

Providing Guaranteed Behaviors for Groups of Low-Capability Mobile Agents

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Luther A. Tychonievich

July 2013

Abstract

I consider the problem of designing algorithms for coordinated groups of low-capability, error-prone mobile agents. It is my thesis that some important collective behaviors of groups of low-capability mobile agents can be guaranteed by the application of provably-correct distributed algorithms. This thesis is demonstrated by the provision of new algorithms that guarantee mobile agent behavioral properties in the face of noise and agent error, as well as proofs that characterize the requirements of particular tasks.

The contributions of this dissertation include both algorithms for solving classes of tasks that are foundational to processes involving groups of mobile agents and proofs regarding the impact limited capabilities have on the ability of agents to perform various tasks.

The capability proofs lie in two principle areas. The first set of proofs relate to the differences between stigmergic and broadcast communication; these proofs bound the time and number of additional agents required to emulate broadcast communication using stigmergy. The second set of proofs bound the capabilities needed to ensure that agents are able to locate one another in an unknown environment. In addition to these stand-alone proofs, there are also proofs accompanying each algorithm presented.

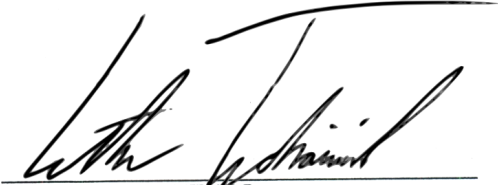
The main classes of tasks solved relate to agents forming and maintaining a group. A family

of algorithms is provided, each of which guarantees rendezvous in bounded time for some class of agent capabilities. For agents with bounded-error knowledge of time and place these rendezvous algorithms are within a logarithmic term of being asymptotically optimal. A technique is presented for generating pair-wise cohesion constraints for various agent types; these constraints provide each agent with a set of allowable behaviors that provably keep the agents connected. A set of related communication protocols achieve global cohesion using an underlying pair-wise cohesion algorithm, allowing swarms of agents to move cohesively without being constrained by connection topology. Finally, a set of protocols is presented for having agents form into groups even when the agents do not trust one another.

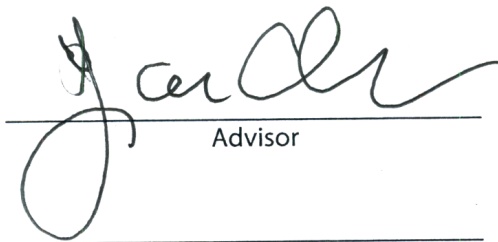
Taken together, the algorithms and proofs in this dissertation contribute to the understanding of how agent capability and behavior are related, significantly improve the scope of problems that can be solved via provable algorithms, and improve several existing time bounds.

APPROVAL SHEET

Providing Guaranteed Behaviors for Groups
of Low-Capability Mobile Agents
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy (Computer Science)

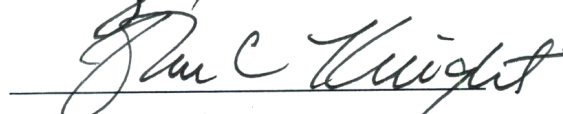


AUTHOR

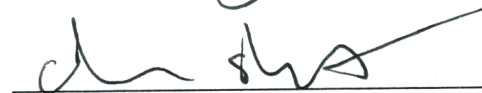


Advisor

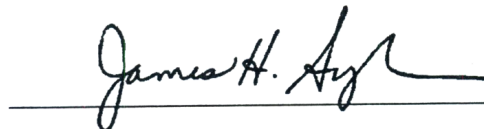








Accepted for the School of Engineering and Applied Science:



Dean, School of Engineering and Applied Science

This dissertation is dedicated to Markham.

Acknowledgements

I cannot imagine a better dissertation advisor than James P. Cohoon. He accepted my scattered interests and allowed me to pursue them while also helping to continually refocus me on the task of completing my degree. He was ever willing to find the leaks in incomplete proofs and to suggest alternative viewpoints. Wherever a particularly sound argument appears in this dissertation, James Cohoon's handiwork is evident.

The members of my examining committee each provided helpful comments that improved the quality of my writing. Worthy Martin provided encouragement early on that helped me define my dissertation scope appropriately. John Knight enthusiastically encouraged me to consider additional ways my research could be applied; though few of his suggestions made it into this dissertation, they will likely steer the future of this work. abhi shelat was instrumental in helping me see ways that my theorems and proofs could be clearer. And I am particularly indebted to Ellen Bass for her extensive suggestions on wording and presentation; almost every page of this dissertation has been improved by her suggestions.

It was my father, Louis P. Tychonievich, who first suggested to my mind the possibilities of mobile agents and phenomenological artificial intelligence. Many are the evenings I stood with him by the white board, talking about maze navigation, collision avoidance, and combining

multiple objectives and constraints into a single behavioral policy.

My time as a masters' student at Brigham Young University was invaluable, being my first experience with research and a broad introduction in many projects. My advisor, Robert P. Burton, gave me the freedom to begin exploring AI formally within the context of my thesis on simulations for multidimensional time. Formal mentoring roles given me by Robert P. Burton and Sean Warnick and research projects supervised by Robert P. Burton, Michael Jones, and Sean Warnick in several topics were educational experiences and invaluable in helping me to develop the maturity to stick to a single topic for the years required to attain a Ph.D.

Finally, I am deeply indebted to the members of my family, my local church congregation, my fellow graduate students, and my Psandbox compatriots for their friendship and encouragement throughout. I researched without them, but they kept me sane as I did so.

Contents

Contents	vii
List of Figures	x
1 Introduction	1
1.1 Organization of this Document	3
1.2 Terminology	3
1.3 Related Work	7
1.3.1 Stigmergy	7
1.3.2 Rendezvous	9
1.3.3 Cohesion	12
1.3.4 Coalescence	16
2 Stigmergy	17
2.1 Adjacent Communication	20
2.2 Remote Communication	23
2.2.1 Couriers	24
2.2.2 Post Office	25
2.3 Site Watching	26
2.3.1 Revised Site-Watching Problem	27
2.3.2 Stigmergic Site Watching is Possible	28
2.3.3 $N > M$ Necessary	31
2.3.4 $\Delta T \geq t_c$ Necessary	31
2.3.5 Necessary and Sufficient	32
2.4 Space and Time Synchronization	33
2.4.1 Clock Synchronization	34
2.4.2 Coordinate Frame Synchronization	36
2.5 Conclusion	39
3 Rendezvous	41
3.1 Definitions and Notation	43
3.2 Necessary Capabilities	47
3.2.1 Individuality	47
3.2.2 Temporality	48
3.2.3 Search	50
3.2.4 Limited Drift	53

3.3	Parameterized Algorithm Families	54
3.3.1	Without Uncertainty	57
3.3.2	Skewed Clocks	58
3.3.3	Noise or Nondeterministic Search	60
3.3.4	Position Drift	61
3.3.5	Variable Clocks	64
3.3.6	Individual Clocks	65
3.4	Cooperative Rendezvous	67
3.5	Conclusion	68
4	Cohesion Constraints	71
4.1	Terminology	73
4.2	Cohesion Predicate	74
4.2.1	Liveness, Composability, and the “null” behavior \mathcal{N}	76
4.2.2	The Induced Distance Function d_C	78
4.2.3	Cohesion Predicate	79
4.3	Computation Strategy	83
4.3.1	Polynomial Approximation	84
4.3.2	Boolean Bernstein Branch-and-Bound	86
4.4	Examples	91
4.4.1	Holonomic agents	91
4.4.2	Car-like Agents	92
4.5	Conclusion	95
5	Local and Global Cohesion	97
5.1	Local Cohesion Without Communication	99
5.1.1	Distance-based Local Cohesion	99
5.1.2	Problems with Local Cohesion	100
5.2	Global Cohesion with Trust	102
5.2.1	Limited Malice	105
5.3	Conclusion	108
6	Coalescence	111
6.1	Terminology	112
6.1.1	Swarms and Hives	112
6.1.2	Knowledge, Malice, and Identity	113
6.2	Small Swarms	115
6.2.1	Full Trust	115
6.2.2	Noise-Free Malice	115
6.2.3	Noise and Malice	120
6.3	Large Swarms	124
6.4	Trustworthy Hives	125
6.5	Untrustworthy Hives	130
6.6	Conclusion	131

Contents	ix
7 Conclusion	133
A Glossary	137
Bibliography	145

List of Figures

1.1	Organization of document	4
2.1	Communication types	18
2.2	Message Combination	19
3.1	Rendezvous	42
3.2	Proof of Theorem 3.2	48
3.3	Basic rendezvous with $\mathcal{R}_{1,1}$	58
3.4	Doubling proof	59
3.5	Handling noise with $\mathcal{R}_{2,1}$ and $\mathcal{R}_{2,2}$	60
3.6	Handling drift with \mathcal{D}_1 and \mathcal{D}_2	62
3.7	Worst-case variable-speed clocks	64
4.1	Kinds of cohesion-breaking scenarios	74
4.2	Using X and Y to define \mathcal{N} and d_C	77
4.3	Predicates (4.11), (4.12), and (4.13) in planes and graphs	80
4.4	Line-of-sight predicates (4.15) and (4.16)	83
5.1	Relative neighborhood graphs	99
5.2	Swarm topologies	100
5.3	Swarm trapping around obstacles	101
5.4	Messages sent by Algorithm 5.1	102
5.5	Break requests resolved according to Theorem 5.1	105
5.6	A four-cut of a four-connected graph	106
5.7	Proof to Theorem 5.3	108
6.1	Failure of blacklisting	116
6.2	Worst-case blacklisting	119
6.3	Delaying Malice in Blacklist	121
6.4	Hive coalescence algorithm	129

Chapter 1

Introduction

This dissertation deals with algorithms that provably achieve various mobility-based objectives in distributed groups of mobile agents with limited capabilities. The algorithms developed are of two kinds: constraints and proscriptions. Constraints provably preclude all behaviors that would result in agents violating their objectives; as such they may be used in conjunction with other constraints and task-specific behavior selection routines. Proscriptive algorithms identify a single behavior for each agent and are presented primarily to demonstrate that various objectives can be achieved with limited resources. In addition to these two classes of algorithms, I also present proofs demonstrating that particular sets of capabilities preclude achieving various objectives.

All of the algorithms I present are designed to provide bounds on what agents can do given explicit assumptions about the environment and capabilities of the agents involved. Average- or expected-case efficiency is not my primary concern; rather, I demonstrate efficient worst-case bounds and algorithms that provably achieve their objectives. These results are often not as efficient as the behavior of known probabilistic and heuristic algorithms, but in the worst case they solve the problems they are presented within a bounded time.

The algorithms and proofs discussed in the following chapters solve various problems facing homogeneous decentralized groups of imprecise mobile agents. The reasons for considering groups of homogeneous, decentralized, and imprecise agents are given below.

Groups of agents are attractive for many reasons. It can be cheaper to produce many simple agents than one more complex agent. Distributing sensors and actuators across multiple agents can result in smaller, more mobile agents. A group of agents can provide multiple perspectives in sensation and cover more area in search than could a single agent. Collectives can be designed that scale with the available resources and handle agent failures. Additionally, some tasks are inherently parallel and require simultaneous sensation and actuation at distinct locations.

Although agents may be heterogeneous in practice, I address only homogeneous agents. The least common denominator of a heterogeneous group of agent's capabilities provides a homogeneous set of shared capabilities; algorithms that rely only on this homogeneous set allow worst-case bounds to be developed. More efficient algorithms may be designed that utilize agent versatility, but the kernel of shared capabilities is sufficient for the worst-case bounds I demonstrate.

Each agent in a homogeneous decentralized group is comparable to each other agent in terms of capabilities and programming. Homogeneity is attractive because only a single agent design is needed, simplifying algorithms and proofs as well as manufacturing. Decentralized behaviors can be made robust to failures because no single agent is uniquely important to the behavior of the group.

Real-world agents often face a variety of imprecision, uncertainty, and noise. Agents might misidentify a non-agent as an agent; might fail to move exactly as intended; might suffer from error in sensor measurements as to their location, orientation, surroundings, or timing; and so forth. The majority of prior works either ignore this imprecision to arrive at proofs or present

algorithms that handle imprecision with heuristic or probabilistic algorithms that are not guaranteed to succeed every time they are executed. This dissertation considers algorithms with proofs that handle imprecision, as well as bounds on how much imprecision can be handled by these algorithms.

1.1 Organization of this Document

In this dissertation I consider several related problems, presenting proofs and algorithms for each. Many of these problems relate to forming and maintaining swarms or hives of agents. I also consider some of capabilities that can be provided through **stigmergy**, or depositing messages in the environment.

The tasks associated with forming and maintaining a swarm or hive may be considered in three parts. **Rendezvous** has agents discover the location of other agents. **Coalescence** has (groups of) agents that are aware of one another decide if and how they will combine into a single group. **Cohesion** is the constraint on a swarm of agents that requires they not separate as they move.

These broad classes of problems, and their various sub-problems and results, are outlined visually in [Figure 1.1](#).

1.2 Terminology

The fields of artificial intelligence (AI) and robotics use various terms in different ways depending on which topics and approaches are being discussed. Several of the terms I use throughout this

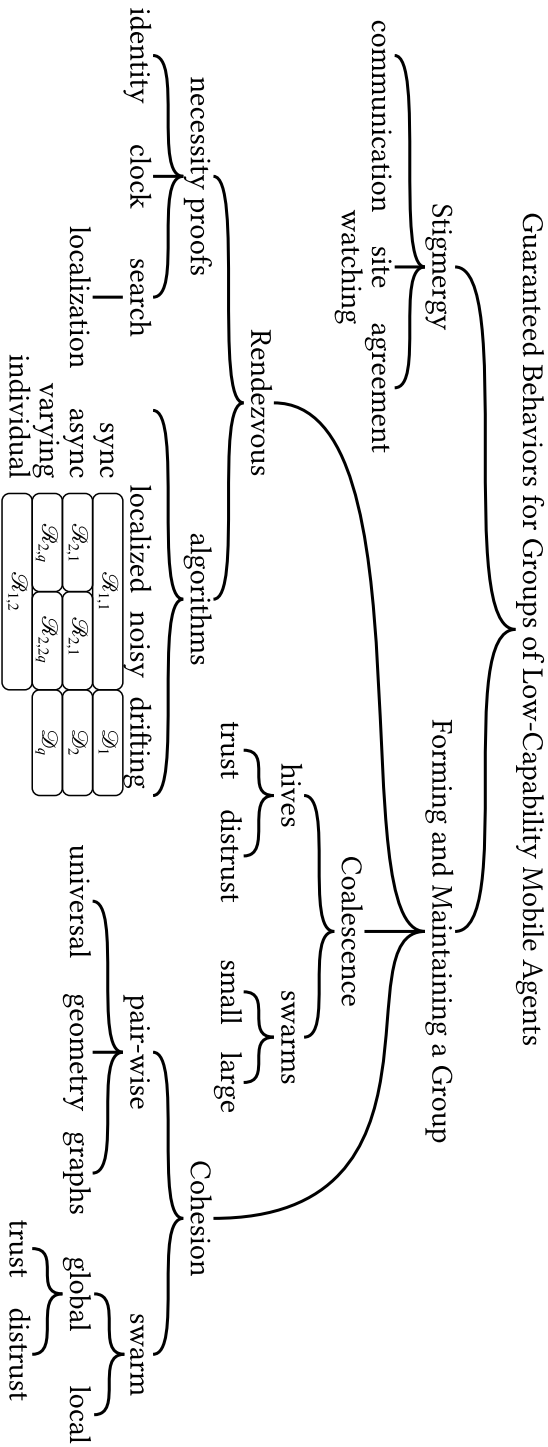


Figure 1.1: Visual hierarchy of the contents of this document.

document are defined in this section. Terms that apply to only one or two solutions are defined in their respective chapters. Additionally, a glossary is found in [Appendix A](#).

Many of the results in this dissertation apply to broad classes of agents, not just to those operating in geometric spaces or in the nodes and edges of a graph. Several terms are defined broadly; unspecified elements of these definitions are not constrained and not utilized in the dissertation's algorithms and proofs.

I refer to agents throughout this text. Agents are distinct computational entities with a well-defined state at any given time. The states of agents are sufficient to compute a meaningful distance metric, such that smaller distances correspond to greater likelihood of successful communication, sensation, etc. The agents are mobile, meaning they are capable of adjusting their states in ways that significantly impact inter-agent distance. One example of such an agent might be an autonomous robot, with its state being position, orientation, and speed; another example might be a job in a distributed computation with distance a function of communication costs and interdependence of jobs. Although my focus and examples are principally robots and similar agents moving through a continuous geometric space, the algorithms I present work for any agent that satisfies the below definitions.

Let S be the set of all possible agent **states**. Let there be a well-defined (though not necessarily computable) **distance** function $d : S \times S \rightarrow \mathbb{R}_0^+$ such that $d(s, s) = 0$ for any state $s \in S$. Two states s_1 and s_2 represent the same **position** if and only if $d(s_1, s_2) = 0$; position itself need not be otherwise defined. Some algorithms require that the distance function satisfy the **triangle inequality**; that is, $d(s_1, s_2) + d(s_2, s_3) \geq d(s_1, s_3)$.

The mobility of agents is formalized by their behavior over time.

Let **time** be represented by a possibly-discrete subset of real numbers. I assume that the environment behaves predictably such that absolute time is neither interesting nor distinguishable except by appeal to a single global clock. I use the symbol t to refer to both fixed points in time (e.g., “the state of the agent at time t ”) and time intervals or durations (e.g., “ t time units after t_0 ”).

Let the set of **behaviors** \mathcal{B} be all the ways an agent can choose to transition from one state to another over time.¹ Let the notation $B_s(t)$ mean the state an agent reaches when starting in state s and executing behavior B for t time units. $B_s(0) = s$ if B is defined at state s ; not all behaviors need to be applicable at all states. Let the **domain** of B , denoted $\text{dom}(B)$, be the set of states for which B is defined. A behavior B is **live** if, for all s in its domain and all $t \geq 0$, $B_s(t) \in \text{dom}(B)$. A behavior is **universal** if its domain is S . All universal behaviors are live.

I assume that the distance between an agent’s current and future states is limited by the time that passes. In particular, there is some fixed function f satisfying the following properties

$$f(0) = 0 \quad \wedge \quad \frac{\partial}{\partial t} f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+ \quad (1.1)$$

such that

$$\forall t \geq 0 \quad \forall B \in \mathcal{B} \quad \forall s \in \text{dom}(B) \quad d(s, B_s(t)) \leq f(t). \quad (1.2)$$

This property generalizes the idea that motion is local, but still allows for discontinuities in time and in agent state.

An **agent** is defined by a state $s \in S$ and a set of achievable behaviors \mathcal{B} . Each agent is assumed to be able to select its own behavior from the set \mathcal{B} , possibly with a time delay before

¹This definition of “behavior” corresponds to the common English usage of “what something does” and is not the same as the more limited usage in the discipline of behavior-based algorithms.

the new selection becomes active.

This dissertation considers **algorithms** that take, as input, data about the environment local to the agent and produces, as output, a behavior to follow. I use the term “algorithm” to refer to both the bounded-time process of translating one set of inputs into one set of outputs and to the potentially non-terminating evolution of agent states produced by repeatedly using the same algorithm to select new behaviors as inputs change. Which meaning is intended will be clear from context.

For physical agents, inputs would likely come from signal processing routines interpreting low-level sensor data and outputs provided to dynamic controllers producing low-level actuator commands. Many of the algorithms explicitly handle inaccuracies of various kinds to account for the imprecision of these routines.

1.3 Related Work

1.3.1 Stigmergy

The Oxford English Dictionary defines “stigmergy” as “the process by which the results of an insects’ activity act as a stimulus to further activity” [48]. It thus includes both passive stigmergy where the functional results of an action also act as the communication (e.g., dig at the end of tunnels) and active stigmergy where otherwise-nonfunctional stimuli are deposited for the purpose of communication (e.g., ant pheromone trails). Although passive stigmergy has been used in agent control publications [6, 26, 34] the bulk of explicitly stigmergic work (including this dissertation) has investigated active stigmergy.

Algorithms have been presented that use active stigmergy to solve a variety of problems. In Cazangi et al. [7], agents visit a sequence of cites in order while avoiding static obstacles. Menezes et al. [41] use stigmergy to reduce the probability of repeat visitation in collaborative exploration of an environment; Panait and Luke [49] use it for the related problem of collaboratively foraging for resources. Potential fields can also be simulated using active stigmergy [57, 68].

Several researchers have investigated how to implement pheromone-like functionality in physical robots without having the robots physically mark the environment. Mamei and Zambonelli [39] and Mamei and Zambonelli [38] have suggested techniques for using RFID tags distributed throughout the environment to emulate active stigmergy. Van Dyke Parunak et al. [68] instead have a subset of the agents stage themselves across the environment to mediate stigmergic communication, allowing the agents to operate in areas not explicitly prepared for them. Garcia et al. [22], Howden and Hendtlass [27] and Shiloni et al. [60] have each agent store a map of virtual pheromones and share their maps (using a common coordinate system) with any agents within communication range. Communication-based approaches can result in lost updates if the communication network becomes disconnected, but it is fully mobile and does not require large numbers of stationary agents.

Shiloni et al. [60] present a set of proofs comparing the computing and mobility power of “ants”—bounded-memory agents with only stigmergic communication—and “elephants”—agents with unbounded computational resources and instant global communication. They prove that elephants are capable of doing anything ants can do and also show that, since ants can simulate Turing machines, they can also do most of what elephants can do, including group computation, counting the number of agents available establishing a common coordinate frame, and meeting at a single location.

Shiloni et al. [60] also introduce the “site-watching” problem, which elephants can handle but ants cannot. This problem requires the group of agents to react to the first (and only the first) of a series of events which may occur at several scattered locations. They prove that n elephants can handle this problem with n sites, but that n ants cannot.

My Contributions

This dissertation continues the theoretic work begun by [Shiloni et al.](#) I demonstrate that the site watching problem can be handled by n agents if a delay in reaction is permissible, and that the delay may be decreased by the addition of more agents. I also investigate other models of communication and stigmergic agents not constrained to a grid. I discuss how agents can agree on location and timing; how they can communicate while moving as a group; and how they can emulate delayed global communication. These results both help refine the distinction between “ant”-like agents and “elephant”-like agents and enable the direct conversion of a variety of non-stigmergic algorithms into the stigmergic domain.

1.3.2 Rendezvous

Schelling [58] introduced the rendezvous problem, and it has been investigated in many settings since then. Rendezvous algorithms may be separated into those that help agents find one another and those that help mutually aware agents coordinate mobility to co-locate; I consider only the finding problem. Algorithms within the finding class of rendezvous problems may be characterized as either randomized or deterministic. Because I am interested in developing worst-case performance bounds, I consider only deterministic algorithms herein. A survey of randomized

rendezvous algorithms is provided by Alpern and Gal [3]. Several asymptotic analyses of coalescence based on randomized rendezvous algorithms are also available [4, 18, 53, 54].

Rendezvous has been investigated in many specialized environments, such as lines and graphs. On a line or ring, tight bounds on runtime are known for pairs [1, 2, 4] and larger groups of agents [40]; these bounds are in $O(T \log n)$ where T is the time required to traverse the initial separation between agents and n is the number of agents involved. Fraigniaud and Pelc [19] demonstrated a tight $O(\log N)$ space-bound for deterministic rendezvous in a tree, where N is the number of nodes in the tree. Dessmark et al. [13] showed time bounds for synchronous and asynchronous agents on trees ($O(N + \log n)$) and rings ($O(T \log n)$ if synchronous, $O(N + T \log n)$ if asynchronous).

Rendezvous in more general graphs and networks has been extensively studied and deterministic algorithms developed for partial and full asynchrony, for indistinguishable agents in visibly-distinct starting locations, and for agents with distinct identities [10, 11, 16, 32, 40]. In the absence of timing errors, these algorithms fit the $O(T \log n)$ time bound demonstrated for lines and rings, where T is generalized to be the time required to search deep enough to reach the other agent rather than simple distance.

In a geometric setting, deterministic rendezvous has been achieved using several approaches. Rendezvous can be reduced to landmark identification and ranking algorithms if such landmarks are available [35, 56]. Stigmergy allows agents to modify the environment to create their own landmarks, as well as depositing other helpful environmental information [60]. In a noise-free environment with a known orientation, rational coordinates in Euclidean space can be reduced to a graph [11], giving access to a variety of graph rendezvous algorithms.

Several researchers have investigated rendezvous with errors in timing information [11, 40].

I consider the multi-agent bounded-time rendezvous problem for agents with uncertainty in positioning (including orientation) and timing information. To my knowledge, I am the first to consider bounded-time rendezvous algorithms for agents experiencing uncertainty in positioning.

My Contributions

This dissertation presents a family of algorithms that achieve tractable worst-case rendezvous. My approach is based on agents alternating searching and waiting based on agent identifying numbers. The patterns of searching and waiting are designed to handle agents with imprecise positioning and timing and are applicable to geometric as well as graph settings. Marco et al. [40] also used a version of alternating searching and waiting, but their technique is restricted to synchronous noise-free agents in graphs. I achieve the same $O(T \log n)$ time bound that noiseless line- and graph-based algorithms achieved, but do so in general environments. Czyzowicz et al. [11] also considered the geometric setting with a broader class of timing errors, but their approach does not handle positioning error and has a super-exponential runtime.

In addition to presenting tractable algorithms for the rendezvous of geometric agents experiencing positioning error, I present novel bounds on the uncertainty that can be handled by any such algorithm. I believe I am the first to present bounds on permissible error.

Portions of my work in rendezvous have appeared previously in Tychonievch and Cohoon [65].

1.3.3 Cohesion

Many algorithms assume that agents remain within contact of one another without providing guarantees of cohesion. Others integrate cohesion into the design of tasks-specific algorithms, a useful technique but one difficult to transfer to new problems. I review here previously-published techniques that explicitly address cohesion.

Overly Constrained or Non-Verified.

Formations are a common approach for ensuring that a collection of agents remain in proximity to one another. By specifying the relative location of each agent, formations ensure cohesion. Broadly, formation algorithms can be broken down into three categories: those where each agent is aware of the target formation [20, 23, 30, 37, 46], those where each agent knows its specific position within the formation [29, 62], and those where the constituent agents are oblivious to the formation being maintained [12, 14]. Unfortunately, formations do not allow individual agents to pursue their own objectives.

One of the earliest and best-known cohesion algorithms is Reynold's Boids [55]. Boids, like the many flocking, herding, and swarming algorithms that have followed it, provides a holistic model of agent behavior which guarantees a group remains connected in the absence of other maneuvering objectives [9, 70]. Many approaches have augmented flocking algorithms with additional objectives (see, e.g., Astengo-Noguez and Velzquez [5], Gurfil [25]); however, these augmented algorithms do not satisfy the antecedents of existing proofs that flocks provide cohesion, and revised guarantees of cohesion have not been presented.

Switching laws choose between two or more distinct behaviors depending on the some predicate. Cohesion can be provided by switching between following mission objectives and mov-

ing toward other agents based on an estimate of cohesion being in danger of being broken [28, 42, 51, 69]. Unfortunately, switching laws can suffer from livelock. For non-holonomic agents the predicates can also be overly conservative, significantly reducing agent mobility.

Potential fields can combine mission objectives and cohesion (e.g., Li et al. [36], Zavlanos and Pappas [71, 73]); these provide smoother operation than switching laws but verifying the behavior of agents under the influence of multiple fields is a difficult process in general [31]. Composable guarantees that can be developed for individual fields and still applied when that field is combined with mission-specific objective have yet to be demonstrated.

Guarantees for Some Agents.

Ganguli et al. [21] present geometric constraints that prevent agents from entering areas that might block line-of-sight to currently visible agents. Their method takes into account partial occlusion, but it is restricted to 2D first-order holonomic agents in polygonal environments.

Cornejo and Lynch [8] provide a filter on agent behavior that ensures moving agents do not separate. Their technique assumes synchronous first-order holonomic agents with explicit communication. They utilize communication to attempt to agree on aggressive behaviors and use a conservative fall-back behavior in case of disagreement. Their fall-back behavior is a particular point on the edge of the midpoint predicate presented in Section 4.2.3 (assuming that the agents use $\vec{v} = \vec{0}$ as their \mathcal{N}).

“Backward” Cohesion: Collision Avoidance.

Cohesion is related to collision avoidance: collision avoidance prevents agents getting too close together while cohesion prevents agents getting too far apart. Many collision avoidance algo-

rithms apply heuristics with some additional padding; these approaches are unsuited for the more constrained decision space of cohesion. Even non-heuristic collision avoidance algorithms generally consider only expected neighbor behaviors because universally-quantified collision avoidance is over-constrained. Cohesion, being cooperative, can instead consider all permissible neighbor behaviors.

Two collision avoidance techniques share noteworthy characteristics with how I guarantee cohesion: reciprocal velocity obstacles and generalized reactive navigation. *Reciprocal velocity obstacles* [61, 67] are cooperative, using the fact that each agent is executing the same algorithm to streamline agent behavior. *Generalized reactive navigation* [66] has mechanisms for explicitly considering all possible neighbor behaviors for any continuous geometric model of agent behavior. Both of these works informed the cohesion algorithms in [Chapter 4](#).

Global Cohesion

Local cohesion algorithms are often paired with a global technique for selecting which pairs of agents may safely break connection with one another without jeopardizing the connectivity of the group. Vazquez and Malcom [69] and Zavlanos and Pappas [72, 73] achieve global cohesions by communicating the entire network connectivity to each agent and then selecting a set of cuts to be made; Zavlanos and Pappas [73] also discusses market-based consensus algorithms for selecting cuts.

My Contributions

Like Ganguli et al. [21] and Cornejo and Lynch [8], I present constraints that prevent agents from breaking cohesion under well-defined conditions without requiring the agents to follow a partic-

ular objective within the set of cohesion-maintaining behaviors. My techniques are more general than theirs in that they apply to both holonomic and nonholonomic agents in any dimensionality; to agents experiencing noise in their sensors and actuators; and to agents in graphs, non-uniform geometries, and any other abstract environment with a well-behaved distance function.

Like van den Berg et al. [67], my cohesion approach embeds a notion of reciprocity; my model of reciprocity is more general than theirs so that it may apply to a broader class of agents than does their first-order geometric model.

My cohesion algorithm is similar to my earlier collision avoidance algorithm [66] in that it uses polynomial approximations to make an explicit mathematical statement of its objective computable. The approximations I use for cohesion are much more precise than they were for collision avoidance, requiring me to base the computational aspect on Bernstein polynomials instead of Sturm sequences. The resulting algorithm is far more general and more precise than my previous work.

I also introduce a notion of liveness to cohesion to express the idea that not all currently-connected groups are even capable of remaining connected. To my knowledge, I am the first to consider liveness in cohesion.

My approach to global cohesion is based on comparable identities instead of market-based decisions, and only requires each agent to recall a set of messages received in a finite time window: no agent needs to accumulate knowledge of the connectivity graph. It can also handle the presence of malicious agents without violating guarantees of cohesion.

Portions of my work in cohesion have appeared previously in Tychonieivch and Cohoon [63] and Tychonieivch and Cohoon [64].

1.3.4 Coalescence

Coalescence is rarely discussed as a problem in its own right. Many of the papers listed in [Section 1.3.2](#) suggest that coalescence can be achieved by an approach like the one I outline in [Section 6.2.1](#). As discussed in that section, this is true only if the agents are universally trustworthy and able to agree on and follow a leader agent.

Coalescing trustworthy agents in a graph-like environment that also contains malicious agents has been investigated by Dieudonné et al. [16]. They provide algorithms for noiseless agents to coalesce in the presence of a minority of malicious agents. Their algorithms allow the agents to detect that they have completed coalescing by having *a priori* information about the number of trustworthy and malicious agents.

My Contributions

Like [Dieudonné et al.](#), I investigate coalescence of agents that are not all trustworthy. I present techniques that can handle arbitrary numbers of malicious agents, noisy geometric environments, and swarms larger than the sensing radius of their agents. I gain these advances at the expense of handling weakly Byzantine agents (i.e., ones that cannot lie about their own identities) where [Dieudonné et al. \[16\]](#) could also handle strongly Byzantine agents (i.e., ones able to lie about their own identities).

Portions of my work in coalescence have appeared previously in Tychonieivch and Cohoon [63].

Chapter 2

Stigmergy

Active stigmergy is communication between agents that is deposited at a particular location instead of being broadcast at a particular time. Agents with broadcast communication are known to be able to perform some tasks that agents with stigmergic communication cannot. In this chapter I explore that difference in more detail, providing algorithms and proofs that characterize the delays and additional agents needed to perform various stigmergic tasks. I also characterize several classes of communication based on their propagation through spacetime and discuss the localization information contained in each.

The word “stigmergy” was introduced by Grassé [24] in 1959 to describe how insects are stimulated to act based on the results of early actions by themselves or other insects. Since then the term has been used to refer to two different kinds of communication. One of these, which I refer to as *passive* stigmergy, is communication through the primary outcomes of an activity, such as termites digging at the end of a tunnel or masons laying bricks on top of brick already laid. The other, which I refer to as *active* stigmergy, is communication through modifications

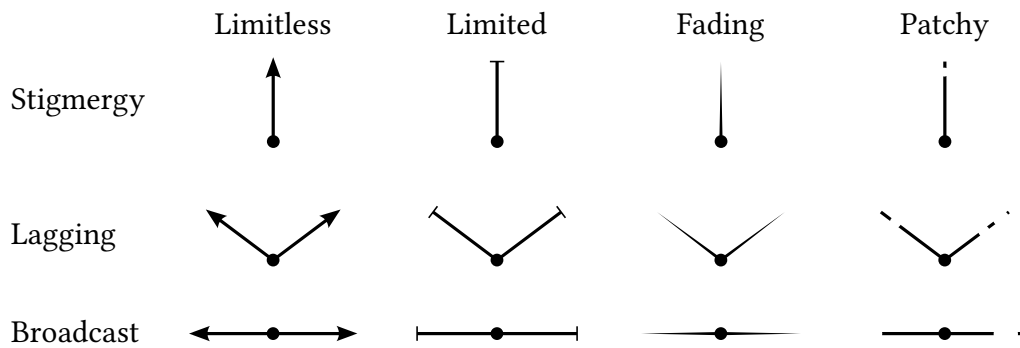


Figure 2.1: Communication types' reach in space-time. Space is shown on the horizontal axis, time on the vertical axis.

to the environment for the purpose of communication, such as ants communicating through pheromones or Ariadne laying thread through the Minotaur's labyrinth[52].

This chapter contains a variety of contributions in the area of active stigmergy in static environments. This type of stigmergy is a model of agents that broadcast their communication through time instead of through space. Stigmergy and other forms of communication may be characterized by how the signal propagates through spacetime and how the signal changes as it gets farther from its origin. **Broadcast** communication travels spans space at a single time; **stigmergy** spans time at a single location; and **lagging** broadcast travels in space over time. A **limitless** signal never fades or dies, reaching all space and/or time; a **limited** signal reaches only a fixed duration and/or distance; a **fading** signal reduces in strength as it propagates away from its origin; and a **patchy** signal is sensed at close range but stops being sensed in an unpredictable fashion. The twelve combinations of these properties are illustrated in [Figure 2.1](#). This set could be extended, but is general enough for the theoretic contributions of this chapter.

In addition to modeling the reach of stigmergic signals, the interaction of conflicting signals deposited in the same environment is important in many algorithms. The simplest model is **unbounded** stigmergy, where any number of signals may be deposited at each location and each

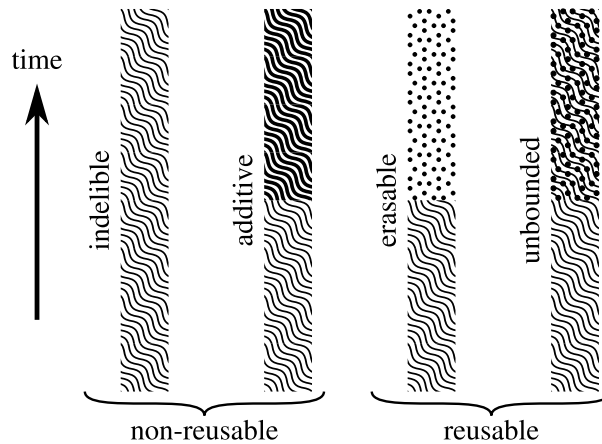


Figure 2.2: Illustration of different models of combining messages. Time increases going up. In each case, an agent first deposits a “wavy” message. From left to right, we have an indelible message, an additive message, an erasable message being replaced by a “dotted” message, and an unbounded region accepting a “dotted” message as well as the “wavy” one.

may be simultaneously detected and understood. Comparison to ant pheromones suggests an **additive** model where the signal in each area is a set of real values and agents may choose to increase any one of those values (by depositing more of that pheromone). Every additive model of which I am aware also models fading stigmergy. **Erasable** signals are available in many digital implementations of stigmergy: each agent has the ability to reset the signal in its location freely. The most restrictive model is **indelible** stigmergy, where once a region is given a message that region becomes unavailable for later signal deposition. These models are illustrated in [Figure 2.2](#)

I also use the term **reusable** to mean either erasable or unbounded and **non-reusable** to mean either additive or indelible.

Except as otherwise noted, this chapter assumes the agents exist within a d -dimensional Euclidean environment; with the exception of a few graph-based maze-solving algorithms, this assumption predominates the literature on active stigmergy. I also assume that agents can make use of previously-published techniques for laying and following trails [7, 41, 49] and for locating other agents [60].

2.1 Adjacent Communication

This section is devoted to establishing the following theorem:

Theorem 2.1 (Adjacent Communication). *Without loss of generality, synchronized stigmergic agents may be assumed to be able to communicate directly when adjacent to one another, with time spent proportional to the bits of information exchanged. If stigmergy is not erasable, this communication may require the agents to move a distance proportional to the number of bits of information they exchange.*

A proof of [Theorem 2.1](#) is presented at the end of this section. The rest of this section discusses the four lemmas on which that proof is based.

In this section I use the following terms:

- k is the bits of information that are conveyed through a single write action.
- t_r is the time sufficient to read k bits of information.
- t_w is the time sufficient to write k bits of information.
- d_k is the distance needed between adjacent k -bit information packets in the environment.
- d_a is the “diameter” of each agent. Agents need not be uniform or circular; d_a merely approximates the space an agent occupies. If d_k is larger than a physical agent, let d_a be d_k instead of the diameter of an agent.
- t_k is a time sufficient for an agent to move d_k .
- t_a is a time sufficient for an agent to move d_a .

I consider two adjacent agents, A_l and A_r . I define “leftward” to mean “the direction A_l is relative to A_r ,” with “rightward” being the opposite direction and “forward” being one of the directions

perpendicular to “leftward” and “rightward,” chosen arbitrarily but consistently by both A_l and A_r .

There are two main considerations in demonstrating adjacent agent communication. The first is if agents can observe messages deposited beyond their physical location (the sensing radius is at least d_a) or not (the sensing radius is less than d_a). The second is if stigmergy is reusable or not. I next present algorithms allowing communication for each of the four cases, with lemmas establishing each.

Lemma 2.1. *Any pair of agents that can sense outside the area they occupy and can reuse space for depositing messages can communicate b bits in time $\left\lceil \frac{b}{k} \right\rceil (t_w + t_r)$ when adjacent, without moving.*

Listing 2.1

1. Each agent writes k bits
 2. Each agent reads k bits from the other’s space
-

Proof. Each application of the algorithm in Listing 2.1 causes adjacent agents to transfer k bits in time $(t_w + t_r)$ without moving. Because space is reusable, the agents can repeat Listing 2.1 $\left\lceil \frac{b}{k} \right\rceil$ times to transfer b bits. Because the agents are synchronized, each agent reads the other’s written bits before those bits are re-written. □

Lemma 2.2. *Any pair of agents that can sense outside the area they occupy, even those that cannot reuse space for depositing messages, can communicate b bits in time $\left\lceil \frac{b}{k} \right\rceil (t_w + t_r + t_k)$ when adjacent, moving $\left\lceil \frac{b}{k} \right\rceil d_k$ forward while doing so.*

Listing 2.2

1. Each agent writes k bits
 2. Each agent reads k bits from the other's space
 3. Each agent moves d_k forward
-

Proof. Each application of the algorithm in Listing 2.2 causes adjacent agents to transfer k bits in time $(t_w + t_r + t_k)$ and end up d_k forward of where they began. Because agents move after writing, agents can repeat Listing 2.2 $\left\lceil \frac{b}{k} \right\rceil$ times to transfer b bits. □

Lemma 2.3. *Any pair of agents that can reuse space for depositing messages, even those that cannot sense outside the area they occupy, can communicate b bits in time $2 \left\lceil \frac{b}{k} \right\rceil (t_w + t_r + t_a)$ when adjacent, with only local motion.*

Listing 2.3

1. A_l reads then writes k bits
 2. Each agent moves d_a leftward
 3. A_r reads then writes k bits
 4. Each agent moves d_a rightward
-

Proof. Each application of the algorithm in Listing 2.3 causes adjacent agents to transfer k bits in time $2(t_w + t_r + t_a)$ and end where they began. Because space is reusable, the agents can repeat Listing 2.3 $\left\lceil \frac{b}{k} \right\rceil$ times to transfer b bits. □

Lemma 2.4. *Any pair of agents, even those that can neither sense outside the area they occupy nor reuse space for depositing messages, can communicate b bits in time $2 \left\lceil \frac{b}{k} \right\rceil (t_w + t_r + t_a + t_k)$ when adjacent, moving $\left\lceil \frac{b}{k} \right\rceil 2d_k$ forward (with additional local motion) while doing so.*

Listing 2.4

1. A_l writes k bits
 2. Each agent moves d_a leftward
 3. A_r reads k bits
 4. Each agent moves d_k forward
 5. A_r writes k bits
 6. Each agent moves d_a rightward
 7. A_l reads k bits
 8. Each agent moves d_k forward
-

Proof. Each application of the algorithm in [Listing 2.4](#) causes adjacent agents to move $2d_k$ forward and transfer k bits in time $2(t_w + t_r + t_k + t_a)$. Because agents move after writing, agents can repeat [Listing 2.4](#) $\left\lceil \frac{b}{k} \right\rceil$ times to transfer b bits. □

Given these four lemmas, the proof of [Theorem 2.1](#) is straightforward.

Proof of [Theorem 2.1](#). By Lemmas 2.1–4, adjacent agents with any radius of sensation and any kind of active stigmergy can communicate in time proportional to the bits of information transferred. If stigmergy is not reusable, the agents also move a distance proportional to the bits of information transferred. □

In the remainder of this chapter I assume that, given [Theorem 2.1](#), adjacent communication is one of the abilities of stigmergic agents.

2.2 Remote Communication

It is typical to assume that agents with stigmergic communication do not possess any form of long distance communication. In this section I discuss two ways that long distance communication may be emulated if additional time and agents are available.

2.2.1 Couriers

Consider a group of agents at distinct locations who wish to communicate. Assume also that additional agents are available to facilitate this communication. Then the agents can simulate any communication network by having the additional agents act as information couriers.

Let $A_s = a_1, a_2, \dots, a_n$ be the set of agents that wish to communicate and $A_c = c_1, c_2, \dots, c_m$ be the set of additional agents available. Courier communication proceeds as follows:

1. A trail is placed between a_i and a_j if and only if a_i and a_j should have network connectivity.
2. Each additional agent c_i is assigned a set of trails to navigate. This may be done by laying a different signal or pattern of signals on each trail. If communication connections are bidirectional, both directions of each trail are assigned. Multiple agents may be assigned to the same trail.
3. Each agent c_i establishes a cyclic order in which it walks the trails it is assigned. Additional navigational trails may be added to facilitate moving between assigned trails.
4. Each agent repeatedly
 - 4.1. Moves to the agent at the head of its next trail
 - 4.2. Uses local communication to discover what message the agent wants to send
 - 4.3. Follows the trail to the agent at the other end
 - 4.4. Uses local communication to deliver the message

Courier communication is rarely practical to implement. The communication delay is proportional to the total distance couriers need to travel. The network connectivity graph must either lack intersecting edges or must have a high-enough information density for crossing edges to be handled. If the communicating agents move, either because of their mission objective or because

stigmergy is non-reusable, then the courier agents need to follow increasingly-lengthy paths from the end of their established trails to the current position of the agent they are seeking. I present courier communication here as part of an exploration of the functions that may be implemented by stigmergic agents rather than as a practical solution to a real-world problem.

2.2.2 Post Office

A small modification of the courier design can avoid many of the caveats that make courier communication undesirable.

Designate a single region of the environment to be the “post office.” Let each agent have a path from its current location to the post office. Communication occurs when an agent follows the trail from its current location to the post office, reads the messages there, writes its own message, and then follows the trail back to its previous location.

Post office communication has many advantages. It does not rely on additional agents (though additional agents could be used to parallelize the working, traveling, and communicating steps). It does not use up space “in the field” for communication. Trails never need to overlap since they form a tree structure with the post office as the root. It allows simulation of arbitrary communication topologies with the addition of “to” markings in the messages deposited. It allows individual agents to prioritize between communication and work, and so on.

Many different implementations of post office communication may be envisioned, and many existing protocols can be adapted to apply to stigmergic post offices. When stigmergy is non-reusable, however, it is important to either (1) pre-allocate enough trail-free space for all the

messages that will arise during execution or (2) have a deterministic protocol for locating additional post office space when the existing region fills up.

2.3 Site Watching

Shiloni et al. [60] introduced the site-watching problem as an example of a task for which agents with global broadcast communication can solve but stigmergic agents cannot. For completeness, I quote their definition of the problem. Their definition depends on a notion of a **call**, defined earlier in their paper as an event occurring in a particular grid cell at a particular moment in time.

Definition 2.1 (LimitedKServer). *Let $R = r_1, \dots, r_N$ be a set of mobile robots with sensing radiuses of $\frac{M^2}{2N}$ each, all are positioned on a finite $M \times M$ grid such that all sensing radiuses are disjoint and their union covers the whole grid. Let $C = c_1, \dots, c_x$ be a set of calls and y be a positive integer such that $y \leq x \leq 2yM$ where y is known, but neither C nor x is known. Assume that within a finite period of time t , x calls are made such that no two calls are made in parallel. Assume that every robot $r_i \in R$ can answer a call immediately only within its sensing radius and that each call lasts for one time cycle only. We say that an algorithm A succeeds if and only if for every sequence of x calls, A answers exactly y ; otherwise, A fails.*

—Shiloni et al. [60, p. 87]

Their proof that stigmergic agents cannot succeed at this task depends on the fact that there are no additional agents available to facilitate communication [60, p. 87].

In this section I first re-pose the site-watching problem in a form that does not rely on discrete time or on a two-dimensional grid. I then demonstrate that the addition of more agents allows solution of the site-watching problem if and only if the immediacy of responses is also relaxed. Finally, I combine this problem with the previous sections to explore the relative power of stigmergic and broadcast communication.

2.3.1 Revised Site-Watching Problem

Definition 2.1 is posed within the discrete-time grid-based abstraction utilized by Shiloni et al. [60]. I present below a slightly different problem definition that does not make either discretization assumption.

Definition 2.2 (Site Watching). *Let $P = p_1, \dots, p_M$ be a set of M sites each of which may generate one or more events; let $A = a_1, \dots, a_N$ be a set of N stigmergic agents; let the sites be positioned such that each agent can sense at most one site at a time. Let $C = c_1, \dots, c_x$ be a set of calls where each c_i happens at some site p_j at some time t_i . Let y be a positive integer such that $y < x$ where y is known but neither x nor C is known. Assume that within a finite period of time t , x calls are generated and that any call $c_j = (p_i, t_j)$ can be observed immediately and answered by any agent who has the generating site p_i within its sensor radius. Assume that each call c_j can only be answered within a fixed ΔT of the time t_j when it was generated. An algorithm A succeeds if and only if for every sequence of x calls, A answers exactly y ; otherwise, A fails.*

This definition allows me to state the following theorem:

Theorem 2.2. *Stigmergic agents can succeed at the Site Watching problem if and only if ΔT is large enough and $N > M$.*

The remainder of [Section 2.3](#) presents the pieces needed to prove [Theorem 2.2](#); the full proof is given in [Section 2.3.5](#).

2.3.2 Stigmergic Site Watching is Possible

The following algorithms succeed at the Site Watching problem with $N = M + 1$ and a large ΔT .

Listing 2.5

1. Place each agent a_i at site p_i
 2. Have agent a_{M+1} follow a trail to walk a circuit of the sites
 3. Agent a_{M+1} has a counter, initially set to 0
 4. Other agents each have an list capable of storing y calls, initially empty
 5. When a_i observes call c_j
 6. Add c_j to a_i 's list (if that list has room)
 7. When a_i and a_{M+1} meet
 8. For each c_j in a_i 's list,
 9. If a_{M+1} 's counter is less than y
 10. Answer c_j
 11. Increment a_{M+1} 's counter
 12. Remove c_j from a_i 's list
-

Lemma 2.5. *If the Site Watching problem is defined with a ΔT that is large enough for an agent to make a complete circuits of the sites, then [Listing 2.5](#) succeeds at Site Watching.*

Proof. Let t_C be the time needed to traverse a complete circuit of all sites. Each call c_j issued at site p_i at t_j will

1. Wait in a_i 's list until $t_{j1} < t_j + t_C$
2. Be answered if and only if fewer than y calls have already been answered

Additionally, each call will be answered at most once because of line 12.

Thus, exactly y calls will be answered, each within t_c of being issued, for any sequence of x calls. □

Listing 2.6 Post-office Site Watching

1. Place each agent a_i at site $p_{(i \bmod M)}$
 2. Each agent has a list capable of storing y calls, initially empty
 3. When call c_j occurs at site p_i
 4. If an agent $a_{k>M}$ is at p_i ,
 5. One such a_k
 6. adds c_j to its list
 7. follows the trail to the post office
 8. Otherwise,
 9. a_i stores c_j on in its list (if there is room)
 10. When a_i visits the post office
 11. For each call c_j in a_i 's list
 12. If the post office's count is less than y
 13. Increment the post office's count
 14. Otherwise,
 15. remove c_j from a_i 's list
 16. a_i returns to $p_{(i \bmod M)}$
 17. When a_k returns from the post office to site p_i
 18. a_k answers each call on its list
 19. If there are calls on a_i 's list,
 20. Copy a_i 's list to a_k 's list
 21. Clear a_i 's list
 22. a_k follows the trail to the post office
-

Extensions to [Listing 2.5](#) can make use of other communication paradigms to trade-off between N and ΔT . In particular, post office communication is well suited to the Site Watching problem.

Lemma 2.6. *If the Site Watching problem is defined with a ΔT that is large enough for any agent to make two round trips to the post office then for any $N \geq 2M$ [Listing 2.6](#) succeeds at Site Watching.*

Proof. Let t_p be sufficient time to make a round trip to the post office.

Given that $N = 2M$, each call c_j issued at site p_i at t_j will

1. Establish a post office within the convex hull of the sites
2. Possibly wait in a_i 's list until $t_{j1} < t_j + t_p$

3. Be in $a_{k>M}$'s list until $t_{j2} < t_j + \frac{3}{2}t_P$
4. Be discarded if y calls have already been queued for answering
5. Otherwise, be in $a_{k>M}$'s list until $t_{j3} < t_j + 2t_P$ and then be answered.

Thus, exactly y calls will be answered, each within $2t_P$ of being issued, for any sequence of x calls. □

Lemma 2.7. *If the Site Watching problem is defined with a ΔT that is large enough for any agent to make a single round trip to the post office, then for any $N \geq (y + 1)M$ Listing 2.6 succeeds at Site Watching.*

Proof. Let t_P be sufficient time to make a round trip to the post office.

Given that $N = (y + 1)M$, each call c_j issued at site p_i at t_j will

1. Be in $a_{k>M}$'s list until $t_{j1} < t_j + \frac{1}{2}t_P$
2. Be discarded if y calls have already been queued for answering
3. Otherwise, be in a_k 's list until $t_{j2} < t_j + t_P$ and then be answered.

Thus, exactly y calls will be answered, each within t_P of being issued for any sequence of x calls. □

Because the post office is within the convex hull of the sites, $t_T \leq t_P \leq 2t_T$ where t_T is the time needed to travel one-way between the most distant pair of sites. Thus, the time used by Listing 2.6 or Listing 2.6 is within a constant factor of the minimal time needed for coordinate between the various sites.

2.3.3 $N > M$ Necessary

Shiloni et al. [60, pp. 87–88]’s Theorem 6 states “There is no ant algorithm which solves Limited-KServer when running N ants.” Their proof applies equally to Site Watching. For completeness, and to have consistent internal theorem numbering, I re-pose their theorem and proof below.

Lemma 2.8. *For any $N \leq M$, there is no stigmergic algorithm that solves Site Watching. [60, pp. 87–88]*

Proof. If any site is not observed at any time, that site might generate an event in that time which cannot be answered. Since x might equal y , that event might be essential for success. Thus, Site Watching cannot be solved by any algorithm that leaves any site unobserved at any time.

If $N < M$ then at any given point in time some site is unobserved. Thus, no algorithm can solve Site Watching if $N < M$.

If $M = N$ then there is only one agent per site. If an agent leaves its site then that site is unobserved for a period of time. If no agent leaves its site then the agents cannot communicate and have no way to coordinate which calls should be answered. Thus, no algorithm can solve Site Watching if $N = M$. □

2.3.4 $\Delta T \geq t_C$ Necessary

Although not stated as a theorem, Shiloni et al. [60] suggests that the instantaneous nature of answers also presents a difficulty for stigmergic agents. I formalize this suggestion below.

Let t_C be the time needed to communicate one-way between the most distant pair of sites. This is proportional to their separation: either the time needed for an agent to move between the

two, or, if agents can communicate without moving, it is the time needed to read and write times the number of agents needed to form an unbroken chain between the two sites.

Lemma 2.9. *If Site Watching is defined with a $\Delta T < t_C$ then there is no stigmergic algorithm that solves Site Watching.*

Proof. Consider two sites, p_1 and p_2 , such that communication between them takes t_C . Consider a time t at which $y - 1$ calls have been answered and after which no additional calls will occur. Either p_1 or p_2 or both could generate a call at time t .

Consider an algorithm A that will decide to answer calls in $\Delta T < t_C$. Then A must decide to answer the call at p_1 before $t + t_C$, but $t + t_C$ is the earliest that the presence or absence of a call at p_2 could reach p_1 (and vice versa). Thus A must decide whether to answer each call independently of the other. Consider the following cases:

- If A answers neither call then it fails.
- If A answers the call at p_1 but not p_2 then it fails if only p_2 generates a call.
- If A answers the call at p_2 but not p_1 then it fails if only p_1 generates a call.
- If A answers the call at p_1 and p_2 then it fails if both calls were generated.

Thus there is no A that solves Site Watching with $\Delta T < t_C$. □

2.3.5 Necessary and Sufficient

The lemmas in the preceding sections allow a proof of [Theorem 2.2](#), re-stated below:

Theorem 2.2. *Stigmergic agents can succeed at the Site Watching problem if and only if ΔT is large enough and $N > M$.*

Proof. By Lemma 2.9, $\Delta T \geq t_C$ is necessary. By Lemma 2.6 and Lemma 2.7, $\Delta T \propto t_C$ is sufficient if $N \geq 2M$.

By Lemma 2.8, $N > M$ is necessary. By Lemma 2.5, $N = M+1$ is sufficient if $\Delta T \propto Mt_C$. \square

Note that the bound is not tight. I have not shown that $\Delta T = t_p$ is the smallest achievable nor that $\Delta T = t_C$ is achievable. I have also not shown if $\Delta T \propto t_C$ can be achieved for $M < N < 2M$. However, I have established asymptotically tight bounds for both ΔT and N ; tightening the constants is left to future work.

2.4 Space and Time Synchronization

Many existing algorithms for groups of agents assume some form of shared coordinate frame in time and space. This includes algorithms that assume GPS or other localization sensors as well as algorithms posed in a grid-based discrete-time environment.

In this section I consider how agents that lack an *a priori* shared coordinate frame, epoch, or landmark may develop a common sense of location and scale in space and time as well as orientation in space. Different communication models will facilitate different elements of this synchronization effort.

I consider communication-only based techniques in this section. Other approaches, such as observing a common external landmark, reacting to a common globally-visible event, or reacting to non-communication interactions such collisions, are beyond the scope of this chapter.

2.4.1 Clock Synchronization

Under global broadcast, clock synchronization is as simple as having one agent broadcast its current time. This message is instantly received by all agents, resetting all clocks.

Under lagging broadcast, clock synchronization may be achieved by something like the Network Time Protocol: a request is sent and returned with both agents timestamping it at both ends. These methods are simple and reliable and well documented elsewhere [43].

Under stigmergy, consider a message i left by agent B at B 's time t_{bi} and observed by agent A at A 's time t_{ai} . This observation informs A that

- $t_{bi} < t_{ai}$; and
- $t_{ak} < t_{bi}$ if A had previously visited the area at time t_{ak} without seeing the message.

Depending on the duration of the signal, the observation may also convey

- $t_{ai} < t_{bi} + \Delta t$ if stigmergy is temporary with known duration ΔT ;
- $t_{ai} \approx t_{bi} + f^{-1}(s)$ where s is signal strength if stigmergy fades according to some known function $f : \text{time} \rightarrow \text{signal strength}$; or
- some probability function if stigmergy is patchy with a known distribution.

Theorem 2.3. *Consider stigmergic agents with limitless stigmergy or with limited, fading, or patchy stigmergy with unknown expiration patterns. Call two locations the “same” if a signal deposited in one can be detected by an agent located in the other. Agents that never visit same locations less than T time units from one another can never achieve clock synchronization tighter than $\pm T$.*

Proof. Assume that agents A and B share full knowledge of a common coordinate system and each know both agents' full behavior functions such that at time t agent A is at location $s_A(t)$ and agent B is at location $s_B(t + t_B)$ for some constant but unknown t_B .

Suppose $s_A(t_0)$ and $s_B(t_1)$ are the same location. Then either

- A left a message B reads, meaning $t_B < t_1 - t_0$;
- B left a message A reads, meaning $t_1 - t_0 < t_B$; or
- neither agent left a message, in which case nothing is learned.

No other relative time information is available to the agents.

Consider a constraint $t_1 - t_0 \geq t_B$. The actual time between the agents being in the same location to create this constraint was $T' = |t_0 - (t_1 + t_B)|$. If $T < T'$ then it is also true that $t_1 - t_0 \pm T \geq t_B$.

Thus, if A and B never visit same locations less than T time units from one another then they can never achieve clock synchronization tighter than $\pm T$. □

If the rate of stigmergic fading is known, then clock synchronization is as simple as writing the initial strength of a signal and then computing the delay when the signal is seen.

If the duration of temporary stigmergy is known then clock synchronization can be accomplished by being at the same location a signal is created or ceases to exist.

If the probability distribution of patchy stigmergy is known then a probabilistic model of clock synchronization can be created by observing many signals in the period where they might or might not be present.

The scale of clocks (i.e., the relative rate at which different agents' counters increase) can be determined via two or more distinct synchronizations.

2.4.2 Coordinate Frame Synchronization

Limitless broadcast contains no information about location and can only be used to establish location if combined with some other location-providing information, such as GPS signals, landmarks, seeing other agents, etc.

Stigmergy provides straightforward location synchronization: stigmergy only communicated between agents with a common location. This is true no matter how the stigmergy lasts, even fading and patchy stigmergy.

All other forms of communication I consider in this chapter (limited, fading, or patchy broadcast or any lagging communication) can be used to establish (approximate or probabilistic) bounds on the distance between agents. Fading signals provide distance directly via signal strength. Limited signals provide distance at the moment they come into or out of range. Patchy signals provide distance probabilistically provided that the distribution on signal reception is known. Lagging signals provide it through the same send-and-receive timing that provides lagging clock synchronization assuming that the speed of lagging signal propagation is known.

Full coordinate frame synchronization can be derived from a set of distance or location matches. Because this computation is not numerically stable, precise results require either many independent samples or a few samples taken at large distances from one another.

In general, the coordinate frame synchronization problem seeks to find a matrix \mathbf{A} and vector \vec{b} such that, given a location \vec{x} in one agent's coordinate space $\mathbf{A}\vec{x} + \vec{b}$ is the corresponding location in the other agent's coordinate space.

Location matches provide the information $\vec{y} = \mathbf{A}\vec{x} + \vec{b}$ for particular \vec{x} and \vec{y} . In n -dimensional space, $n + 1$ linearly independent matches determine a unique \mathbf{A} and \vec{b} . \mathbf{A} may be found by

subtracting the first pair from every other pair (to remove the \vec{b} term) and then concatenating the vectors to form the matrix equation $\mathbf{Y} = \mathbf{A}\mathbf{X}$; column j of \mathbf{Y} is the $(j + 1)$ th \vec{y} minus the first \vec{y} and likewise for \mathbf{X} . Since the matches were linearly independent, \mathbf{X} is invertible and this equation may be solved via $\mathbf{A} = \mathbf{Y}\mathbf{X}^{-1}$. Given \mathbf{A} , \vec{b} can be found via $\vec{b} = \vec{y} - \mathbf{A}\vec{x}$ for any matched pair \vec{x} and \vec{y} .

If measurements are not linearly-independent, or if they are approximate or noisy, least-squares estimates of \mathbf{A} and \vec{b} may be found by taking many samples and using the Moore-Penrose Pseudo-inverse[44, 50] instead of the standard matrix inverse.

Distances provide the information $\|\vec{y} - \mathbf{A}\vec{x} - \vec{b}\|_2^2 = d^2$ for particular \vec{x} , \vec{y} , and distance d .

Expanding the ℓ_2 norm gives

$$\left(\vec{y}'\vec{y}\right) - d^2 - 2\left(\vec{y}'(\mathbf{A}\vec{x}) + \vec{y}'\vec{b}\right) + \left((\mathbf{A}\vec{x})'(\mathbf{A}\vec{x}) + 2\vec{b}'(\mathbf{A}\vec{x}) + \vec{b}'\vec{b}\right) = 0, \quad (2.1)$$

a quadratic equation in $n^2 + n$ unknowns. With $n^2 + n$ samples the resulting system of equations cannot be solved directly, but can be efficiently solved using Newton's method. For example, consider the two-dimensional case. A single observation gives the equation

$$\begin{aligned} y_1^2 + y_2^2 - d^2 - 2 \begin{bmatrix} y_1 & y_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 2y_1b_1 + y_2b_2 \\ + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 2 \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b_1^2 + b_2^2 = 0 \end{aligned} \quad (2.2)$$

which may be expanded to the form

$$\begin{aligned}
& y_1^2 + y_2^2 - d^2 - 2x_1y_1a_{11} - 2x_1y_2a_{21} - 2x_2y_1a_{12} - 2x_2y_2a_{22} \\
& - 2y_1b_1 - 2y_2b_2 + x_1^2a_{11}^2 + x_1^2a_{21}^2 + 2x_1x_2a_{11}a_{12} + 2x_1x_2a_{21}a_{22} + x_2^2a_{12}^2 + x_2^2a_{22}^2 \\
& + x_1a_{11}b_1 + x_1a_{21}b_2 + x_2a_{12}b_1 + x_2a_{22}b_2 + b_1^2 + b_2^2 = 0. \quad (2.3)
\end{aligned}$$

If six samples are taken with linearly-independent \vec{x} , \vec{y} , and d the result is six vector-valued equations in six unknowns: $\vec{f}(a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2) = \vec{0}$. The Jacobian matrix \mathbf{J} for \vec{f} is the six-by-six matrix with rows whose transpose are

$$\begin{bmatrix}
-2x_1y_1 + 2x_1^2a_{11} + 2x_1x_2a_{12} + x_1b_1 \\
-2x_2y_1 + 2x_1x_2a_{11} + 2x_2^2a_{12} + x_2b_1 \\
-2x_1y_2 + 2x_1^2a_{21} + 2x_1x_2a_{22} + x_1b_2 \\
-2x_2y_2 + 2x_1x_2a_{21} + 2x_2^2a_{22} + x_2b_2 \\
-2y_1 + x_1a_{11} + x_2a_{12} + 2b_1 \\
-2y_2 + x_1a_{21} + x_2a_{22} + 2b_2
\end{bmatrix} \quad (2.4)$$

Given estimate $\vec{ab} = (a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2)$ one iteration of Newton's iteration[47] updates the estimate to be $\vec{ab} - \mathbf{J}(\vec{ab})^{-1}\vec{f}(\vec{ab})$.

If measurements are approximate or noisy and more than six samples are available, the Moore-Penrose Pseudo-inverse[44, 50] may be used instead to create a least-squares estimate of \vec{ab} .

2.5 Conclusion

Broadcast communication propagates messages through space; Stigmergy propagates messages through time. This section explored several aspects of that distinction.

I have extended the work of Shiloni et al. [60]; they demonstrate that broadcast communication, if coupled with global location information, is capable of fully emulating stigmergic communication but that for at least some problems stigmergic communication cannot accomplish the same tasks as broadcast communication. I analyze the limitations more closely, demonstrating how stigmergic agents may emulate local and global communication and using those tools to show that the task identified by Shiloni et al. can be solved if and only if both additional agents and additional time are available.

I also investigated how the propagation of communication through spacetime may be used by agents to agree on a common coordinate frame in time and space. Limitless broadcast communication is the only form that does not contain explicit information about space, and that all contain explicit information about time. Given any other model of communication, communication can be used to synchronize clocks and coordinate frames. However, for stigmergy establishing a common time frame requires agents to meet in spacetime.

There are more questions to be asked about the characteristics of different communication propagation models. I have provided several models of communication that can be built out of stigmergy, but there may be others that are more efficient in space used, in time required, or in number of agents involved. I showed that Site Watching can be solved to within a factor of 2 of the optimal time and agent count; it would be nice to extend this construction to techniques for simulating arbitrary broadcast-based algorithms stigmergically with some approximation of op-

tinality. Further investigation of how communication-based synchronization of time and space can be achieved with patchy and/or non-uniform-fading signals might make the theoretic results in this chapter more applicable to real-world robotics and other applications.

Chapter 3

Rendezvous

Rendezvous has mutually-oblivious agents locate one another in an unknown environment. In this chapter I show that agents experiencing uncertainty in timing and location can be guaranteed to rendezvous in finite time without relying on environmental features. I also demonstrate worst-case time bounds for rendezvous with various models of imprecise timing and position, and I present algorithms achieving these bounds. In addition to timing, I also bound the uncertainty agents in featureless environments can handle while still guaranteeing finite-time rendezvous with theoretic upper bounds and algorithmic realization of these bounds. Together, these bounds on runtime and uncertainty help refine our understanding of the rendezvous problem.

The objective of rendezvous algorithms is to have mobile agents locate one another in an unknown or featureless environment. Rendezvous is a foundational problem in cooperative artificial intelligence: before agents can coordinate their actions, individual agents must come close enough together to initiate contact.

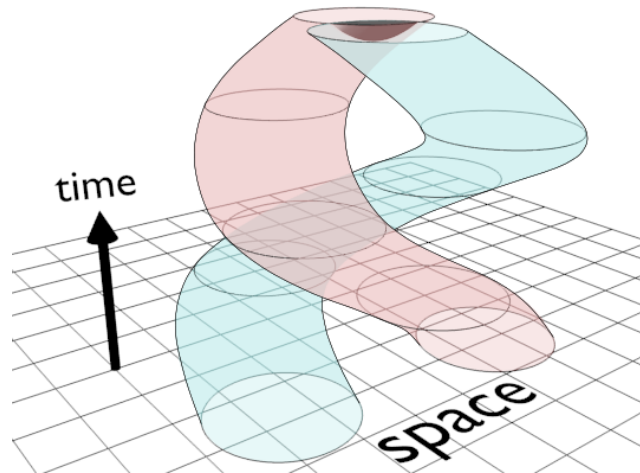


Figure 3.1: Rendezvous as the space-time intersection of sensing volumes. Each circle's diameter is equal to the sensing radius of the agents.

Since its introduction fifty years ago [58], the rendezvous problem has received considerable attention from theoreticians and multi-agent artificial intelligence researchers. Several sub-problems have been identified. I address the rendezvous search problem, where pairs of mutually-oblivious agents seek proximity in space. Rendezvous can be viewed as the task of ensuring that the sensing areas of pairs of agents overlap in space-time, as illustrated in Figure 3.1. Rendezvous is more involved than searching for a static objective because of the need to overlap in space-time, not just in space.

Rendezvous is often posed in the context of the coalescence or gathering problem. Coalescence has initially-scattered agents seek to combine into a single a swarm or flock. For fully trustworthy agents that are able to communicate at short range, coalescence is no more difficult than rendezvous; rendezvous remains central to swarm creation and repair in other settings as well. The coalescence problem and its relationship to rendezvous is discussed in Chapter 6.

Prior work in rendezvous has focused on probabilistic algorithms, on landmark identification and ranking, and on deterministic algorithms for noiseless line- and graph-like environments

(see [Section 1.3.2](#)). This chapter extends prior work by investigating deterministic finite-time rendezvous algorithms for agents experiencing noise in both timing and locating information in environments without landmarks.

This chapter presents a collection of procedures that cause each agent in an arbitrarily-large set to move within sensor range of each other agent. The procedures vary in complexity, efficiency, and in the assumed capabilities of the agents. Each procedure is accompanied by a proof that agents will rendezvous with one another in bounded time. I also present bounds on the uncertainty that may be present in any agent participating in bounded-time rendezvous. These minimal capabilities provide a lower bound on the rendezvous problem for agents facing uncertainty.

3.1 Definitions and Notation

This chapter uses the notation introduced in [Section 1.2](#), particularly the notion of state $s \in \mathcal{S}$, behavior $B_s(t)$, distance d , and the use of “algorithm” to refer to a deterministic behavior selection process. This chapter does not require that the distance function satisfy the triangle inequality.

The **capabilities** of an agent are characterized by the sensors and actuators the agent may access and by the uncertainty of each.

Consider the reference time t and let the state of agent i be $s_i(t)$. Let agent i 's **clock** generate an estimate of the current time $\hat{t}_i(t)$ and let its means of **mobility** allow it to generate an estimate of its state $\hat{s}_i(t)$.

I consider the following classes of capabilities:

- **Mobility:** Agents have some knowledge of and control over their own location. I distinguish between three classes of mobility:

- **Exact:** No uncertainty in location:

$$\forall i, t : d(s_i(t), \hat{s}_i(t)) = 0. \quad (3.1)$$

This class is commonly assumed in prior graph- and grid-based algorithms.

- **Noisy:** Location error within fixed bounds:

$$\forall i : \exists c : \forall t : d(s_i(t), \hat{s}_i(t)) \leq c. \quad (3.2)$$

This class includes GPS and similar imprecise location-sensing devices.

- **Drifting:** Location error increases over time:

$$\forall i : \exists \epsilon : \forall t : d(s_i(t), \hat{s}_i(t)) \leq \epsilon(t), \quad (3.3)$$

where $\epsilon(t)$ is a monotonically non-decreasing function called the agent's **drift**. Most often, the drift increases with some function of distance traveled:

$$\epsilon(t) = f \left(\int_{t_0}^t d(s_i(\tau), s_i(\tau + d\tau)) \right). \quad (3.4)$$

Dead reckoning falls into this class.

- **Detection:** There is some distance r such that any two agents whose separation is no more

than r are aware of one another. Although agents might be aware of one another at greater distances, worst-case analysis of our algorithms do not rely on longer-distance detection. Detection is technically a form of communication (see, e.g., Dieudonné et al. [15]), but is not useful for establishing rendezvous because it does not occur until after rendezvous is achieved.

- **Clock:** Each agent has some notion of the passage of time. I distinguish between four classes of clocks:

- **Synchronized:** Clocks show the same time:

$$\forall i, j, t : \hat{t}_i(t) = \hat{t}_j(t). \quad (3.5)$$

Synchronicity is implicitly assumed in most prior algorithms that do not explicitly mention timing differences.

- **Skewed:** Clocks progress at a shared rate from distinct initial values:

$$\forall i, j : \exists c : \forall t : \hat{t}_i(t) - \hat{t}_j(t) = c. \quad (3.6)$$

Algorithms that may start at different times often assume this timing model.

- **Individual:** Clocks progresses at different, but fixed, rates:

$$\forall i, j : \exists a, b : \forall t : \hat{t}_i(t) - a\hat{t}_j(t) = b. \quad (3.7)$$

This is true of, e.g., a clock that gains a few minutes a day.

- **Variable:** The rate of each clock varies over time with some limited amount of variability:

$$\forall i, j : \exists c : \forall t : \left| \frac{\partial}{\partial t} \hat{t}_i(t) - \frac{\partial}{\partial t} \hat{t}_j(t) \right| \leq c. \quad (3.8)$$

While agents may not be able to *measure* it, I do assume the *existence* of a single global time t . Agents moving at relativistic speeds and other scenarios where a global time cannot be defined are not explicitly considered in the algorithms and proofs presented in this chapter.

This chapter investigates the rendezvous problem, which may be defined mathematically as follows:

Definition 3.1 (Rendezvous). *Let $s_i(t)$ be the state of an agent following algorithm A_i for t time units after starting in state $s_i(0)$.*

*Two algorithms, A_i and A_j , **rendezvous with** one another for a given r if, for every arbitrary pair of starting states $s_i(0)$ and $s_j(0)$, there exists a non-negative time t^* such that $d(s_i(t^*), s_j(t^*)) \leq r$, where t^* is bounded by some fixed finite function of $d(s_i(0), s_j(0))$ and r is a distance at which detection is guaranteed.*

*The set of algorithms A is said to be a **family of rendezvous algorithms** or to have the **rendezvous property** for a given r if, for any arbitrary pair $i \neq j$, A_i and A_j rendezvous with one another for that r .*

Where no confusion will result, r is not explicitly mentioned in the discussions that follow.

To facilitate discussion of worst-case bounds on rendezvous time, the definition of rendezvous incorporates a worst-case time bound by requiring a bounding function on t^* . Every reference to rendezvous in the remainder of this chapter refers to this bounded-time definition of rendezvous.

3.2 Necessary Capabilities

It is clearly not the case that every arbitrary set of agents may be made to rendezvous in finite time. This section contains a set of proofs describing particular capabilities that must be present in any agent able to achieve deterministic rendezvous in general environments. In particular, for some environments,

- Rendezvous requires each agent's algorithm be unique, either through unique computation or via the presence of unique inputs or parameterization.
- If $|A| > 2$, rendezvous requires either some kind of clock or a global coordinate system.
- Rendezvous is at least as difficult as finding static targets in bounded time.
- Bounds on mobility drift are required to be able to find static targets in bounded time, and thus also required for rendezvous.

Each of these requirements is addressed in its own section below.

3.2.1 Individuality

Theorem 3.1. *For any bounded-time rendezvous algorithm, the algorithm relies on one of the following assumptions:*

- *agents have prior knowledge of other agent's state;*
- *agents execute distinct algorithms;*
- *agents are guaranteed to receive distinct sensations; or*
- *agents are guaranteed to traverse distinguishable environments.*

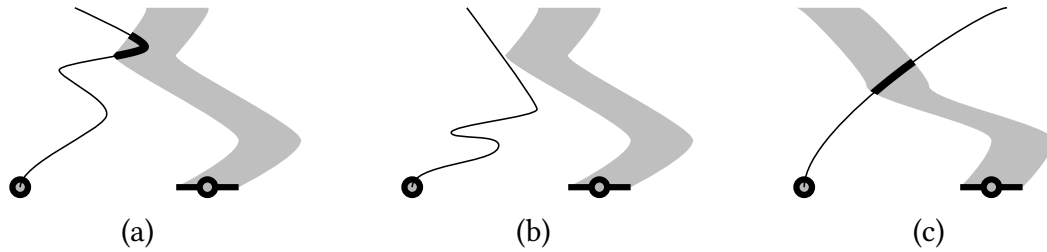


Figure 3.2: Illustration of the proof to **Theorem 3.2**, with time on the vertical axis and space on the horizontal. The thin line is p_i , shaded area is V_j . In (a) and (c) the thicker line is $p_i \cap V_j$. The rendezvous in (a) is prevented in (b) by clock error. In (c) the agents travel the reverse of one another's paths through space, meaning clock error can prevent rendezvous.

This theorem follows trivially from the observation that agents moving in lock-step never rendezvous. Agents executing the same algorithm at the same time with the same sensations and indistinguishable environments will move in lock-step if their initial orientations are equivalent.

Theorem 3.1 is elementary, but it does highlight the fact that individuality must come from somewhere. For rendezvous to work in featureless environments, this individuality must be built into the agents. Rather than rely on particular environmental features, the solutions to the rendezvous problem in this chapter utilize a unique identifier for each agent to parameterize the rendezvous algorithms the agents execute, ensuring that each agent behaves individually.

3.2.2 Temporality

Theorem 3.2. *For any bounded-time rendezvous algorithm applicable to three or more agents, at least one of the following must be true:*

- *The algorithm only works in single-dimensional environments;*
- *The algorithm relies on some estimate of the passage of time; or*
- *The algorithm relies on some enumeration of possible coordinate systems.*

Proof. If each pair of agents in a set of three or more are guaranteed to rendezvous, then at least two agents are mobile. Consider two such mobile agents, i and j . Let $p_i = \{(s_i(t), t)\}$ be the path through spacetime traveled by agent i and $V_j = \{(x, t) : d(x, s_j(t)) \leq r\}$ be the volume of spacetime guaranteed to be observed by agent j . For rendezvous to be guaranteed, the path must intersect the volume; that is $p_i \cap V_j \neq \emptyset$, as illustrated in [Figure 3.2](#). Note the choice of labels i and j is arbitrary; $p_i \cap V_j \neq \emptyset \Leftrightarrow p_j \cap V_i \neq \emptyset$ because r is constant and d is commutative.

If agents do not have any estimate of the passage of time then the t coordinate of each agent's motion cannot be controlled. In particular, p_i might be $\{(s_i(t), f(t))\}$ for any monotonically non-decreasing function f . Thus, in the absence of clocks rendezvous is guaranteed only if it is guaranteed for every f ; mathematically, clockless rendezvous is guaranteed if and only if the following implication holds:

$$(f(x) > f(y) \Rightarrow x > y) \Rightarrow \left(\exists t : d(s_i(t), s_j(f(t))) \leq r \right) \quad (3.9)$$

Czyzowicz et al. [11] demonstrated that (3.9) is satisfied only if each state in a prefix of each agent's trajectory is within r some state in the prefix of the other agent's trajectory, a property of paths which they call "tunnels." Tunnels of this sort can be guaranteed only if each agent knows, or can guess, the others' coordinate system.

Hence, rendezvous requires either an estimate of time or a finitely-enumerable set of possible coordinate systems. □

Many prior rendezvous algorithms assume both a single global coordinate system (typically via graph nodes or grid cells) and knowledge of the passage of time (often in discrete global clock cycles). However, only one of the two conditions in [Theorem 3.2](#) are required for deterministic

rendezvous. Czyzowicz et al. [11] demonstrated that a clock is not needed if the other agent's coordinate system comes from an enumerable set by presenting an algorithm that repeatedly guesses all possible coordinate systems of other agents. Their approach suffers from runtime polynomial in the cardinality of the set of possible coordinate systems; it is not tractable for open geometric environments. I demonstrate in this chapter that knowledge of other agents' possible coordinate systems is not needed if a clock is present through a set of algorithms that guarantee rendezvous even if mobility is noisy or drifting.

Theorem 3.2 does not specify the accuracy of clocks, but does imply that a clock must exist in some form. As my solutions handle some limited clock inaccuracy (see [Section 3.3.5](#) and [Section 3.3.6](#)) they provide a sufficient bound. I am unaware of any necessary bound on clock accuracy.

3.2.3 Search

Before presenting a proof that search is required I first define what I mean by search.

Definition 3.2 (Search). *An algorithm S is a **search algorithm** if, for every arbitrary starting state $s(0)$ and target location p , there exists a non-negative finite time t^* such that $d(s(t^*), p) \leq r$, where t^* is upper bounded by some fixed finite function of $d(s(0), p)$.*

Search algorithms can equivalently be characterized as algorithms that rendezvous with stationary agents (see [Definition 3.1](#)).

Theorem 3.3. *There exist environments such that for any family of rendezvous algorithms A there exists a search algorithm S that can be executed by any agent with sufficient capabilities to execute algorithms in A ; S requires asymptotically no more space or time than the most expensive $A_i \in A$.*

Proof. The existence of S for three classes of environments is demonstrated in [Lemma 3.1](#), [Lemma 3.2](#), and [Lemma 3.3](#) below. □

[Theorem 3.3](#) implies that search is generally no harder than rendezvous. I use this to build rendezvous algorithms that use search algorithms as a subroutine. This allows me to ignore the details of the environment that a search algorithm must consider and focus instead on the extra complications that arise during rendezvous.

There are environments where search cannot be reduced to rendezvous: for example, if the entire environment is a single smoothly-sloping hill then climbing to the top of the hill guarantees rendezvous with time proportional to the radius of the hill, while search requires time proportional to the area of the hill.

The environments where rendezvous is at least as hard as search include the three most common classes of environments in mobile agent research, as demonstrated by the following lemmas.

Lemma 3.1 (Featureless vector spaces). *Assume that the set of agent behaviors is closed under vector addition and subtraction of agent location; that d is an induced norm over that vector space; and that agents do not receive location-dependent inputs such as GPS signals. Then for any any family of rendezvous algorithms A , for any arbitrary $A_i, A_j \in A$, $s(t) = s_i(t) - s_j(t) + s_j(0)$ is a search algorithm.*

Proof. The definition of A_i and A_j rendezvousing is:

$$\forall s_i(0), s_j(0) \exists t^* d(s_i(t^*), s_j(t^*)) \leq r. \quad (3.10)$$

Because d is an induced norm, subtracting $s_j(t) - s_j(0)$ from both paths does not change the distance:

$$\forall s_i(0), s_j(0) \exists t^* d(s_i(t^*) - (s_j(t^*) - s_j(0)), s_j(0)) \leq r \quad (3.11)$$

which is the definition of $s(t) = s_i(t) - s_j(t) + s_j(0)$ being a search algorithm. \square

Lemma 3.2 (Unconstrained graphs). *Assume agents move along edges of a connected graph and can determine its structure only locally. Then for any any family of rendezvous algorithms A , for any two algorithms in a set A achieving rendezvous, at least one is a search algorithm.*

Proof. Assume two agents are executing a pair of rendezvous algorithms but that neither is executing a search algorithm. Since neither is searching, there exists some graph where each agent has some node it never visits. Because the graph structure is known only locally, neither agent can know how much of the graph is inaccessible without traversing its excluded node(s). In particular, every path between the two agents might include a subpath of length at least r accessible only through nodes each agent will not traverse. But this situation contradicts our assumption of rendezvous. Thus, at least one of agent is executing a search algorithm. \square

Lemma 3.3 (Cluttered Environments). *Assume agents navigate an environment that may be modeled as the union of a set of convex polytopes, but is otherwise unconstrained. Then for any any family of rendezvous algorithms A , there exists a search algorithm S that requires asymptotically no more space or time than the most expensive $A_i \in A$.*

Proof. Both featureless vector spaces and undirected embeddable graphs can be modeled as the union of a set of convex polytopes. Thus, by both [Lemma 3.1](#) and [Lemma 3.2](#), search is reducible to rendezvous. \square

3.2.4 Limited Drift

Theorem 3.4. *For any finite sensing radius r , searching a sufficiently-large area in an n -dimensional Euclidean environment requires drift to accumulate no more quickly than $\Theta\left(\sqrt[n]{x}\right)$, where x is the distance the agent travels.*

Proof. We show that if error is in $\omega\left(\sqrt[n]{x}\right)$ then at some point the agent will no longer be able to make progress in expanding the area it knows it has searched.

Observe that the surface area a of area A is in $O\left(\sqrt[n]{A^{n-1}}\right)$. Observe also that the distance x the agent must travel to be guaranteed to sense area A is in $O(A)$ (because r is finite).

Consider the last Δx traveled by an agent. Although that motion will add $\Omega(\Delta x)$ newly viewed area to the searched region, it will also result in $\Delta\epsilon = \epsilon(x) - \epsilon(x - \Delta x)$ more uncertainty in the agent's position. This uncertainty shrinks the area the agent knows it has viewed by $\Delta\epsilon$ times the surface area a of the known-visited area.

For sufficiently large A , $\Omega(\Delta x) < (a)(\epsilon(x) - \epsilon(x - \Delta x))$. This may be shown by re-writing:

$$\begin{aligned}\Omega(\Delta x) &< (a)(\epsilon(x) - \epsilon(x - \Delta x)) \\ \Omega(\Delta x) &< O\left(\sqrt[n]{O(x)^{n-1}}\right) \left(\omega\left(\sqrt[n]{x}\right) - \omega\left(\sqrt[n]{x - \Delta x}\right)\right)\end{aligned}\tag{3.12}$$

Since $n \geq 1$, we can consolidate the asymptotic classes by introducing a new variable $k \in \omega(1)$ to obtain

$$\Delta x < \sqrt[n]{x^{n-1}} \left(\sqrt[n]{x} - \sqrt[n]{x - \Delta x}\right) k\tag{3.13}$$

which can be further rewritten as

$$\begin{aligned}
\Delta x &< kx - k\sqrt[n]{x^n - x^{n-1}\Delta x} \\
kx - \Delta x &> k\sqrt[n]{x^n - x^{n-1}\Delta x} \\
(kx - \Delta x)^n &> k^n x^n - k^n x^{n-1} \Delta x \\
k^n x^n - \Theta(k^{n-1} x^{n-1} \Delta x) &> k^n x^n - k^n x^{n-1} \Delta x \\
-\Theta(k^{n-1} x^{n-1} \Delta x) &> -k^n x^{n-1} \Delta x,
\end{aligned} \tag{3.14}$$

the final form of which is true by the definition of k being in asymptotic class $\omega(1)$.

Since the last Δx traveled by the agent did not increase the area it knows it has seen, it must have already known it had seen A before traveling the final Δx . But that means the same argument holds for the previous Δx . By induction the agent can never have become confident it had searched area A . Hence, for sufficiently large A , ϵ cannot be in $\omega\left(\sqrt[n]{x}\right)$. \square

Because drift must be limited for deterministic search, by [Theorem 3.3](#) it must also be limited for rendezvous.

3.3 Parameterized Algorithm Families

This section contains techniques that achieve rendezvous for several different classes of agents. Each of these techniques achieves rendezvous for a different class of positioning and timing inaccuracy, demonstrating a set of sufficient capabilities to match the necessary capabilities proven in the previous section. The techniques also provide achievable bounds on the runtime of ren-

Listing 3.1 $\mathcal{R}_{p,q}$: Rendezvous without drift

1. Repeat until rendezvous achieved:
 2. Repeat p times:
 3. For each bit x in the key:
 4. If $x = 1$: search and return q times
 5. Otherwise: wait for q time steps
 6. Double the time step
-

Listing 3.2 \mathcal{D}_q : Rendezvous with drift

1. Repeat until rendezvous achieved:
 2. For each bit x in the key:
 3. If $x = 1$: search and return q times
 4. Otherwise: wait for q time steps
 5. Increase the time step to double the search radius
-

deztvous algorithms. Because my objective is to minimize worst-case performance, I intentionally ignore various valid but unrelated average-case optimizations.

While the correctness and runtime of each technique are separately addressed for each class of agents, all the techniques belong to one of the two parameterized families of algorithms presented in [Listing 3.1](#) and [Listing 3.2](#) as $\mathcal{R}_{p,q}$ (\mathcal{R} for “rendezvous” as it is the main family of rendezvous algorithms) and \mathcal{D}_q (\mathcal{D} for “drift” as it is designed to handle agents with limited mobility drift).

Both families of algorithms are based on ideas of keys, bits, time steps, and searching. Each agent’s behavior is broken into **time steps**. In each time step an agent either remains stationary or it executes a **search subroutine** to explore as large an area as possible while still returning to its starting point before the time step ends (or as close thereto as mobility error allows). Which action it takes is based on the **bits** of a binary **key**. On a 1 bit the agent executes q distinct searches; on a 0 bit the agent remains stationary for q times steps. The agent also periodically increases its time step; the length of time that an agent’s time steps remain a single duration is

called a **cycle**. In $\mathcal{R}_{p,q}$ a cycle lasts for p complete runs through the bits of the key—i.e., for a b' bit key a cycle is $b'pq$ time steps long. In \mathcal{D}_q a cycle lasts only for a single bit. \mathcal{R} and \mathcal{D} also differ in how much the time step is increased between cycles, as noted in the listings and explored more in [Section 3.3.4](#).

Rendezvous between two agents may occur at any time. Agents that start or drift close together may rendezvous when both are waiting; two agents may also rendezvous while both are searching. Rather than attempting to characterize the likelihood of these possibilities for a particular environment and set of agent capabilities, I establish worst-case bounds based only on rendezvous that occur when one agent is waiting and the other agent searches its waiting location.

As justified by [Theorem 3.1](#), I assume that each agent has access to a unique b -bit identifier, where 2^b is an upper bound on the number of agents that might want to rendezvous with one another. From that b -bit identifier I derive a b' -bit key, the bits of which are used in \mathcal{R} and \mathcal{D} . The properties that must be satisfied by these keys are specified for each set of agent capabilities separately in the following sections. Letting i be the identifier and k the key, the required properties may be achieved as follows

- The keys are distinct. Using $k = i$ and $b' = b$ suffices.
- If a key is not zero, its low-order bit is set. This may be achieved by $k = 2i + 1$ and $b' = b + 1$.
- Each key begins and/or ends with a common n -bit sequence. This may be achieved by concatenating bits onto i , giving $b' = b + n$.
- The keys are taken from a shift-free set.

This last property requires some explanation.

Definition 3.3 (Shift-free). *Let x , y , k , and b be non-negative integers, x and y be less than 2^b , and k be less than b . The following are all defined with respect to b :*

- $x \text{ rotl}_b k$ is the k -bit **circular shift** of x w.r.t. b , defined as $(2^k x \bmod b) + \lfloor 2^{k-b} x \rfloor$.
- x and y are **shift-similar** w.r.t. b if $\exists k : x \text{ rotl}_b k = y$.
- x is **shift-minimal** w.r.t. b if $\forall k : x \text{ rotl}_b k \geq x$
- A set of b -bit numbers is **shift-free** if no two elements of the set are shift-similar w.r.t. b .

A set of keys taken from a shift-free set are called **shift-free keys**. Without loss of generality, I assume all shift-free keys are shift-minimal.

The largest shift-free set with respect to b contains more than $\frac{2^b}{b}$ elements. Thus, there are mappings from b -bit identifiers to $\lceil b + \log_2(b) + 1 \rceil$ -bit shift-free keys. Such mappings are not trivial to describe; a simpler mapping takes a b -bit identifier x and uses $2x + 1$ as a $(2b + 1)$ -bit shift-free key.

3.3.1 Without Uncertainty

The simplest rendezvous approach works for synchronized agents with exact mobility. This situation establishes a baseline from which the other algorithms in this chapter may be understood.

Theorem 3.5. *Any two agents with distinct keys executing $\mathcal{R}_{1,1}$ at the same time with the same time step will rendezvous with each other in less than $4b'$ times the time required for one agent to search the location of the other, or within b' times the initial time step.*

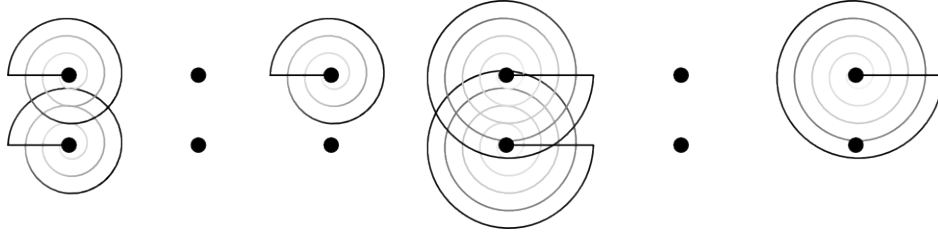


Figure 3.3: Rendezvous of agents without uncertainty with keys 101 and 100 using $\mathcal{R}_{1,1}$. The initial time step is too small; after doubling the timestep the agents rendezvous in the first distinct bit of their keys.

Proof. Let t_0 be the initial time step and T be the time step required for a search by one agent to find the initial location of the other agent. If the time step is initially $t_0 < T$, the time step will be in $[T, 2T)$ after completing the last cycle with a timestep $S < T$, which happens no later than $(t_0 + 2t_0 + 4t_0 + \dots + S)b' < 2b'S < 2b'T$.

Once the time step is at least T , rendezvous will occur within $b'T$. This follows because distinct keys must differ in some bit. Thus, there must be some time during which one agent is searching (the active bit of its key is 1) while the other agent is waiting (its active bit is 0). During that bit, the agent searching will pass within r of the agent waiting, resulting in rendezvous.

Final runtime is bounded by $2b'T$ to achieve $t \in [T, 2T)$ and another $2b'T$ to perform b' searches at that timestep, for a total worst-case runtime of $\max\{4b'T, b't_0\}$. \square

3.3.2 Skewed Clocks

Consider agents that have exact mobility and whose search subroutines are deterministic in the order in which they search the environment, but that have skewed clocks. Because some distinct numbers are not distinguishable if not synchronized (e.g., 01 and 10), these agents will need to use shift-free keys. To demonstrate that $\mathcal{R}_{2,1}$ will guarantee rendezvous, first consider the following lemma.

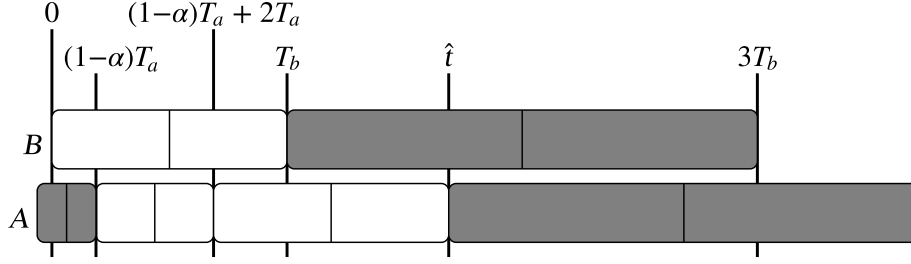


Figure 3.4: Illustration of doubling proof. A rounded box is drawn around each cycle, divided at the point where the key repeats; A's T_a cycle and both $2T_b$ cycles are shaded.

Lemma 3.4 (Doubling). *For any two agents A and B with cycle durations $T_a \leq T_b$, if T_a and T_b differ by a power of two at the beginning of B's cycle then at least half of their $2T_b$ cycles overlap.*

Proof. Call the start of B's cycle $t = 0$. Let $\alpha \in [0, 1)$ be the portion of A's T_a -duration cycle that is completed prior to $t = 0$. A's cycle reaches $2T_b$ at $\hat{t} = (1 - \alpha)T_a + 2T_a + 4T_a + \dots + T_b$. Observe that $T_b < \hat{t} < 2T_b$, so \hat{t} must fall within the first half of B's $2T_b$ cycle $[T_b, 3T_b]$.

This proof is illustrated in [Figure 3.4](#). □

Theorem 3.6. *For any two agents with skewed clocks and a repeatable search algorithm, $\mathcal{R}_{2,1}$ achieves rendezvous within two doublings of the initially-longer time step or the end of the first cycle where the search areas are sufficiently large.*

Proof. Because the keys are shift-free and because each agent searches the same location at the same time within each 1 bit, rendezvous is guaranteed if they share a time step that is large enough for them to find one another's resting locations for one full iteration through the bits of their keys. Because $\mathcal{R}_{2,1}$ repeats the key twice per cycle, it is sufficient to show that they eventually share a time step for half a cycle.

By [Lemma 3.4](#), both agents will have overlapped for at least half a cycle by the end of the second doubling of the longer time step. Because they will continue to overlap for at least half

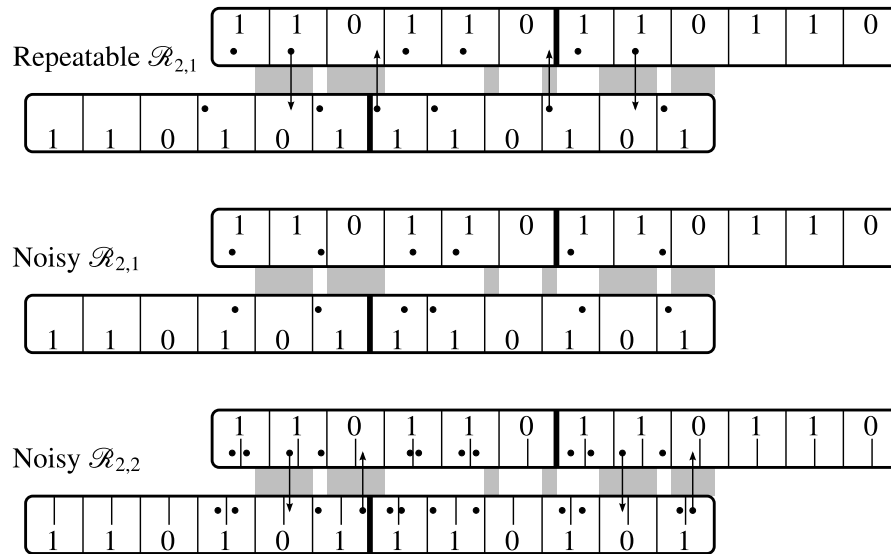


Figure 3.5: Comparison of $\mathcal{R}_{2,1}$ and $\mathcal{R}_{2,2}$. Dots represent the moment within a 1 bit when a searching agent passes the resting location of the other agent. Grey bands are times when search, if performed, guarantees rendezvous. Arrows are times when rendezvous occurs.

a cycle every cycle thereafter, if that first time-step overlap does not have large enough searches to generate rendezvous then the first search that is large enough will result in rendezvous. \square

$\mathcal{R}_{2,1}$ with shift-free keys results in rendezvous in twice the time that $\mathcal{R}_{1,1}$ did (or more if the initial timestep is large): $\max\{8b'T, 6b't_0\}$, where T is the time required for one agent to search the location of the other and t_0 is the larger of the two initial timesteps.

3.3.3 Noise or Nondeterministic Search

If the search subroutine is non-deterministic and/or the mobility suffers from noise then we cannot rely on passing a target's waiting location at the same time within each bit. Timing uncertainty within a search can prevent rendezvous unless an entire search occurs while the other agent is waiting. For agents with skewed clocks and mobility noise, $\mathcal{R}_{2,2}$ with shift-free keys

searches twice per bit, ensuring that at least one entire search occurs within the other agent's waiting period.

Theorem 3.7. *For any two agents with shift-free keys, accurate asynchronous clocks, and either experiencing bounded positioning error or unrepeatable search algorithms, $\mathcal{R}_{2,2}$ achieves rendezvous within four doublings of the initially-longer time step or the end of the first cycle where the search areas are sufficiently large.*

Proof. By searching twice per bit, $\mathcal{R}_{2,2}$ ensures that at least one entire search occurs within the other agent's waiting period. Full-search overlap means variations in the details of individual positioning or search pattern will not prevent rendezvous.

The rest of the proof mirrors that of [Theorem 3.6](#). □

$\mathcal{R}_{2,2}$ takes twice as long as $\mathcal{R}_{2,1}$, discussed in the previous section: rendezvous is achieved within $\max\{16b'T, 12b't_0\}$.

3.3.4 Position Drift

When an agent's sense of location becomes less accurate as the agent moves, it becomes necessary to increase the search area more quickly than drift can accumulate. This means increasing the search radius after every bit, as is done in the \mathcal{D} family of algorithms. The \mathcal{D} algorithms differ from the \mathcal{R} algorithms by doubling the search radius after every bit instead of doubling the time step after every cycle. In n -dimensional Euclidean environments, doubling the radius means multiplying the time step by 2^n .

Theorem 3.8. *For any agents who, after searching radius R in all directions, accumulate mobility error not exceeding mR for some constant m ,*

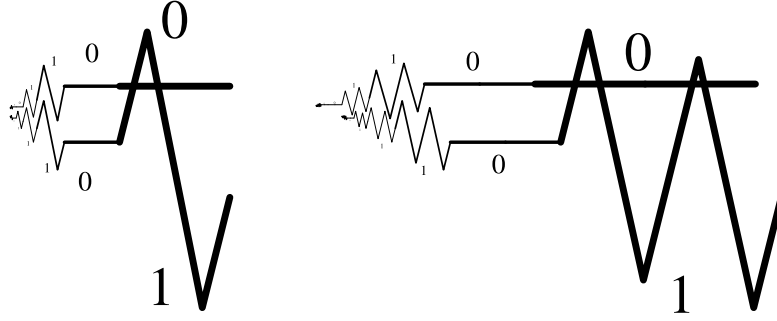


Figure 3.6: Example of 1D agents rendezvous with drift and keys 1100 and 1101. On the left are synchronized agents using \mathcal{D}_1 ; on the right are skewed agents using \mathcal{D}_2 .

- \mathcal{D}_1 and unique keys guarantee rendezvous if the agents have synchronized clocks and $m < \frac{1}{2}$.
- \mathcal{D}_2 and unique keys guarantee rendezvous if the agents have skewed clocks and $m < \frac{1}{4}$.

Proof. Consider two arbitrary synchronized agents with initial separation d_0 and an initial search that explores x in every direction. The first bit can cause each agent to drift mx , meaning $d_1 \leq d_0 + 2mx$. The second bit explores $2x$ and might lead to $d_2 \leq d_0 + 2mx + 4mx$. The i th bit explores $2^i x$ and might lead to separations of $d_i \leq d_0 + 2mx(1 + 2 + 4 + \dots + 2^{i-1}) = d_0 + 2mx2^i - 2mx$. Rendezvous will occur no later than the first differing bit (which must exist since the keys are unique) once $2^i x \geq d_0 + 2mx2^i - 2mx$.

If agents have skewed clocks, by [Lemma 3.4](#) at least half of their cycles will overlap within a single doubling of the initially-longer time step. Because two searches are executed per bit, drift accumulates twice as quickly as it does for synchronized agents: the i th bit explores $2^i x$ and might lead to separations of $d_i \leq d_0 + 4mx(1 + 2 + 4 + \dots + 2^{i-1}) = d_0 + 4mx2^i - 4mx$. Rendezvous will occur no later than the first differing bit (which must occur since the keys are shift-free) once $2^i x \geq d_0 + 4mx2^i - 4mx$.

Combining the above two statements, rendezvous is guaranteed for \mathcal{D}_q once $2^i x > d_0 + 2qmx2^i - 2qmx$. Solving for i gives

$$2^i x > d_0 + 2qmx2^i - 2qmx \quad (3.15)$$

$$2^i(x - 2qmx) > d_0 - 2qmx$$

If $2qmx > d_0$ then rendezvous will occur no later than the first non-equal bit. Otherwise,

$$2^i > \frac{d_0 - 2qmx}{x - 2qmx} \quad (3.16)$$

$$i > \log_2 \left(\frac{d_0 - 2qmx}{x - 2qmx} \right)$$

The maximum number of cycles before rendezvous is thus $i + b'$, which is a finite number because $m < \frac{1}{2q}$ and both d_0 and x are finite. \square

This proof is overly conservative in assuming all bits are 1, or equivalently that error grows with time rather than distance, accumulating even while the agent is at rest on a 0 bit. In practice somewhat larger drifts should be permissible if drift does not accumulate during 0 bits since at least one key has at least one 0 bit.

The time required for rendezvous with drift is not simply expressed. For n -dimensional Euclidean environments doubling the search radius increases the time by a factor of 2^n , giving a runtime in $O\left(2^{nb'i}\right) = O\left(\left(\frac{d_0 - 2qmx}{x - 2qmx}\right)^{nb' i}\right)$: hyperbolic in drift, polynomial in the number and initial separation of agents. In other environments the relationship between search radius and time may result in a different bound.

1	1	0	0	0
1	1	1	0	0

Figure 3.7: Worst-case behavior of variable-speed clocks can cause no search/wait pairings to ever occur.

3.3.5 Variable Clocks

When agents have variable clocks, some time steps might be briefer than others. This variability can cause the previous algorithms to never result in rendezvous. Consider, for example, an agent with key 00111 and another with key 00011. If the first agent's clock runs more quickly on its 1 bits and the other agent's more quickly on its 0 bits, rendezvous might not happen; this worst-case scenario is illustrated in [Figure 3.7](#). The worst-case pair of keys is generally $2^a - 1$ and $2^{a+1} - 1$, where $a = \left\lfloor \frac{1}{2}b' \right\rfloor$.

Lemma 3.5. *Given each agent has a clock with a unit of time varying within $[x, y]$, there exists a pair of shift-free keys such that then the minimal 1/0 overlap's duration is $x + \left\lfloor \frac{1}{2}b' \right\rfloor (x - y)$.*

This lemma follows the squeeze-and-stretch argument above.

Theorem 3.9. *Consider any pair of agents, each having a clock with a unit of time that varies within $[x, y]$. Let $a = \left\lfloor \frac{1}{2}b' \right\rfloor$ and $q = \left\lfloor \frac{y}{x - a(y - x)} \right\rfloor$. Then rendezvous is guaranteed by*

- $\mathcal{R}_{2,q}$ if the pair of agents have precise mobility;
- $\mathcal{R}_{2,2q}$ if the pair of agents have mobility noise; and
- \mathcal{D}_q if the pair of agents have mobility drift no greater than $\frac{1}{2q}R$.

Proof. By [Lemma 3.5](#), searching q (or $2q$) times per bit ensures one (or two) search/wait overlaps in the same cycle that searching once (or twice) per bit would for skewed clocks. The rest of the proof parallels those of [Theorem 3.6](#), [Theorem 3.7](#), and [Theorem 3.8](#) above. \square

Worst-case runtime is q times the runtimes for skewed, noisy, and/or drifting rendezvous, respectively.

Larger clock variability could be handled by additional restrictions on keys or by using a pattern of searches and waits within each bit. I do not investigate particulars of these approaches in this dissertation.

3.3.6 Individual Clocks

When each agent's clock runs at a consistent but individual pace, potentially different from each other agents' clock, rendezvous can be assured by using keys with guaranteed placement of some 0 and 1 bits.

Theorem 3.10. *Assume that each agent has a unique b -bit identifier, and that its $(2b + 2)$ -bit key is, in order, a 1 bit, b 0 bits, another 1 bit, and the identifier. Using this key, $\mathcal{R}_{1,2}$ guarantees rendezvous.*

Proof. Call the agent with the initially longer-period clock s (for “slow”) and the other f (for “fast”). Call the first 1 bit in f 's key f_1 , the second 1 bit f_2 , the b zeros f_0 and the b -bit unique number f_n . Let f'_1 refer to the first one bit of f 's second pass through its key. Define s_1, s_2 , etc., similarly.

Consider the first cycle of f beginning when $f : s$ differ by no more than $1 : 2$. I demonstrate rendezvous by a large set of trivial cases. For succinctness, I use the phrase “ f_x in s_y ” to mean that more than half of the f_x bit falls within s_y .

The proof now proceeds by an exhaustive enumeration of cases, each of which is fairly trivial.

1. f_1 in s_1 is divided into sub-cases based on f_2 :

1.1. f_2 in s_0 results in rendezvous during f_2 .

1.2. f_2 in s_2 means $f : s$ is no less than $b : b + 1$. Consider two additional sub-sub-cases:

1.2.1. f'_1 in s_n means s'_1 will be in f'_0 , resulting in rendezvous.

1.2.2. f'_1 in s'_1 means each bit of f_n is in the corresponding bit of s_n , resulting in rendezvous because $f_n \neq s_n$.

2. f_1 in s_0 results in rendezvous during f_1 .

3. f_1 in s_2 is divided into three sub-cases based on f_2 :

3.1. f_2 in s_n is divided into sub-cases based on f'_1 .

3.1.1. f'_1 in s'_1 results in rendezvous during the next cycle by the first case above.

3.1.2. f'_1 in s'_0 results in rendezvous during f'_1 .

3.2. f_2 in s'_1 means each bit of s_n is in f_0 , resulting in rendezvous because $s_n \neq 0$.

4. f_1 in s_n is divided into sub-cases based on f_2 :

4.1. f_2 in s_n Since $f : s$ differ by no more than $1 : 2$, f'_1 must lie within s'_0 , resulting in rendezvous.

4.2. f_2 in s'_1 is divided into sub-sub-cases based on f'_1 .

4.2.1. f'_1 in s'_0 results in rendezvous during f'_1 .

4.2.2. f'_1 in s'_2 means each bit of f_n is in s'_0 , resulting in rendezvous because $f_n \neq 0$.

4.3. f_2 in s'_0 results in rendezvous during f_2 .

Each level of the case enumeration above is complete; other cases are all precluded either by f being faster than s or by f being no more than twice as fast as s . The longest delay before rendezvous is s''_1 in f''_0 for case 3.1.1 followed by case 1.2.1. □

Final runtime is bounded by $24b'T$: $4b'T$ to achieve 1:2 timing, $4b'T$ to reach f' , $8b'T$ to reach f'' , and $8b'T$ to reach to reach the end of f''_0 .

3.4 Cooperative Rendezvous

This chapter has addressed the issue of individual rendezvous. If hives of agents can cooperate to rendezvous with other hives the task becomes much easier: one or more agents can wait to be found while the others search for other hives. Thus rendezvous for hives reduces to the underlying search algorithm plus some technique for division of labor. Such division of labor can be performed without communication utilizing emergent formation algorithms [12, 14, 45].

Groups of agents organized as in a swarm rather than a hive (that is, agents that need to remain close to one another as they move) can use the rendezvous algorithm presented in this chapter. The only coordination needed is to select a common key or, equivalently, follow a single leader. Besides a possible increase of r with multiple agents involved, no additional efficiency is created by rendezvousing swarms instead of single agents.

3.5 Conclusion

This chapter has investigated the rendezvous problem. I have demonstrated that it is possible for agents to find one another in an unknown, unbounded environment in finite time. I have characterized capabilities required of any agents that execute bounded-time rendezvous algorithms and have presented algorithms ensuring rendezvous for agents with slightly more generous capabilities. For agents with consistent clocks and no positioning drift, these algorithms are tractable and achieve the known asymptotically-optimal time bounds from line- and graph-like environments.

The algorithms I have presented work in any environment by relying on an environment-specific search function I did not define. While I proved that such a search function must exist for any agents that are capable of achieving bounded-time rendezvous, I did not prove that using such a function is optimal. Development of environment-specific rendezvous algorithms that are more efficient than my general approaches is an area for future investigation.

I also did not demonstrate a lower bound on the quality of clocks needed for rendezvous. Czyzowicz et al. [11] demonstrated that there is no such lower bound if super-exponential runtimes and shared orientation are considered. I added a few more data points showing the interplay of runtime and clock accuracy in [Section 3.3.5](#) and [Section 3.3.6](#); continuing to characterize this trade-off is an area for future investigation.

Many earlier algorithms assume some shared global orientation or global position. The algorithms and proofs I present do not assume or make use of this kind of information. Potential speedups that such information might enable is an area for future investigation. Given knowledge of a global space-time location, some variation of landmark search might offer more efficient rendezvous than the search-based approaches I have presented.

Rendezvous is a key sub-problem in achieving agent aggregation. That use is discussed in more detail in [Chapter 6](#).

Chapter 4

Cohesion Constraints

Cohesion requires that a group of moving agents remain a single group as they move.

Cohesion is generally achieved by selective application of a cohesion constraint which prevents pairs of agents becoming separated. In this chapter I develop a technique for creating computationally-tractable distance-based cohesion constraints for broad classes of agents. The constraints are provably both safe (do not break cohesion) and live (lead to states where the same constraints continue to apply). They also do not depend on explicit communication, nor do they specify which cohesion-maintaining behavior an agent may select.

The objective of cohesion algorithms is to have mobile computational agents already in proximity of one another remain proximal as they move. Cohesion is assumed in many decentralized artificial intelligence (AI) tasks, as it allows the group to communicate and react together. As a piece of other tasks, cohesion algorithms should not fully dictate agent mobility, instead letting the other tasks take as much control as possible.

Many decentralized AI algorithms assume cohesion without providing mechanisms to maintain it; others provide cohesion in either an overly-restrictive or non-general way. Flocking and formation algorithms maintain connectivity by dictating significant portions of each agent's actions. Potential fields are verifiable and are extensible with additional objectives, but the extensions are often not verifiable. Some algorithms combine cohesion with other agent objectives, but such approaches are not readily transferable to new agents or objectives.

There are two basic approaches to providing separable, transferable guarantees of cohesion. Switching laws alternate between cohesion-maintaining behaviors and task-specific behaviors. Cohesion constraints reduce the set of behaviors from which a task-specific objective may select to those which are known to maintain cohesion. This chapter presents a new technique for designing cohesion constraints that has significantly more versatility than previous techniques.

Cohesion constraints apply between pairs of agents and restrict their set of available behaviors to a set that guarantees the pair remains connected. Ideally a cohesion constraint should be **safe** (guarantees cohesion in the short term), **live** (leads to states where the constraint remains applicable), **composable** (multiple constraints on the same agent have a non-empty intersection), **tractable** (computable in limited time), and **loose** (leaves a large set of behaviors to choose from). I prove that the technique in this chapter generates constraints satisfying the first four of these properties for agents with distance-based connectivity requirements. I also demonstrate that looseness can depend on details of agent design and can negatively impact tractability.

This chapter discusses cohesion constraints. Building full cohesion algorithms using cohesion constraints is discussed in [Chapter 5](#).

4.1 Terminology

In addition to the general definitions presented in [Section 1.2](#), this chapter makes frequent use of additional notation and terminology.

Piecewise-defined behaviors figure prominently in this section. For conciseness, I introduce the notation $t_0[A|_B]$, where A and B are behaviors, to mean the behavior where the agent follows A until time t_0 and then switches to B :

$$t_0[A|_B]_s(t) \triangleq \begin{cases} A_s(t) & 0 \leq t \leq t_0 \\ B_{A_s(t_0)}(t - t_0) & t_0 < t. \end{cases} \quad (4.1)$$

Observe that s is in the domain of $t_0[A|_B]$ if and only if s is in the domain of A and $A_s(t_0)$ is in the domain of B .

A state m is a **midpoint** between two other states s_1 and s_2 if $d(s_1, m) = d(s_2, m) = \frac{1}{2}d(s_1, s_2)$. Most geometry-based distance functions have midpoints; graph-based distance functions often do not.

Two agents with states s_1 and s_2 are **r -connected** if $d(s_1, s_2) \leq r$. Where confusion will not result, the term **connected** is shorthand for r -connected where r is a fixed constant small enough that any pair of agents no more than r apart can sense one another.

A **cohesion constraint** requires that a pair of neighbors remain connected. In particular, pairs of agents satisfying a cohesion constraint which are connected at time t will also be connected at time $t + \epsilon$ for all sufficiently-small positive ϵ .



Figure 4.1: Agents can be disconnected by distance (left-most agent) or by entering a state where future distance disconnection is inevitable (right-most agent). Cohesion constraints prevent agents from entering these situations.

4.2 Cohesion Predicate

This section presents a technique for creating computable predicates $C : \mathcal{S}^2 \times \mathcal{B} \rightarrow \{\text{true}, \text{false}\}$.

The purpose of C is to identify a subset of behaviors that guarantee cohesion is maintained. A particular C may be constructed for a given definition of \mathcal{S} , \mathcal{B} , and r .

For the predicates in this chapter to function, there must be a known fixed time interval Δt that is the largest possible delay between consecutive decisions of a single agent. Without such a bound, only trivial cohesion constraints can be shown to be live.

Defining a cohesion predicate is complicated by the fact that not all agents currently close together might even be able to remain close. In particular, it is not possible to provide behaviors that guarantee connected agents remain connected without restricting the definition of \mathcal{S} , d , r , and/or \mathcal{B} . This may be demonstrated by agents that are currently close together but who are traveling in opposite directions with momentum that will carry them far apart (see [Figure 4.1](#)).

To handle less-maneuverable agents I discuss a restricted sense of connection that considers guaranteed-future-separation to be disconnected. This is done by defining a default behavior \mathcal{N} which will cause agents to enter a state where they stop moving away from one another and a special distance function d_C that encodes the idea “how far the agents might get if they follow \mathcal{N} .” The remainder of this section then used d_C instead of d when discussing distance. Terms

r , d_C -connected, d_C -connected, and d_C -cohesion constraint are defined to be the same as the corresponding terms without d_C (see Section 4.1), except that they use d_C instead of d to measure distance.

Although conceptually and practically d_C is defined in terms of \mathcal{N} , the properties that each predicate C must satisfy to ensure cohesion are existence proofs and thus work backward, defining \mathcal{N} as a witness to the success of C in maintaining d_C -connectivity. These mathematical properties are:

1. d_C -based connectivity must imply d -based connectivity. Formally,

$$\forall s_1, s_2 \in \mathcal{S} \quad d_C(s_1, s_2) \geq d(s_1, s_2). \quad (4.2)$$

The distance function d_C defines the notion of C -connectivity. The set of agents C -connected to A is a subset of the set of agents connected to A .

2. C must be safe; that is, behaviors that satisfy C must not break C -connectivity. Formally,

$$\begin{aligned} (d_C(s_1, s_2) \leq r \wedge C(s_1, s_2, \mathbf{B}) \wedge C(s_2, s_1, \mathbf{B}')) \implies \\ \forall t \in [0, \Delta t] \quad d_C(\mathbf{B}_{s_1}(t), \mathbf{B}'_{s_2}(t)) \leq r. \end{aligned} \quad (4.3)$$

3. C must be live and composable; that is, any agent must always be able to select a behavior that satisfies C for all of its C -connected neighbors. Formally,

$$\exists \mathcal{N} \in \mathcal{B} \quad \forall s_1, s_2 \in \mathcal{S} \quad d_C(s_1, s_2) \leq r \Rightarrow C(s_1, s_2, \mathcal{N}). \quad (4.4)$$

The existence of behavior \mathcal{N} is used to guarantee that agents do not enter a state from which cohesion cannot be maintained. In practice, d_C is constructed based on a particular \mathcal{N} as discussed in given in [Section 4.2.1](#) and [Section 4.2.2](#) below.

If tractability is relaxed, greater looseness can be gained by using

$$\forall s_1 \in \mathcal{S} \quad \exists \mathcal{N} \in \mathcal{B} \quad \forall s_2 \in \mathcal{S} \quad d_C(s_1, s_2) \leq r \Rightarrow C(s_1, s_2, \mathcal{N}) \quad (4.5)$$

but the alternating quantifiers and contextually-defined \mathcal{N} make that looser property difficult to compute.

These three properties ensure that a subset of initially-connected agents (defined by a special distance function d_C) will maintain connectivity for as long as they continue to select behaviors satisfying C at least every Δt time units, and also guarantee that a selection is always possible.

The remainder of this section describes how to achieve each property.

4.2.1 Liveness, Composability, and the “null” behavior \mathcal{N}

Guaranteeing that agents maintain cohesion throughout future times requires ensuring that each agent can select a cohesion-maintaining behavior no matter how long it has been operating or how many neighbors it has. This assurance can be provided through the definition of a “null” behavior \mathcal{N} , a safe default that any agent can adopt at any time without sacrificing cohesion.

In pathological cases there may not be any suitable \mathcal{N} ; an example might be if every distinct pair of trajectories diverges. I consider only agents and environments for which some \mathcal{N} exists such that agents following \mathcal{N} do not increase in distance from one another.

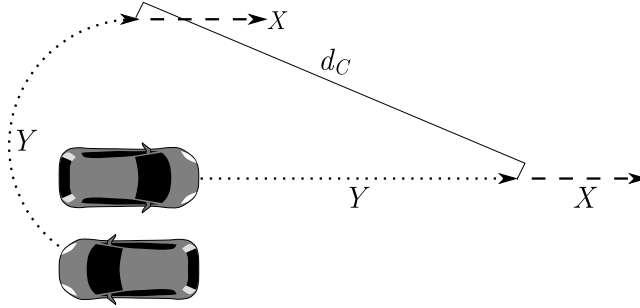


Figure 4.2: Example definitions of \mathcal{N} and d_C . X is “move east”; Y is “turn toward the east”. Together they define $\mathcal{N} = t \left[\begin{smallmatrix} Y \\ X \end{smallmatrix} \right]$. The maximum separation while following \mathcal{N} occurs in $[0, t^*]$ and defines d_C .

Many d_C and \mathcal{N} exist such that (4.4) is satisfied. I present below one particular approach for defining a \mathcal{N} given the existence of some behavior that does not increase separation. The approach discussed ensures that agents following the \mathcal{N} it produces reach a maximum d -separation in a bounded time. This bound is utilized in the computational approach in [Section 4.3](#).

1. Define a live (but not necessarily universal) behavior X that satisfies the property

$$\forall t \geq 0 \quad d(s_1, s_2) \geq d(X_{s_1}(t), X_{s_2}(t)). \quad (4.6)$$

Examples of X might be staying in place, moving in lockstep, or moving along convergent trajectories.

2. Define a universal behavior Y that moves an agent from any state to a state that is in the domain of X . For example, if X was “remain stationary” then Y might be “decelerate.”
3. Let \mathcal{N} be the universal behavior $t^* \left[\begin{smallmatrix} Y \\ X \end{smallmatrix} \right]$ where t^* is the time an agent must follow behavior Y before reaching a state in the domain of X .

Figure 4.2 illustrates how X and Y combine to create \mathcal{N} and how \mathcal{N} defines d_C . Note that for a particular set of agents, some \mathcal{N} result in smaller d_C than do others (e.g., in Figure 4.2 “north” would give a smaller d_C than does “east”). The optimal \mathcal{N} is a global property of the entire group, depends on the details of \mathcal{B} , and is beyond the scope of this dissertation.

4.2.2 The Induced Distance Function d_C

In this section I define d_C in terms of the behavior \mathcal{N} , whose existence is required in (4.4). How to select a \mathcal{N} is discussed in Section 4.2.1.

Let d_C be the maximum distance agents would separate from one another while both following \mathcal{N} :

$$d_C(s_1, s_2) \triangleq \sup_{t \geq 0} \left\{ d(\mathcal{N}_{s_1}(t), \mathcal{N}_{s_2}(t)) \right\}. \quad (4.7)$$

Because $\mathcal{N}_s(0) = s$ (true of all behaviors; see Section 1.2), $d(s_1, s_2) = d(\mathcal{N}_{s_1}(0), \mathcal{N}_{s_2}(0)) \leq d_C(s_1, s_2)$; thus, the d_C defined in (4.7) satisfies (4.2).

Using the particular d_C defined in (4.7) to expand (4.4) yields

$$\sup_{t \geq 0} \left\{ d(\mathcal{N}_{s_1}(t), \mathcal{N}_{s_2}(t)) \right\} \leq r \Rightarrow C(s_1, s_2, \mathcal{N}). \quad (4.8)$$

I use (4.8) in place of (4.4) in the remainder of this chapter.

The following lemma will be useful in establishing properties of predicates.

Lemma 4.1. *For any d that satisfies the triangle inequality, d_C also satisfies the triangle inequality; that is,*

$$d(a, b) + d(b, c) \geq d(a, c) \implies d_C(a, b) + d_C(b, c) \geq d_C(a, c). \quad (4.9)$$

Proof. Consider a time¹ t' at which $d_C(a, c) = d(\mathcal{N}_a(t'), \mathcal{N}_c(t'))$. By (4.7), we know that $d_C(x, y) \geq d(\mathcal{N}_x(t'), \mathcal{N}_y(t'))$ for all x, y . Thus,

$$\begin{aligned} d_C(a, b) + d_C(b, c) &\geq d(\mathcal{N}_a(t'), \mathcal{N}_b(t')) + d(\mathcal{N}_b(t'), \mathcal{N}_c(t')) \\ &\geq d(\mathcal{N}_a(t'), \mathcal{N}_c(t')) = d_C(a, c). \end{aligned} \tag{4.10}$$

□

4.2.3 Cohesion Predicate

This section presents several predicates on behaviors with the property that any two connected agents, each following a behavior that satisfies one of the predicates, will remain connected until one or both selects another behavior.

In describing the predicates, a represents the initial state of the agent selecting behavior A , while n is the initial state of its neighboring agent.

Adversarial: An agent whose behavior satisfies the “adversarial” predicate cannot be separated from the other agent within Δt even if the other agent actively works to achieve separation.

$$\forall t \in [0, \Delta t], B \in \mathcal{B} \quad d_C(A_a(t), B_n(t)) \leq r. \tag{4.11}$$

If either agent’s behavior satisfies (4.11) then both agents remain connected until they have time to select a new action.

¹Technically I should use $\lim_{t \rightarrow t'}$ for asymptotically defined d_C ; I use the looser notation here to simplify the presentation.

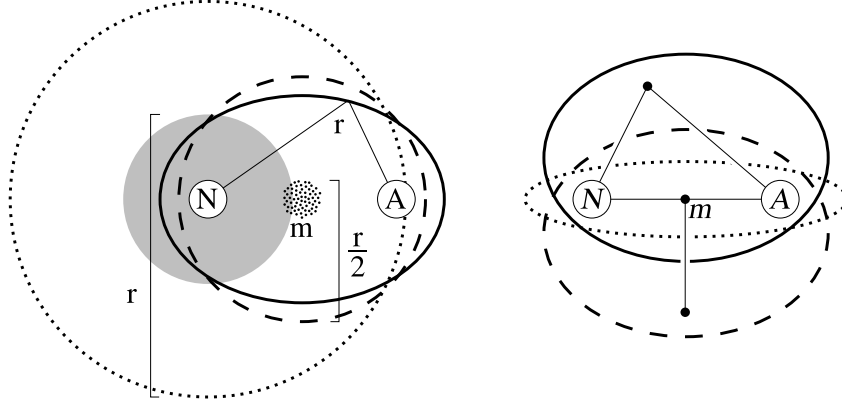


Figure 4.3: Illustration of cohesion predicates for agent A with \mathcal{N} being “remain in place”. The outlined regions satisfy (4.11) (dotted), (4.12) (solid), and (4.13) (dashed). On the left is a geometric version; the shaded region represents where neighbor N can reach in Δt and the cluster of dots being the set of midpoints between A and N . On the right is a graph version; graph-based agents often lack midpoints, but one is displayed for completeness.

Triangle: The “triangle” predicate is so named because it utilizes the triangle inequality.

$$\forall t \in [0, \Delta t] \quad d_C(A_a(t), \mathcal{N}_a(t)) + d_C(A_a(t), \mathcal{N}_n(t)) \leq r. \quad (4.12)$$

On the Euclidean plane, (4.12) forms an ellipse. If both agent’s behaviors satisfy (4.12) then both agents remain connected until they have time to select a new action.

Midpoint: The “midpoint” predicate is defined only if every pair of states has at least one midpoint. Let $m : \mathcal{S}^2 \rightarrow \mathcal{P}(\mathcal{S})$ be a function returning the set of midpoints of two states.

$$\forall t \in [0, \Delta t] \quad \forall m_t \in m(\mathcal{N}_a(t), \mathcal{N}_b(t)) \quad d_C(A_a(t), m_t) \leq \frac{r}{2}. \quad (4.13)$$

On the Euclidean plane, (4.13) forms a circle. If both agent’s behaviors satisfy (4.13) then both agents remain connected until they have time to select a new action.

These three predicates are illustrated in Figure 4.3. In noise-free Euclidean geometries the

midpoint is unique and (4.13) subsumes (4.12); in other settings the two may be independent (see Figure 4.3).

Theorem 4.1. *Given the properties of r and \mathcal{N} , any set of agents satisfying the following properties will remain d_C connected.*

- *Each agent selects a behavior at least every Δt time units.*
- *An agent selects a particular behavior only if at the time it is selected, for each neighbor of the agent, that behavior satisfies either (4.11) or (4.12).*

Furthermore, there is always some behavior that each agent may select.

The same holds if all of the agents use (4.13) instead of (4.12).

Proof. A group of connected agents remain connected as long as each pair of neighboring agents remain connected until they lose their neighborhood status by another agent coming between them. It is thus sufficient to demonstrate that pairs of agents remain d_C connected for at least Δt .

By definition of d_C -connectivity, (4.11) guarantees connection with any agent (including those satisfying (4.12) or (4.13)).

Two agents both satisfying (4.12) remain within r of one another by the triangle inequality with either $\mathcal{N}_a(t)$ or $\mathcal{N}_n(t)$ as the third state.

Two agents both satisfying (4.13) remain within r of one another by the triangle inequality with any m_t as the third state.

Such a selection is always possible because \mathcal{N} is universal and satisfies both (4.12) and (4.13).

□

Define C as either $C(a, n, A) = (4.11) \vee (4.12)$ or $C(a, n, A) = (4.11) \vee (4.13)$. Given an agent in state a with m neighbors with states $\{n_1, n_2, \dots, n_m\}$, then by [Theorem 4.1](#) the subset of behaviors

$$\left\{ A \in \mathcal{B} \mid \bigwedge_{i=1}^m C(a, n_i, A) \right\} \quad (4.14)$$

is nonempty (it contains at least \mathcal{N}) and maintains cohesion with all m neighbors.

A note about line-of-sight

This chapter discusses distance-based cohesion. Line-of-sight is not a well-behaved distance function because it does not satisfy the triangle inequality. However, if a line-of-sight-maintaining \mathcal{N} is available then a line of sight cohesion predicate can be defined.

Let $O(t)$ be a set of all obstacle points at time t . Let $\overline{\mathbf{x}, \mathbf{y}}$ be the line segment connecting points \mathbf{x} and \mathbf{y} ; $\ell(\overline{\mathbf{x}, \mathbf{y}}, \mathbf{z})$ be the point on the line segment that is closest to point \mathbf{z} ; and $\langle \vec{\mathbf{x}}, \vec{\mathbf{y}} \rangle$ be an inner- or dot-product.

The adversarial constraint for line-of-sight is straightforward:

$$\forall t \in [0, \Delta t] \quad \forall B \in \mathcal{B} \quad \overline{A_a(t), B_n(t)} \cap O = \emptyset. \quad (4.15)$$

The other suitable constraint defines a line-of-sight maintaining region based on \mathcal{N} in a manner similar to [\(4.13\)](#) and [\(4.12\)](#):

$$\forall t \in [0, \Delta t] \quad \forall \mathbf{o} \in O(t + t_2) \quad \left\langle A_a(t) - \mathbf{o}, \ell\left(\overline{\mathcal{N}_a(t), \mathcal{N}_n(t)}, \mathbf{o}\right) - \mathbf{o} \right\rangle \leq r. \quad (4.16)$$

These predicates are illustrated in [Figure 4.4](#). They satisfy the same general structure as [\(4.11\)](#),

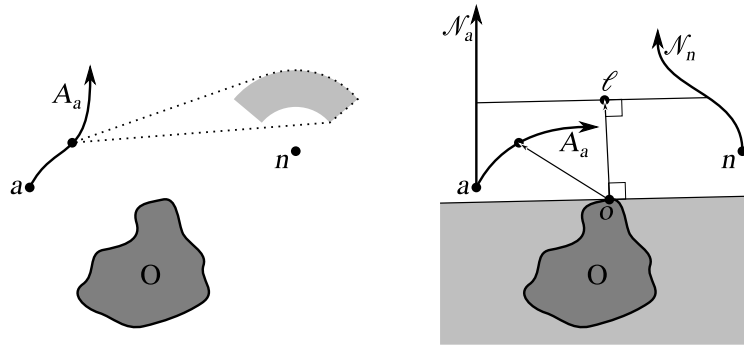


Figure 4.4: Illustrations of (4.15) (left) and (4.16) (right). For (4.15) the set of possible positions of the other agent (gray) define a region (dotted outline) that must not intersect any obstacle. For (4.16), the points $\mathcal{N}_a(t)$, $\mathcal{N}_b(t)$, and \mathbf{o} define a region where line of sight might not be maintained (gray) and thus should not contain $A_a(t)$.

(4.13), and (4.12) and may be used in their place or as additional predicates on top of a distance-based cohesion predicate.

An alternative approach to line-of-sight cohesion is to create a graph-based approximation of the environment such that locations that do not have line of sight to one another have a graph distance in excess of r . Ideally, such a graph would have some edges of length less than r ; techniques for designing such graphs are beyond the scope of this dissertation.

The details of line-of-sight cohesion are beyond the scope of this dissertation. I mention (4.15), (4.16), and line-of-sight graphs more to indicate directions for future investigation than as a completed element of this dissertation.

4.3 Computation Strategy

In general, the various components of the predicates used in [Theorem 4.1](#) need not be computationally tractable. This section contains a technique for forming tractable, arbitrarily-precise approximation functions under the following assumptions:

- \mathcal{B} can be represented as a bounded region of \mathbb{R}^n for some finite n .
- The d -distance between two agents following behavior \mathcal{N} reaches a maximum within a known time bound.
- (4.11), (4.12), and (4.13) are inequalities of piecewise-smooth functions of B and t .
- Errors in sensors and mobility are bounded by piecewise-smooth functions of B and t .

These conditions are satisfied by every geometric-, grid-, and graph-based model of agent behavior of which I am aware. They are sufficient to allow the use of polynomial approximation and the Bernstein branch-and-bound method to conservatively determine the truth of each predicate to within arbitrary precision.

The results in this section rely on extensive research on functional approximation and polynomial computation. Sederberg [59] and Dorato et al. [17] provide overviews for interested readers.

4.3.1 Polynomial Approximation

The computational approach I present takes as input a quantified Boolean expression of bounded-domain polynomial inequalities. It is known that any inequality of piecewise-smooth functions can be approximated as polynomial inequalities to any given precision; a brief review of relevant techniques follows.

Closed-form conversion

Piecewise expressions including addition, scaling, and rational exponents of real-valued variables can be converted to polynomial inequalities in closed form.

Rational exponents can sometimes be replaced by integer powers in simple inequalities, with the possible addition of new terms; for example $a \leq \sqrt{b}$ becomes $a^2 \leq b \vee a < 0$. For more involved expressions like $a^{3/7} + b \leq c^{1/3}$ approximations may be more appropriate than integer power expansion.

Piecewise expressions can always be re-written as disjunctions and expanded; for example,

$$\begin{cases} f_1 & g_1 \\ f_2 & \neg g_1 \end{cases} \leq \begin{cases} p_1 & q_1 \\ p_2 & \neg q_1 \end{cases} \quad (4.17)$$

is equivalent to

$$(\neg g_1 \vee \neg q_1 \vee f_1 \leq p_1) \wedge (g_1 \vee \neg q_1 \vee f_2 \leq p_1) \wedge (\neg g_1 \vee q_1 \vee f_1 \leq p_2) \wedge (g_1 \vee q_1 \vee f_2 \leq p_2). \quad (4.18)$$

Absolute values, sign functions, and other piecewise expressions can be handled similarly.

Approximations & Error

For functions that include non-algebraic operations, such as trigonometric functions, utilize polynomial approximation. Polynomials can be used to approximate any smooth function to any arbitrary level of precision over any finite interval; the error of such an approximation is no more than $\frac{c^n}{(n+1)!}$, where n is the order of the polynomial and c is proportional to the “roughness” of the approximated function. Creating such an approximation generally requires only the evaluation of the function at $n + 1$ points, with no symbolic analysis required. A more detailed discussion, including a formal definition of “roughness”, example algorithms, and convergence analysis can be found in most approximation texts (e.g., [59, pp. 111-114]).

Errors can enter computations through sensor noise, through inexact agent mobility, and through the error in functional approximations. Any bounded uncertainty in any constant, variable, or function can be handled in my computational model by introducing a new variable with the appropriate domain. Non-constant errors on sensor data might itself require approximation, but approximation error has polynomial (and often constant) bounds so the process of adding error terms does terminate.

In some cases it may be clear which value of a domain matters for conservative approximation and no new variable need be created; for example, if d_C is subject to error in $[-\epsilon t, \epsilon t]$ then (4.11) can be written as

$$\forall t \in [0, \Delta t], B \in \mathcal{B} \quad d_C(A_a(t), B_n(t)) + \epsilon t \leq r \quad (4.19)$$

instead of the more computationally-involved

$$\forall t \in [0, \Delta t], B \in \mathcal{B}, x \in [-\epsilon, \epsilon] \quad d_C(A_a(t), B_n(t)) + xt \leq r \quad (4.20)$$

4.3.2 Boolean Bernstein Branch-and-Bound

Inequalities of multivariate polynomials have been investigated at length by feedback control researchers and several computational approaches have been proposed; see Dorato et al. [17] for an overview. When the domain is known, Dorato et al. [17] recommend the Bernstein branch-and-bound (BBB) technique. This approach has the added advantage of being easy to modify to handle Boolean expressions of inequalities. The remainder of this section briefly reviews the BBB method and my extension of it to handle Boolean expressions.

Listing 4.1 Convert power- to Bernstein-basis

1. Given power-basis polynomial
 2. For each variable x_i with order n_i and target domain $[a_i, b_i]$
 3. Let $x_i = (b_i - a_i)y_i + a_i$
 4. Re-write P as $\sum_{j=0}^{n_i} c_j y_i^j$
 5. Let $d_j = \binom{n_i}{j} c_j$
 6. For k from 1 up to n_j
 7. For l from n_j down to k
 8. Add d_{l-1} to d_l
-

Recall that any order- n polynomial can be written in Bernstein basis as

$$P(x) = \sum_{i=0}^n c_i \binom{n}{i} \frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n} \quad (4.21)$$

where the c_0, c_1, \dots, c_n are the Bernstein coefficients over the interval $[a, b]$. For multivariate polynomials with variables x_1, x_2, \dots, x_m , where the order of the polynomial in x_j is n_j , the Bernstein basis is similarly defined:

$$\sum_{\substack{i_k \in \{0, 1, \dots, n_k\} \\ k \in \{0, 1, \dots, m\}}} c_{i_1, i_2, \dots, i_m} \prod_{j=1}^m \binom{n_j}{i_j} \frac{(b_j - x_j)^{n_j - i_j} (x_j - a_j)^{i_j}}{(b_j - a_j)^{n_j}} \quad (4.22)$$

where each c_{i_1, i_2, \dots, i_m} is a Bernstein coefficient over the m -dimensional interval $x_j \in [a_j, b_j]$.

Bernstein bases have several useful properties computationally:

- P interpolates the “corner” coefficient c_{i_1, i_2, \dots, i_m} where each i_k is either 0 or n_k .
- $\min c \leq \min_{x_j \in [a_j, b_j]} P$ and $\max_{x_j \in [a_j, b_j]} P \leq \max c$.
- The de Casteljau algorithm (see [59, §15.2] and Listing 4.2) can split the domain of any variable at any point with linear accuracy, giving two new Bernstein polynomials over the

two new domains.

These three properties are sufficient to define the Boolean BBB method.

The Boolean Bernstein branch-and-bound algorithm requires each polynomial inequality to be in the form $P(x_1, x_2, \dots, x_m) \leq 0$ with P in the Bernstein basis. Because addition, subtraction, and multiplication of Bernstein basis polynomials are computationally straight-forward and numerically more stable than converting from power to Bernstein basis, I suggest using Bernstein polynomials throughout. Conversion from power- to Bernstein-basis is also possible (though less numerically stable) using the algorithm in [Listing 4.1](#). This algorithm is based on inverting a hierarchy of hodographs, as is discussed in more detail in Sederberg [59, §3.3].

Given a set of Boolean expressions of quantified Bernstein-basis polynomial inequalities, the Boolean BBB proceeds as follows:

1. Evaluate each quantified polynomial inequality as true or false if possible.
 - Replace universally quantified inequalities with true if they are true for all coefficients.
 - Replace universally quantified inequalities with false if they are false for any corner coefficient.
 - Replace existentially quantified inequalities with true if they are true for any corner coefficient.
 - Replace existentially quantified inequalities with false if they are false for all coefficients.
2. Evaluate Boolean operators with at least one Boolean literal as an operand (e.g., “ $X \implies$ false” becomes “ $\neg X$ ”).

Listing 4.2 Multivariate de Casteljau algorithm

1. Given Bernstein-basis polynomial P with coefficients p_{i_1, i_2, \dots, i_m} , a variable x_j , and a $t \in (0, 1)$
 2. Create two copies of P , A and B .
 3. For each k_1 from 1 up to n_j
 4. For each k_2 from n_j down to k_1
 5. For each multi-index I with $i_j = k_2$
 6. Let J be I with $i_j = k_2 - 1$
 7. Replace a_I with $(1 - t)a_I + ta_J$
 8. For each multi-index I with $i_j = n_j - k_1$
 9. Replace b_I with a_I
 10. A and B are P with the domain of x_j split at t .
-

3. If any polynomials remain, use the de Casteljau algorithm to split the range of one variable in half and recur.

The last step above makes use of the following two identities

$$\begin{aligned}
 (\forall x \in [a, b] \ Z) &\equiv (\forall x \in [a, c] \ Z) \wedge (\forall x \in [c, b] \ Z) \\
 (\exists x \in [a, b] \ Z) &\equiv (\exists x \in [a, c] \ Z) \vee (\exists x \in [c, b] \ Z)
 \end{aligned}
 \tag{4.23}$$

to double the number of terms and force the Bernstein coefficients to converge toward the polynomials they define.

The only case where the Boolean BBB method splits an interval is when all the corner coefficients have the same sign but some other coefficient has a different sign. Listing 4.2 describes the de Casteljau algorithm for splitting the interval of one variable of a Bernstein-basis polynomial. In general, the best x_j is the variable along which the most coefficient chains have a sign change and a good t to pick is where the maximal sign change in the coefficients occurs (e.g., for coefficients $[-1, 3, 4, -1]$ a good choice for t is $\frac{2}{3}$ because that corresponds to the highest coefficient (4)).

Listing 4.3 The Boolean Bernstein Branch-and-Bound algorithm

1. Given a Boolean expression of quantified Bernstein-basis polynomial inequalities
2. Let Z be the multi-index $0, 0, \dots, 0$
3. Repeat up to N times:
 4. Replace quantified inequalities that are true or false with “true” or “false”, respectively.
 5. Evaluate as much of the Boolean expression as possible.
 6. For each remaining polynomial P
 7. For each $j \in \{1, 2, \dots, m\}$
 8. Initialize c_j to 0
 9. For each multi-index I with $i_j = 0$
 10. Let J be I with $i_j = n_j$
 11. If $p_I p_Z > 0$ and $p_J p_Z > 0$
 12. For each $k \in \{1, \dots, n_j - 1\}$
 13. Let J be I with $i_j = k$
 14. If $p_I p_Z < 0$
 15. Increment c_j by n_j
 16. *break*
 17. Pick a j for which c_j is maximal
 18. Pick a k maximizing the p_I with $i_j = k$ that differ in sign from p_Z .
 19. Get A and B from the Multivariate de Casteljau algorithm with P , x_j , and $t = \frac{k}{n_j - 1}$
 20. Use (4.23) to replace P with A and B
 21. If any polynomials remain, return “false”; otherwise return the remaining Boolean value

Although the BBB method is numerically stable and requires few applications of de Casteljau’s algorithm for well-conditioned polynomials, there are polynomials for which it converges poorly. All of these poorly-convergent polynomials are “close to” unsatisfied, in that some small perturbation will yield an unsatisfied expression. If the BBB method does not converge quickly enough, the expression can be considered unsatisfied with minimal error.

A complete version of the Boolean BBB method is presented in [Listing 4.3](#).

4.4 Examples

Two examples illustrate how the technique in this chapter can be used to create cohesion-maintenance algorithms. The first is selected for its simplicity and can be seen as a generalization of the technique of Cornejo and Lynch [8]. The second is selected as a simple nonholonomic example applicable to many car-like vehicles and robots. The two examples are intended to be informative, not exhaustive; the technique can also work on, e.g., directed and undirected graphs, uniform and nonuniform manifolds of any finite dimension, and other domains with finite state and well-defined distance functions.

4.4.1 Holonomic agents

Consider an agent that can move in any direction in an n -dimensional Euclidean environment. Let S be agent position, $\vec{x} \in \mathbb{R}^n$. Let \mathcal{B} be bounded-speed velocities, $\vec{v} \in \mathbb{R}^n$ where $\|\vec{v}\| \leq s'$.

“Do not move” is a sensible null behavior, it being the centroid of the behavior space: \mathcal{N} is $\vec{v} = \vec{0}$. This \mathcal{N} provides a straightforward Euclidean distance: $d_C(s_1, s_2) = d(\vec{x}_1, \vec{x}_2) = \|\vec{x}_1 - \vec{x}_2\|_2$.

Because the distance function is geometrically defined, (4.13) subsumes (4.12). It is thus sufficient to convert (4.11) and (4.13) into polynomial inequalities. In this example, the predicates can be converted into polynomials without approximation. (4.11) becomes

$$\forall t \in [0, \Delta t], \|\vec{v}_n\|_2 \leq s' \quad \left\| \vec{x}_a - \vec{x}_n + (\vec{v}_a - \vec{v}_n)t \right\|_2^2 - r^2 \leq 0 \quad (4.24)$$

and (4.13) becomes

$$\forall t \in [0, \Delta t] \quad \left\| \vec{x}_a + \vec{v}_a t - \frac{\vec{x}_a + \vec{x}_n}{2} \right\|_2^2 - \frac{r^2}{4} \leq 0. \quad (4.25)$$

These inequalities can be used directly with the BBB method to evaluate if a particular behavior is guaranteed to ensure cohesion.

Holonomic agents are unusual in that the BBB method is not required. The inequalities are quadratic and admit direct solution; they also define spheres which can be handled via geometric techniques. In more involved cases closed-form solutions are not available and the BBB method is more important.

4.4.2 Car-like Agents

Car-like agents are traditionally modeled as non-holonomic, being unable to move sideways. A car-like agent's state is speed, heading, and position (s, θ, \vec{x}) ; its behavior is a forward acceleration a bounded by $|a| \leq a'$ and a signed curvature c bounded by $|c| \leq c'$.

The basic distance function d is Euclidean distance $\|\vec{x}_1 - \vec{x}_2\|_2$.

To find d_C , I first construct \mathcal{N} from X and Y as outlined in [Section 4.2.1](#). Let X be a fixed heading θ_\emptyset and speed $s_\emptyset > 0$; let Y independently turn to that heading and accelerate to that speed. This Y is not optimal since orientation can change more rapidly at higher speeds, but it simplifies the presentation and, since $s_\emptyset > 0$, it does move any agent into a state in the domain of X . I also assume the agents can distinguish between orientations θ and $\theta + 2\pi$ to reduce the number of terms presented below.

I again use [\(4.13\)](#) instead of [\(4.12\)](#) because the distance function is geometric. What remains is deriving piecewise polynomials for $B_s(t)$, $\mathcal{N}_s(t)$, and d_C .

For a general $B = (a(t), c(t))$ and $s = (s_0, \theta_0, \vec{x}_0)$, the following definite integrals define $B_s(t)$:

$$s(t) = s_0 + \int_0^t a(\tau) d\tau \quad (4.26)$$

$$\theta(t) = \theta_0 + \int_0^t c(\tau) s(\tau) d\tau \quad (4.27)$$

$$\vec{x}(t) = \vec{x}_0 + \int_0^t s(\tau) \vec{f}(\theta(\tau)) d\tau, \quad (4.28)$$

where the “forward” vector $\vec{f}(\theta) \triangleq (\cos(\theta), \sin(\theta))$.

Observe that $d_C(A_a(t), B_b(t))$ over $[0, t_0]$ is the same as $d(t_0[A_{\mathcal{N}}], t_0[B_{\mathcal{N}}])$ over $[0, t_0 + t^*]$, where $\mathcal{N} = t \begin{bmatrix} Y \\ X \end{bmatrix}$. It is thus sufficient to handle the general expression $t_0 \begin{bmatrix} a, c \\ \mathcal{N} \end{bmatrix}_{s_0, \theta_0, \vec{x}_0}$, where t_0 is 0 for \mathcal{N} and Δt for A and B in (4.11) and (4.13).

$$a_Y = \text{sign}(s_\emptyset - s_0 - at_0) a' \quad (4.29)$$

$$t_a = \frac{1}{a'} |s_\emptyset - s_0 - at_0| + t_0 \quad (4.30)$$

$$a(t) = \begin{cases} a & t \leq t_0 \\ a_Y & t_0 < t \leq t_a \\ 0 & t_a < t \end{cases} \quad (4.31)$$

$$s(t) = \begin{cases} s_0 + at & t \leq t_0 \\ s_0 + at_0 - a_Y t_0 + a_Y t & t_0 < t \leq t_a \\ s(t_0) + (t - t_0) a_Y & t_0 < t \leq t_a \\ s_\emptyset & t_a < t \end{cases} \quad (4.32)$$

The above speeds formalize the notion of accelerating or decelerating to s_\emptyset as quickly as possible after t_0 . The following heading information formalizes turning toward θ_\emptyset as quickly as possible

after t_0 , as illustrated in **Figure 4.2** and is complicated by the fact that speed impacts the rate of heading change. Thus even the heading before \mathcal{N} kicks in (i.e., $\theta(t_0)$) has several terms.

$$\theta(t_0) = \theta_0 + c s_0 t_0 + \frac{1}{2} c a t_0^2 \quad (4.33)$$

$$c_Y = \text{sign}(\theta_\emptyset - \theta(t_0)) c' \quad (4.34)$$

$$\theta_{a_Y}(t) = \theta(t_0) + c_Y s(t_0)(t - t_0) + \frac{1}{2} a_Y c_Y (t - t_0)^2 \quad (4.35)$$

$$t_1 = \frac{-c_Y s(t_0) + \sqrt{c_Y^2 s(t_0)^2 - 2 a_Y c_Y (\theta(t_0) - \theta_\emptyset)}}{c_Y a_Y} + t_0 \quad (4.36)$$

$$t_2 = t_1 + \frac{\theta_\emptyset - \theta_{a_Y}(t_a)}{s_\emptyset c_Y} \quad (4.37)$$

$$t_c = \max(t_1, t_2) \quad (4.38)$$

$$c(t) = \begin{cases} c & t < t_0 \\ c_Y & t_0 \leq t < t_c \\ 0 & t_c \leq t \end{cases} \quad (4.39)$$

In the above, the orientation of the agent while it is both accelerating and turning is θ_{a_Y} . It stops turning at time t_1 if it stops turning before it stops accelerating; otherwise it stops turning at time t_2 . There are two cases for θ depending on the relative order of t_c and t_a , but both may be expressed in the same piecewise equation because the third piece only applies when $t_a < t_c$.

$$\theta(t) = \begin{cases} \theta_0 + c s_0 t + \frac{1}{2} c a t^2 & t \leq t_0 \\ \theta(t_0) + c_H s_0 t + c_H a t_0 t - c_H a_h t_0 t + \frac{1}{2} c_H a_h t^2 & t_0 < t \leq \min(t_a, t_c) \\ \theta_\emptyset + s_\emptyset c_H (t - t_c) & t_a < t \leq t_c \\ \theta_\emptyset & \text{otherwise} \end{cases} \quad (4.40)$$

These expressions for a , c , and θ can be plugged into (4.28) to obtain the following expression for \vec{x} :

$$\vec{x}(t) = \vec{x}_0 + \int_0^t \begin{cases} (s_0 + a\tau) \vec{f} \left(\theta_0 + cs_0\tau + \frac{1}{2}ca\tau^2 \right) d\tau & t \leq t_0 \\ (s(t_0) + (\tau - t_0)a_h) \vec{f}(\theta_{a_h}(\tau)) d\tau & t_0 < t \leq \min(t_a, t_c) \\ (s_\emptyset) \vec{f}(\theta_\emptyset + s_\emptyset c_H(\tau - t_c)) d\tau & t_a < t \leq t_c \\ (s(t_0) + (\tau - t_0)a_h) \vec{f}(\theta_\emptyset) d\tau & t_c < t \leq t_a \\ (s_\emptyset) \vec{f}(\theta_\emptyset) d\tau & \max(t_a, t_c) < t \end{cases} \quad (4.41)$$

The distance function d_C is thus $\max_{t \in [0, \Delta t]} \|\vec{x}_1(t) - \vec{x}_2(t)\|_2$.

Because the integration in the expression for \vec{x} cannot be solved in closed form, a polynomial approximation of \vec{f} can be used to obtain a piecewise-polynomial integral instead. The error terms resulting from this approximation are also polynomial; for example, the error in the first piece is $(s_0\varepsilon t + \frac{1}{2}\varepsilon at^2)$, where ε is the (constant) error of the polynomial approximation. These errors can be added into the inequalities (4.11) and (4.13) as discussed in Section 4.3.1 to obtain conservative approximations.

All that remains is to split the piecewise functions, square roots (in t_1 and d), and sign dependence (in a_H , t_a , and c_H) to obtain a Boolean expression of polynomial inequalities suitable for the BBB method.

4.5 Conclusion

This chapter has presented a technique for generating computable predicates ensuring cohesion for wide varieties of mobile agents. The technique relies on

- a distance function satisfying the triangle inequality,

- the ability to sense other agents within a fixed distance,
- known error bounds on sensor and actuator behavior,
- known upper bound on the time between consecutive behavior selections, and
- the ability to evaluate polynomials and Boolean expressions.

The technique thus works for both geometric and graph-based environments with or without uncertainty and does not require any form of explicit communication.

The most obvious extension of this work is to develop additional constraints for distance functions where the triangle inequality does not hold. A classic example of such an environment is line-of-sight in environments with static obstacles. Some preliminary work in this direction is provided in [Section 4.2.3](#); however, additional investigation into line-of-sight versions of \mathcal{N} , line-of-sight given uncertainty, and related issues are needed.

Another extension would be to handle other maneuverability constraints, such as collision avoidance. Such an extension would likely resemble the reciprocal velocity obstacle technique [\[67\]](#) extended to arbitrary maneuvers and uncertainty in a manner similar to the generalized reactive navigation method [\[66\]](#). Conceptually all that is required is to insert these constraints into the Boolean expressions of polynomial inequalities; however, the assurance that some satisfying behavior exists will require additional investigation.

Cohesion constraints are an important part of local and global cohesion algorithms. These algorithms are discussed in greater detail in [Chapter 5](#).

Chapter 5

Local and Global Cohesion

Cohesion requires that a group of moving agents remain a single group as they move.

Local cohesion ensures cohesion by requiring that pairs of agents maintain connection until the interposition of a third agent connected to each. Global cohesion ensures cohesion by requiring that pairs of agents maintain connection if failing to do so would split the group in two. In this chapter I develop techniques for ensuring local and global cohesion using an underlying cohesion predicate, local communication, and bounded memory. I also explore notions of global cohesion in the face of malicious agents.

Cohesion is the property of a swarm of agents remaining a single swarm as it moves. Cohesion depends on a notion of connectivity; typically connection means that agents are within sensor distance of one another, sometimes also suggesting line-of-sight. Each agent of a cohesive group must not become disconnected from the other agents in that group.

Local cohesion algorithms ensure cohesion of a swarm in a decentralized manner: each agent's cohesion-maintaining behaviors are informed only by its own local observations. Global cohesion involves coordination across an entire group of agents to allow for non-local cohesion-

maintaining behaviors such as the breaking of rings into lines. Both may be constructed using cohesion constraints that keep a pair of connected agents connected. A discussion of one possible family of cohesion constraints is given in [Chapter 4](#).

These terms may be formally defined as follows.

Definition 5.1 (Cohesion). *Given a notion of two agents being **connected**, define the following:*

Let G be a graph with a node for each agent and an edge between each pair of connected agents.

*Two agents belong to the same **swarm** if they are in the same connected subgraph of G .*

*A **cohesion constraint** between two connected agents, if observed by both agents, ensures that the agents remain connected as they move.*

*A swarm exhibits **local cohesion** if no agent A will break connection with an agent B unless A can see a connected path to B that is not breaking.*

*A swarm exhibits **global cohesion** if no agent A will break connection with an agent B unless there exists a connected path to B that is not breaking.*

A variety of cohesion algorithms have been published and are reviewed in [Section 1.3.3](#). My approach to local cohesion is similar to many previously-published approaches and is included herein primarily for completeness. My approach to global cohesion does not require agents to learn the full connectivity graph of the swarm, requiring less memory than previous approaches. My technique for guaranteeing cohesion with malicious agents present is, to my knowledge, the first to solve this problem.

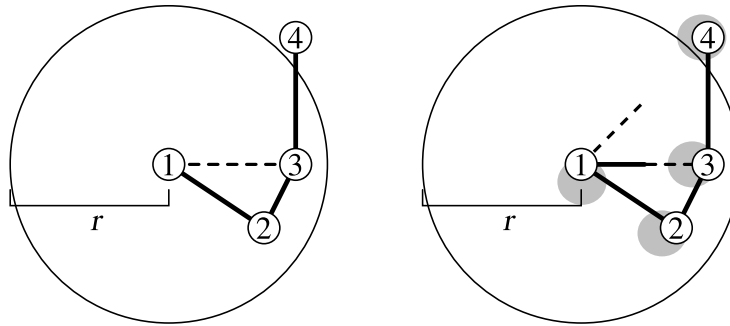


Figure 5.1: Heavy lines show neighbor relationships; dashed lines are connections that are not neighbors; the large circle is the area connected to agent 1. On the left, agents 1 and 3 are not neighbors because agent 2 is between them. On the right, noise causes agents to appear to be anywhere within the shaded circles. Agent 1 thinks it might be connected to agent 4 and neighbors with agent 3, but agents 4 and 3 disagree.

5.1 Local Cohesion Without Communication

Local cohesion requires that an agent stay connected with each of the agents to which it is currently connected unless it can observe another connected path to that agent and knows that that other path is not breaking. When agents do not explicitly communicate their plans (or do not trust others' communications), knowing that another path is not breaking is challenging. If a distance function is available, however, local cohesion may be readily achieved through neighbor computations.

5.1.1 Distance-based Local Cohesion

Given three states s_1 , s_2 , and b , state b is said to be **between** s_1 and s_2 if $d(s_1, b) < d(s_1, s_2) \wedge d(s_1, s_2) < d(b, s_2)$. Two agents are **neighbors** if they are connected and no third agent is between them.

For agents experiencing sensor uncertainties, let neighbor status be conservatively interpreted: agent A treats agent N as a neighbor if A believes N might be connected to it and A

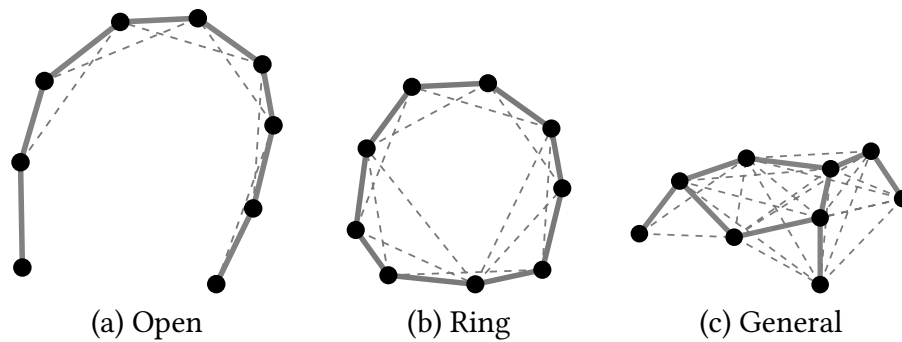


Figure 5.2: Several swarms; neighbors are displayed in bold solid lines, connections in dashed lines. While open configurations (a) can change into rings (b) freely, and both may change to and from general swarms (c), rings cannot readily change into open configurations.

cannot identify a third agent that is definitely between A and N (see [Figure 5.1](#)).

Local cohesion is maintained when each agent uses a cohesion constraint to remain connected to each neighbor. Since the graph of (mutually-suspected) neighbors is always a connected subgraph of the graph of the graph of connected agents, this form of local cohesion guarantees swarm remains cohesive.

Local cohesion can be trivially extended to handle multiple intermediate agents instead of just one in the definition of “between.” The single-intermediate-agent version above is used for clarity of presentation.

5.1.2 Problems with Local Cohesion

Local cohesion maintains some connections that could be safely broken without separating the swarm. While locally cohesive swarms can reconfigure themselves, they cannot break redundant connections in non-local loops (see [Figure 5.2](#)). Since swarms can create non-local loops, the inability to break them leads to irreversible behavior that can cause locally-cohesive swarms to be trapped, as illustrated in [Figure 5.3](#).

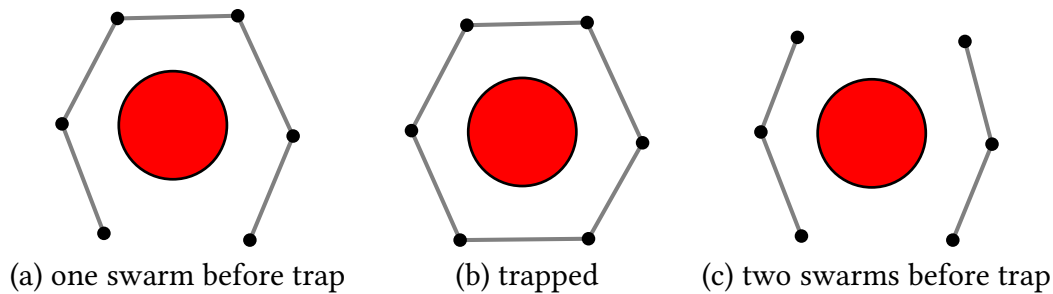


Figure 5.3: A swarm becoming trapped around an obstacle. Local cohesion prevents the the swarm from leaving the obstacle.

The simplest technique for preventing trapping behaviors is to forbid irreversible events within a swarm: only already-connected agents may move into a neighboring configuration. However, this technique is overly restrictive, preventing swarms from merging and broken swarms from reforming.

Communication within the swarm can be used to detect the difference between swarm merges and non-local joins, but such communication does not prevent trapping when two swarms meet in multiple locations (see [Figure 5.3\(c\)](#)). Multiple potential join points could be arbitrated using distributed mutual exclusion (see, e.g., [33]), but such an approach is more involved than full global cohesion.

Even if inter-swarm merges are ignored or handled appropriately, preventing intra-swarm merges reduces swarm mobility: for example, a merge-free swarms must snake through a forest single-file instead of moving through it like a crowd. When no small obstacles exist, local cohesion works well, but for swarms in more cluttered environments global cohesion, which does not have difficulty with non-local breaks, is a better solution.

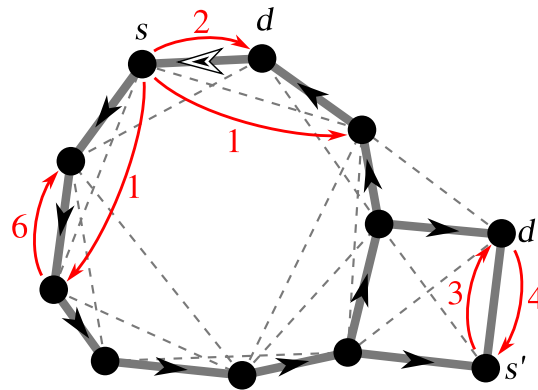


Figure 5.4: Illustration of [Algorithm 5.1](#). Black arrow heads indicate messages $(*, s, d, t)$ sent during protocol. Red arrows are a sample of messages heard but ignored, with the reason noted next to the arrow. We assume that $s' > s$ and s' sent (s', s', d', t') before receiving $(*, s, d, t)$. The outlined arrowhead represents the break approval message.

5.2 Global Cohesion with Trust

Global cohesion allows agents to break connection as long as such a break will not split the swarm in two. Detecting which neighbors may be safely broken requires communication between agents to ensure that both (1) there is another path between the agents in question and (2) that the other path is not breaking.

Global cohesion can be achieved by having each agent discover the entire connectivity of the swarm, but such an approach requires memory at least proportional to the size of the swarm. If each agent has a unique communicable identity and if all identities are subject to a total order, a constant-memory approach can be used to determine if a proposed break is permissible.

The basic idea of this approach is to send break requests through the neighbor graph. Each request is sorted based on the identity of its participating agents; requests are only forward across edges in the neighborhood graph if the corresponding edge is either not participating in a break request or has a lower identity than the current request. This idea is achieved by the protocol presented in [Algorithm 5.1](#) and illustrated in [Figure 5.4](#).

Algorithm 5.1 Global Cohesion

Each agent may move freely provided it does not break connection with any neighbor.

If an agent desires to break with a neighbor, it broadcasts a message seeking an alternate connection to that neighbor within the swarm. This takes the form (from, source, destination, timestamp). When A wishes to separate from neighbor B it broadcasts $(i_A, i_A, i_B, now + \Delta t)$. If A receives (i_B, i_A, i_B, t) or broadcasts (i_A, i_B, i_A, t) then A and B have permission to break connection.

When agent A receives message (i_f, i_s, i_d, t) it sends message (i_A, i_s, i_d, t) unless any of the following are true:

1. A and f are not neighbors or have already agreed to break;
2. $i_f = i_s$ and $i_d = i_A$ (edges can not approve themselves);
3. A has received $(i_f, i_f, i_A, t_1 \geq now)$ and $i_f > i_s$ (ignore messages from across an edge that a higher-priority neighbor is trying to break);
4. A has sent $(i_A, i_A, i_f, t_1 \geq now)$ and $i_A > i_s$ (ignore lower-priority messages from across an edge you are trying to break); or
5. $t < now$ (messages expire)

To reduce duplicate communication, we can also add

6. A already sent (i_A, i_s, i_d, t) (messages do not cycle);

but that condition is not required for correctness.

When an agent sends a message with timestamp t , it is not permitted to initiate a new message until after t expires.

Items 1, 2, 4, and 5 of [Algorithm 5.1](#) require only finite memory. Item 6 in [Algorithm 5.1](#) involves potentially n^2 memory, where n is the number of agents in the swarm, but is not required for correct functionality. Item 3 might involve memory proportional to the number of neighbors, which is small in practice but can be unbounded in general. However, recalling only a finite set of break requests and ignoring new messages once that set is filled will still maintain cohesion.

[Algorithm 5.1](#) makes explicit reference to a timestamp, implying a globally-synchronized clock. This assumption is generally supported by the existence of good clock synchronization algorithms. However, asynchronous clocks may be used instead if communication across the entire network of agents has a known upper time bound T . In that case, timestamps may be omitted and agent should not initiate any messages until T after sending a message.

Theorem 5.1. *For any fully connected connectivity graph, [Algorithm 5.1](#) allows only breaks that leave the connectivity graph fully connected. If messages do not time out, [Algorithm 5.1](#) forbids only one break for each set of requests that would split the swarm in two.*

Proof. Let $N = (V, E)$ be the graph with edges between neighboring agents. It is never the case that, for $e_i, e_j \in E$, both break request i crosses edge e_j and break request j crosses edge e_i unless either i or j has expired.

Consider break request e_i . Let $N_i = (V, \{e_j \in E : j < i\})$ be the graph containing only edges that will forward break request e_i . If e_i is a bridge on N_i , the break request will be denied and e_i will be a bridge after all break requests are resolved. Otherwise, the break request will be approved and not break the graph. □

Undiscussed by [Algorithm 5.1](#) is how to determine when to send a message and with what initial timestamp. Timestamp doubling can be used to ensure that eventually messages last long

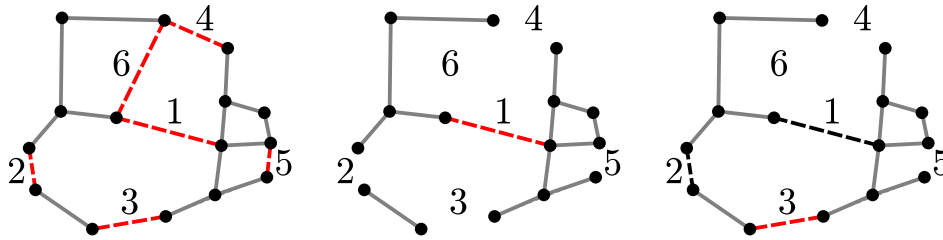


Figure 5.5: **Theorem 5.1.** On the left is N with break requests in red dashed lines. In the center, e_1 is a bridge on N_1 so it cannot break. On the right e_3 is not a bridge in N_3 so it can break. The forbidden breaks are precisely those required to connect the graph.

enough to reach their destinations; alternatively, expired messages can be forwarded so that edges discover that their message expired rather than being blocked. The frequency with which messages are initiated is limited only by the communication medium utilized; heuristics for selecting message initiations and durations are beyond the scope of this dissertation.

5.2.1 Limited Malice

When some agents cannot observe other agents directly, they are dependent on intermediaries to convey information about the swarm status. Thus, global cohesion depends on some significant portion of each swarm being trustworthy.

Communication within a swarm is interrupted if a vertex cut of the swarm is malicious. Swarms with malicious vertex cuts can be constructed for k -connected graphs with arbitrary k , and may be created with only k malicious agents and with an arbitrary minimum path length between malicious agents; one such construction is illustrated in [Figure 5.6](#). I thus require the distribution of malicious agents to be such that in any vertex cut of the graph there must be at least $m + 1$ trustworthy agents and each trustworthy agent is connected to no more than m malicious agents. These requirements mean that if an agent forwards messages it hears from at least

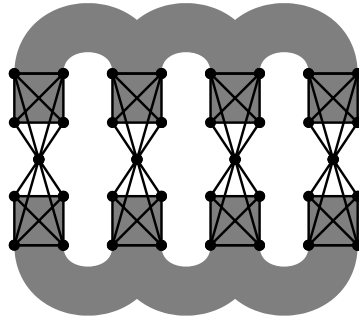


Figure 5.6: Example construction of a four-vertex cut using four widely separated agents in a four-connected graph. Wide gray curves represent arbitrarily long four-connected paths of agents.

$m + 1$ distinct agents, then messages set by trustworthy agents will reach all other trustworthy agents in the graph.

Let **k -neighbors** refer to connected agents that have fewer than k agents between them.

Theorem 5.2 (Local Cohesion Despite Malice). *If there are no more than m malicious agents connected to each trustworthy agent then applying a cohesion constraint between each pair of $(2m + 1)$ -neighbors ensures there are $m + 1$ trustworthy agents per vertex cut in the connectivity graph.*

Proof. Consider any vertex cut in the graph. Consider two agents, one on each side of the cut, who are connected to the same agent in the cut. All of the agents between those two agents are in the cut. The agents would never have allowed fewer than $2m + 1$ agents between them. Since a malicious agent cannot impact the neighbor relationship of agents with which it is not connected, one malicious agent is required per missing intermediary. Thus, at most m of those $2m + 1$ agents could be missing or malicious, leaving at least $m + 1$ that must be trustworthy. \square

The following modification of [Algorithm 5.1](#) also maintains $m + 1$ trustworthy agents per cut provided no more than m malicious agents are connected to any given agent. [Algorithm 5.2](#) requires agents remember all non-expired messages they have received, implying that memory will scale with communication bandwidth and with maximum timestamp duration. [Algorithm 5.2](#)

Algorithm 5.2 Non-local Trustworthy Cuts

Each agent may move freely provided it does not break connection with any $(2m + 1)$ -neighbor. Agents seeking to break with a neighbor broadcast a message as in [Algorithm 5.1](#).

When agent A receives message (i_f, i_s, i_d, t) it ignores messages per items 1–6 in [Algorithm 5.1](#); non-ignored messages are added to an internal set of non-expired messages. If the set now contains either $(2m + 1)$ entries (i_x, i_s, i_d, t) with distinct i_x or a single entry (i_s, i_s, i_d, t) then A broadcasts (i_A, i_s, i_d, t) .

Agents A and B may break when B sends message (i_B, i_A, i_B, t) provided A had earlier sent message (i_A, i_A, i_B, t) .

When an agent sends a message with timestamp t , it is not permitted to initiate a new message until after t expires.

also assumes agents are at most weakly byzantine—that is, agents are not able to deceive others as to their own identity when sending messages.

Theorem 5.3. *Breaks permitted by [Algorithm 5.2](#) will never introduce cuts with fewer than $m + 1$ trustworthy agents.*

The following proof is illustrated in [Figure 5.7](#).

Proof. Consider each cut \mathcal{C} introduced when [Algorithm 5.2](#) approved a break between agents A and B ; assume the break request was initiated by A .

Suppose A and/or B is malicious. Since both $\mathcal{C} \cup \{A\}$ and $\mathcal{C} \cup \{B\}$ were cuts before the break, each had $m + 1$ trustworthy agents; since either A or B is malicious, \mathcal{C} must contain $m + 1$ trustworthy agents.

Suppose both A and B are trustworthy. An agent will only have approved the break if it received the request from $2m + 1$ connected agents. Since each agent is connected to at most m malicious agents, each agent that approved the break must have received the request from at least $m + 1$ trustworthy agents. Since neither A nor B would have transferred the message across \mathcal{C} , for B to have approved the message some agent X on B 's side of \mathcal{C} must have received the

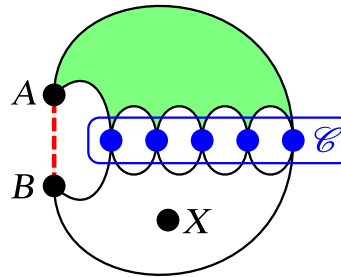


Figure 5.7: Illustration of proof to [Theorem 5.3](#). A break request is sent by A and propagates through A 's side of vertex cut \mathcal{C} . If the request is to make it to B , it must reach some arbitrary first node X on B 's side of \mathcal{C} , requiring $m + 1$ trustworthy requests from agents in \mathcal{C} .

message from at least $m + 1$ trustworthy agents in \mathcal{C} . Thus, \mathcal{C} must contain $m + 1$ trustworthy agents and breaking the connection between A and B will leave at least $m + 1$ trustworthy agents in each cut. □

5.3 Conclusion

Both local and global cohesion can be built relatively simply off of any cohesion constraint. I introduced in particular a finite-memory approach to global cohesion and methods of providing both local and global cohesion for swarms containing a limited group of malicious agents.

I used malice in this chapter as the worst-case behavior of a malfunctioning agent. If some agents are actively working against swarm objectives, cohesion itself might not be the right objective. A single malicious agent can control the behavior of the entire cohesive swarm if the rest of the swarm is unwilling to abandon it.

Additional research could explore algorithms that allow malicious agents to be ignored. As a simple example, if each agent in a flock tracks the median instead of mean behavior of its neighbors then a small number of malicious agents are ignored. Developing similar techniques

for algorithms that (unlike flocking) are composable with other objectives is an area for future investigation.

Additionally, much of the chapter can be streamlined if malicious agents can be identified and ignored explicitly. Techniques for recognizing and ignoring malicious intent are discussed in [Chapter 6](#).

Chapter 6

Coalescence

Coordinated groups of agents may be characterized as either swarms (continuously connected) or hives (intermittently connected). Agents coalesce when they create or join such a coordinated group. Swarm coalescence for correctly-behaving agents is just a combination of rendezvous and cohesion. For agents that are not fully trustworthy and/or for agents forming a hive, the process of coalescing can be more involved. In this chapter I develop a set of coalescence algorithms creating both swarms and hives with varying levels of trust in other agents' correct behavior.

Agents coalesce when they change from a state of acting independently to a state of acting as a group. Coalescence algorithms are of two kinds: those that form a cohesive swarm, meaning that agents maintain continuous interconnectivity; and those that form a hive, meaning that agents only periodically connect with other members of the hive.

Swarm and hive coalescence both involve at least two steps. First, agents must locate one another, either through rendezvous (see [Chapter 3](#)) or through simple search, depending on the nature of the groups in question. Second, agents that have found one another must agree on a

coordinated behavior, including how to communicate future changes to other members of the group. If there is a potential for some agents to be malicious or malfunctioning, a third step requires agents to recognize and react to such untrustworthy agents in a way that the whole collective can agree with.

In this chapter I provide approaches to each of these tasks for both hives and swarms, with proofs accompanying each approach.

6.1 Terminology

6.1.1 Swarms and Hives

Assume that there is a notion of two agents being **connected**, meaning they can sense one another and, if they have access to inter-agent communication, also communicate with one another. Let the **connectivity graph** have a node for each agent and an edge between each pair of connected agents. Let a **cluster** of agents be the nodes of a connected component of the connectivity graph. Let a **clique** of agents be the nodes of a clique in the connectivity graph—that is, every pair of agents within the clique are connected.

A group of agents belong to the same **swarm** if they are currently and will remain part of a single cluster. A group of agents belong to the same **small swarm** if they are currently and will remain part of a single clique.

Defining a hive is more involved.

Definition 6.1 (Hive). *Define a predicate $c(A_1, A_2, t)$ which is true if and only if agents A_1 and A_2 are connected at time t .*

Define H recursively as follows:

$$\begin{aligned}
 H_0(A, t_0, t) &= \{(A, t_1) \mid t_0 \leq t_1 \leq t\} \\
 H_{i>0}(A, t_0, t) &= \{(A_1, t_2) \mid \exists A_2 \text{ s.t. } (A_2, t_1) \in H_0(A, t_0, t) \wedge \exists t_2 \in [t_1, t] \text{ s.t. } c(A_1, A_2, t_2)\} \quad (6.1) \\
 H(A, t_0, t) &= \bigcup_{i=0}^{\infty} H_i(A, t_0, t).
 \end{aligned}$$

H_i is the set of agents who can receive a message from A in i hops in the specified time window; H is the set of all agents that can receive a message from A in the specified time window.

A **hive** has the property that every agent can get a message to every other agent in finite time.

A set of agents \mathcal{A} is a **hive** at time t_0 with time-varying **lag** $T(t)$ if

$$\forall A \in \mathcal{A} \quad \forall t \geq t_0 \quad \mathcal{A} \subseteq H(A, t, t + T(t)). \quad (6.2)$$

6.1.2 Knowledge, Malice, and Identity

In general, two agents observing the same event may disagree on what they observed and may not be able to determine the exact truth of every predicate they desire. Let **noise-free** mean that any agents observing the same events will agree on what those events imply; **noisy** means that agents observing the same events may disagree. An agent A is **sure** or **1-sure** of a predicate x if its observations are sufficient to establish x is true. For noisy situations, define k -surety inductively: an agent A is **k -sure** of predicate x if it is sure of the predicate “any agent observing the same events as I would be at least $(k - 1)$ -sure of x .” **0-sure** means not sure.

An agent is **malicious in T** if, during time window T , it acts in a way not permitted by the algorithm it is supposed to be following. An agent is **malicious** if it is malicious in some future

time window. An agent is **trustworthy** if it is not malicious.

A finite observation can at most verify that an agent is not malicious in some time window T ; this provides no guarantee it will not be malicious after T . Thus, there is no way to deduce that an unknown agent or swarm is trustworthy without some type of oracle or *a priori* knowledge about trustworth. This leads to the following theorem.

Theorem 6.1 (Identifiability). *Coalescence in the presence of an unbounded but finite number of malicious agents requires that each agent can identify each other encountered agent and have sufficient storage to remember the identity of all malicious agents it has encountered.*

Proof. At least one agent must adjust its behavior when agents rendezvous or they cannot coalesce. By the pigeonhole principle, any change in behavior will delay the time to rendezvous with some other agents. If the same malicious agent can repeatedly rendezvous with an agent, it can repeatedly delay that agent rendezvousing with other, non-malicious agents.

Preventing repeatedly reacting to encounters with the same malicious agent requires that each agent be both able to identify the agents with which it rendezvous and to remember the malicious subset of the agents it encounters. □

I assume that every agent has a unique **identity** that is visible to all other agents; this means malicious agents are at most weakly Byzantine. An identity is **communicable** if it can be serialized into the same form by every observer, including the agent possessing the identity. Agent identities are **comparable** if all agents agree on a total ordering of identities. All of my algorithms assume comparable identities; all except those for small swarms also assume communicable identities.

6.2 Small Swarms

In a **small swarm** the entire connectivity graph is a single clique; that is, every agent is connected to, and can thus observe, every other agent. For coalescence, **small swarm** means that if all of the agents were to coalesce, they could form a small swarm and remain a small swarm as they move through the environment. I demonstrate that this potential for mutual-observation allows the development of algorithms that guarantee coalescence of all correctly-behaving agents without requiring any explicit communication, even if there an arbitrary number of the agents are malicious.

6.2.1 Full Trust

The simplest coalescence algorithm works for small swarms with no noise or malice. When n agents rendezvous, the one with the maximum identity becomes the leader. The leader continues its rendezvous search uninterrupted, while the other agents follow it. This general approach has been discussed by several authors, often in the context of motivating the rendezvous problem (see [Section 1.3.2](#)).

6.2.2 Noise-Free Malice

When observations of agents are noise-free, each agent in a swarm will agree upon the malice of the leader provided they have all observed a sufficient quantity of the leader's actions. The simplest extension of the full-trust rendezvous in this case is to have each agent maintain a blacklist of agents it has observed acting maliciously. Individual blacklists, such as those illustrated in [Figure 6.1](#), fail because a malicious agent can manipulate different agents' blacklists separately.



Figure 6.1: Failure of two example approaches to individual blacklisting where $A > B > C$ and A is malicious. If agents do not join swarms with blacklisted leaders (a) then A can play nice with C to ensure B and C do not coalesce. If agents join swarms with any non-blacklisted members (b) then A can follow B until C comes along; then, when A and B 's rendezvous searches differ, C follows A while B follows itself.

Individual blacklists that track entire swarms instead of individual agents do work, but require exponential space (there are $n!$ possible swarms of n agents).

I propose instead a shared-blacklist approach that makes three guarantees: trustworthy agents never leave a swarm, sufficiently malicious agents never lead a swarm for too long, and each swarm increases in size between times a malicious agent leads it.

The core of this approach is a specialized blacklist that stores two kinds of elements: “arrivals” and “bans.” The API of this blacklist is given in [Listing 6.1](#).

Lemma 6.1 (Blacklist Properties). *The blacklist described in [Listing 6.1](#) satisfies the following properties:*

1. *There is at most one (ban, A) and one (arrive, A) in the list for each agent A .*
2. *An element (ban, A) never appears above an element (arrive, A) for a single agent A .*
3. *If (ban, A) and (arrive, A) are both in the list, there is an element (arrive, B) between them.*

Proof. All three properties are trivially satisfied by an empty list. By induction, if each API procedure maintains each property then the properties are maintained for all blacklists.

Listing 6.1 Blacklist

Initialize with an empty list.

Procedure “ban(A)”:

1. If (ban, A) is in the list, remove it.
2. If there is an (arrive, A) element in the list,
3. Let B be the set of bans below (arrive, A) and above any other arrival.
4. For each element (x , A') above (arrive, A) from bottom to top,
5. If x is “ban”, add A' to B ;
6. Otherwise if $A' \in B$, remove element (x , A') from the list;
7. Otherwise break out of this loop early.
8. Remove (arrive, A) from list.
9. Push (ban, A) on top of list.

Predicate “isBanned(A)”:

1. For each element (t , B) in list starting at the top,
2. If t is “arrive”, return false.
3. If t is “ban” and $A = B$, return true.
4. Return false.

Procedure “see(A_1, A_2, \dots)”:

1. Initialize Q to be an empty container.
 2. Initialize b to false.
 3. For each A_i ,
 4. If (arrive, A_i) is in the list, ← never true as used by Algorithm 6.1
 5. ignore A_i .
 6. If isBanned(A_i), add A_i to Q ;
 7. Otherwise,
 8. Push (arrive, A_i) on top of list
 9. Set b to true.
 10. If b is true,
 11. Push (arrive, Q_i) on the top of list for each Q_i in Q .
-

Algorithm 6.1 Noise-free Small Swarms

Each agent maintains a blacklist as described in [Listing 6.1](#). Each agent identifies as the “leader” the visible agent (including itself) with the greatest identity which is not currently banned by the blacklist.

If an agent considers itself to be the leader, it follows its individual rendezvous search (see, e.g., [Section 3.3.1](#)). Otherwise it follows the leader, verifying that the leader is following its prescribed rendezvous search.

Each agent handles the following events:

Agent A leaves swarm: Invoke the “ban(A)” method of the blacklist.

Encounter swarm A_1, A_2, \dots : Invoke the “see(A_1, A_2, \dots)” method of the blacklist. If that invocation changed the blacklist, clear estimation information¹ used to verify the leader’s behavior.

Line 5 of procedure “see” is never exercised in this step because the agents in a new swarm are either new or were banned when they left to join the new swarm.

Leader A fails to follow its prescribed search: Invoke “ban(A)” and identify a new leader.

The predicate “isBanned” does not modify the blacklist, and hence maintains each property.

Both “ban(A)” and “see(A_1, A_2, \dots)” remove old elements before adding new ones, maintaining property 1.

Both “ban(A)” and “see(A_1, A_2, \dots)” remove any elements (arrive, A) from beneath a new element (ban, A), maintaining property 2.

The loop on line 4 of “ban(A)” removes arrivals that were separated from a ban by an arrival that is being removed, maintaining property 3.

“see(A_1, A_2, \dots)” first pushes all (arrive, A_i) nodes that are either not matched by a (ban, A_i) or are separated from their (ban, A_i) by an existing arrival element, which maintains property 3.

After adding such an (arrive, A_i) the procedure may add some additional (arrive, A_j); these are separated from their (ban, A_j) by the just-added (arrive, A_i), maintaining property 3. □

¹For k -surety to change, each agent must have some type of estimator or filter that converts observations into bounds on an observed agent’s state (see [Section 6.1.2](#)). The details of such an estimator will depend on the particular sensors and set of possible behaviors of the agents in question; however, any estimator can be reset to the state it had before processing its first observation.

see(1): ①	see(4): ①②③④	see(5): ④①②③⑤
ban(1): ①	see(1): ①②③④①	see(1): ④①②③⑤①
see(2): ①②	ban(1): ②③④①	ban(1): ④②③⑤①
see(1): ①②①	see(2): ②③④①②	see(2): ④②③⑤①②
ban(1): ②①	see(1): ②③④①②①	see(1): ④②③⑤①②①
ban(2): ①②	ban(1): ②③④②①	ban(1): ④②③⑤②①
see(3): ①②③	ban(2): ③④①②	ban(2): ④③⑤①②
see(1): ①②③①	see(3): ③④①②③	see(3): ④③⑤①②③
ban(1): ②③①	see(1): ③④①②③①	see(1): ④③⑤①②③①
see(2): ②③①②	ban(1): ③④②③①	ban(1): ④③⑤②③①
see(1): ②③①②①	see(2): ③④②③①②	see(2): ④③⑤②③①②
ban(1): ②③②①	see(1): ③④②③①②①	see(1): ④③⑤②③①②①
ban(2): ③①②	ban(1): ③④②③②①	ban(1): ④③⑤②③②①
ban(3): ①②③	ban(2): ③④③①②	ban(2): ④③⑤③①②
	ban(3): ④①②③	ban(3): ④⑤①②③

Figure 6.2: A single example blacklisting sequence achieving worst-case performance: $(2^{m+1} - t)t$ time and $t + 2m - 1$ space, where t is the number of trustworthy agents and m is the number of malicious agents. In this example, agents 1, 2, and 3 are malicious; agents 4 and 5 are not. Observe that after the arrival of each new agent all of the changes effected by malicious agents prior to that point may be repeated.

Theorem 6.2. *Any set of agents, S , following [Algorithm 6.1](#), are guaranteed to coalesce into a small swarm despite any number of malicious agents M provided that*

- *all observations made by agents in S are noise-free,*
- *all identities of agents in $S \cup M$ are comparable,*
- *the underlying rendezvous algorithms used by each agent in S is repeating, and*
- *any agent in S observing a behavior of another agent that would prevent rendezvous eventually becomes 1-sure it has done so.*

Proof. Because [Algorithm 6.1](#) clears estimation information when each new agents arrives, and since perceptions are noise-free, all agents in the swarm agree on when other agents should be banned.

Because bans happen synchronously and because “isBanned” only considers bans that have happened after the last non-banned agent arrived, [Algorithm 6.1](#) ensures that each pair of non-banned agents agree on the value of “isBanned(A)” for every agent A in the swarm. Each pair thus also agrees on which agent is the leader.

All trustworthy agents follow the same leader correctly and are thus never banned by other trustworthy agents following [Algorithm 6.1](#).

A malicious agent can impede a rendezvous only by (1) failing to execute its prescribed search as the leader or (2) joining the swarm as a new leader. In [Algorithm 6.1](#), both of these types of impeding behaviors change the configuration of the blacklist. Given m malicious and n trustworthy agents, a single blacklist may go through at most $(2^{m+1} - 1)n$ configurations, so malicious agents can only prevent a finite number of rendezvous opportunities. Since rendezvous is repeating, preventing a finite number of rendezvous is insufficient to prevent swarm coalescence.

Therefore, [Algorithm 6.1](#) guarantees coalescence of all trustworthy agents. □

Although the runtime of [Theorem 6.2](#) is exponential in the number of malicious agents, it can guarantee rendezvous even if only two trustworthy agents are present within a large population. This ability to handle majority-malicious swarms can also be extended to agents experiencing noise, as is outlined in the next section.

6.2.3 Noise and Malice

[Algorithm 6.1](#) does not work when agents may disagree on when to ban a leader. Agreeing on banning decisions in a noisy environment requires agents that can unambiguously share some

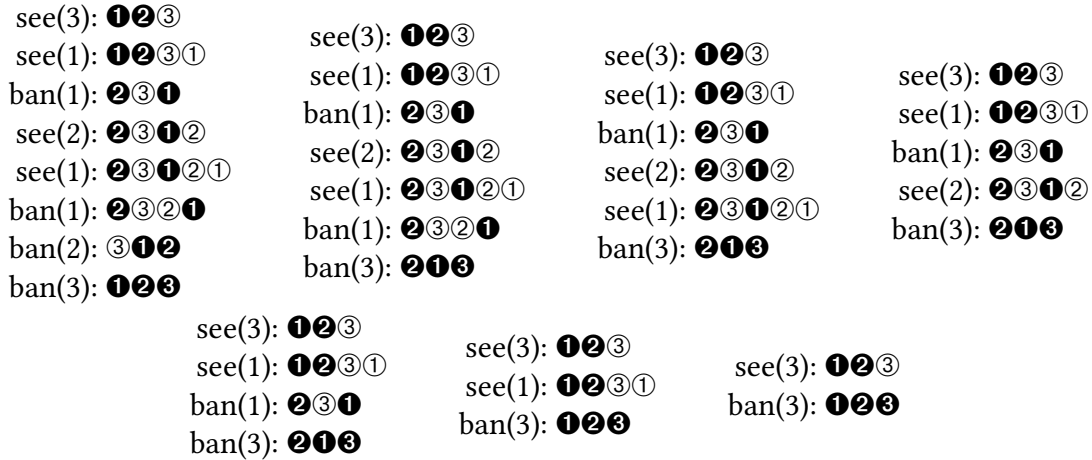


Figure 6.3: A newly-arrived agent can allow malicious agents to repeat their earlier difficulties, but only as long as the new agent has not acted maliciously. This figure demonstrates that the sooner an agent acts maliciously the less impact the set of malicious agents can have.

information. In particular, I consider agents that can broadcast the k by which they are k -sure that the leader of the swarm is malicious.

If every agent is at least 1-sure that the leader is malicious then the leader may be banned by the entire swarm and a new leader selected.

An agent that is k -sure that the leader is malicious should not desert an agent that is $(k - 1)$ -sure; but that agent will not desert an agent who is $(k - 2)$ -sure, etc. I refer to this set of agents as a **surety-chain**. The blacklist data structure can be easily extended to handle surety chains. In addition to the (arrive, A) and (ban, A) elements it now also stores (sure, (A, k, B)) elements meaning “agent A is k -sure agent B acted maliciously.” The additional functionality of the augmented blacklist are outlined in [Listing 6.2](#).

[Algorithm 6.2](#) implicitly assumes that agents store estimation information for every other agent in the swarm. This assumption is manifest in including B in the messages sent. It is possible that the computational or storage costs of the estimator used will not allow this level of storage. If agents store only estimation information for the current swarm leader, [Algorithm 6.2](#)

Listing 6.2 Noisy Blacklist

Include “see,” “ban,” and “isBanned” from Listing 6.1, with the following modifications:

- Removing (arrive, A) also removes all (sure, (A , *, *)) and (sure, (*, *, A)).
- If two or more surety elements differing only in k are in the list with no arrivals between them, remove all but one having the largest k .

Procedure “message(A , k , B)”:

1. If (sure, (A , $k - 1$, B)) is in the list, remove it.
2. Push (sure, (A , k , B)) on top of list.
Determine R , the set of agents no longer part of this agent’s surety chain.
3. Initialize R be to the set of all non-banned agents referenced in the list.
4. Initialize Q to {self}. \leftarrow queue of agents to remove from R
5. As long both R and Q are not empty,
 6. Remove an element A_1 from Q ;
 7. Remove element A_1 from R ;
 8. For each element A_2 in $R \setminus Q$,
 9. If trust(A_1 , A_2), add A_2 to Q .
10. For each agent A_i in R , invoke ban(A_i).

Function “trust(A , B)”: \leftarrow does A trust B ?

$$k\text{-surety}(A, B) = 0 \wedge k\text{-surety}(B, A) = 0 \wedge \forall C \left| k\text{-surety}(A, C) - k\text{-surety}(B, C) \right| \leq 1$$

Function “ k -surety(A , B)”: \leftarrow for what k is A k -sure that B is malicious?

1. For each element (t , X) in list starting at the top,
 2. If t is “arrive”, return 0.
 3. If t is “sure”,
 4. Let (A_1 , k , A_2) = X ;
 5. If $A_1 = A$ and $A_2 = B$, return k .
 6. If t is “ban”, return ∞ .
 7. Return 0.
-

Algorithm 6.2 Noisy Small Swarms

Each agent maintains a blacklist as described in [Listing 6.2](#). Each agent identifies as the “leader” the visible agent (including itself) that is not currently banned by the blacklist having the greatest identity.

If an agent considers itself to be the leader, it follows its individual rendezvous search. Otherwise it follows the leader, verifying that the leader is following its prescribed rendezvous search.

Agent A should handle the following events:

Agent B leaves swarm: Do nothing (subsequent arrivals will also be ignored by line 5 in [Listing 6.1](#)’s “see” procedure).

Encounter swarm A_1, A_2, \dots : Invoke the “see(A_1, A_2, \dots)” method of the blacklist. If that invocation changed the blacklist, clear estimation information used to verify other agents’ behavior.

A becomes k -sure that B is not behaving correctly: Broadcast (A, k, B) to all agents (including self).

Receive (A_1, k, A_2) : Invoke the “message(A_1, k, A_2)” method of the blacklist.

still works without modification, as the proof of [Theorem 6.3](#) below does not depend on non-leader estimation. If only leader state is estimated then the messages can be simplified to (A, k) instead of (A, k, B) .

Theorem 6.3. *Any agents following [Algorithm 6.1](#), are guaranteed to coalesce into a small swarm despite any number of malicious agents provided that*

- *all identities are comparable*
- *identities are communicable,*
- *rendezvous is repeating,*
- *agents agree on when other agents arrive, and*
- *any agent observing a behavior that would prevent rendezvous eventually becomes k -sure it has done so, for every $k \geq 1$.*

Proof. An agent does not ban another agent until it is sure that all the agents it trusts are also going to ban it.

For an agent to be k -sure the leader is malicious but not ban it requires there be a chain of agents that are $k - 1$ -sure, $k - 2$ -sure, etc, down to 0-sure. Since all trustworthy agents will be at least $k - 1$ -sure, such a chain must contain at least $k - 1$ malicious agents. Eventually an agent in a swarm of m malicious agents will become $m + 2$ -sure that the leader is malicious, by which point the leader will be banned.

Agents might not agree on the definition of “leave the swarm;” hence, agents that leave cannot be banned directly; however, such departures will be handled through the usual surety mechanism. Ignoring departures and re-arrivals beyond broadcasting surety causes both those who see departures and those who do not to have compatible blacklists.

The rest of the proof mirrors the proof of [Theorem 6.2](#). □

One antecedent to [Theorem 6.3](#) is that any agent observing a behavior that would prevent rendezvous eventually becomes k -sure it has done so for arbitrary k . Most rendezvous algorithms can be made to handle bounded errors by decreasing the used perception radius to a value below the actual radius available to the agent. For any asymptotically-accurate estimation filter, this finite buffer will provide the requisite k -surety.

6.3 Large Swarms

When swarms are large enough that they do not form a single clique, coalescence is a combination of rendezvous and cohesion rather than rendezvous and leader-following. Cohesion contains an

implicit level of trust as there is no immediate way to verify whether agents are reacting to other agents outside of sensor range or are feigning such interactions.

All components of large swarm coalescence have been solved elsewhere in this dissertation, as outlined below.

When agents all trust one another, coalescence is a simple combination of three subproblems: rendezvous search (see [Section 3.3](#)), global cohesion (see [Section 5.2](#)), and consensus building to select which agent's rendezvous search algorithm should direct the swarm. Because rendezvous presupposes agents have known unique identities (see [Theorem 3.1](#)), consensus building is as simple as deferring to the agent with the greatest identity.

When some agents in a swarm are malicious, coalescence is achieved by the combination of a malice-tolerant cohesion algorithm and communication policy (see [Section 5.2.1](#)) and a leader malice identification algorithm (see [Section 6.2.2](#) and [Section 6.2.3](#)).

6.4 Trustworthy Hives

According to [Definition 6.1](#), a hive is groups of agents that are able to share messages with all other agents in the hive. One way to achieve this kind of hive is by having every agent be aware of a “home location,” a central place where agents go to communicate. The hive algorithms I present below assume such a home location.

Assuming that agents are capable of communicating locations to other agents, coalescence of trustworthy hives of agents can be performed quite efficiently. By having an agent remain at the home location, rendezvous can be reduced to the simpler problem of static search. By having

that home agent coordinate inter-agent communication and task allocation, duplication of effort can be almost totally avoided.

Hive coalescence is more efficient if every meeting of agents from distinct hives results in the two hives combining into a single hive. This level of efficiency is made difficult because agents from a single hive might encounter many other hives before the message of their mutual meeting can be communicated to the rest of the hive. In particular, hive X might have decided to join hive Y but hive Y might depart to join hive Z before hive X can reach Y 's previous home location. This situation does not prevent coalescence— X and Z will eventually locate one another as well—but it is an avoidable inefficiency.

Algorithm 6.3 is a technique for hive coalescence that can avoid all double discoveries using linear memory or can avoid most, but not all, double discoveries using only finite memory. In both cases, memory is also needed to execute a search of the environment. In vector spaces, remembering a single coordinate (which requires logarithmic space in the size of the environment to be searched) is sufficient to execute a search (as a Searcher in **Algorithm 6.3**) and to keep track of what region of the environment needs to be searched next (as a Director in **Algorithm 6.3**). The space required for the search portion of **Algorithm 6.3** may vary in other environments.

Conceptually, **Algorithm 6.3** has hives with directors that stay put and coordinate searchers to locate other hives. When hives encounter one another, one hive directs all its agents to join the other.

Several of the interactions in **Algorithm 6.3** are illustrated in **Figure 6.4**.

Algorithm 6.3 Hive Coalescence (part 1 of 2)

Assume that each agent is able to track the passage of time (to within bounded error) and its position in the environment. Each agents also remembers (R, I, L, t_1, t_2, S) , where

- R is its current role, which is one of the following:
 - **Director:** Remains at its home. The purpose of a director is to coordinate searchers.
 - **Searcher:** Travels to and executes a search. If the search is completed, travels to its home.
 - **Redirector:** Waits in place. When $now = t_2$, becomes a searcher instead.
- I is its hive identity, which is the maximal identity of any agent in the hive.
- L is its hive's home location.
- t_1 and t_2 are points in time.
- S is the state of an in-progress search, either an individual agent's search (for agents with the Searcher role) or the combined search efforts of the hive (for agents with the Director role).

Except as defined below, an agent changes state and role when it encounters another agent. How they change depends on the relative order of the two agents' hive identities. In the following list, each entry " $X \stackrel{\cong}{\approx} Y$ " defines what happens when an agent $A = (R, I, L, t_1, t_2, S)$ encounters an agent $A' = (R', I', L', t'_1, t'_2, S')$ where $R = X$, $R' = Y$, and $I \stackrel{\cong}{\approx} I'$.

Searcher = Director: Updates the director's wait time and assigns a new search.

1. Replace t'_2 with $\max\{t_1, t'_2\}$.
2. Replace S' with the next area that needs searching.
3. Update S to reflect the work that will be done by S' .
4. Replace t'_1 with the time needed for A' to complete S' and return home.
5. Replace t'_2 with $\max\{t_1, t'_2\}$.

Searcher > Director: The director becomes a redirector and the searcher remembers how long its hive needs to wait.

1. Replace t'_1 with t'_2 plus the time needed to reach L from L' .
2. Replace t_1 with $\max\{t_1, t'_1\}$.
3. Replace R' with Redirector, I' with I , and L' with L .

Searcher < Director: The searcher converts to the new hive and immediately searches out its old hive's director.

1. Replace t_1 with the time needed to travel to L' and back again.
2. Replace S with an unfinished search that will visit just L .
3. Replace I' with I and L' with L .
4. Replace t'_2 with $\max\{t'_2, t_1\}$.

Searcher = Redirector: Searchers from and redirectors to the same hive ignore one another.

(continued...)

Algorithm 6.3 Hive Coalescence (part 2 of 2)

(...continued)

Searcher < Redirector: If the redirector is at the searcher's home location, the searcher simply converts to the new hive.

1. Replace I with I' and L with L' .

Otherwise, the searcher converts to the new hive and immediately searches out its old hive's director.

1. Replace t_1 with the time needed to travel to L and then to L' .
2. Replace S with an unfinished search that will visit just L .
3. Replace I' with I and L' with L .
4. Replace t'_1 with $\max\{t'_1, t_1\}$.

Searcher > Redirector: This is the case that needs linear memory or is lossy. The two agents know of three hives, each three of which may require an agent's presence before all may be visited. Since this case may arise an arbitrary number of times for each agent, the third hive cannot be remembered with bounded memory. Thus, I simply have the redirector forget about its old hive:

1. Replace t'_1 with t'_2 plus the time needed to reach L from here.
2. Replace t_1 with $\max\{t_1, t'_1\}$.
3. Replace I' with I and L' with L .

As a lossless but unbounded-memory alternative, the searcher could maintain a list of the old home locations of all redirectors it encounters which could be used to create specially-targeted searches by its director.

Searcher = Searcher: Searchers from the same hive ignore each other.

Searcher < Searcher: Meeting a searcher from another hive is treated similarly to meeting a director from another hive.

1. Replace t_1 with the time needed to travel to L and then to L' .
 2. Replace S with an unfinished search that will visit just L .
 3. Replace I with I' and L with L' .
 4. Replace t'_1 with $\max\{t_1, t'_1\}$.
-

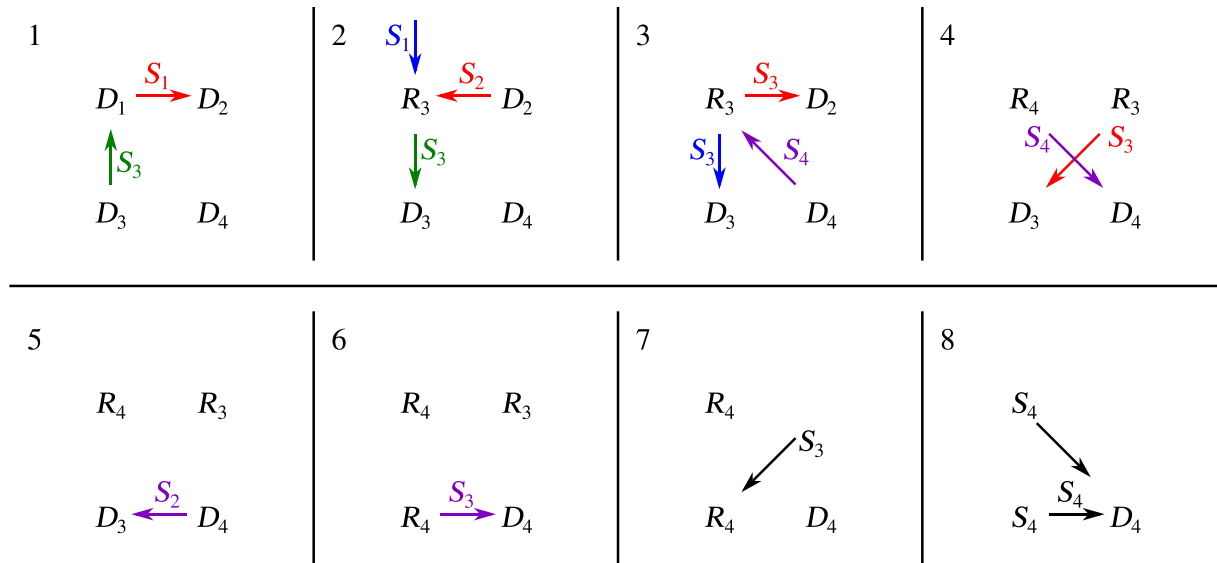


Figure 6.4: Illustration of several of the interactions in [Algorithm 6.3](#). Searchers sent on new searches by directors are not shown to avoid cluttering the images. Frames 5 and 6 happen immediately in the linear-memory version but since search is exhaustive happen eventually in the bounded-memory version too.

Theorem 6.4. *For any two hives $i < j$ of agents following [Algorithm 6.3](#), if any agent in hive i encounters an agent in hive j then each agent in hive i will join some hive $\geq j$ within bounded time.*

A full proof of [Theorem 6.4](#) involves a tedious case analysis. An outline of the proof's significant elements follows.

Proof. When an agent visits its home location, it finds a director or redirector there. This is enforced by directors' and redirectors' t_2 , which causes them to wait for all agents of which they are aware; and searchers' and redirectors' t_1 , which are used to update directors' and redirectors' t_2 to ensure they wait for all agents of newly arriving hives.

When agents from different hives meet, at least one hive thereafter merges with another. This is enforced by each " $<$ " interaction in [Algorithm 6.3](#).

Directors remain in place and will be found by a search unless their hive joins another first.

Thus, no agent is deserted by the rest of its hive and every hive eventually encounters each other hive, so all agents eventually coalesce. □

6.5 Untrustworthy Hives

Within a hive with limited trust, agents must verify the behavior of other agents. Verification may be accomplished by having groups of agents perform each role in concert, where the group is large enough to trust a majority of the agents involved.

However, if even a single agent in each hive may be untrustworthy then it is possible that any other hive located is made of entirely malicious agents that have left their original hives and congregated elsewhere. Hence, all hives in a limited-trust environment are suspect of being purely malicious.

Coalescence of hives that do not trust one another must proceed in the same full-verification-and-blacklist pattern as swarm coalescence: a following hive must shadow every action of a leading hive and blacklist it when it behaves incorrectly. If each hive operates with few enough groups that every hive may have the same number of groups, then the same blacklist techniques outlined in [Listing 6.1](#) and [Listing 6.2](#) may provide for coalescence of untrustworthy hives.

The only efficiency gained by untrustworthy hives over untrustworthy swarms is the more efficient rendezvous available when some agents can remain in place and when search can be parallelized.

6.6 Conclusion

I demonstrated that coalescence is more involved than rendezvous and cohesion when agents are untrustworthy or when the efficiency of hives searching in parallel is desired. I presented algorithms for small swarm coalescence in the face of an arbitrary number of malicious agents, with or without precise information. I also presented an algorithm for the efficient coalescence of hives without malicious agents, one which allows every agent to act in parallel and ensures that no information learned by any agent is lost to the hive.

For hives with malice and large swarms I observed that coalescence is essentially just a combination of rendezvous and cohesion, and outlined how that combination can be used.

Other work has investigated a stronger notion of malice in noise-free graphs with known numbers of agents [16]. It would be interesting to see if that model of strongly-Byzantine agents can be adapted to noisy geometric environments and/or swarms too large to be simultaneously observed.

While the theory of rendezvous, cohesion, and coalescence improve, there remains significant work to be done before these algorithms become practical for real robots. Among the issues to be investigated before such tests can be made are details of message passing frequency, models of agent state estimation, and budgeting of physical resources.

This chapter is the first to address provable coalescence of geometric agents with malice and the first to investigate the interplay of noise and malice on coalescence.

Chapter 7

Conclusion

In this dissertation I have investigated several important theoretic problems regarding the behavior and capabilities of groups of mobile computing agents that are individually limited in what they can accomplish. The contributions of this dissertation together create a clearer picture of what groups of agents can accomplish.

In [Chapter 2](#) I investigated the impact of stigmergic vs. broadcast communication. I improved previous bounds on the ability of stigmergic agents to simulate broadcast-based algorithms, demonstrating that, allowing for some delays and additional agents, the capabilities of both types of agents are equivalent. I also investigated the space- and time-localization information contained in different kinds of communication and discussed how communication itself could be used to synchronize agents' clocks and coordinate frames.

In [Chapter 3](#) I explored the problem of having noisy mutually-oblivious agents with only local sensations locate one another in unknown environments, be they geometric or graph-based. I demonstrated a set of necessary capabilities that any agents achieving worst-case time-bounded rendezvous must have and showed algorithms that achieved worst-case time-bounded

rendezvous for those capabilities. When agents did not accumulate positional error as they moved and had access to an accurate clock, these algorithms achieved optimal time bounds that had been proven for much more limited environments. Together, these contributions demonstrated that rendezvous is not only tractable but efficient when limited-noise localization is available.

In Chapters 4, 5, and 6 I built a series of algorithms and proofs that together demonstrate that agents with limited local sensation can provably form and remain in a single cohesive group. I demonstrated how pairs of agents could provably maintain interconnectivity, how groups of agents could select pairs to remain connected in a way that kept the group connected without getting stuck, and how groups could recognize and/or cope with malfunctioning or malicious agents that might try to keep a group from coalescing or remaining cohesive. These algorithms and proofs rely only on simple notions of distance that apply to Euclidean and non-Euclidean geometries as well as graphs and other environments where motion is local and a distance function can be defined.

The various elements of this dissertation have advanced the theoretic understanding of mobile agents on several fronts and provided the first provably-correct algorithms for many common mobile agent tasks. However, there is much that remains to be done. Lower- and upper-bounds I have developed provide the first bounds on the worst-case complexity of several tasks, but most are not tight. Proofs of correctness rely on far more general definitions of motion and sensation than previously attempted in theoretic work, including ideas of noise and error, but are not yet as general as the models used in physical robot design. Most of my work is explicitly designed to be environment-agnostic and function in graphs as well as geometric environments, leaving open extensions into particular restrictions of environments such as planar graphs or cluttered maps. Other ideas for future work are contained at the end of each chapter of this dissertation.

The proofs and algorithms I have developed have pushed the theoretic boundaries of mobile agent design closer to practice than they were previously.

Appendix A

Glossary

Active (stigmergy) Communication through environmental changes effected by actions with no other objective besides stigmergy. Ant pheromones are an example of active stigmergy.

Additive (stigmergic communication) A finite space may hold only a finite set of signals, but the strength of those signals may be increased by later agents.

Algorithm Any deterministic behavior selection process. The phrase “algorithm” is used to refer to both the finite steps that map a given set of sensations to short-term actions and the overall process of applying the same algorithm to each successive set of sensations to create ongoing behavior.

Agent An agent is a mathematical abstraction of a robot, avatar, or other entity with the ability to sense other agents and its environment and to select its own behavior.

Each agent is defined by a state $s \in S$ and a set of allowable behaviors \mathcal{B} .

\mathcal{B} The set of behaviors an agent can achieve.

Behavior The behavior of an agent is how its states evolve over time. Each behavior is contingent on the initial state of the agent as well as environmental inputs. If $B \in \mathcal{B}$ is a behavior then $B_s(t)$ is the state an agent initially in state s would enter after following behavior B for t seconds.

Behaviors may have various domains and may or may not be live and/or universal.

This definition is based on the common usage “what something does” and not the technical definition used in behavior-based artificial intelligence algorithms.

Between x is between of y and z if and only if $d(x, y) < d(y, z)$ and $d(x, z) < d(y, z)$.

Broadcast (communication) Propagating through space without appreciable delay in time.

Connected Two agents are connected if they sense one another. A set of agents are connected if the graph of pair-wise agent connections within the set is a connected graph.

In some cases, “connected” suggests a distance less than r for a fixed r rather than any form of sensation.

d A distance function defined on agent states $d : S \times S \rightarrow \mathbb{R}_0^+$. For all distance functions, $d(s, s) = 0$. Some distance functions also satisfy the triangle inequality.

Where appropriate, d is assumed to work on agents and positions as well as states.

Distance see d .

Detect see Sense.

Domain (of behavior) The domain of a behavior is the set of initial states over which it is defined.

Drifting (mobility) The uncertainty of an agent in its own location increases the as the agent travels.

Erasable (stigmergic communication) A finite space may hold only a finite amount of information, but the information stored may be changed freely.

Exact (mobility) An agent with exact mobility can move to any specified location without appreciable error.

Fading (communication) Reducing in strength as it propagates through space-time.

Hive A hive is a group of agents who share knowledge of a common base location.

Holonomic The allowable behaviors of a holonomic agent do not depend on its orientation. This can either be because the agent can move sideways or because it can turn quickly enough that orientation can be ignored in practice.

Indelible (stigmergic communication) A finite space may hold only a finite amount of information, and once written that information cannot be changed or added to.

Individual (clock) Agents have individual clocks if each agent's clock rate is fixed but not necessarily the same as that of other agents.

Lagging (communication) Propagating through space with an appreciable delay in time.

Limited (communication) Reaching only space-time near the originating agent.

Limitless (communication) Reaching all space (broadcast) or all future time (stigmergy) without appreciable variation.

Live (behavior) Let $\text{dom}(B)$ be the domain of behavior B . B is said to be live if and only if

$$B_s(t) \in \text{dom}(B) \text{ for all } s \in \text{dom}(B) \text{ and } t \geq 0.$$

Malicious A theoretic abstraction of “not behaving according to the designed algorithm(s).”

Midpoint x is a midpoint of y and z if and only if $d(x, y) = d(y, x)$, $d(x, z) = d(z, x)$, $d(y, z) =$

$$d(z, y), \text{ and } d(x, y) = d(x, z) = \frac{1}{2}d(y, z).$$

Noisy (mobility) An agent with noisy mobility can move to within a fixed distance of any specified location.

Noise-free (sensation) Agent X has noise-free sensation if and only if $X[x] = x$.

Order (of agent) An n th-order agent can directly control the n th derivative of its position. A 0th order agent teleports. Humans are approximately 1st-order, able to change direction and speed of motion almost instantly. Cars and airplanes are 2nd order, able to control acceleration.

Passive (stigmergy) Communication through environmental change resulting as the side-effect of independently desirable activities. For example, determining how the other tugboats are moving based on the motion of the freighter is passive stigmergy.

Patchy (communication) Detectable at some locations/times and not others according to some unknown function of space, time, and/or stochasticity.

Position Formally, two agents share the same position if the distance between their states is 0. Conceptually, position is the portion of the state of an agent that specifies “where” it

is. Practically, the position of an agent is either a point in \mathbb{R}^n for agents operating in a Euclidean environment, or an edge or node for agents operating in graphs.

\mathbb{R} The set of real numbers. \mathbb{R}^+ is the set of real numbers greater than 0; \mathbb{R}_0^+ is the set of real numbers greater than or equal to 0.

r The distance at which sensation becomes assured. That is, for agents with states s_a and s_b , $d(s_a, s_b) \leq r$ implies the agents sense one another.

Rendezvous Two agents rendezvous when they move so as to be able to sense one another. Two algorithms rendezvous if agents following them from any starting states are guaranteed to rendezvous.

Reusable (stigmergic communication) Either erasable or unbounded.

s A particular state of a particular agent.

\mathcal{S} The set of all agent states (for a particular type of agent).

Search (algorithm) A search algorithm will cause an agent following it to sense every location within a time relative to the distance between the location and the initial location.

Sense An agent senses another if it has sufficient information to determine the existence of approximate location of the other agent. The particular sensors and processing used to develop this information is not specified in this document.

Skewed (clock) Agents have skewed clocks if they progress at the same rate from different initial values.

Small Swarm A group of agents where each agent is able to observe every other agent. The connectivity graph of a small swarm is a single clique. Also refers to coalescence algorithms that work provided the agents, once coalesced, could form a small swarm.

State The state of an agent are the pieces of information that define it at a particular point in time. State may include position, heading, computational memory, etc.

Stigmergy Stigmergy is communication between agents, or between the same agent at different times, facilitated by the environment. See also Active (stigmergy) and Passive (stigmergy).

Stigmergy (communication) Propagating through time without moving through space.

Swarm A swarm is a group of agents that are now connected and will behave so as to remain connected in the future.

Synchronized (clock) Agents have synchronized clocks if they always agree on the current time.

t A point in time or a time delta, depending on context.

Time May be either continuous or discrete, but is causal (past events do not depend on the future) and predictable (the same sequence and pacing of stimuli result in the same outcome no matter the absolute time at which they occur).

See also the discussion of time-locality of behavior in [Section 1.2](#) on page 6

Triangle Inequality An optional property of distance functions where $\forall s_1, s_2, s_3 \in \mathcal{S} \quad d(s_1, s_2) + d(s_2, s_3) \geq d(s_1, s_3)$.

Unbounded (stigmergic communication) A finite space may hold only an unlimited amount of information; new information may be added to existing information freely.

Universal (behavior) A behavior is universal if its domain is S . All universal behaviors are also live.

Variable (clock) Agents have variable clocks if their relative rate of progression varies over time.

Bibliography

- [1] S. Alpern. The rendezvous search problem. *SIAM Journal of Control and Optimization*, 33(3):673–683, 1995.
- [2] S. Alpern and S. Gal. Rendezvous search on the line with distinguishable players. *SIAM Journal on Control and Optimization*, 33:1270–1276, 1995.
- [3] S. Alpern and S. Gal. *The theory of search games and rendezvous*. International Series in Operations Research and Management Science. Kulwer Academic Publisher, 2002.
- [4] E. J. Anderson and S. Essegaiier. Rendezvous search on the line with indistinguishable players. *SIAM Journal on Control and Optimization*, 33:1637–1642, 1995.
- [5] C. Astengo-Noguez and L. Velzquez. A vectorial approach on flock traffic navigation. In *Artificial Intelligence, 2008. MICAI '08. Seventh Mexican International Conference on*, pages 300–304, oct. 2008.
- [6] R. Beckers, O. E. Holland, and J.-L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In R. Brooks and P. Maes, editors, *Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 181–189, Cambridge, MA, 1994. MIT Press.
- [7] R. R. Cazangi, F. J. Von Zuben, and M. F. Figueiredo. Autonomous navigation system applied to collective robotics with ant-inspired communication. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 121–128, New York, NY, USA, 2005. ACM. ISBN 1-59593-010-8. doi: <http://doi.acm.org/10.1145/1068009.1068026>.
- [8] A. Cornejo and N. Lynch. Connectivity service for mobile ad-hoc networks. *Self-Adaptive and Self-Organizing Systems Workshops, IEEE International Conference on*, 0:292–297, 2008. doi: <http://doi.ieeecomputersociety.org/10.1109/SASOW.2008.62>.
- [9] F. Cucker and S. Smale. Emergent behavior in flocks. *Automatic Control, IEEE Transactions on*, 52(5):852–862, may 2007. ISSN 0018-9286.
- [10] J. Czyzowicz, A. Kosowski, and A. Pelc. How to meet when you forget: Log-space rendezvous in arbitrary graphs. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC 2010)*, pages 450–459, 2010.

- [11] J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 22–30, 2010.
- [12] X. Défago and A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 97–104, New York, NY, USA, 2002. ACM.
- [13] A. Dessmark, P. Fraigniaud, and A. Pelc. Deterministic rendezvous in graphs. In *Proceedings of the 11th European Symposium on Algorithms (ESA 2003)*, pages 184–195, 2003.
- [14] Y. Dieudonné, O. Labbani-Igbida, and F. Petit. Circle formation of weak mobile robots. *ACM Trans. Auton. Adapt. Syst.*, 3(4):1–20, 2008.
- [15] Y. Dieudonné, S. Dolev, F. Petit, and M. Segal. Deaf, dumb, and chatting asynchronous robots. In T. Abdelzaher, M. Raynal, and N. Santoro, editors, *Principles of Distributed Systems*, volume 5923 of *Lecture Notes in Computer Science*, pages 71–85. Springer Berlin / Heidelberg, 2009. doi: 10.1007/978-3-642-10877-8_8.
- [16] Y. Dieudonné, A. Pelc, and D. Peleg. Gathering despite mischief. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 527–534, 2012.
- [17] P. Dorato, K. Li, E. Kosmatopoulos, P. Ioannou, and H. Ryaciotaki-Boussalis. Quantified multivariate polynomial inequalities. The mathematics of practical control design problems. *Control Systems, IEEE*, 20(5):48–58, Oct 2000. ISSN 1066-033X.
- [18] P. Dykiel. Asymptotic properties of coalescing random walks. Technical Report 2005:15, Uppasala University, December 2005.
- [19] P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Springer Lecture Notes in Computer Science*, pages 242–256, 2008.
- [20] C.-H. Fua, S. Ge, K. D. Do, and K.-W. Lim. Multirobot formations based on the queue-formation scheme with limited communication. *Robotics, IEEE Transactions on*, 23(6):1160–1169, dec. 2007. ISSN 1552-3098.
- [21] A. Ganguli, J. Cortes, and F. Bullo. Multirobot rendezvous with visibility sensors in non-convex environments. *Robotics, IEEE Transactions on*, 25(2):340–352, april 2009. ISSN 1552-3098.
- [22] A. Garcia, C. Li, and F. Pedraza. Rational swarms for distributed on-line bayesian search. In *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*, pages 1–8, Piscataway, NJ, USA, 2007. IEEE Press. ISBN 978-963-9799-08-0.
- [23] S. Ge and C.-H. Fua. Queues and artificial potential trenches for multirobot formations. *Robotics, IEEE Transactions on*, 21(4):646–656, August 2005. ISSN 1552-3098.

- [24] P.-P. Grassé. La construction du nid et les interactions inter-individuelles chez les bellisitermes natalenis et cubitermes sp. la théorie de la stigmergie: essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.
- [25] P. Gurfil. Evaluating uav flock mission performance using dudek's taxonomy. In *American Control Conference, 2005. Proceedings of the 2005*, pages 4679 – 4684 vol. 7, june 2005.
- [26] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, String 1999.
- [27] D. Howden and T. Hendtlass. Collective intelligence and bush fire spotting. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 41–48, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-130-9. doi: <http://doi.acm.org/10.1145/1389095.1389102>.
- [28] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor. Maintaining network connectivity and performance in robot teams. *Journal of Field Robotics*, 25(1-2):111–131, 2008.
- [29] G. Kaminka, R. Schechter-Glick, and V. Sadvov. Using sensor morphology for multirobot formations. *Robotics, IEEE Transactions on*, 24(2):271 –282, april 2008. ISSN 1552-3098.
- [30] S. Kloder and S. Hutchinson. Path planning for permutation-invariant multirobot formations. *Robotics, IEEE Transactions on*, 22(4):650 –665, aug. 2006. ISSN 1552-3098.
- [31] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE Conference on Robotics and Automation*, pages 1398–1404, April 1991.
- [32] D. Kowalski and A. Malinowski. How to meet in anonymous network. In *13th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2006)*, volume 4056 of *Springer Lecture Notes in Computer Science*, pages 44–58, 2006.
- [33] A. Kshemkalyani and M. Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, March 2011.
- [34] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Trans. Auton. Adapt. Syst.*, 1(1):4–25, 2006. ISSN 1556-4665. doi: <http://doi.acm.org/10.1145/1152934.1152936>.
- [35] A. Lazanas and J.-C. Latombe. Landmark-based robot navigation. In *Proceedings of Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 816–822, july 1992.
- [36] X. Li, D. Su, J. Yang, and S. Liu. Connectivity constrained multirobot navigation with considering physical size of robots. In *Proceedings of the International Conference on Automation and Logistics*, pages 24–29, Chongqing, China, August 2011.
- [37] Y. Li, K. Yuan, and W. Zou. Nonholonomic mobile robot formation control with kinodynamic constraints. In *PCAR '06: Proceedings of the 2006 international symposium on Practical cognitive agents and robots*, pages 200–211, New York, NY, USA, 2006. ACM.

- [38] M. Mamei and F. Zambonelli. Physical deployment of digital pheromones through rfid technology. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1353–1354, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi: <http://doi.acm.org/10.1145/1082473.1082769>.
- [39] M. Mamei and F. Zambonelli. Pervasive pheromone-based interaction with rfid tags. *ACM Transactions on Autonomous and Adaptive Systems*, 2(2):4, 2007. ISSN 1556-4665. doi: <http://doi.acm.org/10.1145/1242060.1242061>.
- [40] G. D. Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 335(3):315–326, 2006.
- [41] R. Menezes, F. Martins, F. E. Vieira, R. Silva, and M. Braga. A model for terrain coverage inspired by ant's alarm pheromones. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 728–732, New York, NY, USA, 2007. ACM. ISBN 1-59593-480-4. doi: <http://doi.acm.org/10.1145/1244002.1244164>.
- [42] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas. Maintaining connectivity in mobile robot networks. In *Proceedings of the 11th International Symposium on Experimental Robotics*, pages 117–126, 2009.
- [43] D. L. Mills, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4: Protocol and algorithms specification. IETF RFC 5905, June 2010. URL <https://tools.ietf.org/html/rfc5905>.
- [44] E. H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26(9):394–395, 1920. doi: <http://dx.doi.org/10.1090/S0002-9904-1920-03322-7>.
- [45] N. Moshtagh, N. Michael, A. Jadbabaie, and K. Daniilidis. Vision-based, distributed control laws for motion coordination of nonholonomic robots. *Robotics, IEEE Transactions on*, 25(4):851–860, aug. 2009. ISSN 1552-3098.
- [46] M. Namvar and F. Aghili. Adaptive force-motion control of coordinated robots interacting with geometrically unknown environments. *Robotics, IEEE Transactions on*, 21(4):678–694, August 2005. ISSN 1552-3098.
- [47] I. Newton. *Methodus fluxionum et serierum infinitarum*. 1664–1671.
- [48] Oxford English Dictionary. *Oxford English Dictionary*, entry “stigmergy, *n.*”. Oxford Press, second edition, 1989.
- [49] L. Panait and S. Luke. A pheromone-based utility model for collaborative foraging. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 36–43, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 1-58113-864-4. doi: <http://dx.doi.org/10.1109/AAMAS.2004.25>.
- [50] R. Penrose. A generalized inverse for matrices. In *Proceedings of the Cambridge Philosophical Society*, volume 51, pages 406–413, 1955.

- [51] G. A. S. Pereira, A. K. Das, and V. Kumar. Decentralized motion planning for multiple robots subject to sensing and communication constraints. In *Proceedings of the 2003 International Workshop on Multi-Robot Systems*, pages 267–278, 2003.
- [52] Plutarch. *Theseus*, pages 15–19.
- [53] S. Poduri and G. S. Sukhatme. Achieving connectivity through coalescence in mobile robot networks. In *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*, pages 1–6, Piscataway, NJ, USA, 2007. IEEE Press.
- [54] S. Poduri and G. S. Sukhatme. Latency analysis of coalescence in robot groups. In *IEEE International Conference on Robotics and Automation*, pages 3295–3300, 2007.
- [55] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIG-GRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: <http://doi.acm.org/10.1145/37401.37406>.
- [56] N. Roy and G. Dudek. Collaborative exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, September 2001.
- [57] J. A. Sauter, R. Matthews, H. Van Dyke Parunak, and S. A. Brueckner. Performance of digital pheromones for swarming vehicle control. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 903–910, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi: <http://doi.acm.org/10.1145/1082473.1082610>.
- [58] T. Schelling. *The strategy of conflict*. Oxford University Press, Oxford, England, UK, 1960.
- [59] T. W. Sederberg. *Computer Aided Geometric Design*. Brigham Young University, 2012. <http://hdl.lib.byu.edu/1877/2822>.
- [60] A. Shiloni, N. Agmon, and G. A. Kaminka. Of robot ants and elephants. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 81–88, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.
- [61] J. Snape, J. van den Berg, S. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *Robotics, IEEE Transactions on*, 27(4):696–706, aug. 2011. ISSN 1552-3098.
- [62] P. Tabuada, G. Pappas, and P. Lima. Motion feasibility of multi-agent formations. *Robotics, IEEE Transactions on*, 21(3):387–392, June 2005. ISSN 1552-3098.
- [63] L. A. Tychonievch and J. P. Cohoon. Coalescing swarms of limited capacity agents: Meeting and staying together (without trust). *IAENG International Journal of Computer Science (IJCS)*, 39(3):254–260, 2012.
- [64] L. A. Tychonievch and J. P. Cohoon. Cohesion: Keeping independently-moving agents close together. Technical Report CS-2012-03, University of Virginia, 2012.

- [65] L. A. Tychonievich and J. P. Cohoon. Guaranteeing rendezvous of oblivious limited-capability mobile agents. Technical Report CS-2012-04, University of Virginia, 2012.
- [66] L. A. Tychonievich, R. P. Burton, and L. P. Tychonievich. Versatile reactive navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009 (IROS 2009)*, pages 2966–2972, St. Louis, MO, October 2009. IEEE.
- [67] J. van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928 –1935, may 2008.
- [68] H. Van Dyke Parunak, S. Brueckner, and J. Sauter. Digital pheromone mechanisms for coordination of unmanned vehicles. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 449–450, New York, NY, USA, 2002. ACM. ISBN 1-58113-480-0. doi: <http://doi.acm.org/10.1145/544741.544843>.
- [69] J. Vazquez and C. Malcom. Distributed multirobot exploration maintaining a mobile network. In *Proceedings of the 2nd International IEEE Conference on Intelligent Systems*, volume 3, pages 113–118, 2004.
- [70] D. Yamins. Towards a theory of “local to global” in distributed multi-agent systems (ii). In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 191–198, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi: <http://doi.acm.org/10.1145/1082473.1082502>.
- [71] M. Zavlanos and G. Pappas. Dynamic assignment in distributed motion planning with local coordination. *Robotics, IEEE Transactions on*, 24(1):232 –242, feb. 2008. ISSN 1552-3098.
- [72] M. Zavlanos and G. Pappas. Distributed connectivity control of mobile networks. *Robotics, IEEE Transactions on*, 24(6):1416 –1428, dec. 2008. ISSN 1552-3098.
- [73] M. M. Zavlanos and G. J. Pappas. Distributed connectivity control of mobile networks. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3591–3596, New Orleans, LA, USA, December 2007.