Breaking Symmetry with Agents of Different Speeds

Evan Huus (Carleton University)
eapache@gmail.com

April 3, 2013

Abstract

Breaking symmetry is a fundamental problem in distributed computing, and has immediate applications in many classes of algorithms including leader election, rendezvous and others. Feinerman et al. [3] showed that given two mobile agents travelling at different speeds on a ring, rendezvous can be achieved by using the speed difference to break symmetry. They presented upper and lower bounds on the time of reaching rendezvous in two versions of this scenario.

While the problems considered by Feinerman et al. were relatively straightforward ones, they note in their conclusion that the uses of their general technique are potentially much broader. In this paper we explore the results of Feinerman et al., then survey other problems and models in which their technique might be applicable.

1 Introduction

1.1 Background

The problem of breaking symmetry is a general one in distributed computing, with a wide number of applications. Many common problems require some form

of symmetry breaking to even be solvable, and having some level of asymmetry frequently opens up faster algorithms than can be constructed in a purely symmetric system.

As such, a great deal of effort and research has gone into various methods of breaking symmetry in common situations. Random bits can often be used of course, but they are frequently a limited resource and do not necessarily lead to deterministic time bounds, so more inventive approaches have been explored. One extremely common area in which much of this research has occurred is in the context of mobile agent rendezvous.

The rendezvous problem with mobile agents is fairly simple on its face: given a collection of two or more mobile agents in a network, can they meet (achieve rendezvous) and if so, how fast? However, the problem has an enormous number of variants and details to explore. For example, much depends on the knowledge available to the agent. Knowledge of the size or topology of the network, the starting distance between the agents, or the number of agents to rendezvous can all affect available algorithms. Agent memory is also a frequent concern; if memory is severely limited then many algorithms simply cannot be run.

1.2 Rendezvous with Different Speeds

Differences in agent speed are a common consideration in the rendezvous problem, but most often as an obstacle to rendezvous. For example, in many models such as [1] the agents can move at arbitrary speeds. This may sometimes be useful but cannot be used to break symmetry, since if the agents are deterministic then they will all move at the same speed as each other until symmetry is broken some other way. In other models such as [2] the speed of the agents is controlled by an adversary. In such models speed differences may in fact be an active hindrance to rendezvous, as the adversary can adjust agent speeds to

prevent rendezvous where it would otherwise be possible.

In the paper of Feinerman et al. they took a slightly different model where the speed difference between the agents is both fixed and known. In this situation they were able to treat the speed difference as a tool for breaking symmetry, rather than as an obstacle to be overcome, allowing them to achieve an algorithm for relatively fast rendezvous in their model.

As they note in their conclusion, Feinerman et al. considered only two of the most basic rendezvous models in their paper, while their technique has potentially much broader applications. In this paper we will summarize and explain the results of Feinerman et al. and then explore other models and problems where this type of symmetry-breaking technique may be of use.

1.3 Results

We consider four different applications of the symmetry-breaking technique used by Feinerman et al. We first consider the obvious extension of the original problem to the problem of achieving rendezvous with k different agents on a ring. We consider three different models, and show that under the two more powerful of these rendezvous is possible even when the agents don't have sufficient knowledge to rendezvous in the normal case (the weakest model provides no apparent benefit).

We also briefly examine the leader election problem; the application of their technique to this problem is simple and obvious. We then explore the case where instead of the agents having different speeds, the ring has different speeds. We show that this case can actually be considered equivalent to leader election and the multiset sorting problem in auxiliary rings that can be constructed from the original.

Finally, we consider applications to rendezvous outside the ring. We show

how rendezvous can be achieved in a torus using the same memory as an existing algorithm but with a constant factor (related to c) better running time. We also show that the obvious application of this principle to rendezvous in trees results in memory requirements that are quadratically worse than existing algorithms.

2 Breaking Symmetry with Different Speeds

In this section we lay out the precise models used by Feinerman et al. and explain their proofs and results from [3].

2.1 Model

The two models considered by Feinerman et al. differed only in one key point, so we will describe first the base model that they had in common. The underlying network graph is a simple ring of some arbitrary length n. The ring is *anonymous* in that specific nodes and edges have no identifying characteristics, and it is *oriented* in that an agent on the ring can tell clockwise from counter-clockwise. Movement on the ring is continuous, so rendezvous can happen at any point on the ring and it is not possible for agents to pass each other without rendezvous occurring.

An adversary places on the ring two deterministic mobile agents, identical but for their speed. Each agent x has a speed denoted by $s(x)$, which is the inverse of the time it takes x to travel one unit of length on the ring. An agent x can at any time be stopped or be moving at speed $s(x)$ in either direction. The agents do not have unique identities, though we will refer to one as A and the other as B for descriptive convenience. Without loss of generality, assume that A is the faster agent, and normalize $s(B) = 1$. Then let $c = s(A) = s(A)/s(B)$ which implies $c > 1$.

Agents have some knowledge available to them. They know n, the size of the

ring, as well as c, the ratio between their speeds. Agents do *not* know whether they are the fast agent (A) or the slow agent (B). Each agent has a pedometer that allows them to count how far they have travelled, and agents can of course detect rendezvous.

With this base model established, Feinerman et al. consider two variations. In the first, no communication is allowed between the agents. In the second, pebbles may be dropped by the agents at a point on the ring, and subsequently detected by other agents. They also consider the "white board" model and show that in this general model white boards are little stronger if at all than simply dropping pebbles.

It is worth noting that Feinerman et al. never mention the memory requirements for the agents in any version of their model. Their algorithms require some fairly non-trivial calculations on the part of the agents, so it's not immediately clear what bound can be put on an agent's memory. For simplicity's sake we will proceed as if the agents have unlimited memory.

2.2 Results

2.2.1 Without Pebbles

In the case where no pebbles or other communication is allowed, Feinerman et al. provide an algorithm that has a rendezvous time of $\frac{cn}{c^2-1}$ and show that this time is optimal in the current model.

Begin by considering a simpler (and slightly slower) algorithm that they call *Distributed Race* (DR). In DR, all agents start moving clockwise and proceed until rendezvous is reached. It is obvious that the distance between the agents shrinks by $c-1$ every unit of time, so for some clockwise distance d between A and B this algorithm takes $\frac{d}{c-1}$. Since $d \leq n$ this algorithm takes at most $\frac{n}{c-1}$ time.

Now consider a revised version of DR. In this revised version, an agent moves clockwise for $k = \frac{cn}{c^2-1}$ steps (note that this is *steps*, not *time*, so two agents with different speeds will traverse k steps in different amounts of time). If after k steps rendezvous has not been reached, the agent turns around and moves counter-clockwise until rendezvous. To see the running time of this algorithm, consider when the two agents will turn around. Due to their normalized speeds, B will turn around at time k and A will turn around at time k/c (both assuming rendezvous has not yet been reached).

If rendezvous has already been reached at time k/c then we are trivially below our stated time of $\frac{cn}{c^2-1}$, so we need only consider the case when A turns around. Clearly at time k/c when A turns around, B has not yet turned around. This means there is some time interval of length $k - k/c$ where B is still travelling clockwise while A is travelling counter-clockwise. During this stage of the algorithm, the distance between them shrinks by $c + 1$ units distance per unit of time, so the total distance travelled in this interval is:

$$\begin{aligned}
(c+1)\cdot(k-k/c) &= (c+1)\cdot k(1-1/c) \\
&= (c+1)\cdot\frac{cn}{c^2-1}\cdot\left(1-\frac{1}{c}\right) \\
&= \left(c-\frac{c}{c}+1-\frac{1}{c}\right)\cdot\frac{cn}{c^2-1} \\
&= \left(c-\frac{1}{c}\right)\cdot\frac{cn}{c^2-1} \\
&= \frac{c^2n-n}{c^2-1} \\
&= \frac{(c^2-1)n}{c^2-1} \\
&= n
\end{aligned}$$

Therefore they have closed the distance by n (the entire ring) and must have reached rendezvous by time $k = \frac{cn}{c^2-1}$.

Now we must show that this time is optimal. To simplify we can assume that the agents begin walking clockwise in any algorithm without loss of generality. Let $\hat{t}$ be the rendezvous time of an algorithm (the maximum time over all placements). Now consider the situation where the agents start at the same point on the ring and cannot detect rendezvous; they execute their algorithm even though they may have already reached rendezvous in the normal scenario. Let T be the time needed in this hypothetical execution for the agents to have been every distance d apart, $d \in [0, n)$.[1] It can be shown easily by contradiction that $T \leq \hat{t}$, since if there were some distance d' not covered then the algorithm would not produce rendezvous when the adversary started the agents at distance d' apart. This means that T is a lower bound on the rendezvous time.

We must also note two other facts about our hypothetical. Trivially, the distance between A and B can change by at most $c+1$ for each unit of time (this is also true in the normal scenario). More interestingly, the distance between the two agents at time x is at most $x(c-1)$. This holds because if A is at some point after its kth step (and therefore time k/c) then B is at the same point after its kth step (which is time k) since the agents are executing the same algorithm from the same starting position and never detect rendezvous. Therefore A only has time $k - k/c$ to travel away from that point before B arrives. A travels at speed c, so it can travel at most $c(k - k/c) = k(c-1)$ steps in that time.

If the agents never turn then the time it takes for them to cover all distances in $[0, n)$ is trivially $n/(c-1)$. However, the agents are allowed to turn arbitrarily during the algorithm. The fastest way for the agents to cover all distances is for the faster agent A to turn after some small constant distance x (in particular, x may be zero) so that the majority of the time the agents are moving in opposite directions.[2] This gives a bound of $T \leq \frac{x}{c-1} + \frac{n}{c+1}$. By applying our second point

[1] In the original paper Feinerman et al. go to great lengths to formalize this by constructing auxiliary rings and colouring them, but I feel that over-complicates the matter.

[2] When moving in opposite directions the distance between the agents changes at speed

from the previous paragraph, we also get that $T(c-1) \geq n - x$. Combining these inequalities to remove x and then solving for T gives us the bound we are looking for.

$$\hat{t} \geq T \geq \frac{cn}{c^2 - 1}$$

2.2.2 With Pebbles

The results for the case using pebbles are less precise. They first show in two steps that any algorithm in the whiteboard model requires $\max\left\{\frac{n}{2(c-1)}, \frac{n}{c+1}\right\}$ time.

Showing the second term in the maximum is fairly simple. Consider that in time less than $\frac{n}{c+1}$, agent A can only travel distance less than $\frac{c \cdot n}{c+1}$. This means there is a contiguous piece of the cycle of length at least $n - \frac{c \cdot n}{c+1} = \frac{n}{c+1}$ that A cannot visit. Agent B, however, can only travel distance less than $\frac{n}{c+1}$ in that time period, so an adversary can easily locate B such that it never leaves the 'gap' not visited by A.

Showing the first term is somewhat more complicated. Consider the case where the agents are located at distance $n/2$ apart on the ring (that is, they are located symmetrically). Additionally consider the case where the agents execute their algorithm for some i steps and then terminate (note that this is steps, not time, so it takes B longer than A to execute the i steps). It can be shown easily by induction that for $i < \frac{cn}{2(c-1)}$ the resulting situation is still symmetric. Now we can show by contradiction that any algorithm must take at least $\frac{n}{2(c-1)}$ time.

Assume that there is some algorithm which produces rendezvous in time $t < \frac{n}{2(c-1)}$, so the agents meet at time t at some location we will call u. Note that t is trivially less than $\frac{cn}{2(c-1)}$, which means if we were to take t as the number of steps in our previous hypothetical then the resulting situation would still be

$c+1$, whereas when moving in the same direction the distance between them changes only at speed $c-1$.

symmetric.[3] Agent B executes one step per unit of time, which means that in the hypothetical it terminates exactly when reaching u. It therefore follows that in the hypothetical A terminates at time t/c at symmetric distance $n/2$ from u. In the real algorithm, then, A has to travel distance $n/2$ in time $t - t/c$ in order to rendezvous with B at u. However, this implies that $c(t - t/c) \geq n/2$ which contradicts the assumption of $t < \frac{n}{2(c-1)}$, proving the lower bound.

With the lower bound of the white board case shown, Feinerman et al. then give an algorithm using only pebbles that takes an almost-tight time of $\max\left\{\frac{n}{2(c-1)}, \frac{n}{c}\right\}$.[4] In this algorithm, each agent drops a single pebble at its initial location then begins walking clockwise. The first time an agent reaches a pebble, it turns around if the distance it has walked so far is strictly less than $\tau := \min\{n/2, n/c\}$. There are three cases to consider, letting d be the initial clockwise distance between A and B.

1. $d = \tau$

   In this situation neither agent turns around, which means the algorithm effectively devolves into *Distributed Race* which was presented previously. We already showed that distributed race achieves rendezvous in $d/(c-1)$ time, and since d is either $n/2$ or n/c that means the algorithm takes either $n/2(c-1)$ if $c \leq 2$ or $n/c(c-1)$ if $c > 2$, which are both within the stated bounds.

2. $d < \tau$

   In this situation A will turn and B will not. A reaches B's starting point and turns at time d/c, at which point B has travelled distance d/c from that point. When A turns, the two agents have to close a distance of

[3] This step was very confusing in the original paper. The thing to realize is that while t is introduced as a time, the hypothetical requires it to be a number of steps instead, and which units to use must be deduced entirely from context.

[4] It is actually tight for the case $c \geq 2$, since that implies the first term is used, and only the second term differs from the optimal case.

$n - d/c$, which they do at speed $c + 1$. Therefore it takes additional time of $\frac{n-d/c}{c+1}$ to reach rendezvous and the total time is that plus the initial d/c, which works out to $\frac{d+n}{c+1}$. When $d = \tau$ and $c \leq 2$ then the rendezvous time is $\frac{3n}{2(c+1)}$ which is less than $\frac{n}{2(c-1)}$. If $d = \tau$ and $c > 2$ then the rendezvous time works out to n/c. Either way these are valid upper bounds since $d < \tau$ for the current case.

3. $d > \tau$

   In this situation A never turns. If the agents meet before B reaches A's pebble then rendezvous time is at most $n-d$ (since that is the time it would take B to reach A's pebble). Also note that this situation is equivalent to *Distributed Race* which gives us exact time of $d/(c-1)$. Combining these two facts gives us $d/(c-1) \leq n - d$ which can be rearranged to show that the rendezvous time is at most n/c.

   If B reaches A's pebble before rendezvous then things are slightly more complicated. B must turn, because $n - d < \tau$ (to see this consider the two cases $c \leq 2$ and $c > 2$). At this point A has travelled distance $c(n - d)$, so the two agents are distance $n - c(n - d)$ apart. The two agents are closing that distance at speed $c + 1$, giving us a rendezvous time of

$$(n - d) + \frac{n - c(n - d)}{c + 1} = \frac{2n - d}{c + 1}.$$

   Since B turns when it reaches A's pebble $n - d < n/c$, so the rendezvous time is

$$\frac{n + (n - d)}{c + 1} < \frac{n + n/c}{c + 1} = \frac{n}{c}.$$

3 Further Exploration

Having stepped through Feinerman et al.'s original results we now explore other possible applications of their method.

3.1 Multiple Agents with Different Speeds

3.1.1 Possible Models

One of the most obvious directions to explore with this new concept is to simply extend the existing problem to the case of $k > 2$ agents. In this situation there are several obvious possible sub-cases, although only a few of them end up being interesting. If there are no guarantees about the uniqueness of the agents' speeds, for example, then there is little to say as it simply devolves into the situation where there are no speed differences. We consider three potentially interesting models, in increasing order of apparent power:

1. There is at least one agent with a guaranteed unique speed.
2. There is one agent with a guaranteed unique speed, and that speed is either maximal or minimal in the set of agents.
3. Every agent has a unique speed.

3.1.2 Single Unique Speed

In the case where one agent is guaranteed to have a unique speed, but that speed is not necessarily maximal or minimal then our options seem fairly limited. For descriptive simplicity we will call this agent U, though it's important to note that the agents themselves are not aware of this label.

In general it is not obvious that this case wins us anything at all over the case where all the agents have the same speed. The problem is that when two agents meet, comparing their speeds doesn't necessarily tell them anything

about which agent is or is not U. If the agents have different speeds, either one of them could be U or neither of them could be U; it's impossible to tell without having met every other agent. Even remembering the speeds of the agents met so far isn't helpful, since there's no way to tell if you've met two different agents with the same speed, or simply the same agent twice.

This case has the same basic problem as the simpler case where all agents have the same speed: without knowing at least one of k or n there is no way to determine when rendezvous has been reached. If k or n is known to the agents then perhaps the existence of U can be used to give a slightly faster algorithm, but the benefit does not appear likely to be very large.

3.1.3 Maximal Unique Speed

In this slightly stronger case, the agent with the unique speed is also guaranteed to have the maximal speed (or symmetrically the minimal speed). We will denote this agent M.

Unlike the previous case, this case does provide some advantage over the basic case. When two agents meet, they can always learn something about whether one of them is or is not M: if they have the same speed then neither of them is M, since M's speed is unique, and if one is faster then the slower agent is not M, since M's speed is maximal. This information can be used to construct an algorithm that can deterministically reach rendezvous even in the case where neither k nor n are known to the agents.

We now sketch an example of one such potential algorithm, which I will call the *Herding Algorithm*.[5] The herding algorithm, intuitively, uses M to 'herd' all the slower agents to rendezvous, like a sheepdog herding sheep. All agents require $2\log(s(M)) + 2$ bits of memory: enough to store their own speed, the maximum speed of any agent they have encountered so far, and one of four

[5] Whether or not this algorithm is optimal would be an interesting topic for future research.

states.

All agents begin by moving until a rendezvous with some other agent occurs.[6] When two agents rendezvous, they exchange and compare their speeds. Both agents update their maximum encountered speed if necessary. On an agent's first rendezvous the slower agent stops and the faster agent continues (or both stop if they are the same speed). On subsequent rendezvouses the faster agent 'herds' the slower agent towards the location of its own first rendezvous. The algorithm stops when agent M has made a complete cycle after its first rendezvous, since it is then guaranteed to have herded all other agents to that location.

Each agent storing their maximum encountered speed is necessary so that M knows when it is done its cycle, and so that herded slower agents know when they have reached their 'pen'.

3.1.4 All Distinct Speeds

In this case, all agents have unique speeds. This means, of course, that the herding algorithm is still valid, though it may be possible to do better. However, I do not believe that to be the case.[7] For all agents to reach rendezvous, the rendezvous has to occur at a specific spot on the ring, and that spot has to be communicated to or calculated by each agent somehow. It seems that this requires all agents to be in communication with a leader, which is already possible in the previous weaker case. To show that it is possible to do better in this case you would have to show that two agents that are not M can gain more information from a rendezvous if they know their speeds will be distinct, and this is not the case. Formally proving this conjecture would, I think, make

[6] Note that this algorithm does not require orientation. In the worst case this opening step devolves to Feinerman et al.'s *Distributed Race* algorithm.

[7] To clarify, it may be possible to do better than the herding algorithm in the second model, but I don't believe it possible to do better than the second model in the third.

for an interesting paper.

3.2 Leader Election

The leader election problem is of course intimately related to the rendezvous problem. In any of the three cases considered previously then leader election is trivial (assuming rendezvous can be achieved in the first place). This holds with 2 agents or with $k > 2$ agents. The agents simply elect the agent with the highest unique speed.

3.3 Friction

3.3.1 Models and Requirements

Rendezvous when the ring has *friction* is another interesting variation on the original problem, though the model is not quite the same. Instead of assigning different speeds to our two agents, we instead partition the ring into $m \geq 2$ 'patches' and assign each patch p a *coefficient of friction* $f(p) \geq 1$ (where $1 \leq p \leq m$). When an agent is travelling over patch p, its normal speed of 1 is reduced by friction to $1/f(p)$. We will call the points where neighbouring patches meet 'transition points'.

First, we will assume that either the agents already know the frictional coefficient and size of each patch (implying knowledge of n) or that they are able to calculate the frictional coefficient and size of each patch (requiring knowledge of n). In either case the agents require non-trivial memory, a pedometer, and the ability to detect the frictional coefficient of their current location. Without these properties it is not obvious how the agents could use the friction to break symmetry, although the friction may still provide a running-time benefit given some other method of breaking symmetry.

3.3.2 Equivalence with Leader Election

Interestingly, this problem can be solved by solving the leader election problem[8] on one of several auxiliary rings. This is actually fairly easy to see by construction of the auxiliaries in question. Each patch in the base ring becomes a node in the auxiliary, and each transition point in the base ring becomes an edge in the auxiliary. Each auxiliary node is assigned the value either of its associated patch's size or of its associated patch's frictional coefficient. If the agents can solve the general leader election problem on this auxiliary ring then they can elect a leader patch or leader transition point and rendezvous at that location. By equivalence with the *Multiset Sorting Problem*, this is possible if and only if the sequence of frictional coefficients is aperiodic.

3.3.3 Some Interesting Special Cases

While the problem can generally be solved using leader election on an auxiliary ring, this takes non-trivial time and memory. There are two special cases worth mentioning that can be solved more efficiently, although of course both of these cases imply aperiodic frictional coefficients, and as such could be solved by the more general algorithm.

- If there is guaranteed to be at least one patch with a unique frictional coefficient or size then the agents can rendezvous at the midpoint of that patch.[9]

- If there is a single distinct transition point between patches where the delta between the two sizes or frictional coefficients is largest (or, symmetrically, smallest), then rendezvous can occur at this point.

[8] Leader election of a vertex or an edge, not of an agent.

[9] As usual, if multiple patches have unique values then the rendezvous occurs at the patch with the lowest (or, symmetrically, highest) such unique value.

3.4 Outside the Ring

We now turn back to the original simple two-agent rendezvous problem (recall agent A with speed c and agent B with speed 1) but considering the case where the network is not a ring. We consider only the simple application of overlaying a ring on the underlying network and using the algorithms already given by Feinerman et al. More powerful results may be possible by working directly with the underlying topology.

3.4.1 Rendezvous on a Torus

Chapter 6.2 of [4] considers an $n \times m$ oriented torus network. In that monograph, Kranakis et al. provide an algorithm for rendezvous that requires $O(\log n + \log m)$ memory and uses one unmovable token per agent. This algorithm achieves rendezvous in time $O(nm)$ or halts when rendezvous is not possible.

Interestingly, $O(\log n + \log m)$ memory is trivially sufficient for the agents to overlay a ring on the torus.[10] This allows them to run one of the algorithms originally given by Feinerman et al. Consider *Distributed Race*, the slower of these, which takes time $d/(c-1)$ but requires no additional memory. Since the size d of the overlayed ring is nm, this leads to an algorithm that also takes $O(nm)$ time but with several distinct advantages. First, it requires no tokens. Second, for large c this algorithm takes only a very small constant fraction of the time taken by the original algorithm in [4]. Finally, this algorithm is always capable of achieving rendezvous, even in the case where the token-based algorithm cannot.

[10] The agents must simply traverse the current row horizontally then move down a row and repeat. This requires $\log n$ bits to detect when the agent has traversed the current row and must move down. The addition of the $\log m$ bits makes it possible to detect when the agent has traversed every row (and thus the entire 'ring').

3.4.2 Rendezvous on a Tree

Overlaying a ring on a tree is not as simple. Although it is possible via a traversal of the tree, the amount of memory required to perform this traversal is in fact $O(\log^2 n)$ bits as discussed in [5]. This is quadratically larger than the $\Theta(\log n)$ bits required for the rendezvous algorithm discussed in [4]. Additionally, the length of this ring overlay is $2(n-1)$ since there are $n-1$ edges in a tree and each edge is traversed twice. So it seems that at least the simple application of this principle does not provide any benefit over existing algorithms in a tree.

4 Conclusion

This paper has provided a summary and explanation of the work by Feinerman et al. It also explores a number of other models and potential applications for the symmetry-breaking technique they introduced. The herding algorithm we present for solving k-agent rendezvous is worthy of further study, especially if it can be proved that it (or some small variant) is optimal in that model.

The 'friction' model is also interesting, though it appears to be equivalent to an existing heavily-studied problem on auxiliary rings. Here also there is room for future work. In non-ring networks we considered rendezvous by the construction of overlay rings. This simple application has already provided an interesting result on torus networks, though the results for tree networks were less promising. There are still likely interesting results to show when considering the underlying topology directly.

Feinerman et al. mention "many natural ways of further exploring this idea" in their original paper. We have taken some basic steps in a couple of obvious directions, but there are still many problems left unexplored.

References

[1] Jurek Czyzowicz, David Ilcinkas, Arnaud Labourel, and Andrzej Pelc. Asynchronous deterministic rendezvous in bounded terrains. *CoRR*, abs/1001.0889, 2010.

[2] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *CoRR*, abs/1301.7119, 2013.

[3] Ofer Feinerman, Amos Korman, Shay Kutten, and Yoav Rodeh. Rendezvous of agents with different speeds. *CoRR*, abs/1211.5787, 2012.

[4] E. Kranakis, D.D.M. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring: An Introduction*. Synthesis Lectures on Distributed Computing Theory Series. Morgan & Claypool Publishers, 2010.

[5] Evangelos Kranakis and Danny Krizanc. Mobile agents and exploration. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.