Mobile Agent Rendezvous in a Synchronous Torus

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

[1] School of Computer Science, Carleton University, Ottawa, Ontario, Canada. kranakis@scs.carleton.ca
[2] Department of Mathematics, Wesleyan University, Middletown, Connecticut 06459, USA. dkrizanc@caucus.cs.wesleyan.edu
[3] Department of Computer Science, National Kapodistrian University of Athens, Athens, Greece. emarkou@cs.ntua.gr

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

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

1 Introduction

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

The problem is well studied for several settings. When the nodes of the network are uniquely numbered, solving the rendezvous problem is easy (the

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

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

two agents can move to a node with a specific label). However even in that case the agents need enough memory in order to remember and distinguish node labels. Symmetry in the rendezvous problem is usually broken by using randomized algorithms or by having the mobile agents use different deterministic algorithms. (See the surveys by Alpern [1] and [2], as well as the recent book by Alpern and Gal [3]). Yu and Yung [13] 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 [5] mark the starting points of the agents, they still rely on randomized algorithms or different deterministic algorithms to solve the rendezvous problem.

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 [13] have considered attaching unique identifiers to the agents while Dessmark, Fraigniaud and Pelc [6] added unbounded memory; note that having different identities allows each agent to execute a different algorithm. Other researchers (Barriere et al [4] and Dobrev et al [7]) have given the agents the ability to leave notes in each node they visit. In another approach each agent has a stationary token placed at the initial position of the agent. This model is much less powerful than distinct identities or than the ability to write in every node. Assuming that the agents have enough memory, the tokens can be used to break symmetries. This is the approach introduced in [11] and studied in Kranakis et al [10] and Flocchini et al [8] for the ring topology. In particular the authors proved in [10] that two agents with one unmovable token each in a synchronous, n-node oriented ring need at least $\Omega(\log \log n)$ memory in order to do rendezvous with detection. They also proved that if the token is movable then rendezvous without detection is possible with constant memory.

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

1.1 Model and Terminology

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

any other agents. The agents follow the **same** deterministic algorithm and begin execution at the same time.

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

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

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

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

Let $U = \{(n_1/2, 0, ..., 0), (0, n_2/2, ..., 0), ..., (0, 0, ..., n_d/2)\}$, where each n_i is even, be a set consisting of d vectors in d-dimension. The *distance* between two nodes on a d-dimensional torus is a d-vector the ith of element of which is $\min\{(x_i - y_i), (n_i + y_i - x_i)\}$ where wlog $x_i \geq y_i$ are the ith co-ordinates of the nodes.

[4] The first known algorithm designed for graph exploration by a mobile agent, modeled as a finite automaton, was introduced by Shannon [12] in 1951.

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

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

Theorem 1 is a generalization of Theorem 1 in [10] which states that it is impossible for two agents equipped with one unmovable token each, to rendezvous in a ring with n nodes if their initial distance is $n/2$, where n is even.

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

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

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

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

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

1.2 Our Results

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

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

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

1.3 Outline of the Paper

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

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

In Section 4 we discuss the results and state some open problems. Due to space limitations, the proofs, formal algorithms and figures have been omitted in this extended abstract.

2 Memory Lower Bounds of Rendezvous

2.1 Preliminary Results

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

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

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

2.2 An $\Omega(\log n)$ Memory Lower Bound for Rendezvous Using One Movable Token

Lemma 3. *Consider two mobile agents with σ states. They each have a token (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find an oriented $n \times n$ torus, where $n = \Omega(\sigma^2)$ and place the agents so that if they can not move tokens then they cannot rendezvous.*

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

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

□

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 3 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.

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

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

- 1: go right until you meet the second token;
- 2: move the token down;
- 3: repeat from step 1;

Nevertheless the goal is again to place the agents in a way that they could meet only their own token. To achieve this we place the agents so that in a phase

which starts when the agents move their tokens, up to the moment that they move their tokens again they do not meet each other's token.

Lemma 4. *Consider two mobile agents with σ states. They each have a token (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find an oriented $n \times n$ torus, where $n > 8\sigma^3 = \Omega(\sigma^3)$ and place the agents so that even if they can move tokens they cannot rendezvous.*

This implies the following theorem:

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

Proof. Suppose that the agents have a memory of r bits. Hence they can have at most 2^r states. By Lemma 4 as long as $n > 8\sigma^3$ the agents cannot perform rendezvous. Hence, the agents need at least $r = \Omega(\log n)$ memory in order to perform rendezvous. □

2.3 An $\Omega(\log n)$ Memory Lower Bound for Rendezvous Using $O(1)$ Unmovable Tokens

Lemma 5. *Consider two mobile agents with σ states. They each have two tokens (identical to each other). Then we can always (for any configuration of the automatons, i.e. states and transition function) find a $n \times n$ oriented torus, where $n = \Omega(\sigma^2)$ and place the agents so that if they cannot move tokens they cannot rendezvous.*

Proof. (**Sketch**) In view of Lemmas 2, 3 we can select the torus and the starting positions so that an agent will visit at most $(\sigma+1)(1+\sigma n)$ nodes until it decides to release its second token and up to that point does not meet the other's token. Its second token will have to be released at a 'short' distance from the first one since an agent cannot count more than σ. Using similar arguments as in the proof of Lemma 3 one can show that there are at least $n^2 - 5(\sigma + 1)(1 + \sigma n)$ pairs of starting nodes that satisfy property π. □

This implies the following theorem:

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

Applying similar arguments we can prove the following lemma and theorem:

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

Proof. (**Sketch**) Using similar arguments as in the proof of Lemma 3 one can show that there are at least $n^2 - \frac{k(k+2)}{2}(\sigma+1)(1+\sigma n)$ pairs of starting nodes that satisfy property π. □

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

3 Rendezvous

3.1 Rendezvous with Detection (RVD) in a $n \times n$ Torus Using One Token and $O(\log n)$ Memory

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

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

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

The above result can be extended for the case of an arbitrary $n \times m$ torus. The main difference in that case is how the agents decide if they have started on the same ring or not: they again explore one by one the horizontal rings. They will meet a token while going down (passing from one horizontal ring to the next) if and only if they have started on the same ring. Otherwise, they will meet a token while going right (before finishing the exploration of a horizontal ring). They can solve the Rendezvous with Detection problem in $O(nm)$ steps as long as they have $O(\log n + \log m)$ memory each.

3.2 Rendezvous with Detection in a $n \times n$ Torus Using Two Movable Tokens and Constant Memory

We define Procedure HorScan which will be used in our algorithms.

In this procedure the agent stops immediately after it meets a token. So for example, if after it goes right it meets a token then it stops immediately; it does not go up.

Procedure HorScan

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

Procedure FindTokenHor

1: **repeat**
2: HorScan
3: **if** you meet token up **then**
4: HorScan
5: go one step down and drop (or move) the second token
6: **end if**
7: **until** you meet a token down or right

We also use Procedure FindTokenHor.

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

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

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

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

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

Therefore the agent exits procedure FindTokenHor knowing that it has started either in the same ring with the other agent (if it met a token after it moved down) or in different rings (if it met a token after it moved right). Procedure FindTokenHor needs $O(n^2)$ time units.

Below is a high level description of the algorithm RVD2n which solves the RVD problem in a $n \times n$ torus. The two agents search one by one the horizontal rings of the torus (using Procedure FindTokenHor) to discover whether they have started in the same ring. If so, then they execute a procedure which appeared

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

in [9], for rendezvous with detection in a ring using two tokens and constant memory. Otherwise they try to 'catch' each other on the torus using a path, marked by their tokens. If they do not rendezvous then they search one by one the vertical rings of the torus (using a procedure similar to FindTokenHor for searching one by one the vertical rings). They again try to 'catch' each other on the torus. If they do not meet this time they declare rendezvous impossible. The algorithm takes $O(n^2)$ time.

Theorem 7. *The Rendezvous with Detection problem on a $n \times n$ torus can be solved by two agents using two movable tokens and constant memory each, in time $O(n^2)$.*

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

3.3 Rendezvous without Detection in a $n \times m$ Torus Using Two Movable Tokens and Constant Memory

We give now algorithm RV2mn which is a RV algorithm for two agents with constant memory in a $n \times m$ torus. Algorithm RV2mn, at first, copies algorithm RVD2n. If no rendezvous occurs and no decision is made about its impossibility (i.e. the agents have started in different rings), the algorithm instructs the agents to mark a rectangle with their tokens on the torus and then execute Procedure Pendulum: they try to shrink the rectangle and eventually meet which will happen unless they had started at distance $(n/2, m/2)$ (in that case the algorithm runs forever).

In fact one of the following things could happen: either the agents rendezvous, or they detect that they are in the same ring and their distance is half the size of the ring or the algorithm runs forever (in that case they are at horizontal distance $n/2$ and vertical distance $m/2$). Algorithm RV2mn needs $O(n^4 + m^4)$ time.

Theorem 8. *The Rendezvous without Detection problem on an arbitrary $n \times m$ torus can be solved by two agents using two movable tokens and constant memory each, in time $O(n^4 + m^4)$.*

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

Lemma 7. *If after the horizontal and vertical scanning of Algorithm RV2mn the agents do not rendezvous and $\frac{n-1}{10} \leq m \leq 2n + 17$ then their distance is $(n/2, m/2)$ and therefore rendezvous is impossible.*

Hence by Lemma 7 if we knew that $\frac{n-1}{10} \leq m \leq 2n + 17$ then algorithm RV2mn would be a RVD algorithm for the $n \times m$ torus.

3.4 Rendezvous with Detection in a $n \times m$ Torus Using Three Movable Tokens and Constant Memory

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

Theorem 9. *The Rendezvous with Detection problem on an arbitrary $n \times m$ torus can be solved by two agents using three movable tokens and constant memory each, in time $O(n^2 + m^2)$.*

4 Conclusions

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

It appears that there is a strict hierarchy on the power of tokens and memory with respect to rendezvous: a constant number of unmovable tokens are less powerful than two movable tokens. While the hierarchy collapses on three tokens (we gave an algorithm for rendezvous with detection in a $n \times m$ torus when the agents have constant memory each), it remains an open question if three tokens are strictly more powerful than two with respect to rendezvous with detection.

It is also interesting that although a movable token is more powerful than an unmovable one (we showed that an agent with one unmovable token cannot visit all the nodes of a torus with a properly selected size unless it has $\Omega(\log n)$ memory, while it could do it with a constant memory if it could move its token) it appears that this power is not enough with respect to rendezvous; the agents with one movable token each, still require $\Omega(\log n)$ memory to rendezvous in the torus.

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

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

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

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

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

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

References

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