Carleton University Honours Project
Knowledge in Rendezvous of Agents with Different Speeds

Evan Huus
eapache@gmail.com
Supervised by Professor E. Kranakis,
Department of Computer Science

April 16, 2014

Abstract

Achieving rendezvous is a common distributed computing problem which can be roughly stated as follows: given k agents in some space, have them move so that they meet ("achieve rendezvous") in the minimal amount of time. This process frequently involves breaking symmetry between the agents; one method of doing this was proposed by Feinerman et al. [2012] and was based on differences in agent speeds. While their paper focused exclusively on the case of two agents, they note in their conclusion that this technique has much broader applications.

In 2013, this author surveyed other applications of the technique of Feinerman et al. as part of an undergraduate course project [Huus, 2013]. The problem was extended to more than two agents in several different ways, and an algorithm was sketched to achieve rendezvous in some of those models. In this paper, we will formally present an improved variant of that algorithm, give some bounds on the theoretic optimal algorithm, and consider several other versions of the problem when the agents have differing capabilities and levels of knowledge.

Acknowledgements

I would like to thank my supervisor, professor Evangelos Kranakis, for his help, support and advice throughout this project.

Contents

Chapter 1

Introduction

1.1 The Rendezvous Problem

The rendezvous problem is an extremely common one in both distributed computing and robotics, and while certain variations on it have been quite well studied (a paper by Alpern [2002] provides a good summary of what had already been accomplished over a decade ago), others are relatively new. In its broadest form, the rendezvous problem can be stated as follows: given k agents in some space, have them move so that they meet in the minimal amount of time. This event is usually known as "achieving rendezvous".

There are of course many versions of this problem, depending on several parameters. The space may be continuous as studied by Czyzowicz et al. [2010], or it may be discrete as studied by Yu and Yung [1996]. It may be a specific shape, such as the ring studied by Kranakis et al. [2010], or it may be a general graph as studied by Dieudonné et al. [2013].

The number of agents and their capabilities also have a substantial effect. Hegarty et al. [2013] explored the scenario where agents can exchange arbitrary information with any agents in their "line of sight", while Sawchuk [2004] considered a much weaker model where the agents were restricted to dropping indistinguishable tokens to mark their current location.

Of particular interest are cases where the agents must all run the same algorithm, which is generally known as the *symmetric rendezvous problem* [Alpern, 2002]. If the agents can run different algorithms (the *asymmetric rendezvous problem*) then the problem is typically much easier, though not always trivial.

1.2 Breaking Symmetry

In many of these variants, rendezvous cannot be achieved without finding some way of breaking the symmetry of the system. For example, in the simple case of two deterministic agents on a ring graph, rendezvous cannot be achieved if the agents are perfectly symmetric; they will always remain exactly the same distance apart. Even in cases where rendezvous can be achieved without it, breaking symmetry often leads to much more efficient algorithms than would otherwise be possible.

The most frequent method for breaking symmetry is to use random bits to make probabilistic choices. Such algorithms provide an expected time to rendezvous, but typically poor worst-case behaviour. There are also many real-world scenarios where no good source of random bits is available; in these cases, various tricks must be used in order to exploit other sources of asymmetry. One such trick is to let the agents drop tokens and count inter-token distances. This method successfully achieves rendezvous as long as the agents do not start equidistant on the ring [Sawchuk, 2004].

Breaking symmetry has applications to other distributed computing problems as well. For example, Schneider and Wattenhofer [2010] developed a new symmetry-breaking technique which gives an exponential speed-up in certain distributed graph colouring problems. The heavily-studied leader election problem is also fundamentally one of breaking symmetry; in order for multiple agents to agree on a single leader, there are many cases where their decisions cannot be symmetric.

1.3 Agent Speed Differences

One source of asymmetry that has recently been receiving more attention is the difference in speed between two otherwise identical agents. While agent speeds have been considered as a problem parameter before, the traditional assumption has been that all agents move at a common fixed speed. Even when agent speed has been allowed to vary, as in the paper by Czyzowicz et al. [2010], agents have typically had a uniform range of possible speeds and have been required to choose their speed deterministically, maintaining the symmetry of the problem.

Recently, Feinerman et al. showed that speed differences can sometimes actually be useful. In their paper, they consider the case of two agents placed on

a ring. The agents are identical in every aspect except their speed, and use this asymmetry to achieve guaranteed rendezvous with good time bounds [Feinerman et al., 2012]. While they focused exclusively on this simple case, Feinerman et al. note in their conclusion that this technique has much broader potential applications, both in rendezvous and in other symmetry-breaking problems.

Following that publication, this author surveyed several other applications of their technique as part of an undergraduate course project on distributed computing. In that project, torus and tree-shaped graphs were explored, as was a variant where the agents have a common speed, but different sections of the ring have varying levels of "friction". Applications to the leader election problem were also touched on [Huus, 2013].

The most interesting direction explored in that project was, however, extending it to more than two agents. Several ways of generalizing the model were discussed, and an algorithm was sketched to achieve rendezvous in the more powerful of those models. In this paper, we will formally present an improved version of that algorithm, give a bound on the optimal algorithm, and consider several variants of the problem when the agents have differing capabilities and levels of knowledge.

1.4 Our Models

Throughout this paper we will consider rendezvous across several different variants of a single core model, similar but not identical to the one used by Feinerman et al. [2012]. In our core model, we have $k \geq 2$ mobile agents placed on a continuous ring of length n. We denote the agents A_1 through A_k for convenience, but emphasize that both the agents and the ring are anonymous. The agents are fully identical except possibly for their speeds.

Each agent A_i has its own maximum speed, denoted s_i. In Feinerman et al. agents could be either stopped or travelling at s_i; we relax this requirement slightly to allow agents to travel at any speed in the interval $[0, s_i]$. Without loss of generality, we normalize $\min(s_1, \ldots, s_k) = 1$ and denote $\max(s_1, \ldots, s_k) = c$, the ratio between the speeds of the fastest and slowest agents.

Agents can travel in either direction on the ring, and can turn without cost. However, the ring is not oriented. Agents can detect when they encounter another agent, as well as whether that agent is travelling in the same or opposite

direction. Co-located agents may exchange arbitrary (though typically small) amounts of information. We wish to explicitly note that this is a deterministic model, and as such agents must have a reason to turn or change speeds, whether that reason is an encounter with another agent, a certain number of steps elapsed, or some other event.

When discussing running-time and optimal bounds, we take an adversarial model where the adversary can choose the initial position and orientation of each agent. The adversary may also choose the maximum speed of each agent, within the restrictions of the speed distributions discussed below.

The above conditions define the portions of our model that remain constant throughout the paper. We generate variants from this base across two dimensions: agent knowledge, and the distribution of agent speeds. In the weakest case of agent knowledge, the agents know nothing except their own speed s_i, and they do not have pedometers or timers. In stronger cases they may know the values of n, k, or c; they may also have pedometers or timers.

The distribution of agent speeds is a more interesting source of differences. We consider the following four possible distributions, sorted in increasing order of apparent power:

Not-All-Identical This model is the weakest possible interesting model. Its only restriction on the adversary's choice of speeds is that there is some pair of agents, A_i and A_j such that $s_i \neq s_j$. No agent is guaranteed to have a unique speed.

One-Unique In this slightly stronger model, the adversary is required to give at least one agent a unique speed, though that speed is not required to be maximal or minimal among the agents.

Max-Unique In this model, the adversary is required to give at least one agent a unique speed, and that speed is required to be maximal (or, symmetrically, minimal) among the agents.

All-Unique In this strongest model, the adversary is required to give each agent a unique speed. Note that since agents are aware of their own speeds, this effectively provides each agent with a unique label, which simplifies a number of problems [Flocchini et al., 2004].

The last component of our model which we want to touch on is the very concept of rendezvous. The obvious, simple definition is that rendezvous occurs when all k agents meet at the same location, but for certain real-world problems this is not quite enough. For example, if some quorum of agents is necessary to execute an algorithm, actually achieving rendezvous is not sufficient if the agents are not aware of that fact. Specifically, if k is not known to the agents then they may not realize that they have achieved rendezvous even when all k agents are present.

Dieudonné et al. [2013] address a more general version of this problem. In their model, all agents have a unique label, and the problem is solved when all agents know all k labels and are aware of that fact. They call this problem Strong Global Learning. Following this terminology, we define the concept of *strong rendezvous* for k agents as the situation where all k agents meet at the same location, and are aware of that fact. Note that in the event of strong rendezvous, all agents are therefore trivially able to calculate k by counting the agents present.

1.5 Our Results

The remainder of this paper is divided into three chapters. In chapter 2 we present the "herding algorithm" for rendezvous of $k \geq 2$ agents in the `Max-Unique` and `All-Unique` models with no additional knowledge. We prove that it achieves rendezvous in time at most $\frac{1}{2}\left(\frac{c+1}{c-1}\right)n$, and that this rendezvous is *strong* in the `All-Unique` model. We also prove that, asymptotically in k, no algorithm can do better than time $\frac{2}{c+3}\left(\frac{c+1}{c-1}\right)n$ in either model.

In chapter 3 we consider variants of the problem with more agent knowledge. We show that knowledge of n is insufficient on its own to make a difference; the algorithm from chapter 2 does not receive any benefits. We also show that knowledge of k is somewhat more interesting; while it does not materially affect the existing algorithm, it does permit the construction of a variant that is capable of achieving rendezvous even in the weaker `One-Unique` model. This variant is presented and a proof of correctness is given, but the running-time is not analyzed.

The cases where the agents have pedometers or know the value of c are also discussed. As with knowledge of n, these additions are insufficient on their

own to provide any obvious advantage. However, we show that both of these elements can be used by the agent to gain useful knowledge, and we conjecture that this knowledge may lead to improved algorithms for certain special cases. Finally, we consider the combined case where the agents have a pedometer and know both c and n. We use this knowledge to construct a variant of the herding algorithm which takes $n/(c^2 - 1)$ less time than the original.

Finally, we present our conclusion in chapter 4 and consider possible areas for future work.

Chapter 2

The Herding Algorithm

In this chapter we present the basic herding algorithm, which achieves rendezvous in the `Max-Unique` model and strong rendezvous in the `All-Unique` model while requiring no additional agent knowledge. As this chapter deals strictly with those two stronger models, we introduce one extra notational convenience: we define A_{max} to be the agent with the unique maximal speed.

We prove the correctness of the presented algorithm, analyze its running-time, and prove a lower bound on the running-time of the optimal algorithm. While our algorithm was developed independently, it turns out that it shares some similarities with a randomized algorithm proposed by Steve Alpern in section 7.4.1 of his paper [Alpern, 2002]. That algorithm belongs in the same family as the one presented here, but to the best of our knowledge was never published in full. Due to its use of random bits, it provides only a probabilistic running-time, as opposed to the guaranteed running-time of our algorithm.

2.1 Presentation of the Algorithm

Intuitively the herding algorithm is quite simple. Agents begin to move until they encounter another agent. When that occurs, the two agents attempt to circumnavigate the ring in opposite directions, in order to "herd" all the other agents to rendezvous. When two herding agents meet, the agent with the faster remembered speed dominates, meaning that eventually A_{max} and some partner will herd all other agents to a single rendezvous point.

Formally, the algorithm requires each agent to maintain in memory a simple state machine consisting of four states. We denote the current state of A_i as σ_i. The algorithm also requires each agent to remember a single agent speed (not necessarily its own), which we denote m_i.

So far we have said nothing bounding the size of agent speeds; since c represents the largest ratio, not the largest absolute speed, the intuitive $\lceil \log c \rceil$ has no actual relation to the amount of memory agents require to store m_i. However, the algorithm only needs enough precision to do comparisons on the speeds. Therefore we can take $\lceil \log k \rceil$ as a bound, since there cannot be more than k distinct speeds in any model. As the state machine requires only a constant number of bits to store one of the five states, this gives us a total memory requirement of $O(\log k)$ bits per agent.

Since there are only four states, we provide an intuitive name and description for each one:

Searching Agents begin in this state, and while in it they simply move in their current direction at their maximum speed until they encounter another agent. At this point they transition to some other state and continue the algorithm. Once an agent has left this state, it never re-enters it.

Herding Agents enter this state when they have reason to believe that they are the fastest agent, A_{max}. This occurs when they have encountered at least one other agent, and all agents encountered so far have been slower than themselves. While in this state, agents attempt to herd all other agents to rendezvous by circumnavigating the ring. They travel at their maximum speed or, if other agents are travelling with them, the maximum speed achievable by the group.

Herded Agents enter this state when they become aware of an agent faster than themselves, and therefore know that they are not the fastest agent. While in this state, agents are moving towards rendezvous with the fastest agent they are currently aware of. As in `Herding` they travel at their maximum speed or, if other agents are travelling with them, the maximum speed achievable by the group.

Penned Agents stop and enter this state when they have reached what they believe to be the rendezvous location. Agents in this state are not moving. Agents in this state may transition back to `Herded` if they are reached by

an even faster agent (thus invalidating their current rendezvous location) or they may terminate if they learn conclusively that their current location is correct.

Initially, each agent sets σ_i to `Searching`, and m_i to s_i. When encountering another agent, the two agents transmit their values for s_i, m_i and σ_i. The agents perform a state transition if necessary, and store the largest of all communicated speeds as their new value of m_i. If an encounter involves more than two agents, every distinct pair of agents present performs this process in arbitrary order.

The rules for state transitions are relatively straightforward, though not trivial. For convenience, we denote the two agents in an encounter as A and B, and without loss of generality we assume $m_A > m_B$, meaning that A's stored speed is initially greater than B's stored speed. The transitions for agent A are as follows:

- If A is in state `Searching` then it transitions to state `Herding`. Agent A turns and begins moving at its maximum speed in the opposite direction.
- Otherwise, agent A does not change its state or direction. If A is in either state `Herding` or state `Herded`, and agent B is slower than any other agent present, then agent A reduces its current speed to match.

The transitions for agent B are slightly more complex:

- If A is in state `Searching` then B transitions to state `Herded`. Agent B begins moving at maximum speed in the direction opposite to A.
- If A is in state `Penned` then B transitions to state `Penned` and stops moving.
- Otherwise, agent B transitions to state `Herded`. It begins moving in the same direction as A, at the smallest speed present in the group.

The case where $m_B > m_A$ is perfectly symmetric, which leaves only the case where $m_A = m_B$ (the agents have the same stored speed). In this case, both agents stop and enter state `Penned`. If an agent enters state `Penned` in the `All-Unique` model, it knows that rendezvous has been achieved and that it may terminate.

2.2 Proof of Correctness

Theorem 1. *When running the herding algorithm in the* `All-Unique` *or* `Max-Unique` *models, all agents eventually stop at the same location on the ring, achieving rendezvous.*

Proof. To see this, we must follow the state transitions of the fastest agent, A_{max}. Like all agents, it starts in state `Searching` and begins moving at maximum speed. Since it is the fastest agent and has a unique speed, by moving at maximum speed it is guaranteed to encounter another agent eventually. At this encounter, following the state transition rules, it transitions to state `Herding` and begins moving at its maximum speed in the opposite directions. The agent it encountered transitions to state `Herded` and does the same. At this point, both agents have s_{max} stored in memory.

Here we know several important facts. First, the two agents are moving towards each other around the entire ring from their initial point of encounter. This means that all other agents must be between them in their direction of movement. Second, we know that no other agent has s_{max} in memory. Agent A_{max} has a unique speed, and has only encountered a single agent, thus only those two agents have s_{max} in memory.

Now consider the segment of ring that lies between the two agents. As the two agents move, the segment gets smaller and each other agent will encounter either A_{max} or its partner. At each such encounter, the agent will be dominated (since its stored speed must be less than s_{max}) and so it transitions to state `Herded` and joins in the circumnavigation of the ring.

Eventually, A_{max} and its partner will meet again. At this point all agents stop and transition to state `Penned`. Rendezvous has been achieved. □

Theorem 2. *In the* `All-Unique` *model, the herding algorithm results in strong rendezvous.*

Proof. Due to theorem 1 we know that all agents must enter state `Penned` when rendezvous is reached. Now we simply show that no agent can enter that state prior to rendezvous being reached. Agents can then know that rendezvous has been reached when they transition to state `Penned`, making the rendezvous strong. Consider some agent A_i. For A_i to enter state `Penned` it must encounter

some other another agent A_j such that either $m_i = m_j$ (case 1), or $m_i < m_j$ and $\sigma_j = \texttt{Penned}$ (case 2).

Since all agents speeds are unique in this model, the first case can only occur when the two agents are connected by some chain of previous encounters. This implies that their stored speed is maximal among all agents in the chain, which means that they are both participating in the same circumnavigation of the ring (headed by the agent A_x such that $s_x = m_i = m_j$). A_x must have encountered some other agent at this point, so must be in state `Herding`.

Since an agent that is participating in a circumnavigation joins the other agents in that process and travels in lockstep with them, A_i and A_j cannot both be travelling in the same "prong" of the herd. This implies that the circumnavigation instigated by A_x must be complete when A_i encounters A_j, meaning that rendezvous has been achieved and that $A_x = A_{max}$.

The second possible case ($m_i < m_j$ and $\sigma_j = \texttt{Penned}$) is handled by induction; since there is no agent faster than A_{max}, that agent must only enter state `Penned` when rendezvous is really achieved. Given this, it must be true for the second-fastest agent as well, and so on for the third-fastest, etc. □

Theorem 3. *In the `Max-Unique` model, the herding algorithm cannot result in strong rendezvous.*

Proof. To show that an agent may enter state `Penned` prematurely in the `Max-Unique` model, consider some agent A_i that is not A_{max}. As in theorem 2, for A_i to enter this state it must encounter another agent A_j such that $m_i = m_j$, or $m_i < m_j$ and $\sigma_j = \texttt{Penned}$. But unlike in theorem 2, the first of these cases can happen in any number of trivial scenarios where rendezvous hasn't been reached. Since an arbitrary number of agents can have the same speed as long as that speed isn't maximal, an adversary can simply place two agents facing each other with the same speed to start.

If an agent enters state `Penned` prematurely in this model, then both agents will transition back into state `Herded` when an agent knowing some larger speed comes along. As such, no agent can never know that some agent with an even larger speed won't eventually arrive and wake it up again. □

Conjecture. We expect that in the `Max-Unique` model, it is impossible to achieve strong rendezvous with any algorithm.

2.3 Time Analysis

Theorem 4. *The herding algorithm finishes in time at most* $\frac{1}{2}\left(\frac{c+1}{c-1}\right)n$.

Proof. The algorithm can be analyzed in two parts. The first part consists of the time before A_{max} encounters any other agent. Since A_{max} is moving at its full speed this devolves in the worst case into the "Distributed Race" discussed by Feinerman et al. [2012]. This part takes at most $n/(c-1)$ time.

The second part is the time for A_{max} and its partner to herd all other agents to rendezvous after their initial meeting. The two agents travel the entire ring (size n) in opposite directions, and each trivially moves with a speed of at least 1. Therefore this step takes time at most $n/2$. Putting the two together we get:

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

□

Remark 1. This bound implies that if the agents have knowledge of n, knowledge of c, and some ability to count time, then strong rendezvous is possible even in the `Max-Unique` model simply by achieving normal rendezvous and waiting for $\frac{1}{2}\left(\frac{c+1}{c-1}\right)n$ time to elapse, thus guaranteeing that there are no other agents left.

Remark 2. The algorithm actually achieves a slightly faster running-time than proved here, because the longer the first part (the distributed race) takes, the shorter the second part (the herding) must take. Assume the first part takes time t. Then any agent with speed 1 must be at least distance $t(c-1)$ "behind" A_{max} at the point of its first encounter. Therefore when it turns, it must spend at least time $t\frac{c-1}{c+1}$ travelling faster than speed 1, so the second part of the algorithm (the herding) must take less than $n/2$ time. Unfortunately, we have not found a concise equation for representing this total tighter running-time.

Remark 3. When $k = 2$ we can give a slightly tighter bound because when circumnavigating the ring there are no other agents to encounter. This means agent A_{max} has no reason to slow down, and ends up travelling at speed c the entire time. The resulting circumnavigation takes time exactly $n/(c+1)$, leading to a bound of

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

2.4 Optimality

In order to prove a useful bound on the optimal algorithm, we first need a few intermediate results whose utility will eventually become clear.

Lemma 1. *An agent may not turn or change its speed except when encountering another agent.*

Proof. We must first observe that agents have very little information to work with. In fact, an agent A_i initially knows only its own speed, s_i and has no way of counting time elapsed or distance travelled. Since the model is deterministic as we noted in section 1.4, this means that agents have a severely limited set of criteria on which they are able to make decisions. In fact, there are only two kinds of events which change the agent's memory and permit it to make new decisions: the beginning of the algorithm, and encountering another agent.

On this argument, an agent may turn or change speeds when encountering another agent (as stated in the lemma) *or* at the very beginning of the algorithm. But an agent's behaviour at the beginning of the algorithm is strictly predictable; since each agent has initial access to only one variable (their own speed) then any condition embedded in the algorithm must involve some constant test. If the condition is constant, however, then it is trivial for the adversary to construct an otherwise-identical example where all agents fail the condition and do not turn or change speeds. Therefore without loss of generality we can assume agents do not turn or change speeds at the beginning of the algorithm. □

We now consider a particular initial layout of agents that is available to the adversary. In this layout, the adversary places the agents clumped together on the ring facing the same direction. The clump is tight (the two "outside" agents are only a tiny distance, ϵ, apart) and the agents are ordered by their speed such that the fastest agent, A_{max} is at the "head" of the clump and the slowest agent is at the tail. The speed of each agent A_i is chosen as $s_i = 1 + \frac{(i-1)(c-1)}{k-1}$. For convenience we shall name this layout "alpha".

Lemma 2. *When starting from layout alpha, no algorithm can achieve the first agent encounter faster than* $(n - \epsilon)/(c - 1)$.

Proof. The proof of this lemma follows trivially from lemma 1. No agent can turn or change its speed prior to the first encounter, so the fastest way of achieving rendezvous is for all agents to travel at maximum speed, taking time $(n-\epsilon)/(c-1)$. This is effectively the Distributed Race algorithm from Feinerman et al. [2012]. □

Note that since first encounter *is* rendezvous when $k = 2$, this is trivially the optimal algorithm. We therefore only consider the case $k > 2$ going forward.

Lemma 3. *When the agents are started in layout alpha, then at the time of first rendezvous the agents are spread equidistant at* $k - 1$ *locations along the ring (we call this layout "beta").*

Proof. First note that by definition we have $s_1 = 1$ and $s_k = c$. From lemma 2 we know it takes time $(n-\epsilon)/(c-1)$ to reach first contact. At this point, A_1 and A_k have encountered one another and occupy the same location. We also know that each A_i has travelled total distance $s_i(n - \epsilon)/(c - 1)$. For any agent A_i, $1 < i \leq k$, compare the distance it has travelled against the distance travelled

by the neighbour starting immediately behind it, A_{i-1}. This difference is:

$$\begin{aligned}
\frac{s_i(n-\epsilon)}{c-1} - \frac{s_{i-1}(n-\epsilon)}{c-1} &= \frac{n-\epsilon}{c-1}\left(1+\frac{(i-1)(c-1)}{k-1}\right) - \frac{n-\epsilon}{c-1}\left(1+\frac{(i-2)(c-1)}{k-1}\right) \\
&= \frac{n-\epsilon}{c-1}\left(\frac{(i-1)(c-1)}{k-1} - \frac{(i-2)(c-1)}{k-1}\right) \\
&= \frac{n-\epsilon}{c-1}\left(\frac{(c-1)((i-1)-(i-2))}{k-1}\right) \\
&= \frac{(n-\epsilon)(c-1)(i-1-i+2)}{(c-1)(k-1)} \\
&= \frac{n-\epsilon}{k-1}
\end{aligned}$$

As A_1 and A_k occupy the same location, there are agents at $k-1$ locations on the ring, each one a distance of $(n-\epsilon)/(k-1)$ from the previous. This is equidistant within the variance of ϵ. □

We ignore the ϵ term going forward as it complicates the equations without materially affecting the result.

Lemma 4. *In position beta, every agent A_i has some other agent A_j at distance at least $\frac{n}{2}\left(\frac{k-2}{k-1}\right)$ from it. Without loss of generality, $i<j$ and $j = i+\lfloor(k-1)/2\rfloor$.*

Proof. Since the agents are equidistant in position beta, we have two cases. When $k-1$ is even, each agent A_i $(1 \le i < (k-1)/2)$ has another agent A_j exactly opposite it on the ring, where $j = i + \frac{k-1}{2}$. This agent is at distance $n/2$ from A_i, and A_i is at distance $n/2$ from it. The lemma holds since $n/2 > \frac{n}{2}\left(\frac{k-2}{k-1}\right)$.

If $k-1$ is odd then the point opposite A_i $(1 \le i < k/2)$ is equidistant between agent A_{j-1} and agent A_j where $j = i + k/2$. These agents are neighbours and are thus at distance $\frac{n}{k-1}$ from each other. Both of these agents are therefore at distance $(n - \frac{n}{k-1})/2 = \frac{n}{2}\left(\frac{k-2}{k-1}\right)$ from A_i. □

Lemma 5. *Asymptotically in k, no algorithm can achieve rendezvous in time better than $\frac{n}{c+3}$ from position beta.*

Proof. Consider any agent on the ring. From lemma 4 we know it must have a roughly-opposite partner agent. These two agents have a combined speed of:

$$\begin{aligned} s_i + s_j &= 1 + \frac{(i-1)(c-1)}{k-1} + 1 + \frac{(j-1)(c-1)}{k-1} \\ &= 2 + \frac{c-1}{k-1}((i-1) + (j-1)) \\ &= 2 + \frac{c-1}{k-1}((i-1) + (i + \lfloor (k-1)/2 \rfloor - 1)) \\ &= 2 + \frac{c-1}{k-1}(2(i-1) + \lfloor (k-1)/2 \rfloor) \end{aligned}$$

Also from lemma 4 we know they must be at least distance $\frac{n}{2}\left(\frac{k-2}{k-1}\right)$ apart. This means that even if they were to head towards each other at their maximum respective speeds, they cannot rendezvous in better than time:

$$\frac{\frac{n}{2}\left(\frac{k-2}{k-1}\right)}{2 + \frac{c-1}{k-1}(2(i-1) + \lfloor \frac{k-1}{2} \rfloor)}$$

This value is obviously decreasing in i, and so must be smallest when $i = 1$. This lets us simplify to:

$$\frac{\frac{n}{2}\left(\frac{k-2}{k-1}\right)}{2 + \frac{c-1}{k-1}\lfloor \frac{k-1}{2} \rfloor}$$

Asymptotically in k (as k goes to ∞), this simplifies further to:

$$\begin{aligned} \frac{\frac{n}{2}\left(\frac{k-2}{k-1}\right)}{2 + \frac{c-1}{k-1}\lfloor \frac{k-1}{2} \rfloor} &= \frac{n/2}{2 + \frac{c-1}{2}} \\ &= \frac{n}{2(2 + \frac{c-1}{2})} \\ &= \frac{n}{4 + c - 1} \\ &= \frac{n}{c+3} \end{aligned}$$

Therefore, asymptotically in k, no algorithm can rendezvous in time better than $\frac{n}{c+3}$ from position beta. □

Theorem 5. *Asymptotically in k, no algorithm can achieve rendezvous in time better than* $\frac{2}{c+3}\left(\frac{c+1}{c-1}\right)n$.

Proof. This result follows simply from lemmas 1, 2 and 5. Given starting position alpha, any algorithm must take at least time $n/(c-1)$ to reach position beta. No algorithm can do better than time $n/(c+3)$ to achieve rendezvous from position beta. Together they sum:

$$\begin{aligned}
\frac{n}{c-1}+\frac{n}{c+3} &= \frac{n(c+3)+n(c-1)}{(c-1)(c+3)} \\
&= \frac{cn+3n+cn-n}{(c-1)(c+3)} \\
&= \frac{2cn+2n}{(c-1)(c+3)} \\
&= \frac{2n(c+1)}{(c-1)(c+3)} \\
&= \frac{2}{c+3}\left(\frac{c+1}{c-1}\right)n
\end{aligned}$$

□

Chapter 3

Stronger Models

In this chapter we consider variants of the problem with more agent knowledge. We consider the cases where the agents know the values of n, k, or c. We also consider the case where the agents have pedometers, and the case where agents have pedometers as well as knowledge.

We construct two variants of the herding algorithm presented in chapter 2. One depends on knowledge of k but is capable of achieving rendezvous even in the weaker `One-Unique` model. The other uses a result of Feinerman et al. to reduce the running-time of the original algorithm by $\frac{n}{c^2-1}$, but requires the agents have pedometers as well as knowledge of n and c.

3.1 Knowledge of n

Giving the agents knowledge of n seems to be relatively useless on its own. The agents do not have pedometers or timers, so they still cannot tell when they've traversed any given fraction of the ring. As the value of n is the same for each agent, it does not provide any asymmetry in its own right, and combining it in some way with an agent's speed provides no more asymmetry than the speed would alone.

Of particular interest is that lemma 1 still holds (with effectively the same proof) when the agents know n. The rest of section 2.4 follows unchanged, meaning that theorem 5 also holds in this model.

3.2 Knowledge of k

Unlike knowledge of n, knowledge of k is obviously useful. It trivially permits the herding algorithm to achieve strong rendezvous in the `Max-Unique` model, while theorem 3 showed that this was impossible otherwise. In fact, knowledge of k makes strong and normal rendezvous equivalent; if normal rendezvous is achieved in any circumstance, the agents can simply count how many other agents are present to make the rendezvous strong.

In this author's previous paper [Huus, 2013], it was conjectured that rendezvous could not be achieved in the `One-Unique` model without some additional agent knowledge. Here we present a variation of the herding algorithm which achieves rendezvous in this model, given that the agents know k and have a non-trivial amount of memory. We call this the "non-maximal variant".

As in chapter 2 we introduce one additional notational convenience. Since there may be multiple agents with the fastest speed in the `One-Unique` model, the label A_{max} is no longer useful. Instead, there is at least one agent who's speed is unique even if it is not necessarily maximal. We therefore denote this agent A_U. We emphasize that, like A_{max}, the agents are not aware of this label.

3.2.1 The Non-Maximal Herding Algorithm

In the non-maximal variant of the herding algorithm, each agent is required to store the set of all speeds it has encountered so far, as well as whether it believes each of those speeds is unique. As there are k agents potentially each with a unique speed, this requires at least $k(1 + \lceil \log k \rceil) = O(k \log k)$ bits of memory per agent.

Each agent requires an additional 3 bits to store one of five states. Four of these states should be familiar from chapter 2; `Searching`, `Herding`, `Herded`, and `Penned` are all nearly the same. The new state we call `Wandering`; when an agent knows it is not A_U but does not have any other information, it begins `Wandering` simply by moving at its maximal speed in some direction; this is necessary to prevent the algorithm from getting stuck.

Agents initialize their memory in the obvious way; their set of speeds contains only their own speed, and that speed is currently believed to be unique. They start in state `Searching`, and behave as expected in that state; each agent

simply begins moving at its maximal speed. Since A_U's speed is unique, it is guaranteed to encounter another agent at some point.

When two agents meet (call them A_i and A_j) they first count the number of agents present and check against k; if rendezvous has been achieved then the agents terminate. Otherwise they exchange their set of known speeds and their state. If A and B have the same speed ($s_A = s_B$) then clearly that speed is not unique and is marked as such by both agents. If one of the agents already has the other's speed in memory, then that speed is also marked as not unique by both. If either agent has seen a speed that the other has not, then that speed and its corresponding uniqueness flag is copied.

At this point the two agents have the same set of speeds in memory. If the agents do not know of any unique speeds (for example they have the same speed, and have only encountered each other) then the agents begin to `Wander`. Otherwise what happens next depends on the agents' states. Assume without loss of generality that prior to the encounter, the speed that both agents now believe to be maximally unique was stored in the memory of A_i (either because it was A_i's speed or because A_i had already encountered an agent knowing that speed).

The transitions here are very similar to the normal algorithm. If $\sigma_i \neq$ `Herding` and the maximal unique speed is s_i, then A_i and A_j both transition to state `Herding` and begin moving at their respective maximum speeds in opposite directions. Otherwise A_j transitions to state `Herded` and beings moving in tandem with A_i. Agent A_i does nothing except potentially slow down to permit A_j to keep up.

Theorem 6. *The non-maximal variant of the herding algorithm achieves strong rendezvous in the* `One-Unique` *model.*

Proof. First note that we need only show that normal rendezvous is achieved. As k is known to the agents, normal rendezvous trivially implies strong rendezvous.

Unlike the normal herding algorithm, where the correctness of the algorithm was relatively intuitive, the correctness of the non-maximal variant is not. To see it, we must follow what happens to the fastest agent (or any one of the fastest, if there are multiple agents with speed s_{max}). We will show that this agent must eventually either:

- realize that its speed is not unique, or

- bring all agents to rendezvous if its speed is unique.

We will then show that this property applies inductively; once it is true for all agents faster than A_i it must eventually become true for A_i. It therefore follows that eventually it will become true for A_U. Since A_U's speed is the largest such speed that is unique, it must eventually bring all agents to rendezvous.

Now consider what happens to some fastest agent A_{max} as the algorithm executes. There must be some agent which is slower than it (if not then all agents have speed s_{max} which contradicts our assumption of at least one agent with a unique speed), so it must eventually encounter some other agent. When this occurs there are two cases. The other agent may already have s_{max} in memory, in which case A_{max} realizes its speed is not unique and we are done. If not then they both transition to state `Herding` and begin moving in opposite directions.

Now A_{max} must eventually encounter another agent with s_{max} in memory. Since every other agent is between it and its partner, we again have several cases:

- If there is no other agent with speed s_{max} then it will eventually encounter its herding partner again with all k agents present, achieving rendezvous.
- If there is some other agent with speed s_{max} then that agent (or an agent it has encountered) must encounter A_{max} first, at which point A_{max} realizes that its speed is not unique.

This suffices to show our base case. Our inductive hypothesis is that for some agent A_i, all faster agents know themselves to not have a unique speed. Now we can take the inductive step.

If A_i already knows its speed isn't unique then we are easily done, so we consider the case where A_i believes its speed to be unique (though not necessarily maximally so). If A_i still believes that some other faster speed is unique then it will continue moving in state `Herding` or `Herded` until it encounters an agent which can tell it otherwise. Once it believes that its own speed is both maximal and unique, then the process it follows is identical to the process described in detail for A_{max} above. On its next encounter it either knows it is not unique, or it begins `Herding`. Once in that state it either herds all agents to rendezvous, or must encounter another agent with the same speed, proving it is not unique. □

3.3 Knowledge of c

When we discuss knowledge of c we specifically do not mean knowledge of the actual fastest speed; as we have already seen, c is defined as the normalized ratio between the fastest and slowest speed. If the agents instead know the actual fastest speed then this leads to a trivial algorithm: all agents know whether or not they are the fastest or not, so the fastest stays put and all other agents move to join. This takes time at most n and requires no memory.

Knowledge of c (the ratio) can potentially be useful in its own right however. Specifically, an agent A_i can use just its knowledge of its own speed s_i combined with its knowledge of c to determine the range of valid speeds for all other agents: no agent can possibly have a speed slower than s_i/c, and no agent can possibly have a speed faster than $s_i c$. When two agents meet, they can exchange bounds and narrow the window of valid speeds. If a fastest agent (with normalized speed c) meets a slowest agent (with normalized speed 1) then they both immediately know that they are the fastest/slowest agents, respectively.

While this knowledge does not obviously lead to a better general-case algorithm in any model, it does potentially lend itself to improvements in specific cases.

3.4 Using Pedometers

Like knowledge of n, giving agents pedometers on their own is relatively useless without something to compare against. However, pedometers do give us one small advantage: since agents know their own speed, the two can be used in combination to construct a timer. As with knowledge of c, this does not obviously lead to any improvements.

3.5 Pedometers and Knowledge

While knowledge of n (section 3.1), knowledge of c (section 3.3) and pedometers (section 3.4) have not proved particularly fruitful lines of enquiry on their own, in combination they produce a much more interesting result. Feinerman et al. proved in their paper that the optimal two-agent rendezvous in this case required time $\frac{cn}{c^2-1}$ [Feinerman et al., 2012]. Using this as a component, we can construct a second variant of the original herding algorithm, which we will call the "fast herding algorithm".

3.5.1 The Fast Herding Algorithm

The fast herding algorithm is in almost all respects identical to the herding algorithm presented in chapter 2. The only difference is in the initial step. Instead of using the plain "distributed race" of Feinerman et al. we have each agent execute the optimal two-agent algorithm given as part of theorem 1 in that paper. As the two-agent case is the degenerate one, this guarantees that A_{max} will reach its first encounter in time at most $\frac{cn}{c^2-1}$. The proof of correctness given for theorem 1 can be trivially adapted to show that this fast variant also achieves rendezvous. More interesting is the running-time.

Theorem 7. *In the* `Max-Unique` *and* `All-Unique` *models, the fast herding algorithm achieves rendezvous in time that is* $\frac{n}{c^2-1}$ *faster than the normal herding algorithm, specifically in time* $\frac{1}{2}\left(\frac{c+1}{c-1}\right)n - \frac{n}{c^2-1}$.

Proof. The proof of this follows the same form as the proof of theorem 4. The initial encounter for A_{max}, as we have seen, can take at most $\frac{cn}{c^2-1}$. The herding to rendezvous after that initial step takes at most time $n/2$. Adding together we get:

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

□

Remark 4. Following remark 1 we can make a similar point here. Since the agents do have knowledge of n, c and a pedometer (which we mentioned in section 3.4 could be used to construct a timer) this implies that even in the `Max-Unique` model, strong rendezvous is achievable by the fast herding algorithm given enough memory. Agents simply run the algorithm and then wait for their timer to expire.

Remark 5. As with the original algorithm (see remark 3) we can give a slightly tighter bound when $k = 2$.

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

Chapter 4

Conclusion

In this paper we have studied the rendezvous problem for $k \geq 2$ agents on the ring, using differences in agent speed to break symmetry and achieve rendezvous with good guaranteed running-times. We extended the simple $k = 2$ case to four different $k \geq 2$ models, and presented the herding algorithm which achieves rendezvous in the two stronger of these models. We also prove a bound on the optimal algorithm in those models.

We further studied the cases where agents had knowledge or capabilities above and beyond those available in the base model. We present two variants on the herding algorithm, one of which makes use of additional knowledge to achieve a better running-time, and one of which is capable of achieving rendezvous even when guarantees about agent speed distribution are weakened.

As with any new and active field, there are many open questions for future research. In particular, this paper has generate several new unsolved problems of interest. Remark 2 suggests that a better running-time bound can be proven for the herding algorithm; a tighter bound on the optimal algorithm also seems like it might be possible.

We additionally conjectured that strong rendezvous is impossible in the `Max-Unique` model without additional knowledge. Theorem 6 showed the contrapositive of this (that strong rendezvous is possible when given additional knowledge), and theorem 3 showed that strong rendezvous is not possible with the original herding algorithm. We suspect a variant of that argument would suffice to show this conjecture.

Bibliography

Alpern, S. (2002). Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795.

Czyzowicz, J., Ilcinkas, D., Labourel, A., and Pelc, A. (2010). Asynchronous deterministic rendezvous in bounded terrains. *CoRR*, abs/1001.0889.

Dieudonné, Y., Pelc, A., and Villain, V. (2013). How to meet asynchronously at polynomial cost. *CoRR*, abs/1301.7119.

Feinerman, O., Korman, A., Kutten, S., and Rodeh, Y. (2012). Rendezvous of agents with different speeds. *CoRR*, abs/1211.5787.

Flocchini, P., Kranakis, E., Krizanc, D., Santoro, N., and Sawchuk, C. (2004). Multiple mobile agent rendezvous in a ring. In Farach-Colton, M., editor, *LATIN 2004: Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 599–608. Springer Berlin Heidelberg.

Hegarty, P., Martinsson, A., and Zhelezov, D. (2013). A variant of the multi-agent rendezvous problem. *CoRR*, abs/1306.5166.

Huus, E. (2013). Breaking symmetry with agents of different speeds.

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

Sawchuk, C. (2004). *Mobile Agent Rendezvous in the Ring*. PhD thesis, Carleton University.

Schneider, J. and Wattenhofer, R. (2010). A new technique for distributed symmetry breaking. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 257–266, New York, NY, USA. ACM.

Yu, X. and Yung, M. (1996). Agent rendezvous: A dynamic symmetry-breaking problem. In Meyer auf der Heide, F. and Monien, B., editors, *ICALP*, volume 1099 of *Lecture Notes in Computer Science*, pages 610–621. Springer.