agents which have unique identifiers, and they solve the problem when an upper bound on the number of nodes of the network is known. The problem is left open when no upper bound on the number of nodes is known.

In another approach, each agent has a stationary token placed at the initial position of the agent. This model is much less powerful than having distinct identities or than having 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 [18] and studied in [17, 13, 14, 19] for the ring topology. In particular, the authors proved in [17] 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 make a rendezvous with detection. They also proved that if the token is movable then a rendezvous without detection is possible with constant memory.

We keep here the same model as in [9, 13, 14, 17, 18] with the exception of the underlying graph topology. Specifically, we study here the following scenario: there are two identical anonymous agents running the same deterministic algorithm in an an anonymous and synchronous oriented torus. As this is one of the weakest models which has appeared in the literature, we would like to study more general topologies, and the selection of the torus is a step towards arbitrary topologies. In particular, we are interested in answering the following questions. What memory do the agents need to solve the rendezvous problem 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 anonymous and 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 an \( n \times m \) 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., all 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 nodes. The agents follow the same deterministic algorithm, and begin execution at the same time and being at the same initial state.

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, once released, it can occupy only the node in which it has been released.

More formally, we consider a mobile agent as a finite Moore automaton\(^1\) \( A = (X, Y, \delta, \lambda, S_0) \), where \( X \subseteq \delta \times C_v \times C_{MA}, Y \subseteq \delta \times \{ \text{drop, take} \}, \delta \) is a set of \( \sigma \geq 2 \) states among which there is a specified state \( S_0 \), called the initial state, \( \lambda : \delta \rightarrow Y, \) \( \delta \) 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, \( \delta = \{\text{up, down, left, right, stay}\} \). (Stay represents the situation where the agent does not move). \( C_v = \{\text{agent, token, empty}\} \) is the set of possible configurations of a node (if there is an agent and a token in a node then its configuration is agent). Finally, \( C_{MA} = \{\text{token, no\textunderscore token}\} \) 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 \( u_0 \), at the initial state \( S_0 \in \delta \). \( S_0 \) determines an action (drop token or nothing) and a direction from 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 from which it entered \( v \), the configuration \( c_v \in C_v \) of node \( v \) (i.e., whether there is a token or an agent in \( v \)), and of course the configuration \( c_{MA} \in C_{MA} \) 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 \( \lambda(S') \), from 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 \( \Theta(\log(|\delta|)) \) bits. The agents know that there are two of them and they also know the number of tokens they have. Memory permitting, an agent can count the number of nodes between tokens, or the total number of nodes of the torus, etc. Since the agents are identical, they face the same limitations on their knowledge of the network.

---

\(^1\) The first known algorithm designed for graph exploration by a mobile agent, modeled as a finite automaton, was introduced by Shannon [19] in 1951.
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 agents start at the same initial state $S_0$ and at the same time. We assume that crossing a link takes one time unit.

A rendezvous occurs when the agents either meet on a network node or simultaneously cross the same network link while moving in opposite directions. We say that the agents can solve the rendezvous with detection (RVD) problem in a torus if, no matter what the initial positions of the agents are, after a finite time either they rendezvous or they stop, declaring that a rendezvous is impossible. We say that the agents can solve the rendezvous without detection (RV) problem in a torus, if they meet whenever a rendezvous is possible. In this case, if a rendezvous is impossible, they may move forever on the torus.

The distance between two nodes $(x_1, x_2)$ and $(y_1, y_2)$ on a two-dimensional torus $n \times m$ is a two-dimensional vector $(d_1, d_2)$, where $d_1 = \min\{|x_1 - y_1|, (n - |x_1 - y_1|)\}$ and $d_2 = \min\{|x_2 - y_2|, (m - |x_2 - y_2|)\}$. An example of two agents in a torus is shown in Fig. 1.

**Theorem 1.** Consider two identical agents placed in a two-dimensional oriented torus $(n \times m)$, so that their distance is either $(n/2, 0)$ (with $n$ even) or $(0, m/2)$ (with $m$ even) or $(n/2, m/2)$ (with $n, m$ even). The agents start at the same initial state and at the same time. Then, no matter how many tokens (movable or unmovable) or how much memory the agents have, it is impossible for the agents to rendezvous at a node or an edge.

**Proof.** Let $D$ be the initial distance of the agents, with $D = (n/2, 0)$ or $D = (0, m/2)$ or $D = (n/2, m/2)$. Since the agents start at the same state $S_0$ and at the same time, as long as they do not release tokens, they simultaneously enter the same state $S_t$ at any time $\tau$, moving in the same direction, thus maintaining their initial distance. Hence, if they release a token, they do it at the same time $\tau_k$, being at the same state $S_k$ and having the initial distance $D$. Thus the distance of the released tokens is also $D$; i.e., for any token $T_A$ released by agent $A$, there is a token $T_B$ released by agent $B$ at distance $D$. At any time $\tau' > \tau_k$, as long as the agents do not meet tokens while they are moving in the same direction, they again simultaneously enter the same state $S_{\tau'}$ maintaining distance $D$. Now, suppose that one of the agents, say $A$, meets a token while moving. Consider the following two cases.

(i) Suppose that $A$ meets token $T_A$ which had been released by $B$ earlier (we recall here that all tokens are indistinguishable, and hence agent $A$ is not aware that $T_B$ has been released by it). In this case, since the agents were moving identically, maintaining distance $D$, agent $B$ must meet token $T_B$ (which had been released by $B$ earlier) at exactly the same time.

(ii) Suppose that $A$ meets token $T_B$ (which had been released by $B$ earlier) at time $\tau_k$. This means that, at time $\tau_k$, agent $A$ is at distance $D$ from token $T_A$. Since up to that time, $\tau_k$, the agents were moving identically (i.e., in the same direction, entering the same states and covering the same distance), agent $B$ is at distance $D$ from token $T_B$. But this is the position where token $T_A$ lies. Hence, at time $\tau_k$, both agents meet tokens.

In other words, they are always simultaneously entering the same states, and their configuration is the same (they both carry the same number of tokens), while the configuration of the node they occupy is always the same (no tokens or the same number of tokens). Therefore, since they act identically, they maintain their distance forever. \hfill \Box

**Theorem 1** is a generalization of Theorem 1 in [17], 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.

Given two mobile agents initially situated at two nodes of an $n \times m$ torus, we formally define the two following problems.
Definition 1. We call the rendezvous with detection (RVD) problem the problem in which the task for the agents is either to meet each other at a node or an edge whenever this is possible (i.e., if their distance is not $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$), and otherwise detect the impossibility of meeting each other and stop moving.

We say that an algorithm $A$ solves the RVD problem (or $A$ is an RVD algorithm) if: (a) $A$ leads the agents to rendezvous at a node or an edge after a finite time, when their initial distance is not $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$, and (b) $A$ halts after a finite number of steps and the agents declare that a rendezvous is impossible, when their initial distance is $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$.

Definition 2. We call the rendezvous without detection problem, or simply the rendezvous (RV) problem, the problem in which the task for the two agents is to meet each other at a node or an edge when their initial distance is not $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$.

Therefore we say that an algorithm $A$ solves the RV problem (or $A$ is an RV algorithm) if the agents rendezvous at a node or an edge when their initial distance is not $(n/2, 0)$ or $(0, m/2)$ or $(n/2, m/2)$. Otherwise, $A$ may run forever.

The input in any of those algorithms is the number of agents (which are anonymous, identical, and start at the same initial state), the number of tokens (which are indistinguishable and may be movable or unmovable) and the information that the underlying graph topology is an anonymous, synchronous and oriented torus. In other words, an RV or RVD algorithm should solve the RV or RVD problem in any anonymous, synchronous and oriented torus. In one of our algorithms we add to the input the information that the torus is square (same number of nodes horizontally and vertically). 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 the cost $CT_{RVD}(A, G, s, s')$ of an RVD algorithm $A$ the minimum time (number of steps plus waiting time of an agent) needed to rendezvous (or to decide that a rendezvous is impossible if this is the case for $s$, $s'$). The cost $CT_{RV}(A', G, s, s')$ of an RV algorithm $A'$ is defined only for non-symmetrical positions $s$, $s'$ and is the minimum time needed to rendezvous. The time complexity of the RVD algorithm $A$ is the maximum cost of $A$ over all pairs $(s, s')$, $T_{RVD}(A, G) = \max_{(s,s')} CT_{RVD}(A, G, s, s')$. The time complexity of the RV algorithm $A'$ is the maximum cost of $A'$ over all non-symmetrical pairs $(s, s')$, $T_{RV}(A', G) = \max_{(s,s')} CT_{RV}(A', G, s, s')$.

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. Specifically, we show the following. Two agents with

1. a constant number of unmovable tokens each cannot rendezvous in an $n \times n$ torus if they have $o(\log n)$ memory;
2. one movable token each cannot rendezvous in an $n \times n$ torus if they have $o(\log n)$ memory;
3. one unmovable token each can solve the rendezvous with detection problem in an $n \times m$ torus as long as they have $O(\log n + \log m)$ memory;
4. two movable tokens each and constant memory can solve the rendezvous (respectively, rendezvous with detection) problem in an arbitrary $n \times m$ torus (respectively, $n \times n$) torus;
5. three movable tokens each and constant memory can solve the rendezvous with detection problem 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 a torus.

1.3. Outline of the paper

In Section 2, we first give some preliminary results concerning the possible ways in which an agent can move in a torus using either no tokens or a constant number of unmovable tokens. Then we prove that the rendezvous without detection problem in an $n \times n$ 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 first give an algorithm for the rendezvous with detection problem in an arbitrary $n \times m$ torus where the agents use one unmovable token and $O(\log n + \log m)$ memory each. We then give an algorithm for the rendezvous with detection problem in an $n \times n$ torus where the agents use two movable tokens and constant memory each. Next we give an algorithm for the rendezvous without detection problem in an arbitrary $n \times m$ torus where the agents use two movable tokens and constant memory each. We prove that, when $m$ and $n$ have a specific relation, this algorithm solves the rendezvous with detection problem. Finally, we give an algorithm for the rendezvous with detection problem in an arbitrary $n \times m$ torus where the agents use three movable tokens and constant memory.

In Section 4, we discuss the results and state some open problems.
2. Memory lower bounds of rendezvous

In this section, we will prove the lower bounds for the memory the agents need in order to rendezvous. We first prove some technical lemmas which we will use in the main theorems.

2.1. Preliminary results

We prove here some lemmas about one mobile agent in an $n \times n$ oriented torus. In particular, we show how many nodes one agent can visit in the torus when it carries no tokens (Lemma 1), or carries one unmovable token (Lemma 2), or carries a constant number of identical unmovable tokens (Theorem 2).

**Lemma 1.** Consider one mobile agent with $\sigma \geq 2$ 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 $n > \sigma$, so that, no matter what the starting position of the agent is, it cannot visit all nodes of the torus. In fact, the agent will visit at most $n(\sigma - 1) + 1$ 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 $v = (v_x, v_y)$ be the node where the agent is located 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 $p_x$ and $p_y$ the horizontal and vertical distance, respectively, between $v$ and $v'$. Since $S$ is the first state repeated, the total number of nodes visited by the agent until it repeats $S$ for the first time is at most $\sigma + 1$. In particular, the total number of nodes visited by the agent after the moment that it first encountered $S$ and until it enters $S$ again (i.e., between visiting nodes $v$ and $v'$ without counting $v'$) is at most $\sigma - 1$ (notice that the initial state $S_0$ occurs only in the beginning).

Once the agent is again at state $S$, it has to repeat the same trajectory $(p_x, p_y)$ and visiting again at most $\sigma - 1$ new nodes until it encounters $S$ again. Label the coordinates of 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

$$(v_x + np_x) \mod n = v_x$$
$$n = v_y$$

This means that the agent is again at node $v$ and state $S$. The agent has to continue moving visiting exactly the same nodes. Up to that moment, the agent has visited at most $(\sigma + 1) + (n-1)(\sigma - 1) - 1 = n(\sigma - 1) + 1 < n^2$ nodes. \hfill $\Box$

Notice that in general it suffices to select an $n \times m$ torus with $n = ap_x$ and $m = ap_y$, where $a$ is such that $ap_x > \sigma$ and $ap_y > \sigma$. After $a$ repetitions of the first repeated state $S$, the agent will be again located at the same node, entering state $S$. Therefore it will visit at most $(\sigma + 1) + (a-1)(\sigma - 1) - 1 = a(\sigma - 1) + 1 < mn$ nodes.

**Lemma 2.** Consider one mobile agent with $\sigma \geq 2$ 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 $n > \sigma^2$, so that, no matter what the starting position of the agent is, it cannot visit all nodes of the torus. In fact, the agent will visit at most $\sigma + (\sigma - 1)^2(n+1) < n^2$ nodes.

**Proof.** We select an oriented $n \times n$ torus, where $n > \sigma^2$. As long as the agent does not release the token, Lemma 1 holds, and the agent visits at most $n(\sigma - 1) + 1 < n^2$ nodes.

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 $\sigma$ nodes. After releasing the token, say at node $v_1$, the agent moves without carrying any tokens. Take the first state $S$ which is repeated after dropping the token. Let $v_3$ be the node where the agent is located when $S$ is encountered for the first time after dropping the token, and let $v_3'$ be the node where the agent is located when $S$ is repeated for the first time. After the release of the token the agent has visited at most $\sigma - 1$ new nodes (notice that state $S_0$ occurs only in the beginning) until it repeats $S$ for the first time (at node $v_3'$). In any phase between two appearances of state $S$, the agent visits at most $\sigma - 1$ new nodes.

Suppose that, after at most $n$ repetitions of $S$, the agent does not meet its token. But then, following exactly the same reasoning as in the previous lemma, the agent will again be located at node $v_3$, having state $S$. Up to that point, it has visited at most $\sigma + (\sigma - 1)n - 1 < n^2$ nodes. After that point, the agent continues moving, visiting exactly the same nodes.

Suppose now that, at some point, the agent sees again its token at node $v_1$. Up to that point, it has visited at most $(\sigma - 1)(n+1)$ nodes. When it meets its token at $v_1$, it could change its orbit, visiting another $(\sigma - 1)(n+1)$ nodes. After at most $\sigma$ times visiting $v_1$, it has to repeat a state. In other words, it could enter at most $\sigma - 1$ different states (thus changing its orbit) when it meets its token. Therefore, it will visit a total of at most $\sigma + (\sigma - 1)^2(n+1)$ nodes, and after that it visits exactly the same nodes. Hence, if we select the size of the torus to be $n > \sigma^2$, then the agent will visit at most $\sigma + (\sigma - 1)^2(n+1) < n^2 + 2n(1-\sigma) + 1 - \sigma < n^2$ nodes. \hfill $\Box$

For the case of more than one unmovable tokens, we can apply again the arguments used in Lemmas 1 and 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 $\sigma$ nodes away from another token. Therefore we get the following.
Theorem 2. Consider one mobile agent with $\sigma$ 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 an $n \times n$ oriented torus, where $n > k\sigma^2$, so that, no matter what the starting position of the agent is, it cannot visit all nodes of the torus. In fact, the agent will visit at most $\sigma + k(\sigma - 1)^2(n + 1) < n^2$ nodes.

Proof. Following the same reasoning as in the previous lemma (Lemma 2), the agent should release the first token after visiting at most $\sigma$ nodes. Then it could visit at most $(\sigma - 1)^2(n + 1)$ new nodes, and release the second token at a distance at most $\sigma$ nodes away from the first token, and so on. After releasing the $k$-th token, it could visit at most $(\sigma - 1)^2(n + 1)$ new nodes before it repeats everything (passing from already visited nodes). Thus, it could visit at most $\sigma + k(\sigma - 1)^2(n + 1) < n^2$ nodes. \(\square\)

We will also need the following technical lemma.

Lemma 3. Let $A$ be an agent with $\sigma$ states and a constant number $k$ of identical unmovable tokens in an $n \times n$ oriented torus, where $n > k\sigma^2$, and let $v$ be a node in that torus. There are at most $\sigma + k(\sigma - 1)^2(n + 1) < n^2$ different starting nodes at which we could have initially placed $A$ so that node $v$ is always visited by $A$.

Proof. Fix a node $v$, and suppose that there are more than $\sigma + k(\sigma - 1)^2(n + 1)$ different starting nodes where $A$ could be initially placed and still visit $v$. Since the starting nodes are different, this means that the distances covered by $A$ to reach $v$ are pairwise different. But this means that $A$ can start from a node $s$ and visit nodes at more than $\sigma + k(\sigma - 1)^2(n + 1)$ different distances (i.e., different nodes), which in view of Theorem 2 is impossible. \(\square\)

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

We first show that two mobile agents with $\sigma$ states and one unmovable token each cannot rendezvous in an $n \times n$ oriented torus, when $n > 2\sigma^2$.

Lemma 4. Consider two mobile agents with $\sigma$ states and one unmovable token each. The tokens are identical. We can always (for any configuration of the automatons, i.e., states and transition function) place the agents in an $n \times n$ oriented torus, where $n > 2\sigma^2$, so that 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 for the first time 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. Since they move in exactly the same way, they see their tokens at the same time, being at the same state, and they continue moving identically. Thus they maintain their initial distance.

(b) Consider the case in which they repeat a state without having seen a token. Take the first state $S$ that they repeat. Suppose that, when first they are at state $S$, at that moment they are at nodes $v_1$ and $v_2$.

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 subcase (i), Lemma 1 holds, and hence they will eventually be located again at nodes $v_1$ and $v_2$, having state $S$. They continue moving identically, following the same routes as before, and therefore maintaining their initial distance for ever. Suppose now subcase (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 $\sigma + (\sigma - 1)^2(n + 1)$ nodes before it repeats everything. We prove that we can initially choose a node to place the other agent $B$ so that either one’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 $\sigma + (\sigma - 1)^2(n + 1)$ nodes visited by the first agent $A$,
- so as to avoid its visiting 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 $n^2 - (\sigma + (\sigma - 1)^2(n + 1))$ (taking $n > 2\sigma^2$) so that $B$’s token is out of reach of $A$. Moreover, by Lemma 3, only $\sigma + (\sigma - 1)^2(n + 1)$ starting nodes could lead agent $B$ to meet $A$’s token. Thus there are at least $n^2 - 2(\sigma + (\sigma - 1)^2(n + 1))$ starting nodes that satisfy the above property.

Therefore, if we choose the size of the torus $n > 2\sigma^2$, then we can place the agents so that if they meet a token it is always their token. Since until then they move identically, they always meet their token at the same time, being at the same state, and continue moving identically. Hence, they maintain their initial distance for ever. The situation has been illustrated in Fig. 2 (orbits of agent $A$ have been drawn with solid lines while orbits of agent $B$ have been drawn with dashed lines). \(\square\)
Fig. 2. Two agents with one unmovable token each cannot see each other’s token.

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 $s$ and $s'$ 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 $s$ and $s'$ satisfy property $\pi$.

If the agents could move the tokens, then it is easy to think of an algorithm in which all nodes of any torus are visited by the same agent. For example, consider the following algorithm for two agents with one movable token each:

1. release the token at the starting node;
2. go right, counting tokens until you meet the second token;
3. move the token down;
4. repeat from step 2.

Nevertheless, in the following scenario, in which the agents can move tokens, we again show that we can 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 and runs to the moment when they move their tokens again, they do not meet each other’s token.

**Lemma 5.** Consider two mobile agents with $\sigma$ states and one movable token each. The tokens are identical. We can always (for any configuration of the automaton, i.e., states and transition function) place the agents in an $n \times n$ oriented torus, where $n > 2\sigma^2$, so that they cannot rendezvous.

**Proof.** In view of Lemma 4, as long as $n > 2\sigma^2$, 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. Since the agents are identical and start at the same state, they will visit their own token and take the decision to move it simultaneously. Since, up to that point, they have maintained their initial distance, they will again place their tokens, maintaining the initial distance, and their new starting positions also have the same distance as before, and thus they continue to satisfy property $\pi$. Therefore, they will never meet each other. 

This implies the following theorem.

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

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

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

We end this section by showing that two mobile agents carrying a constant number of unmovable tokens cannot rendezvous in an $n \times n$ oriented torus unless they have $\Omega(\log n)$ memory.
Lemma 6. Consider two mobile agents with \( \sigma \) states and a constant number of \( k \) unmovable tokens each. All tokens are identical. We can always (for any configuration of the automaton, i.e., states and transition function) place the agents in an \( n \times n \) oriented torus, where \( n > 2k^2\sigma^2 \), so that they cannot rendezvous.

Proof. Take an \( n \times n \) oriented torus, where \( n > 2k^2\sigma^2 \), and place agent \( B \) at a starting node \( s(B) \). If agent \( B \) was alone in the torus it would release its tokens at nodes \( T_1(B), T_2(B), \ldots, T_k(B) \). According to Lemma 3, there are at least \( n^2 - (\sigma + k(\sigma - 1)^2(n + 1)) \) starting nodes at which we can place agent \( A \) so that \( A \) does not visit node \( T_1(B) \). Among these starting nodes (applying again Lemma 3) there are at most \( \sigma + k(\sigma - 1)^2(n + 1) \) nodes that would lead agent \( A \) to token \( T_2(B) \), another at most \( \sigma + k(\sigma - 1)^2(n + 1) \) nodes that would lead agent \( A \) to token \( T_3(B) \), and so on. Therefore, there are at least \( n^2 - 2k(\sigma + k(\sigma - 1)^2(n + 1)) \) starting nodes at which we can place agent \( A \) so that \( A \) does not visit any of the \( T_1(B), T_2(B) \), \ldots, \( T_k(B) \) nodes. We still need to place agent \( A \) at a starting node \( s(A) \) so that \( A \) releases its tokens at nodes \( T_1(A), T_2(A), \ldots, T_k(A) \) not visited by agent \( B \).

Notice that an agent can decide to release a new token at a distance at most \( \sigma \) from a previously released token (an agent cannot count more than \( \sigma \) before it repeats a state). Since \( n > 2k^2\sigma^2 \), for every two different starting nodes \( s(A) \) and \( s'(A) \), agent \( A \) would release its tokens to nodes \( T_1(A), T_2(A), \ldots, T_k(A) \) and \( T_1(A'), T_2(A'), \ldots, T_k(A') \), respectively, where \( T_i(A) \neq T_i'(A), 1 \leq i \leq k \).

Since, in view of Theorem 2, agent \( B \) can visit at most \( \sigma + k(\sigma - 1)^2(n + 1) \) nodes (if \( B \) was alone in the torus), there are at most \( \sigma + k(\sigma - 1)^2(n + 1) \) starting nodes for \( A \) for which \( A \) would place its first token at a node visited by agent \( B \), another at most \( \sigma + k(\sigma - 1)^2(n + 1) \) starting nodes for \( A \) for which \( A \) would place its second token at a node visited by agent \( B \), and so on. Hence, we need to exclude another \( k(\sigma + k(\sigma - 1)^2(n + 1)) \) starting nodes for agent \( A \). Thus, we are left with \( n^2 - 2k(\sigma + k(\sigma - 1)^2(n + 1)) > \sigma k^2n \) (when \( n > 2k^2\sigma^2 \)) starting nodes at which we can place agent \( A \) so that \( A \) does not visit any of the \( T_1(B), T_2(B) \), \ldots, \( T_k(B) \) nodes and agent \( B \) does not visit any of the \( T_1(A), T_2(A), \ldots, T_k(A) \) nodes. Hence, agents \( A \) and \( B \) may visit only their own tokens at the same time and being at the same states, and therefore they maintain their initial distance forever. □

This implies the following theorem.

Theorem 4. Two agents with a constant number of unmovable tokens need at least \( \Omega(\log n) \) memory to solve the RV problem in an \( n \times n \) oriented torus.

3. Upper bounds

In the previous section, we proved that solving the rendezvous without detection problem is impossible even in an \( n \times n \) oriented torus when the agents have a movable token or a constant number of unmovable tokens and \( o(\log n) \) memory each. These results imply the infeasibility of the RV problem (and of course the RVD problem) in an arbitrary \( n \times m \) oriented torus when the agents have one token and \( o(\log n - \log m) \) memory each.

In this section, we investigate and prove that \( O(\log n + \log m) \) memory is enough for agents equipped with one movable token each, in order to achieve a rendezvous with detection in an arbitrary \( n \times m \) oriented torus. Therefore both RV and RVD problems require \( \Theta(\log n + \log m) \) memory in an arbitrary \( n \times m \) oriented torus when the agents have one (movable or unmovable) token each.

We further investigate the situation when the agents have two movable tokens and constant memory each, and we show that in this case the RVD problem can be solved in an \( n \times n \) oriented torus and the RV problem can be solved in an arbitrary \( n \times m \) oriented torus.

Finally, we show that the RVD problem can be solved in an arbitrary \( n \times m \) oriented torus when the agents have three movable tokens and constant memory each.

3.1. Rendezvous with detection (RVD) in an \( n \times m \) torus using one unmovable token and \( O(\log n + \log m) \) memory

We describe an algorithm which solves the RVD problem of two agents equipped with one unmovable token and \( O(\log n + \log m) \) memory each in any \( n \times m \) oriented torus. We remind the reader that the agents do not know \( n \) and \( m \) but, as we will see, they can use their memory to calculate them. 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 counts steps until it meets a token twice. If its counters (measuring intertoken distances) differ, then a rendezvous can be arranged. Otherwise, it does the same in the initial vertical ring. If it does not meet the other agent then it searches the horizontal rings of the torus one by one, counting its steps. If it meets a token while going down passing from one horizontal ring to the other, then it declares that a rendezvous is impossible. Otherwise, if it meets a token while going right in a horizontal ring (which means that the agents must have started in different rings), then, if at least one of its counters counting horizontal or vertical distances from its token is different from \( n/2 \) or \( m/2 \), respectively, a rendezvous can be arranged. Otherwise, it stops and declares that a rendezvous is impossible.

As Algorithm 1 suggests, the agents first execute Procedure SameRing. If they do not meet each other, then either they must have started in symmetrical positions in the same ring or they must have started in different rings. In either
Algorithm 1 Algorithm for RVD in an $n \times m$ oriented torus with one unmovable token and $O(\log n + \log m)$ memory

1: SameRing
2: DifRing

Procedure SameRing
1: leave your token down
2: go right and count steps until you see a token
3: $c_1 \leftarrow$ this number of steps
4: go right and count steps until you see a token
5: $c_2 \leftarrow$ this number of steps
6: if $c_2 \neq c_1$ then
7: Rendezvous(horizontal, $c_1$, $c_2$)
8: else
9: go down and count steps until you see a token
10: $c_3 \leftarrow$ this number of steps
11: go down and count steps until you see a token
12: $c_4 \leftarrow$ this number of steps.
13: if $c_4 \neq c_3$ then
14: Rendezvous(vertical, $c_3$, $c_4$)
15: end if
16: end if

Procedure Rendezvous(ring, ct, ck)
1: if ring = horizontal then
2: if $ck > ct$ then
3: go right
4: else
5: go left
6: end if
7: end if
8: if ring = vertical then
9: if $ck > ct$ then
10: go down
11: else
12: go up
13: end if
14: end if

of those cases, they execute Procedure DifRing. Their exploration finishes after at most $O(nm)$ time, while they need $O(\log n + \log m)$ memory for counting.

Lemma 7. If the agents are located on the same ring of an $n \times m$ oriented torus in non-symmetrical positions, then Procedure SameRing will lead them to a rendezvous.

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 rendezvous. If $c_2 = c_1$, then the agents must have started in the same vertical ring. After $c_3 + c_4$ steps down they meet their token on the vertical ring with $c_3 \neq c_4$. They execute Procedure Rendezvous on the vertical ring, and rendezvous.

In view of Lemma 7, if, after executing Procedure SameRing, the agents do not meet each other, then either they have started in the same ring in symmetrical positions or they have started in different rings. In either such case, it must hold that $c_1 = c_2$ and $c_3 = c_4$.

Lemma 8. If the agents are located on the same ring on symmetrical positions or in different rings of an oriented $n \times m$ torus, then Procedure DifRing is a RVD algorithm.

Proof. The agents explore the other horizontal rings one by one, first going down and then at most $2c_1$ steps to the right. If they find a token while going down (passing from one horizontal ring to the next), then this token is either their token (which means that they have started in the same horizontal ring) or the other's token (which means that they have started in the same vertical ring). In either of these cases they declare that a rendezvous is impossible. If they find a token while
**Procedure DifRing**

1. repeat
2. godown to the next horizontal ring
3. repeat
4. go right
5. \(c_5 \leftarrow \) the number of steps right
6. until \((c_5 = 2c_1) \) OR (you meet a token)
7. until you meet a token
8. \(c_6 \leftarrow \) the number of rings down
9. if (you have met a token while going down) then
10. stop and declare rendezvous impossible
11. else
12. if \(c_6 \neq c_3/2\) then
13. Rendezvous2(\(c_6, c_3/2\))
14. else
15. if \(c_5 \neq c_1/2\) then
16. Rendezvous2(\(c_5, c_1/2\))
17. else
18. stop and declare rendezvous impossible
19. end if
20. end if
21. end if

**Procedure Rendezvous2(\(ct, ck\))**

1. if \(ct < ck\) then
2. reverse horizontal direction and go \(c_5\) horizontally and then vertically until you meet your token and wait
3. end if
4. if \(ct > ck\) then
5. wait
6. end if

An example has been illustrated in Fig. 3. An agent needs \(O(\log n + \log m)\) memory to execute Algorithm 1 since it needs to store only a constant number of counters with values up to \(n\) and \(m\). Algorithm 1 together with Lemmas 7 and 8 implies the following theorem.

**Theorem 5.** The rendezvous with detection problem on an oriented \(n \times m\) 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 in an oriented \(n \times m\) torus using two movable tokens and constant memory

In this subsection, we first give an algorithm which solves the RVD problem in any square anonymous, synchronous and oriented torus. In other words, in this case, the agents know that the torus has the same number of nodes horizontally and vertically, but they do not know this number. We then give an algorithm which solves the RV problem in any \(n \times m\) anonymous, synchronous and oriented torus.

Let us first define and analyze several procedures which will be used in our algorithms. We start with Procedure HorScan.

**Procedure HorScan**

1. repeat
2. go down, right, up
3. until you meet a token

In this procedure, 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.
Fig. 3. Two agents with $O(\log n + \log m)$ memory and one unmovable token each.

Fig. 4. An agent executing Procedure FindTokenHor.

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 move a token there
6: end if
7: until you meet a token down or right

An agent following Procedure FindTokenHor scans the horizontal rings of the torus one by one, as in Fig. 4, 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 both tokens at their starting positions and execute Procedure FindTokenHor. During the first execution of HorScan (step 2 of Procedure FindTokenHor), an agent has to meet a token for the first time, either after it moved down in the first step, or after it moved up or right at a later step (it cannot 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 while moving up, then it has met either one of its own tokens or one of the other's tokens. However, if it executes Procedure 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 at its starting node, and it meets it after it moved up. Furthermore, in this case the down horizontal ring had no tokens. In other words, if an agent meets a token after it moved up, it has met either one of its own tokens (which means that the agents did not start at the same horizontal ring) or one of the other's tokens (which means that the agents started at the same horizontal ring). Now the agent moves one of its tokens one step down (step 5 of Procedure FindTokenHor) and repeats from step 1. Notice that an agent never moves all tokens from its starting node and always moves a token along the vertical ring defined by its starting node. Let us call first ($T_1$) the token that stays at the starting node and second ($T_2$) the token that the agent moves.
Fig. 5. Two agents $A$ and $B$ starting at the same horizontal ring. (a) Each agent releases both its tokens at its starting node. (b) Each agent moves one of its tokens down. (c) Each agent repeatedly moves the same token $T_2$ down. (d) Eventually each agent moves its $(T_2)$ token one horizontal ring above the initial horizontal ring.

Suppose that at some point the agent exits the loop at step 7 by meeting a token while it goes down. This token is either its first token (which means that the agents have started in the same horizontal ring) or the other’s first token (which means that the agents have started in the same vertical ring). Hence, in both cases, meeting a token while going down means that the agents have started in the same ring.

If the agent exits the loop at step 7 by meeting a token while going right, then it is clear that it is the other’s first token and that the two agents have started in different horizontal and vertical rings.

Therefore the agent exits Procedure $\text{FindTokenHor}$ knowing that it has started either in the same ring with the other agent (if it exited the loop at step 7 meeting a token after it moved down) or in different horizontal and vertical rings (if it exited the loop at step 7 meeting a token after it moved right).

We illustrate the above situation by giving a few examples.

**Example 1.** Suppose that the agents start at the same horizontal ring and that they release their tokens at the same time, as in Fig. 5(a). When they execute Procedure $\text{HorScan}$ (at step 2 of Procedure $\text{FindTokenHor}$) for the first time, agent $A$ meets $B$’s tokens while moving up and agent $B$ meets $A$’s tokens while moving up (perhaps at different times). Next they execute Procedure $\text{HorScan}$ (at step 4 of Procedure $\text{FindTokenHor}$) again, and they meet their own tokens while moving up (at the same time since both agents covered the same distance). Then they move a token down (as in Fig. 5(b)) and repeat from step 1 of Procedure $\text{FindTokenHor}$. Now agent $A$ meets $B$’s second token $(T_2(B))$ while moving up and $B$ meets $A$’s second token $(T_2(A))$ while moving up (executing Procedure $\text{HorScan}$ at step 2 of Procedure $\text{FindTokenHor}$). Then they execute Procedure $\text{HorScan}$ at step 4 of Procedure $\text{FindTokenHor}$, and they meet their own second tokens while moving up at the same time. Next, they move their second tokens one step down, as in Fig. 5(c), and repeat from step 1. At some point they will move their second tokens as in Fig. 5(d) (one horizontal ring above the initial ring), and then they will move down (executing Procedure $\text{HorScan}$ at step 2 of Procedure $\text{FindTokenHor}$), meeting (at the same time) a token, and thus exiting the loop at step 7. Hence, they correctly decide that they have started at the same ring.
Fig. 6. Two agents A and B starting at the same vertical ring. (a) Each agent releases both its tokens at its starting node. (b) Eventually each agent places its \((T_2)\) token one horizontal ring above the other’s initial horizontal ring.

Fig. 7. Two agents A and B starting at different horizontal and vertical rings. (a) Each agent releases both its tokens at its starting node. (b) Eventually each agent places its \((T_2)\) token one horizontal ring above the other’s initial horizontal ring.

**Example 2.** Suppose that the agents start at the same vertical ring and that they release their tokens at the same time, as in Fig. 6(a). The agents will always meet (at the same time) a token going up while executing Procedure \texttt{HorScan} at step 2 or 4 of Procedure \texttt{FindTokenHor}, until they place (possibly not at the same time) their second token one horizontal ring above the other’s initial horizontal ring, as in Fig. 6(b). When they next execute Procedure \texttt{HorScan} at step 2 of Procedure \texttt{FindTokenHor}, they meet the other’s first token (possibly not at the same time) while moving \texttt{down}. Hence, they again correctly decide that they have started at the same ring.

**Example 3.** Suppose that the agents start (at the same time) at different horizontal and vertical rings and that they release their tokens as in Fig. 7(a). The agents will always meet (at the same time) a token going up while executing Procedure \texttt{HorScan} at step 2 or 4 of Procedure \texttt{FindTokenHor}, until they place (possibly not at the same time) their second token one horizontal ring above the other’s initial horizontal ring, as in Fig. 7(b). Now, while executing Procedure \texttt{HorScan} at step 2 of Procedure \texttt{FindTokenHor}, they will meet (possibly not at the same time) a token while going \texttt{right}, and they will exit the loop deciding that they have started in different rings, which is correct.

Notice that, in **Examples 2** and **3**, if the initial vertical distance between the agents is 1, then at least one of the agents (or both if the torus consists of only two horizontal rings) will meet the other’s first token immediately after it goes \texttt{down} (in **Example 2**) or \texttt{right} (in **Example 3**) while executing \texttt{HorScan} at step 2 of Procedure \texttt{FindTokenHor}. That is, the agent does not move any of its tokens. However, its decisions are again correct.

We also use Procedures \texttt{VerScan} and \texttt{FindTokenVer} by which the agents scan the vertical rings of the torus one by one, as in Fig. 8. Procedures \texttt{VerScan} and \texttt{FindTokenVer} have exactly the same properties as Procedures \texttt{HorScan} and \texttt{FindTokenHor}, respectively, if we replace direction \texttt{down} with \texttt{right}, right with \texttt{down}, and \texttt{up} with \texttt{left}. If the agents have a guarantee that they have started in different horizontal and vertical rings, then (similarly as before) by executing...
Procedure FindTokenVer they will exit the procedure, meeting (possibly not at the same time) a token while they move down (see Fig. 9). Both procedures FindTokenHor and FindTokenVer need $O(nm)$ time units.

**Procedure VerScan**
1: repeat
2: go right, down, left
3: until you meet a token

**Procedure FindTokenVer**
1: repeat
2: VerScan
3: if you meet token left then
4: VerScan
5: go one step right and move a token there
6: end if
7: until you meet a token down

We also use Procedure RVDRing, which solves the rendezvous with detection problem in an oriented ring for two agents having two tokens and constant memory each. Suppose that the agents release their tokens at the same time and that they start, possibly having a delay as explained below, executing Procedure RVDRing. Then they solve the rendezvous with detection problem (see Fig. 10).

**Procedure RVDRing(direction)**
1: move a token one step to the direction
2: while you are at a node with less than 2 tokens do
3: move to the direction until you meet the fourth token
4: move this token one step to the direction
5: end while
6: go to the direction until you meet a node $v$ with a token
7: if there are 2 tokens at $v$ then
8: return
9: else
10: wait there
11: end if

**Lemma 9.** Consider two mobile agents with constant memory on an oriented ring consisting of $n$ nodes. The agents have two tokens each (identical to each other). Let $d_s \leq n/2$ be the distance between $A$ and $B$ and let $B$ be placed $d_s$ steps to the right of $A$. The agents release their tokens at the same time, and they start executing Procedure RVDRing(right), but possibly $B$ starts with a delay. Then they solve the rendezvous with detection problem in $O(n^2)$ time.

**Proof.** Since there are always four tokens in the ring, each agent always meets (at step 3 of the procedure) the same token. Let us call this token $T_2$ or second. Each agent (say $A$) repeatedly moves the same token $T_2(A)$ until it hits the other’s token.
Fig. 9. Two agents $A$ and $B$ starting at different horizontal and vertical rings. (a) Each agent releases both its tokens at its starting node. (b) Each agent moves one of its tokens right. (c) Eventually each agent moves its $(T_1)$ token one vertical ring left of the other's initial vertical ring.

Fig. 10. In a ring: two agents with constant memory and two movable tokens each solve the rendezvous with detection problem.

$T_1(B)$, which is placed at $B$'s starting node. An agent $Z$ never moves its own or the other's $T_1(Z)$ token. Consider the moment at which an agent $A$, after moving to the right token $T_2(A)$, touches another token (which should be token $T_1(B)$). The total
distance covered by agent $A$ is $1 + (d_x - 1)(n + 1)$. The other agent $B$ has covered at most the same distance, hence moving token $T_2(B)$ at most $d_x$ steps to the right. There are two cases.

(i) Suppose that the initial distance between the agents is $d_x = n/2$ and that they start at the same time. Hence agent $A$ has traveled a total distance of $1 + (n/2 - 1)(n + 1)$ until it touches token $T_1(B)$ at time $\tau$. Since they have traveled for the same time $\tau$, agent $B$ moves its second token $T_2(B)$, having also covered a total distance of $1 + (n/2 - 1)(n + 1)$. But then $B$ should touch token $T_1(A)$ at $\tau$. Now both agents discover this at an additional $n/2$ time, according to the procedure.

(ii) Suppose that the initial distance between the agents is $d_x < n/2$. Hence when agent $A$ touches token $T_1(B)$, having covered a total distance of $1 + (d_x - 1)(n + 1)$, agent $B$ should not be touching $T_1(A)$ (as this is further than $d_x$ steps away from $T_1(B)$). In fact, $B$ needs at least $n + 1$ time units more (at least one round) to touch token $T_1(A)$. Now, according to the procedure, agent $A$ travels for another at most $n - d_x$ time units to meet token $T_2(B)$, and waits for agent $B$, which will eventually come. The situation has been depicted in Fig. 10. Procedure RVDRing takes $O(n^2)$ time. □

Combining those procedures, we now give the main Procedure SearchTorus that will be used in our algorithms. A high-level description of Procedure SearchTorus is the following.

The two agents search the horizontal rings of the torus one by one (using Procedure FindTokenHor) to discover whether they have started in the same ring. If so, then they execute Procedure RVDRing(down) and then Procedure RVDRing(right). 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 the vertical rings of the torus one by one (using Procedure FindTokenVer). They again try to ‘catch’ each other on the torus.

**Procedure SearchTorus**

1: release both tokens
2: FindTokenHor
3: if you meet a token down then
4: (* Same Ring *)
5: go one step up
6: if you see only one token then
7: pick-up token
8: go up until you meet a token
9: release token
10: end if
11: (* RVDRing on the vertical ring with direction down *)
12: RVDRing(down)
13: (* RVDRing on the horizontal ring with direction right *)
14: RVDRing(right)
15: if not rendezvous then
16: stop and declare rendezvous impossible
17: end if
18: else
19: (* Different Ring *)
20: go up until you meet a token
21: go one step down
22: repeat
23: go left, wait 1 time unit
24: until (rendezvous) OR (you meet a token for the second time)
25: if not rendezvous then
26: Synchronize
27: release token
28: FindTokenVer
29: go left until you meet a token
30: go one step right
31: repeat
32: go up, wait 1 time unit
33: until (rendezvous) OR (you meet a token for the second time)
34: end if
35: end if

Let the agents execute Procedure SearchTorus. They release their tokens at their starting positions and follow Procedure FindTokenHor. As argued before, they will exit Procedure FindTokenHor either by finding a token while going down (which means that they have started in the same horizontal or vertical ring) or by finding a token while going right (which means that they have started in different horizontal and vertical rings).
Procedure Synchronize
1: go up
2: repeat
3: wait 1 time unit, go left
4: until you meet a token
5: pick up token
6: if you do not see tokens then (* it means that you were distance > 1 down *)
7: go up until you meet token
8: end if
9: release token
10: FindTokenHor
11: go up
12: go right until you meet token
13: pick up token
14: go down
15: go right until you meet token

Case 1.
Suppose that they exit Procedure FindTokenHor by finding a token while going down.

- Case 1.1: Suppose that they had started in the same horizontal ring and, w.l.o.g., let $d_A \leq n/2$ be the distance to the right between $A$ and $B$. The agents move as explained in Example 1 and Fig. 5. While executing Procedure FindTokenHor, agent $A$ has moved as follows. After $6n$ time units it meets its own tokens for the second time, and then goes one step down. This is repeated $m - 1$ times (until it is one horizontal ring above its starting node). It takes it another $1 + m$ time units to pick up its second token and move back to its starting node. Agent $B$ travels for exactly the same time. Hence, both agents reach their starting points at the same time. They start executing Procedure RVDRing(down) in their vertical rings at the same time, and (since they are alone in their vertical rings) they again finish Procedure RVDRing(down), reaching their starting points at the same time.\(^2\) Then they execute Procedure RVDRing(right) in the horizontal ring. After that, in view of Lemma 9, either they rendezvous (if their initial distance was less than $n/2$) or stop, declaring a rendezvous impossible (if their initial distance was $n/2$), which in view of Theorem 1 is correct.

- Case 1.2: Suppose that they had started in the same vertical ring and, w.l.o.g., let $d_y \leq m/2$ be the distance downwards between $A$ and $B$. The agents move as explained in Example 2 and Fig. 6. While executing Procedure FindTokenHor, agent $A$ has moved as follows. After $6n$ time units it meets its own token for the second time, and then goes one step down. This is repeated $m - 1$ times (until it is one horizontal ring above the ring where $B$'s first token lies). It takes it another $1 + m$ time units to pick up its second token and move back to its starting node. Agent $A$ has traveled a total distance of $(6n + 1)(d_A - 1) + 1 + d_y$. Agent $B$ travels for $6n$ time units until it meets its own token for the second time, and then goes one step down. This is repeated $m - d_y - 1$ times (until it is one horizontal ring above the ring where $A$'s first token lies). It takes it another $1 + m - d_y$ time units to pick up its second token and move back to its starting node. Agent $B$ has traveled a total distance of $(6n + 1)(m - d_y - 1) + 1 + m - d_y$. Since $d_y \leq m/2$, either agent $A$ has reached its starting point first or both agents arrive at their starting nodes at the same time. The agents start executing Procedure RVDRing(down) in their vertical rings, with $B$'s starting possibly delayed. Suppose that, when agent $A$ reaches its starting point, agent $B$ is still executing Procedure FindTokenHor. That is, $B$ will be still moving its second token $T_2(B)$ on the same vertical ring, approaching token $T_1(A)$ from above. As long as $B$ has not yet completed Procedure FindTokenHor and $A$ executes Procedure RVDRing(down), there are always four tokens on this vertical ring, and agent $A$ always moves token $T_2(A)$ down from $T_1(A)$ to $T_1(B)$. If $A$ exits the loop at step 5 of Procedure RVDRing(down) before agent $B$ finishes Procedure FindTokenHor, then $A$ (executing step 6 of Procedure RVDRing(down)) will reach either $T_2(B)$ or $T_1(A)$. After that, in view of Lemma 9, they will rendezvous if and only if their initial distance was less than $m/2$. If they do not rendezvous, the agents execute Procedure RVDRing(right) in their horizontal rings\(^3\) and they stop, declaring a rendezvous impossible, which in view of Theorem 1 is correct.

Case 2.
Suppose that, while the agents execute Procedure FindTokenHor at step 2 of Procedure SearchTorus, one of them is the first to find a token while moving right (as explained in Example 3 and Fig. 7). Then this agent knows that they have started in different horizontal and vertical rings.

Let $d_y$ be the shortest initial distance between the two agents. It holds that $m - 2d_y \geq 0$.

\(^2\) Notice that in this case there are two tokens in each vertical ring and hence the agents will exit Procedure RVDRing(down) at step 8.

\(^3\) Notice that in this case there are two tokens in each horizontal ring, and hence the agents will exit Procedure RVDRing(right) at step 8.
Consider agent $A$ executing Procedure SearchTorus (Fig. 11(a)). While executing Procedure FindTokenHor at step 2 of Procedure SearchTorus, it meets its own token (going up) for the first time after $3n$ time units. It takes it another $3n$ time units to meet its token (node 1 in Fig. 11(a)) for the second time, plus one time unit for going down. This is repeated $d_y - 1$ times (until it is one ring above the ring where $B$'s first token lies) at node 2 in Fig. 11(a). Agent $A$ leaves its $T_2(A)$ token there and then it takes it another $3(d_x - 1) + 2$ time units to meet $B$'s first token (node 3 in Fig. 11(a)). So far, it has spent the following time units.

$$\tau(A)_{(1 \rightarrow 3)} = (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2. \quad (1)$$

Meanwhile, agent $B$ executes Procedure SearchTorus, starting at node 3 (Fig. 11(b)) and reaches node 4 and then node 1 (by executing Procedure FindTokenHor), spending a total time of

$$\tau(B)_{(3 \rightarrow 1)} = (1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2. \quad (2)$$

From (1) and (2), the time difference $\tau(A)_{(1 \rightarrow 3)} - \tau(B)_{(3 \rightarrow 1)}$ is

$$\tau(A)_{(1 \rightarrow 3)} - \tau(B)_{(3 \rightarrow 1)} = (1 + 6n)(2d_y - m) + 3(2d_x - n). \quad (3)$$

Since $2d_y - m \leq -1$ and $d_x \leq n - 1$, we get

$$\tau(A)_{(1 \rightarrow 3)} - \tau(B)_{(3 \rightarrow 1)} \leq (1 + 6n)(-1) + 3(2n - 1) = -7 - 3n < 0.$$ 

Hence, agent $A$ reaches node 3 before $B$ reaches node 1. Therefore, agent $A$ meets first $B$'s first token and knows that they have started in different horizontal and vertical rings.

Then agent $A$ goes up until it meets a token (step 20 of Procedure SearchTorus). Let us calculate the time spent by agent $A$ to reach node 4 (which is on the vertical ring defined by node 3 and one horizontal ring above node 1).

$$\tau(A)_{(1 \rightarrow 4)} = (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + d_y. \quad (4)$$

We first show that token $T_2(B)$ is either at node 4 or upwards of node 4 in that vertical ring and downwards of node 3 or at node 3. Agent $B$ moves its second token $T_2(B)$ downwards in the vertical ring where its first token lies (at node 3) until node 4. Suppose that $B$ places token $T_2(B)$ at node 4 before $A$ gets there. Then $B$ should find token $T_1(A)$ at node 1 and should go upwards until it meets $A$'s second token $T_2(A)$, which has been placed by $A$ at node 2 (or node 1 if $d_y = 1$). Hence, agent $B$, after reaching node 1, goes up for at least $m - d_y + 1$ time units (step 20 of Procedure SearchTorus) until it meets the other's second token (node 2 in Fig. 11). So far it has spent the following time units:

$$\tau(B)_{(3 \rightarrow 2)} = (1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2 + m - d_y + 1. \quad (5)$$

From (4) and (5), the time difference $\tau(A)_{(1 \rightarrow 4)} - \tau(B)_{(3 \rightarrow 2)}$ is

$$\tau(A)_{(1 \rightarrow 4)} - \tau(B)_{(3 \rightarrow 2)} = (1 + 6n)(2d_y - m) + 3(2d_x - n) + 2d_y - m.$$ 

Fig. 11. (a) Agent $A$ executing Procedure SearchTorus until step 24. (b) Agent $B$ executing Procedure SearchTorus until step 24.
Since \( 2d_y - m \leq -1 \) and \( d_x \leq n - 1 \), we get
\[
\tau(A)_{(1 \rightarrow 4)} - \tau(B)_{(3 \rightarrow 2)} \leq (1 + 6n)(-1) + 3(2(n - 1) - n) - 1 < 0.
\]
Hence, agent A reaches node 4 before B reaches node 2. Therefore, token \( T_2(B) \) is either at node 4 or at a node up of node 4 and down of node 3, or at node 3 in the same vertical ring.

Suppose that token \( T_2(B) \) is in a horizontal ring not adjacent to the horizontal ring in which \( T_1(A) \) lies (\( T_2(B) \) could be still at node 3). This means that the other agent B is still searching that ring coming from left to right. Therefore they will rendezvous, since A, after meeting token \( T_2(B) \), goes one step down and then performs the left–wait–1-time-unit operation repeatedly. Hence, the only remaining case is that token \( T_2(B) \) is just one horizontal ring above the ring in which token \( T_1(A) \) lies, as shown in Fig. 11. Hence, agent A has moved up for \( d_y + 1 \) time units (step 20 of Procedure SearchTorus) until it meets the other's second token (node 4 in Fig. 11(a)). Then agent A goes one step down and spends another \( 2(d_x - 1) + 1 \) time units (steps 22–24) until it meets its own first token \( T_1(A) \) for the first time. Thus agent A went back to node 1 having spent a total time of
\[
\tau(A)_{(1 \rightarrow 1)} = (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + d_y + 2 + 2d_x - 1 + 1.
\]

Suppose that the agents do not rendezvous until A meets token \( T_1(A) \) again (and hence exiting the loop at step 24) at an additional \( 2n \) time units. Notice that this means that agent B has reached node 1 before agent A (for the first time). Therefore, from (7) and (2), it should hold that
\[
\tau(A)_{(1 \rightarrow 1)} - \tau(B)_{(3 \rightarrow 1)} > 0 \rightarrow (1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 > 0.
\]

Meanwhile, agent B, after reaching node 2, goes one step down and spends another \( 2(n - d_y - 1) + 1 \) time units (steps 22–24) until it meets its own first token \( T_1(A) \) for the first time. Thus agent B went back to node 3 having spent a total time of
\[
\tau(B)_{(3 \rightarrow 3)} = (1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2 + m - d_y + 2 + 2(n - d_x - 1) + 1.
\]

From (7) and (9), the time difference \( \tau(A)_{(1 \rightarrow 1)} - \tau(B)_{(3 \rightarrow 3)} \) is
\[
\tau(A)_{(1 \rightarrow 1)} - \tau(B)_{(3 \rightarrow 3)} = (1 + 6n)(2d_y - m) + 3(2d_x - n) + 2d_y - m + 2(2d_y - n).
\]

Since \( 2d_y - m \leq -1 \) and \( d_x \leq n - 1 \), we get
\[
\tau(A)_{(1 \rightarrow 1)} - \tau(B)_{(3 \rightarrow 3)} \leq (1 + 6n)(-1) + 3(2(n - 1) - n) - 1 + 2(2(n - 1) - n) = -n < 0.
\]

Hence, agent A reaches node 1 before B reaches node 3.

- Case 2.2: \( m - 2d_y = 0 \).

Let agent B be initially at distance \( d_y \) down from agent A and \( d_x \) to the right, where \( n - 2d_x \geq 0 \).

Consider agent A executing Procedure SearchTorus. From (1), it takes it \( \tau(A)_{(1 \rightarrow 3)} = (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 \) to get to node 3. Meanwhile, from (2) and (3), agent B either reaches node 1 at the same time or has not yet reached node 1. Therefore, agent A meets first B's first token and knows that they have started in different horizontal and vertical rings.

Then agent A goes up until it meets a token (step 20 of Procedure SearchTorus). From (6), we have \( \tau(A)_{(1 \rightarrow 4)} = \tau(B)_{(3 \rightarrow 2)} \leq 0 \). Hence, agent A reaches node 4 before or at the same time as B reaches node 2. Following the same argument as before, we conclude that token \( T_2(B) \) is either at node 4 or at a node up of node 4 and down of node 3, or at node 3 in the same vertical ring. In the last case, as argued before, the agents will meet. Suppose that token \( T_2(B) \) is just one horizontal ring above the ring in which token \( T_1(A) \) lies, as shown in Fig. 11. Suppose that the agents do not rendezvous until A meets token \( T_1(A) \) again (and hence exiting the loop at step 24) at an additional \( 2n \) time units. Notice that this means that agent B has reached node 1 before agent A (for the first time). Therefore, (8) should hold.

Finally, from (7), (9), and (10), we have that
\[
\tau(A)_{(1 \rightarrow 1)} - \tau(B)_{(3 \rightarrow 3)} \leq 0,
\]
meaning that agent A reaches node 1 before or at the same time as B reaches node 3.

Therefore, Procedure SearchTorus correctly instructs, in both subcases above, agent A to go back to node 1, exactly as in Fig. 11(a). Agent B either reaches node 3 at the same time or has not yet reached node 3.

Agent A has traveled for a total number of \( (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + d_y + 2 + 2(d_x - 1) + 1 + 2n \) time units until entering step 25 of Procedure SearchTorus, while agent B traveled a total of \( (1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2 \) time units to reach node 1, \( m - d_y + 1 \) to reach node 2 and another \( 1 + 2(n - d_x - 1) + 1 + 2n \) to reach node 3 and exit the loop at step 24. Procedure Synchronize (Fig. 12(a)) forces agent A to travel another \( 1 + 2(n - d_x) \) time units to reach node 4 (steps 1–4 of Procedure Synchronize). Agent A picks up token \( T_2(B) \) at node 4 and spends another \( (m - d_y) \) time units (unless token \( T_2(B) \) was at node 3 together with token \( T_1(B) \), which means that \( m - d_y = 1 \) to reach node 3 (steps 5–9 of Procedure Synchronize). Now, agent A releases the token at node 3 and executes Procedure FindTokenHor (at step 11), and after \( (1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2 \) time units it reaches node 1, having placed the carried token at node 4. Finally, after \( 2 + n \) time units (steps 12–16) it reaches node 1 carrying a token. Similarly, agent B (Fig. 12(b)) travels for \( 1 + 2d_y \) time units (steps 1–4) to reach node 2, picks up token \( T_2(A) \) at node 2 and then travels another \( (d_x - 1) \) time units (unless token \( T_2(A) \) was at node 1 together with token \( T_1(A) \), which means that \( d_x = 1 \) to reach node 1 (steps 5–9). Then it releases the token at node 1 and executes Procedure FindTokenHor (at step 11), and after \( (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 \) time units reaches node 3, having placed the carried token at node 2. Finally, agent B reaches node 3 after an additional \( 2 + n \).
time (steps 12–16), carrying a token. Notice that, for the whole Procedure $\text{Synchronize}$, the agents do not interfere with each other: agent $A$ only moves in the area downwards of node 3 and upwards of node 1, while agent $B$ only moves in the area downwards of node 1 and upwards of node 3.

The total time spent by agent $A$ is

$$(1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + d_x + 2 + 2(d_x - 1) + 1 + 2n + 1 + 2(n - d_x) + (m - d_y - 1) + (1 + 6n)$$

$$(m - d_y - 1) + 3(n - d_x - 1) + 2 + 2 + n = (1 + 6n)(m - 2) + 3(n - 2) + m - 1 + 2(n - 1) + 3n + 10.$$

The total time spent by agent $B$ is

$$(1 + 6n)(m - d_y - 1) + 3(n - d_x - 1) + 2 + m - d_y + 2 + 2(n - d_x - 1) + 1 + 2n + 1 + 2d_x + (d_y - 1) + (1 + 6n)(d_y - 1) + 3(d_x - 1) + 2 + 2 + n = (1 + 6n)(m - 2) + 3(n - 2) + m - 1 + 2(n - 1) + 3n + 10.$$ 

Hence, the agents reach their initial nodes at the same time carrying a token.

Next (going back to step 27 of Procedure $\text{SearchTorus}$), they release their tokens at their initial nodes and execute Procedure $\text{FindTokenVer}$ (the configuration will be like the one shown in Fig. 9). Finally, they move following steps 29–33 in a similar way as in steps 20–24 (just rotated by 90°).

### 3.2.1. Rendezvous with detection in an oriented $n \times n$ torus using two movable tokens and constant memory

**Theorem 6.** The rendezvous with detection problem on an $n \times n$ oriented torus can be solved by two agents using two movable tokens and constant memory each, in time $O(n^2)$.

**Algorithm 2** $\text{RVD2n}$

1. $\text{SearchTorus}$
2. if not rendezvous then
3. stop and declare rendezvous impossible
4. end if

**Proof.** The agents execute Algorithm $\text{RVD2n}$. If the agents started in the same horizontal or vertical ring, they solve the rendezvous with detection problem as explained above (Case 1). Suppose that the agents started in different horizontal and vertical rings (see Fig. 7). Let, w.l.o.g., agent $B$ be $d_y$ down from agent $A$ and $d_x$ to the right, where $m - 2d_y \geq 0$.

Suppose that $m - 2d_y > 0$. If the agents do not rendezvous until $A$ meets token $T_1(A)$ for the second time, then this means that agent $B$ had reached token $T_1(A)$ before agent $A$. Thus (8) should hold:

$$(1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 > 0.$$ 

However,

$$(1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 
\leq (1 + 6n)(-1) + 3(2(n - 1) - n) + 2(n - 1 - 1) + d_y + 3 = d_y - n - 8.$$
By taking \( n = m \), we have \( d_y - n - 8 = d_y - m - 8 < 0 \). Therefore, (8) does not hold, which means that \( m = 2d_y \). Observe that, if \( m = n \) is an odd number, then they will always rendezvous.

Now they get synchronized, as explained above, and they execute Procedure \texttt{FindTokenVer} (see Fig. 9). Let, w.l.o.g., agent \( B \) be \( d_y = d_y \) down from agent \( A \) and \( d_y' \) to the right, where \( n - 2d_y' \geq 0 \).

Suppose that \( n - 2d_y > 0 \). If they do not rendezvous until \( A \) meets token \( T_1(A) \) for the second time, then this means that agent \( B \) has reached token \( T_1(A) \) before agent \( A \). Therefore (8) should hold (replacing \( d_y \) with \( d_y' \), \( d_y \) with \( d_y' \) and interchanging \( m \) with \( n \)):

\[
(1 + 6m)(2d_y' - n) + 3(2d_y' - m) + 2(d_y' - 1) + d_y' + 3 > 0.
\]

However,

\[
(1 + 6m)(2d_y' - n) + 3(2d_y' - m) + 2(d_y' - 1) + d_y' + 3 \leq (1 + 6m)(-1) + 2(d_y' - 1) + d_y' + 3 \leq d_y' - 5m.
\]

By taking \( n = m \), we have \( d_y' - 5m = d_y' - 5n < 0 \). Therefore, (8) does not hold, which means that \( n = 2d_y' \). Hence, in view of Theorem 1, they can safely decide that a rendezvous is impossible after \( O(n^2) \) total time. 

3.2.2. Rendezvous without detection in an \( n \times m \) oriented torus using two movable tokens and constant memory

We now give an RV algorithm for two agents with constant memory in an arbitrary \( n \times m \) anonymous, synchronous and oriented torus. We remind the reader that the agents do not know \( n \) or \( m \).

The agents first execute Procedure \texttt{SearchTorus}. If no rendezvous occurs and no decision is made about its impossibility\(^4\) (i.e., the agents have started in different rings), then the agents mark a rectangle with their tokens on the torus by executing Procedure \texttt{BuildRectangle}. Then they execute Procedure \texttt{Chase}: they try to catch each other on the previously built rectangle, which will eventually happen unless they had started at distance \( (n/2, m/2) \) (in that case the algorithm runs forever). We give Procedure \texttt{Chase} below.

**Procedure Chase**

1: repeat
2: repeat
3: go down
4: until you meet the third token
5: repeat
6: go right
7: until you meet the third token
8: repeat
9: go up
10: until you meet the third token
11: repeat
12: go left
13: until you meet the third token
14: until rendezvous

We first prove that, if the configuration is as in Fig. 13 and exactly one of the following holds: either the horizontal distance between the agents is \( n/2 \) or the vertical distance between the agents is \( m/2 \), then Procedure \texttt{Chase} leads the agents to meet.

**Lemma 10.** Suppose that there is a rectangle marked by four tokens in an \( n \times m \) torus. Suppose also that exactly one of the following holds: 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 (nodes 1 and 3, respectively, in Fig. 13). The agents start, possibly not at the same time, executing Procedure \texttt{Chase}. Then they will rendezvous in \( O(n^2 + m^2) \) time.

**Proof.** Agent \( A \) starts at node 1 (depicted in Fig. 13(a)), and by following the procedure (steps 2–4) reaches node 2 after \( m + d_y \) time units. Then, following steps 5–7 of the procedure, it reaches node 3 after \( n + d_y \) time units. After steps 8–10, it reaches node 4 after another \( m + d_y \) time units. Finally, after steps 11–13, it reaches node 1 after another \( n + d_y \) time units. Agent \( A \) completes a full round going back to node 1 in a total number of \( 2(n + d_y) + 2(m + d_y) \) time units.

Similarly, agent \( B \) starts at node 3 (depicted in Fig. 13(b)), and by following the procedure (steps 2–4) reaches node 4 after \( 2m - d_y \) time units. Then, following steps 5–7 of the procedure, it reaches node 1 after \( 2n - d_y \) time units. After steps 8–10, it reaches node 2 after another \( 2m - d_y \) time units. Finally, after steps 11–13, it reaches node 3 after another \( 2n - d_y \) time units. Agent \( B \) completes a full round going back to node 3 in a total number of \( 2(2n - d_y) + 2(2m - d_y) \) time units.

\(^4\) Notice that if the agents have started in the same (horizontal or vertical) ring then they solve the RVD problem.
We show that the agents maintain their order at which they arrive at landmark nodes, and if \( d_x = n/2 \) or \( d_y = m/2 \) (but not both) they move closer and closer to each other. Suppose, w.l.o.g., that either \( d_x = n/2 \) or \( d_y = m/2 \) (but not both). As illustrated above, agent \( A \) starts each round at node 1, and consecutively reaches node 2, 3, 4, and again 1 when it exits the loop at step 4, 7, 10, and 13, respectively. Similarly, agent \( B \) starts each round at node 3, and consecutively reaches node 4, 1, 2, and again 3 when it exits the loop at step 4, 7, 10, and 13, respectively. Let agent \( A \) be the one that has to cover less distance than \( B \) to complete its round. Hence, \( A \) approaches any given landmark node more quickly in each round. Eventually, there will be a moment at which agent \( A \) will reach a node \( k \in \{1, 2, 3, 4\} \) while agent \( B \) will be on its way to \( (k + 1) \) modulo 4 having already passed from node \( k \). If these two nodes belong to the same horizontal ring and \( d_x = n/2 \), then their order does not change, since they have to cover the same distance from \( k \) to \( (k + 1) \) modulo 4. If these two nodes belong to the same vertical ring and \( d_y = m/2 \), then their order does not change, since they have to cover the same distance from \( k \) to \( (k + 1) \) modulo 4. We claim that agent \( A \) cannot reach node \( (k + 1) \) modulo 4 first in any of the remaining cases without meeting \( B \). Let us see why.

- **Case 1**: Nodes \( k, (k + 1) \) modulo 4 are in the same vertical ring and \( d_y < m/2 \).
  - **Case 1.1**: \( k = 1 \). Agent \( B \) is on its way to node 2, following steps 8–10, that is, traversing once the vertical ring which contains node 1, approaching node 2 from below. Agent \( A \) starts to follow steps 2–4, that is, traversing once the vertical ring which contains node 1, but having the opposite direction to agent \( B \), and approaching node 2 from above. Hence, agent \( A \) cannot reach node 2 before \( B \) without meeting \( B \).
  - **Case 1.2**: \( k = 3 \). Agent \( B \) is on its way to node 4, following steps 2–4, that is, traversing once the vertical ring which contains node 3, approaching node 4 from above. Agent \( A \) starts to follow steps 8–10, that is, traversing once the vertical ring which contains node 3, but having the opposite direction to agent \( B \), and approaching node 4 from below. Hence, agent \( A \) cannot reach node 4 before \( B \) without meeting \( B \).

- **Case 2**: Nodes \( k, (k + 1) \) modulo 4 are in the same horizontal ring and \( d_x < n/2 \).
  - **Case 2.1**: \( k = 2 \). Agent \( B \) is on its way to node 3, following steps 11–13, that is, traversing once the horizontal ring which contains node 2, approaching node 3 from the right. Agent \( A \) starts to follow steps 5–7, that is, traversing once the horizontal ring which contains node 2, but having the opposite direction to agent \( B \), and approaching node 3 from the left. Hence, agent \( A \) cannot reach node 3 before \( B \) without meeting \( B \).
  - **Case 2.2**: \( k = 4 \). Agent \( B \) is on its way to node 1, following steps 5–7, that is, traversing once the horizontal ring which contains node 4, approaching node 1 from the left. Agent \( A \) starts to follow steps 11–13, that is, traversing once the horizontal ring which contains node 4, but having the opposite direction to agent \( B \), and approaching node 1 from the right. Hence, agent \( A \) cannot reach node 1 before \( B \) without meeting \( B \).

Since agent \( A \) comes closer and closer to agent \( B \) and their order cannot change without \( A \) meeting \( B \), they will eventually meet after \( O(n + m) \) rounds. Hence, they will meet after \( O(n^2 + m^2) \) total time. \( \Box \)

The following algorithm (RV2mn) is an RV algorithm for two agents having constant memory in an \( n \times m \) oriented torus. In fact, one of the following things could happen: 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 \( m/2 \)).

**Theorem 7.** The rendezvous without detection problem on an arbitrary \( n \times m \) oriented torus can be solved by two agents using two movable tokens and constant memory each, in time \( O(n^2 + m^2) \).

**Proof.** Let (w.l.o.g.) agent \( B \) be located initially a distance \( d_x \) down from agent \( A \), where \( m - 2d_x \geq 0 \), and \( d_y \) to the right. The agents follow Algorithm RV2mn. They first execute Procedure SearchTorus. If they have started on the same horizontal
or vertical ring, then they solve the rendezvous with detection problem, as explained above. Suppose that they have started on different horizontal and vertical rings (Fig. 7). We first prove that, if they do not rendezvous after horizontal and vertical scanning, then either $n - 2d_x = 0$ or $m - 2d_y = 0$ (or both).

Suppose that (as described above) the agents exit the loop at step 24 without rendezvous and that $m - 2d_y > 0$. Thus (8) should hold:

$$
(1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 > 0.
$$

Since $m - 2d_y > 0$ and $d_x < n$, we have

$$
(1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 \\
\leq (1 + 6n)(-1) + 3(2(n - 1) - n) + 2(n - 1 - 1) + d_y + 3 = d_y - n - 8.
$$

Hence it should hold that

$$
d_y - n - 8 > 0. \tag{11}
$$

The agents execute Procedure Synchronize and reach their initial nodes at the same time, carrying a token, as explained above. Then they release the tokens at their initial nodes and execute Procedure FindTokenVer (step 28 of Procedure SearchTorus), as in Fig. 9. Suppose that they do not rendezvous. Consider the agent whose horizontal movement (before executing Procedure FindTokenVer) until it reaches the other's agent ring is $d'_x$, where $n - 2d'_x \geq 0$.

Suppose that (as described above) the agents exit the loop at step 33 without rendezvous, and that $n - 2d'_x > 0$. Then, from (8), by interchanging $n$ with $m$ and substituting $d_x$ with $d'_y$ and $d_y$ with $d'_x$, we get

$$
(1 + 6m)(2d'_x - n) + 3(2d'_y - m) + 2(d'_y - 1) + d'_x + 3 > 0.
$$

Since $n - 2d'_x > 0$ and $d'_y < m$, we have

$$
(1 + 6m)(2d'_x - n) + 3(2d'_y - m) + 2(d'_y - 1) + d'_x + 3 \\
\leq (1 + 6m)(-1) + 3(2(m - 1) - m) + 2(m - 2) + d'_x + 3 = d'_x - m - 8.
$$

Hence it should hold that

$$
d'_x - m - 8 > 0. \tag{12}
$$

By adding relations (11) and (12), we should have $d_y + d'_x - n - m - 16 > 0$, which is impossible. Hence, either $m - 2d_y = 0$ or $n - 2d_x = 0$ (or both).

Now the two agents have reached their initial nodes $T_1(A)$ and $T_1(B)$, as shown in Fig. 9(c). As before, agent $A$ exits the loop at step 33 and arrives at node 1 before or at the same time as agent $B$ exits the loop at step 33 and arrives at node 3. Since the agents did not meet, there are three possible configurations, shown in Fig. 14. As we will see, in all cases both agents spend $2m + n$ time units from the time they enter Procedure BuildRectangle until they move a token at step 9.
We could instruct the agents to synchronize again by using a procedure similar to Procedure Synchronize. However, as we will see, this is not necessary at this time.

Let $B$ be the agent that escaped from node $T_1(A)$ before $A$ gets there. Agent $B$ has moved left from token $T_1(A)$ until it meets token $T_2(A)$, then moved one step right and up until it meets token $T_1(B)$ and spent at most another $n - d_x + 1 + 2(m - d_y - 1) + 1 + 2m$ time units before starting Procedure BuildRectangle. Agent $A$ moves up from token $T_1(A)$ until it meets it again, and thus needs $2m$ time units to start Procedure BuildRectangle. If $A$ sees two tokens at its initial node (Fig. 14(b) or (c)), it enters step 6 and spends $n$ time units until it meets token $T_1(A)$ again. If $A$ sees one token at its initial node (Fig. 14(a)), it enters step 4 and spends $n$ time units until it meets token $T_1(A)$ again. Step 8 takes another $2m$ time units (even if $B$ moved a token in that vertical ring, $A$ searches only for nodes with two tokens). We have

$$n - d_x + 1 + 2(m - d_y - 1) + 1 + 2m < 2m + 2m + n.$$ 

Therefore, agent $B$ starts executing Procedure BuildRectangle before agent $A$ moves a token. In all cases, agent $A$ will be the first to reach step 9 of Procedure BuildRectangle and move a token, since after entering Procedure BuildRectangle both agents need $2m + n$ time units until they move a token.

After entering Procedure BuildRectangle, agent $B$ needs another $2m + n + 1$ time units to move a token. Agent $A$ spends 1 time unit to move a token and then another $n$ time units to meet this token again (since, after moving a token, there are exactly two tokens in this horizontal ring). Finally (since after moving a token, there are exactly two tokens in this vertical ring), agent $A$ finishes Procedure BuildRectangle, spending another $2m$ time units. Hence, by the time agent $A$ starts executing Procedure Chase, agent $B$ has finished building the rectangle. In all configurations of Fig. 14, each agent moves a token from its initial node one step to the left, and hence the rectangle will be built correctly. Therefore, the agents start (possibly not synchronized) executing Procedure Chase, which, according to Lemma 10, ends up with a rendezvous unless $d_x = n/2$ and $d_y = m/2$. \hfill \Box

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

**Lemma 11.** If after the horizontal and vertical scanning of Algorithm RV2mn the agents do not rendezvous and $\frac{m-1}{10} \leq n \leq 10m + 1$, then their distance is $(n/2, m/2)$, and therefore a rendezvous is impossible.

**Proof.** The agents execute Algorithm RV2mn. Suppose that they do not rendezvous. Recall that, if the agents do not rendezvous after the horizontal scanning and vertical scanning, then either $m = 2d_y$ or $n = 2d_x$ holds (or both).

**Case 1:** Suppose that $m = 2d_y$ and $n - 2d_x \geq 1$.

Consider agents $A$ and $B$, where $B$ is initially located a distance $d_y$ down of $A$ and $d_x$ to the right. Since the agents did not rendezvous after the vertical scanning, by interchanging $d_x$ with $d_y$ and $n$ with $m$ in (8) we should have

$$(1 + 6m)(2d_y - n) + 3(2d_y - m) + 2(d_y - 1) + d_x + 3 > 0.$$ 

Since $m = 2d_y$, we get

$$(1 + 6m)(2d_y - n) + m - 2 + d_x + 3 > 0$$

$$(1 + 6m)(2d_y - n) + m - 2 + d_x + 3 \leq -(1 + 6m) + m - 2 + \frac{n - 1}{2} + 3 = \frac{n - 1}{2} - 5m.$$
Suppose that \( n \leq 10m + 1 \). Then \( \frac{n-1}{2} - 5m \leq 0 \), which means that the agents rendezvous if \( n - 2d_x \geq 1 \).

Case 2: Suppose that \( n = 2d_y \) and \( m - 2d_y \geq 1 \).

Consider agents \( A \) and \( B \), where \( B \) is initially located a distance \( d_y \) down of \( A \) and \( d_x \) to the right. Since the agents did not rendezvous after the horizontal scanning, (8) should hold:

\[
(1 + 6n)(2d_y - m) + 3(2d_x - n) + 2(d_x - 1) + d_y + 3 > 0.
\]

Since \( n = 2d_x \), we get

\[
(1 + 6n)(2d_y - m) + n - 2 + d_y + 3 > 0
\]

\[
(1 + 6n)(2d_y - m) + n - 2 + d_y + 3 \leq -(1 + 6n) + n - 2 + \frac{m - 1}{2} + 3 = \frac{m - 1}{2} - 5n.
\]

Suppose that \( m \leq 10n + 1 \). Then \( \frac{m-1}{2} - 5n \leq 0 \), which means that the agents rendezvous if \( m - 2d_y \geq 1 \).

Hence, if \( n \leq 10m + 1 \) and \( m \leq 10n + 1 \) and the agents do not rendezvous, then it means that \( m = 2d_y \) and \( n = 2d_x \). \( \square \)

Therefore, by Lemma 11, if we knew that \( n \) is at least about one tenth and no more than ten times \( m \), then Algorithm RV2mn would be an RVD algorithm for the \( n \times m \) torus.

### 3.3. Rendezvous with detection in an \( n \times m \) oriented torus using three movable tokens and constant memory

If the agents have three tokens, then we can modify Algorithm RV2mn to get Algorithm 4, which is an RVD algorithm for an arbitrary \( n \times m \) anonymous, synchronous and oriented torus. The idea is the following. If the agents do not meet while they execute Procedure SearchTorus, they build the rectangle by executing Procedure BuildRectangle, but instead of executing Procedure Chase, they release their third token to the right of their starting position. They travel on the 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 (see Fig. 15). If at that point they see two tokens adjacent, then they declare a rendezvous impossible. Otherwise, they wait until a rendezvous, which will occur in less than \( n + m \) time. Algorithm RVD3mn takes \( O(n^2 + m^2) \) time.

**Algorithm 4 RVD3mn**

1: SearchTorus
2: if not rendezvous then
3:   BuildRectangle
4:   move right and drop the third token
5:   while the token did not hit another token do
6:     go right until you meet a token
7:     go down until you meet a token
8:     go left until you meet a token
9:     go up until you meet a token
10:    go right until you meet a token
11:    move that token one step to the right
12:   end while
13: move token down
14: while the token did not hit another token do
15:   go down until you meet a token
16:   go left until you meet a token
17:   go up until you meet a token
18:   go right until you meet a token
19:   go down until you meet a token
20:   move that token one step down
21: end while
22: go left until you meet a token
23: go up until you meet a token
24: if there are two tokens there then
25:   stop and declare rendezvous impossible
26: else
27:   wait
28: end if
29: end if
Theorem 8. The rendezvous with detection problem on an arbitrary $n \times m$ oriented 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 neither rendezvous nor decide that a rendezvous is impossible. This means that either their horizontal distance $d_x = n/2$ or their vertical distance $d_y = m/2$ (or both). Let, w.l.o.g., agent A be at node 1 before or at the same time as agent B reaches node 3 in Fig. 15. This means that agent B is $d_x \leq n/2$ to the right of agent A and $d_y \leq m/2$ down from agent A. Agent A travels for $(2d_x + 2d_y)(d_x + d_y - 2) + d_x + d_y$ time units until it sees its third token touch a token for the second time (at node 3). The other agent B travels for $(2(n - d_x) + 2(m - d_y))(n - d_x + m - d_y - 2) + n - d_x + m - d_y$ time units until it sees its third token touch a token for the second time.

Suppose, w.l.o.g., that $d_x = n/2$ and $d_y \leq m/2$. If $d_y = m/2$, then the two agents should have started at the same time, and they always move their tokens at the same time, and at the end (when their third token touches a token for the second time) they find out that the other’s token touches another token as well. Therefore they declare a rendezvous impossible.

If $d_y < m/2$, then agent A starts earlier, and is the first to see its third token touching a token for the second time at node 3. At that moment, agent B needs at least one more round to move its third token to node 1. This means at least another $2(n - d_x) + 2(m - d_y) > d_x + d_y$ time units for agent B before reaching node 1. Hence, in $d_x + d_y$ time units, agent A will reach node 1 first, waiting for agent B. $\square$

4. Conclusion

In this paper, we have investigated the number of tokens and memory that two agents need in order to rendezvous in an anonymous oriented torus. We have shown that when the agents have one (movable or unmovable) token each then both the RV and RVD problems in an arbitrary $n \times m$ oriented torus require $\Theta(\log n + \log m)$ memory.

It appears that there is a strict hierarchy on the power of tokens and memory with respect to a rendezvous: a constant number of unmovable tokens is less powerful than two movable tokens. While the hierarchy collapses on three tokens (we gave an algorithm for the rendezvous with detection problem in an $n \times m$ 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 the rendezvous with detection problem. It is also interesting that although a movable token is more powerful than an unmovable one (we showed that an agent with one movable token cannot visit all nodes of any $n \times n$ oriented torus 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 a rendezvous; 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 Fig. 16.

An interesting open problem on the number of tokens for solving the rendezvous problem with or without detection, having constant memory, arises when the torus is not oriented. We conjecture that by using additional tokens one may be able to extend the results of this paper to the case of the unoriented torus.

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, many interesting questions remain open.

- Can we improve the time complexity for the rendezvous without detection problem on an $n \times m$ torus using constant memory? Can we improve the time complexity for the rendezvous with detection problem on an $n \times n$ torus using constant memory?
What is the lower memory bound for two agents with two movable tokens each in order to rendezvous with detection in an \( n \times m \) torus? In particular, can they do it with constant memory?

- Is there a tradeoff between the number of unmovable tokens needed and the memory required to solve the RV problem. Specifically, is it possible to solve the RV problem with a linear number of tokens and constant memory?

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

- What is the situation when the torus is asynchronous?

- An interesting problem is that of many agents trying to rendezvous (or gather) in a torus network.

- Since the knowledge of the underlying graph topology is essential for obtaining deterministic algorithms with tokens, a challenging open problem is that of an arbitrary network. What is the tradeoff between the number of tokens and memory needed by the agents for solving the rendezvous with detection problem in an arbitrary network?

**Acknowledgements**

The authors wish to thank the anonymous referees for their helpful suggestions based on an earlier version of this paper. The first author’s research was supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants. The third author did his work partly while visiting Carleton University (2005–2006). The research was supported in part by an NSERC grant and by PYTHAGORAS project 703/7392 under the EPEAEK program of the Greek Ministry of Educational and Religious Affairs.

**References**


