Optimal memory rendezvous of anonymous mobile agents in a uni-directional ring

L. Gąsieniec[*] E. Kranakis[†] D. Krizanc[‡] X. Zhang[§]

May 13, 2005

Abstract

We study the rendezvous problem with $k \geq 2$ mobile agents in a n-node ring. We present a new algorithm which solves the rendezvous problem for the non-symmetric distribution of agents on the ring. The mobile agents require the use of $O(\log k)-$bit-wise size of internal memory and one indistinguishable token each. In the periodic (but not symmetric) case our new procedure allows the agents to conclude that rendezvous is not feasible. It is known that in the symmetric case the agents cannot decide the feasibility of rendezvous if their internal memory is limited to $\omega(\log \log n)$ bits, see [15]. In this context we show an optimal algorithm that uses $O(\log k + \log \log n)$-size internal memory and a single token available at each mobile agent which allows them to recognize the symmetric case efficiently. Note that both in the periodic as well as in the symmetric case the rendezvous cannot be completed by any deterministic procedure due to problems with breaking the symmetry.

1 Introduction

The mobile agent *rendezvous* problem is a search optimization problem, whereby a number of autonomous deterministic agents starting from two (or more) given nodes of a network are required to rendezvous at the same node or edge (of the network) so as to optimize either the number of steps or the memory used by the robots or a tradeoff of both. The mobile agents are autonomous entities that move along vertices of the network acting on the information collected following a given protocol.

There are instances where the rendezvous problem is easy, e.g., if the nodes of the network have distinct identities, in which case the mobile agents can move to a node with a specific pre-assigned identity. However, even in this case the problem becomes more difficult to solve if the agents do not have enough memory to "remember" and distinguish identities. In general, solutions of the rendezvous problem are challenging under conditions that delimit what the mobile agents can do and/or know about the status of the overall system. We are interested in studying tradeoffs between memory-space used and time required by the mobile agents in order to rendezvous in a ring.

[*]Department of Computer Science, University of Liverpool, L69 7ZF, Liverpool, UK.

[†]Carleton University School of Computer Science, 1125 Colonel By Drive Ottawa, Ontario K1S 5B6, Canada. Research supported in part by NSERC and MITACS grants.

[‡]Computer Science Group, Mathematics Department, Wesleyan University, Middletown, CT 06459 USA.

[§]Department of Computer Science, University of Liverpool, L69 7ZF, Liverpool, UK.

There is a variety of interesting scenarios under consideration that may involve 1) (minimum) number of tokens used by the mobile agents, 2) knowledge of the status of the network and presence of other mobile agents in the system (e.g., number of mobile agents participating in the rendezvous problem under consideration, feasibility of rendezvous for a given starting configuration), 3) knowledge of inter-agent distances at the start, etc. If the rendezvous problem is impossible to solve the mobile agents will be executing a rendezvous algorithm that may never terminate. It is also crucial for accomplishing rendezvous what knowledge the mobile agents have about the network configuration (e.g., size of the network, relation between start positions of the mobile agents).

1.1 Communication model

A number of mobile agents are situated on the nodes of a unidirectional synchronous ring. Each mobile agent owns a token that it may release at given nodes of the ring. The tokens are indistinguishable from each other. At any single time unit, the mobile agent occupies nodes of the ring and may 1) stay at its current position, 2) move left or right from its current position, 3) detect the presence of one or more agents at the node it is currently occupying, 4) detect the presence of one or more tokens at the node it is currently occupying, and 5) release/remove one or more tokens to/from any node it is occupying. The mobile agent may be *released* at any node of the network and the start node of the mobile agent is also called its *home node*. A token is *movable* if it can be moved by any mobile agent to any other node of the network, otherwise a token is not movable in the sense that it can occupy only the home node of the agent that released it. Mobile agents may communicate and exchange information with each other only when they are situated at the same node. We say that one or more mobile agents rendezvous when either they meet at the same node or else traverse the same edge in opposite directions.

In this paper we study the rendezvous problem with $k \geq 2$ mobile agents in a n-node ring. We use the following model of communication. The ring and the agents are anonymous. The links in the ring are unidirectional and there is a sense of direction [13]. The ring consists in n nodes but this size is not known to the nodes of the ring or to the agents. The nodes of the ring are identical, i.e., they do not have distinct identities. The k participating agents use indistinguishable tokens only once to mark their original positions in the ring. The agents move along the ring in synchronous steps. We assume that all agents start to traverse the ring at the same time and they execute the same deterministic algorithm. The agents may communicate and exchange information with one another only when they meet each other at some node. Moreover, when an agent finds a token that has been released at a node in the ring it can not distinguish it from its own token(s) or from the token(s) of the other mobile agents (this is equivalent to identifying tokens with *erasable marks*).

1.2 Related work

There has been considerable literature on the rendezvous problem. The problem initiated with the work of Alpern (see the survey [2]). Research conducted over the years by Alpern and various collaborators [1, 3, 4, 5, 16] concentrated on the optimization problem for probabilistic search in operations research. Additional research with emphasis on either optimization on rendezvous as a competitive game was also conducted by Pikounis and Thomas [17], Anderson and Essegaier [6], Anderson and Fekete [7], Anderson and Weber [8], Baston and Gal [10, 11], Chester and Tutuncu [12].

We are interested in time memory tradeoffs for achieving rendezvous as well as for detecting the feasibility of rendezvous. The main concern in all rendezvous algorithms is how to “break symmetry”. In previous works mentioned above, symmetry in the rendezvous search problem is typically broken either by using randomized algorithms or by mobile agents running different deterministic algorithms [3], e.g., based upon distinct identities. Baston and Gal [10] consider the case where the agents may mark their starting points but they still rely on randomization or different deterministic algorithms to achieve rendezvous. The current token model used to break symmetries was initiated by Sawchuk [18].

Several possibility and impossibility results on the rendezvous problem for two mobile agents were considered in Kranakis et al [14], whereby two mobile agents on an n node ring can use identical tokens to break the symmetry. In Flocchini *et al.* [15] it is proved that the agent rendezvous problem has no solution when both values of k and n are unknown. Since we assume that $k \leq n$ this implies that agents require at least $\log k$ bits to store the value k. Flocchini *et al.* also show that rendezvous is feasible iff the original distribution of the distances between agents on the ring is non-periodic. Further, they show that any deterministic algorithm for an n-node uni-directional ring using a single token to mark the starting positions of the agents requires $\Omega(\log\log n)$ memory to recognize if rendezvous is feasible. They present several unconditional and conditional solutions to the agents rendezvous problem with different memory and time requirements. In particular, they provide an algorithm that performs rendezvous when possible or recognizes that rendezvous is not possible, that requires $O(k\log\log n)$ memory per agent. This result is not optimal for non-constant k.

1.3 Results and organization of the paper

In this paper we show the optimal utilisation of the internal memory by providing an $O(\log k + \log\log n)$ memory deterministic rendezvous algorithm.

The paper is organised as follows. In Section 2 we give a new gathering algorithm which performs rendezvous when possible or recognizes the periodic (non-symmetric) input case. This algorithm requires $O(\log k)$ bits of memory per agent. Note that in the symmetric case this algorithm will not halt. In Section 3 an algorithm for identifying symmetric distributions of mobile agents is given. This algorithm requires $O(\log k + \log\log n)$ bits of memory per agent. In both cases we provide analysis and proof of correctness. Together, the algorithms provide an optimal memory solution to the rendezvous problem on the ring.

2 Efficient Agents Gathering

In this section we give a detailed description of the communication model, the task to be performed and our new algorithmic solution.

2.1 The network model and the gathering task

The k anonymous agents $A_1, A_2, ..., A_k$ are placed on an anonymous directed $n-$node ring R, such that, the distance between the agents A_i and A_{i+1} is equal to S_i, for all $i = 1, ..., k$. Equivalently (depending on the context) we will also refer to the distances $S_1, ..., S_k$ as segments, see figure 1. Note that reference to the agent A_j is always understood as the reference $A_{j \bmod k}$. All the agents are anonymous (indistinguishable) and each of them possesses local memory limited to $\log k$ bits. Moreover, each agent A_i is allowed to leave a permanent mark T_i (indistinguishable token) at its original position on the ring. We assume here that the actions of the agents are totally synchronised. I.e., the agents wake up at the same time and later they proceed in perfectly synchronised time units, called also *time steps*. During each step an agent decides whether

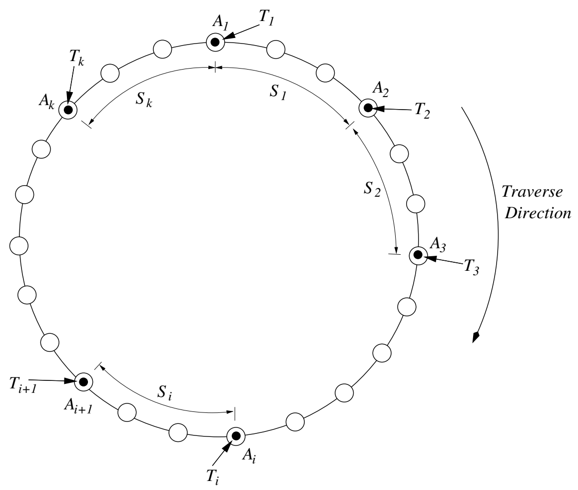

Figure 1: Basic facts about the model

to move to the next node (in clockwise direction) or to remain stationary according to the algorithm and the content of its $O(\log k)-$size memory (in fact, we know that each agent needs exactly $2\log k + 2$ bits). At the conclusion of each step the content of the local memory is updated. We assume also that the agents detect one another only when they meet either at some node or edge of the ring. When an agent moves back to its start point, we say that the mobile agent has finished a full turn or traversal of the ring.

The main task: considered here is *to gather* all participating agents (if possible) at one of the nodes of the ring R at some specific moment in time.

The outcome: of the gathering process depends on the size of the local memory of each agent. When the memory is limited to $O(\log k)$ bits then 1) the ring is entirely symmetric if the original distances between participating agents are uniform and the agents will run forever, 2) if the distribution of agents on the ring is periodic (but not symmetric), the agents will not rendezvous, though they eventually learn that the initial distribution is periodic and they stop. 3) Otherwise, the rendezvous is completed.

In case when the internal memory of each participating agent is limited to $O(\log k + \log\log n)$ bits then also in the symmetric case the agents eventually learn that the initial distribution is symmetric and they stop having this knowledge.

2.2 Basic notation and facts

In this paper, we introduce a new gathering mechanism which is based on a *variable speed* of the agents. In this mechanism, a current speed of an agent A_i depends on the relative index of the segment S_j, see Figure 1, as follows. We say that an agent A_i traverses with the speed $V_i(j)$ if it uses the segment S_{i+j} to calculate its speed. And when this happens, A_i always moves to the next node during each time step while being in S_{i+j} (we say that the agent goes with the *full speed* 1 in S_{i+j}); and it alternates moves with stationary steps (the agent goes with the speed 1/2) in all other segments.

There are three possible states in which the agents find themselves on the move: a *runner state*, a *marker state* and a *loser state*. An agent is called a *runner*, a *marker* and a *loser* in the mentioned states respectively. All the agents start to traverse the ring while being in the *runner state* or in other words being the *runners*. While being a *runner* the agent A_i traverses the ring using the Algorithm Rendezvous (see below). A_i becomes a marker when it is caught by A_{i-1}. When this happens, A_i loses its self-control and becomes the slave of the agent A_{i-1}. The *marker* A_i always follows the instructions of its master A_{i-1}. When the agent A_i is caught by A_j, for j≠i-1, the agent A_i becomes a *loser*. In other words, the agent A_i is switched off until a winner agent picks it up at the final stage of the rendezvous (this could only happen if the distribution of agents on the ring is not symmetric).

The utilization of the internal memory limited to $2\log k + 2$ bits of each agent is performed as follows:

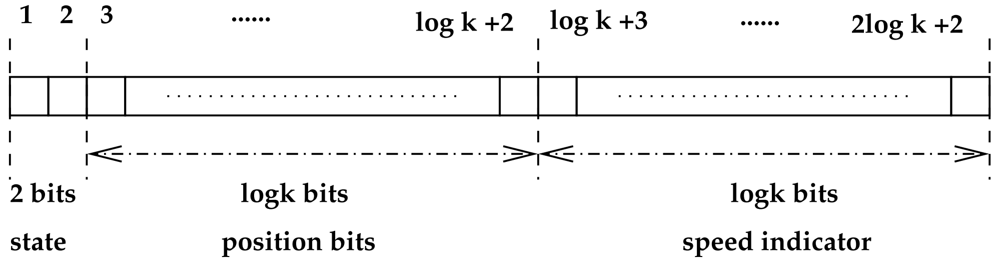

Figure 2: Memory size

1. *State*: 2 bits of memory are used to record agent's state. I.e., $0*$ means the agent is a *runner*, 10 means the agent is a *marker*, and 11 means the agent is a *loser*.
2. *Position bits*: the first chunk of $\log k$ bits is used to count the number (modulo k) of tokens the agent passed from its original position. This value is always between 0 and $k-1$. As soon as this value turns to 0, the agent knows that it is back in its starting position, i.e., a full turn has been accomplished.
3. *Speed indicator*: the second chunk of $\log k$ bits informs the agent in which segment the full speed should be used. This value is also within the range $0, ..., k-1$. In particular, when this value is i, the agent uses full speed on the $(i+1)$th segment from its starting position. When this value turns to 0, the agent is aware of the fact that it already tried all the available speeds. And when this happens, this part of the internal memory is used to count for how many agents are the rendezvous has been completed.

2.3 The algorithm

In this section we present our new gathering algorithm. Each of the participating agents executes the following code splitting the gathering procedure into three consecutive stages. During the Stage (0) every running agent is expected either to catch its direct predecessor or to be caught by one of its successors. We show later that if the initial distribution is not symmetric every agent accomplishes the Stage (0) eventually. Otherwise all agents remain in stage (0) for ever. In the next section we show also how to recognise this case with the optimal $O(\log\log n)$ bits of extra memory. The Stage (1) is used to eliminate (switch off) all agents but one who eventually gathers all (switched off) agents in its original position. We prove that this gathering process is entirely successful if the initial distribution is not periodic. In the periodic case there is more than one winner. The final stage is used to collect all the losers (including their markers), i.e., all switched off agents by the winner(s). If the winning agent collects all $(k-1)$ other agents it knows (so the other collected agents know too) that the gathering process was accomplished successfully. Otherwise the agents know that the case was periodic.

Finally note that since we deal with anonymous agents the indices of agents, segments, and tokens are relative. The code of the algorithm follows.

Algorithm Rendezvous (A_i)

STAGE (0)

Run with the speed $V_i(0)$ until:

(At this point, the agent's *state* in memory is 00. After agent starts running,

the *position bits* in the memory starts to count.)

(1) if A_i catch A_{i+1}

(1.a) make A_{i+1} as the *marker* M_i;

(Change agent A_{i+1}'s *state* from 00 to 10 and reset *position bits* to 0.)

(1.b) A_i goes with M_i to T_i, and leave it there;

(When *position bits* turn to 0, agent A_i knows that it back to T_i now.

Also, *position bits* of the *marker* start to count from 0 again.)

(1.c) go to Stage(0.5);

(2) if A_i is caught or meet the marker of another agent, then A_i gets switched off.

(In the symmetric case, all the agents remain in the Stage (0) for ever.)

If the sequence of segments S_i is not uniform there are at least two neighboring agents, A_i and A_{i+1}, that will traverse the ring using two different speeds $V_i(0) > V_{i+1}(0)$. In this case, after every full turn, the agent A_i gets closer to the agent A_{i+1} by at least one position on the ring. Since the original distance between the respective tokens T_i and T_{i+1} is less than n we know that the agent A_i will catch the agent A_{i+1} (if not caught by another one) within the first n full turns which corresponds to is $O(n^2)$ units of time.

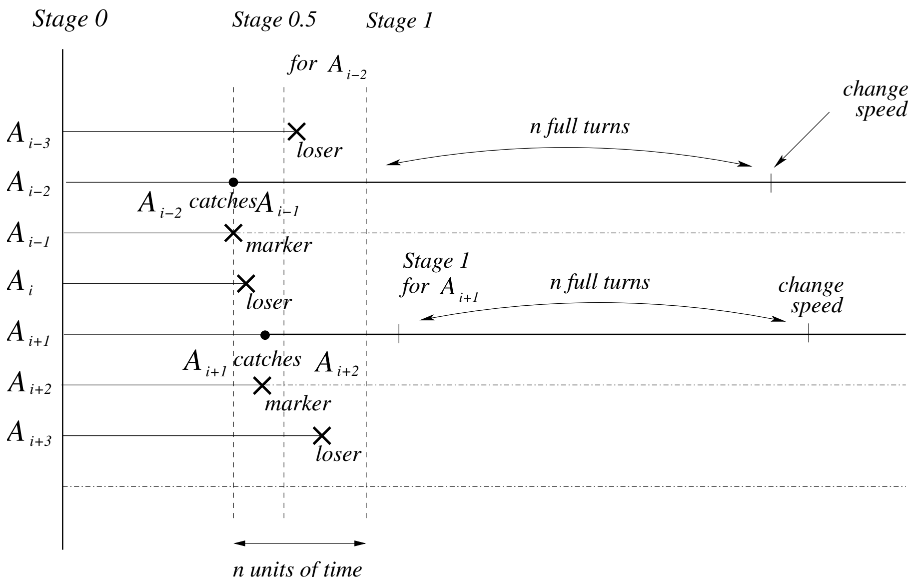

Figure 3: Progress through the Stage (0.5)

STAGE (0.5)

Leave the marker M_i at the position containing T_i, and

run with the full speed for one full turn.

(1) When A_i catch another agent, switch it off;

(2) Go to Stage (1).

After the move of the agent A_i to the Stage (0.5), A_i leaves its marker M_i at the node containing the token T_i and immediately it makes one full turn with the full speed (one node during each time unit). Note that during the n time units (full turn), all other agents who are still in the Stage (0), if they do not manage to find their own marker on time, they will be switched off either by being caught or by encountering the marker of another agent already being in the Stage (0.5).

The main purpose of the Stage (0.5) is to ensure that on the completion of the stage (0.5) by any agent (including the most advanced one) all other agents are either switched off or they are at least in stage (0.5) or later. As the consequence of this we know that the largest difference between the events in which two agents enter the Stage (1) is never larger than n units of time.

STAGE (1)

Run $j = 0, 1, ...(k - 1)$ rounds with the speed $V_i(j)$.

(The *speed indicator* of the memory is used to count from 0 to $k - 1$, which could

let the agent A_i know when it should use full speed to traverse.)

(1) Every time when A_i meets its *maker* M_i, move M_i one node ahead;

(When a *runner* meet a *marker*, it must check both the *position bits* of the *marker*

and itself, if the values are the same, then it knows this is its *marker*.)

(2) When A_i catch another agent, switch it off;

(Change the caught one's *state* to 11.)

(3) Until M_i is back to T_i, then $j = j + 1$;

(*) If A_i is caught by another agent, then it gets switched off.

After entering the Stage (1), an agent A_i traverses the ring using k variable speeds $V_i(j)$, for $j = 0, .., k - 1$, based on the sizes of consecutive segments following the token T_i (i.e., the original position of A_i). More precisely, the ring traversal in the Stage (1) is split into k consecutive rounds (see main loop for), s.t., during the l^{th} round the speed $V_i(l)$ is used for exactly n full turns. Note that the full turns are counted with the help of the marker M_i (see step (1) in the body of the main loop) and the internal memory (1^{st} chunk of $\log k$ bits of the internal memory, see Figure 2). The switch to the next speed is recorded in the internal memory via appropriate update (addition of 1 modulo k to the number represented by the last $\log k$ bits in the internal memory. The current (relative to the original) segment location of the agent is represented by the 1^{st} chunk of $\log k$ bits, which is updated (also via addition of 1 modulo k to the number represented by the bits) on every encounter of another token. We say that an agent A_i accomplishes the Stage (1) successfully if it runs all rounds of this stage without being caught by another agent. This process requires $O(kn^2)$ units of time. The following lemma holds.

Lemma 1 *On the successful completion of the Stage (1) an agent A_i is either a unique winner (in the non-periodic) or one of the several winners (in the periodic case).*

Proof: Consider the non-periodic case and two arbitrary agents A_i and A_j who managed to progress to the

Stage (1). Note that the pattern of segments following the initial positions for both of the agents must be different; otherwise we would have the periodic case. As the result of this there will be a round (with n full turns) in which the two agents will use different speeds. So during this round one of the agents will have to catch the other one if not caught by someone else in the meantime. In other words any agent entering the Stage (1) will have a chance to challenge any other agent in one of the k rounds. And since during each challenge one of the competing agents gets switched off, after k rounds only one survivor is left alive on the ring. And this is the winner who will complete the gathering process in the Stage (2). Note that this whole dueling process is feasible because the maximum time difference between the entry of two agents to the Stage (1) is limited to at most n units of time. And since each rounds is based on n full turns there will be enough time to meet any two competing agents.

We know that in the periodic case the gathering of agents governed by any deterministic procedure is not feasible. In this case the agents entering the Stage (1) are partitioned into several classes of abstraction. The nodes in one class are represented by the same pattern of segments. Note that these agents will never compete (challenge each other) during the k consecutive rounds. However they will challenge every other agent belonging to the other class. As the consequence of this at the end of the Stage (1) there will be survivors in at most one class of abstraction. The survivors will enter the Stage (2) when they will learn that the gathering is not feasible. This process takes n units of time.

STAGE(2) CLEANING-UP STAGE

The survivor A_i traverses for yet another round and collects all the switched off agents on the ring.
If the number of collected agents is $(k-1)$, we know that the rendezvous is successful.
Otherwise, the agent recognises the periodic case.
(Now, the agent A_i do not need to calculate its speed anymore, so it uses its *speed indicator*
to count how many agents it managed to collect.)

Theorem 1 *Gathering of k agents can be performed in the non-periodic case and the periodic but non-symmetric case can be recognised with the help of $O(\log k)$ bits of the internal memory present in each participating agent.*

The time complexity of the gathering process is dominated by the Stage (1) and it is bounded by $O(kn^2)$.

3 How to identify the symmetric distribution

If the distribution of the agents on the ring is entirely symmetric, then there is no deterministic algorithm that can complete the rendezvous task. So, it is important to test the initial distribution of the agents/tokens before the actual use of the rendezvous procedure.

3.1 Handling the symmetric case

In this section we show how to recognise the symmetric case with a help of extra $O(\log \log n)$ memory bits. It has been proved in [15] that this is the minimal memory required to recognise the symmetric case. So our algorithm uses asymptotically optimal size of the internal memory also in the symmetric case.

The algorithm is based on the use of a sequence of prime numbers and a simplified version of the Chinese

Remainder Theorem (see Apostol [9]). The main line of the reasoning is that for any two numbers $0 < n_1 < n_2 < n$ the sequence of the remainders of the integer division by the sequence of the consecutive prime numbers of a length $C \cdot \frac{\log n}{\log \log n}$, for some small constant C must differ eventually. Otherwise the number $0 < n_2 - n_1 < n$ would have more than $C \cdot \frac{\log n}{\log \log n}$ distinct prime divisors which is not feasible for C large enough.

Note also the value of the largest prime in the sequence is linear in $\log n$. In fact we can either consider the remainders based on prime numbers, or alternatively we can use the testing procedure based on the sequence of all consecutive integers in the range $1, .., O(\log n)$, which obviously includes all the primes required in the symmetry testing process.

The algorithm is split into consecutive rounds. During the i^{th} round based on the i^{th} prime (integer) every agent performs a full turn and calculates the remainder of the division of each $|S_i|$ by the prime (integer). If the remainder is the same within each segment, the next prime is taken and the test is now performed within each segment with the new prime, etc. If the remainders are uniform for all primes the agents know that the case is symmetric. If at any time of this process the diverse remainders (based on the same prime) are discovered (i.e., the case is not symmetric), the whole process is halted at the end of the round and all the agents proceed (at the very same moment) to the Stage (0).

The pseudo-code of the symmetry test algorithm follows.

(0) Place the token T_i at the starting position and set the flag to *symmetric-case.*
(Then start traversing the ring with the full speed
in rounds based on consecutive prime numbers.)
(1) Loop for $prime = 2, 3, 5, ..., p_{O(\frac{\log n}{\log \log n})}$ and flag = *symmetric-case.*
  (1.1) Visit consecutive segments $S_i, S_{i+1}, .., S_{i-1}$ and
    (1.1.1) Set counter of steps to 0;
    (1.1.2) Traverse each segment adding after each move one modulo
    $prime$ to the counter of the steps;
    (1.1.3) At the end of the segment test whether the current content
    of the counter of the steps (representing the value of the remainder)
    is consistent with with the content of the counter at all previous segments;
    (to implement this test we need one extra counter based on $O(\log \log n)$ bits
    to remember the uniform value of reminders computed in previous segments)
    (1.1.4) If the test is negative, set the flag to *non-symmetric-case.*

Finally note that to implement this process each agent will need at most $O(\log k + \log \log n)$ bits of memory, since all values to be computed during the symmetry test procedure do not exceed neither k nor $\log n$.

The following theorem follows.

Theorem 2 *Gathering of k agents can be performed in the non-periodic case and all other cases can be recognised with the help of $O(\log k + \log \log n)$ bits of the internal memory present in each participating agent.*

The time complexity of this test in $O(n \log n)$.

References

[1] S. Alpern, Asymmetric Rendezvous on the Circle, Dynamics and Control, 10, pp. 33-45, 2000.

[2] S. Alpern, Rendezvous Search: A Personal Perspective, LSE Research Report, CDAM-2000-05, London School of Economics, 2000.

[3] S. Alpern and S. Gal, The Theory of Search Games and Rendezvous, Kluwer Academic Publishers, 2003.

[4] S. Alpern and J.V. Howard, Alternating Search at Two Locations, LSE OR Working Paper, 99.30, 1999.

[5] S. Alpern and D. Reyniers, The Rendezvous and Coordinated Search Problems, Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, FL, December 1994.

[6] E.J. Anderson and S. Essegaier, Rendezvous Search on the Line with Indistinguishable Players, SIAM J. of Control and Opt., 33, pp. 1637-1642, 1995.

[7] E.J. Anderson and S. Fekete, Two-dimensional Rendezvous Search, Operations Research, 49, pp.107-188, 2001.

[8] E.J. Anderson and R.R.Weber, The Rendezvous Problem on Discrete Locations, Journal of Applied Probability, 28, pp. 839-851, 1990.

[9] T.M. Apostol, Introduction to Analytical Number Theory, Springer Verlag, 1997.

[10] V. Baston and S. Gal, Rendezvous on the Line When the Players' Initial Distance is Given by an Unknown Probability Distribution, SIAM Journal of Control and Optimization, 36, No.6, pp. 1880-1889, 1998.

[11] 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.

[12] E. Chester and R. Tutuncu, Rendezvous Search on the Labeled Line, *Old title*: Rendezvous Search on Finite Domains, Preprint, Department of Mathematical Sciences, Carnegie Mellon University, 2001.

[13] P. Flocchini, B. Mans, and N. Santoro, Sense of Direction in Distributed Computing, Theoretical Computer Science, 291, No. 1, pp. 29-53,2003.

[14] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, and C. Sawchuk, The Rendezvous Search Problem with More Than Two Mobile Agents, preprint, 2002.

[15] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, Multiple Mobile Agent Rendezvous in a Ring, LATIN'2004, pp 599-608.

[16] W.S. Lim and A. Beck and S. Alpern, Rendezvous Search on the Line with More Than Two Players, Operations Research, 45, pp.357-364, 1997.

[17] M. Pikounis and L.C. Thomas, Many Player Rendezvous Search: Stick Together or Split and Meet?, Working Paper 98/7, University of Edinburgh, Management School, 1998.

[18] C, Sawchuk, Mobile Agent Rendezvous in the Ring, PhD Thesis, Carleton University, 2004.