ABSTRACT

Title of dissertation: ADAPTING SWARM INTELLIGENCE FOR THE SELF-ASSEMBLY OF PRESPECIFIED ARTIFICIAL STRUCTURES

Alexander Grushin
Doctor of Philosophy, 2007

Dissertation directed by: Professor James A. Reggia
Department of Computer Science

The self-assembly problem involves designing individual behaviors that a collection of agents can follow in order to form a given target structure. An effective solution would potentially allow self-assembly to be used as an automated construction tool, for example, in dangerous or inaccessible environments. However, existing methodologies are generally limited in that they are either only capable of assembling a very limited range of simple structures, or only applicable in an idealized environment having few or no constraints on the agents' motion. The research presented here seeks to overcome these limitations by studying the self-assembly of a diverse class of non-trivial structures (building, bridge, etc.) from different-sized blocks, whose motion in a continuous, three-dimensional environment is constrained by gravity and block impenetrability. These constraints impose ordering restrictions on the self-assembly process, and necessitate the assembly and disassembly of temporary structures such as staircases. It is shown that self-assembly under these conditions can be accomplished through an integration of several techniques from the field of swarm intelligence. Specifically, this work extends and incorporates computational models of distributed construction, collective motion, and communication via local signaling. These mechanisms enable blocks to determine where to deposit themselves, to effectively move through continuous space, and to coordinate their behavior over time, while using only local information. Further, an algorithm is presented that, given a target structure, automatically generates distributed control rules that encode individual block behaviors. It is formally proved that under reasonable assumptions, these rules will lead to the emergence of correct system-level coordination that allows self-assembly to complete in spite of environmental constraints. The methodology is also verified experimentally by generating rules for a diverse set of structures, and testing them via simulations. Finally, it is shown that for some structures, the generated rules are able to parsimoniously capture the necessary behaviors. This work yields a better understanding of the complex relationship between local behaviors and global structures in non-trivial self-assembly processes, and presents a step towards their use in the real world.

ADAPTING SWARM INTELLIGENCE FOR THE SELF-ASSEMBLY OF PRESPECIFIED ARTIFICIAL STRUCTURES

by

Alexander Grushin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor James A. Reggia, Chair/Advisor
Professor Dana S. Nau
Professor G. W. Stewart
Professor Amitabh Varshney
Professor James R. Anderson, Dean's Representative

© Copyright by
Alexander Grushin
2007

Acknowledgments

Writing a dissertation can sometimes seem like an activity that is purely solitary in nature, but it is not. I owe much gratitude to the many people who, in one way or another, have helped me along the way.

In particular, I feel very fortunate that I have had a chance to work under the advisement of Professor Jim Reggia, whose dedication to his students is truly remarkable. He has provided both a great deal of freedom and a great deal of guidance in my work (not to mention the financial support for it), and I cannot thank him enough.

I am grateful to the other defense committee members for taking the time to get to know this research, for listening to me ramble about it late on a Friday afternoon, and for their valuable comments and suggestions.

A big "thank you" goes to the people with whom I have traveled some stretch of the road towards a higher degree, in particular (but by no means only), to those who shared the daily grind with me in AVW 3136. The conversations (which have ranged from artificial intelligence to what's on TV these days to the true nature of Tao), foosball games and friendship have been invaluable. Thank you for making graduate school a much more positive experience than it would have otherwise been, for all that you have taught me, and for making sure that I did not have to spend time dragging bags full of groceries from the bus stop.

Over the years, many people, both within and outside the Computer Science Department, have taken the time to talk to me about my work, and have kept me on my toes with interesting questions. Whether you made me think of my research in new ways, or simply showed that you are interested to know what I have been up to - I thank you for it.

A note of thanks is also in order for the Department's technical and administrative staff, who have put up with my various technological misadventures, and patiently answered all my questions!

Thank you to my family, who, whether they happen to be a Metro ride away, or live on the other side of the world, have always been there for me. Most importantly, I thank my parents for raising me, for encouraging me (from an early age) to take an interest in the surrounding world, and for all their support.

This research has been funded by NSF award IIS-0325098, and I give my thanks to the National Science Foundation, and to the taxpayers of this country, for keeping me fed for five years! I hope that the following pages give something back.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Self-Assembly

Self-assembly processes are prevalent in nature [122]. At various levels of scale, from the subatomic to the galactic, one can observe the seemingly spontaneous formation of structures that are far more complex than their individual components. In a nuclear fusion reaction, light atoms combine to form heavier elements. These elements may develop chemical bonds, and combine into molecules ever more complex. It appears that the building blocks of life thus emerged, as amino acids combined to form the proteins that constitute living cells. Beyond the level of the individual organism, crowds assemble on streets, and insects form *self-assemblages*, as illustrated in Figure 1.1. Meanwhile, life feeds on the energy provided by the sun, which itself formed from a cloud of gas, and produces this energy by fusing hydrogen into helium.

What drives these various complex processes, in seeming contradiction to the second law of thermodynamics? Of course, this law, which states that entropy (roughly speaking - a measure of disorder) cannot decrease in a *closed* system, remains unbroken: as self-assembling systems gain order, they “release” entropy into the outside world. Still, how such processes yield complex structures remains, in many cases, not entirely understood. It is almost as if some outside hand moves

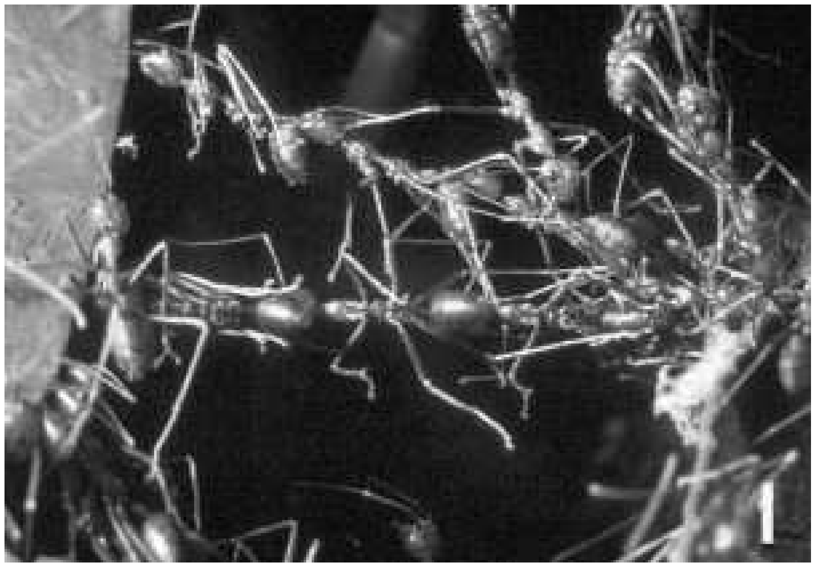


Figure 1.1: Weaver ants (*Oecophylla longinoda*) assembled into chains between leaves. From [5].

individual components into appropriate locations, but like the “invisible hand” of 18th century economist Adam Smith, the overall effect is really the result of the forces (in many cases, forces local in range) that the components exert upon each other. Self-assembly is thus a fundamental form of natural *self-organization* [4, 24].

The fundamental relationship between individual forces and the global structures that they produce has been studied over many centuries by scholars in a variety of fields. In more recent times, the advent of computer technology has provided a novel and powerful angle of attack on the problem: self-assembling systems could now be studied in simulation. In this context, a researcher has control of the individual behaviors of each component, and can vary them to observe their effects upon the global outcome. For example, computational models have been developed to better understand the self-assembly of a bacteriophage [112] and the formation of protocell structures from lipid molecules [32].

While computational approaches have been developed to better understand nature, an "inverse" question arose: can natural principles be applied to govern the self-assembly of *prespecified* structures, without any centralized, controlling entity? In other words, is it possible to endow simulated, and eventually physical components with the computational, sensing and actuation capabilities that will allow them to form a given target structure? If it were possible to do this in an effective manner, the methodology could potentially automate the production of a wide range of useful objects. This would hold particular relevance if construction is to take place in an environment that is dangerous, or altogether inaccessible for human beings. Important examples of possible applications include solar power systems in outer space [101], bases on the lunar surface [22], and various structures at the nanoscale [38, 39, 40, 87, 97].

Within this control context, the *self-assembly problem* can be defined as follows:

Given a set A of agents situated in some environment E, and a set S of acceptable spatial arrangements of these agents, design local control mechanisms that each agent $a \in A$ can follow, subject to the constraints of E, such that the entire collection of agents forms S.

Each *agent* $a \in A$ is a physically embodied or simulated component that has some degree of *autonomy*, i.e., control over its state and behavior [124]. Its embodiment is part of the environment E, which is a set of laws that govern how the agents may act and interact. For example, the environment may prevent agents from moving "through" each other, thus necessitating the use of obstacle avoidance behaviors. The *target structure* S that must be assembled is one in a set of satisfactory arrangements of these agents. Multiple arrangements may be allowed if the structure need not be built precisely. Additionally, the given framework accounts for situations where the agents are physically identical: swapping the positions of two agents will not make a difference, so long as the agents occupy all and only locations that must contain them; also, if there are excess agents available, it may not matter where they are located after the construction is completed.

An effective solution to the self-assembly problem entails finding the relationship between the local behaviors of these agents and the global, structural properties. But formalizing this relationship can be a formidable task: as the agents move through space, interacting with each other and the environment in an attempt to form the target structure, they present a complex system with behavior that is difficult to predict. Here, it is useful to step back and ask: what is the advantage of a distributed control methodology, where the operation of the system is governed by the components themselves, rather than a centralized machine, such as a robot arm that manipulates these components, or a server that receives state information from them, and then sends each component a set of instructions to be executed? Arguably, most self-assembly in nature happens in a non-centralized fashion. Like many natural processes (and unlike the operation of most software systems), distributed self-assembly can potentially exhibit properties such as *parallelism* (components act simultaneously, without being tied down by a centralized controller), *parsimony* (at the level of an individual component, behaviors are simple and elegant), *robustness* (the failure of a single controlling entity does not lead to the failure of the entire system) and *adaptability* (the system can respond to changing conditions). Nature's ability to find solutions that have such attributes has inspired the broad area of *biologically-inspired computing*, which includes paradigms such as *neural networks* [73], *evolutionary algorithms* [11, 75] and of particular relevance to this work, *swarm intelligence* [17, 53].

As elaborated further in Chapter 2, swarm intelligence studies collections of relatively simple, spatially situated agents whose behavior is inspired by that of social animals [17, 53]. In nature, there is a variety of animal species where each organism is relatively simple, in terms of its individual capabilities, but where a population of these organisms is capable of self-organizing [4, 24] in order to perform tasks of surprising complexity. For example, species of wasps exist that construct rather intricate nests [51], such as the one shown in Figure 1.2, in spite of the fact that an individual wasp is not believed to have sophisticated behaviors or a blueprint of the desired structure. Instead, when it is in the vicinity of the nest, a wasp is able to locally detect particular structural patterns, which may trigger a simple, material depositing behavior. By this action, the wasp changes the structure of the nest, possibly creating a new pattern, which may serve as a cue to other wasps; this results in a process of implicit self-coordination known as *stigmergy* [111].

While this nest construction process is not self-assembly, because there is a physical distinction between the agents (i.e., the wasps) and the material (pulp), the two types of processes are very much related [4]. (In fact, if the number of

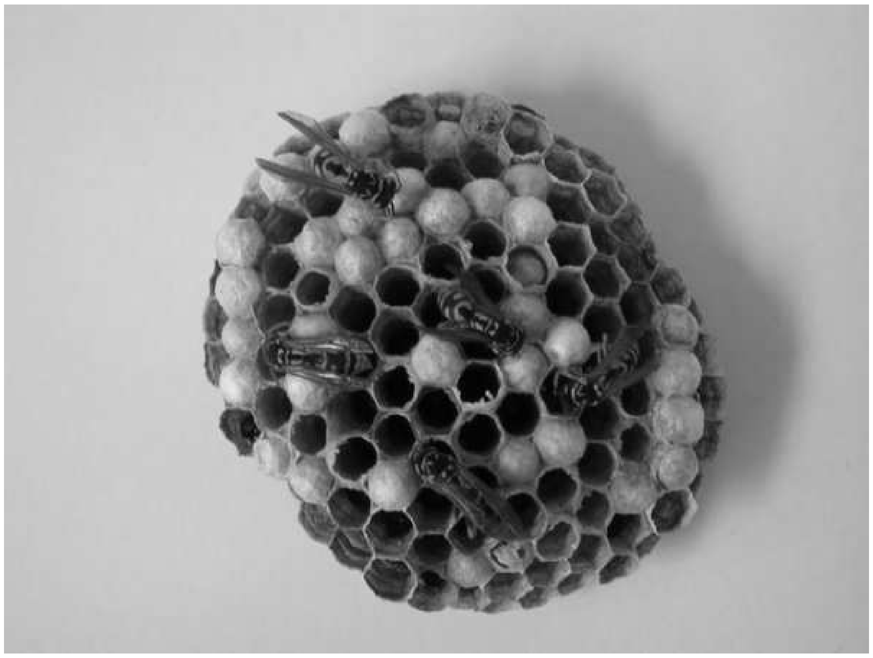


Figure 1.2: A nest of the paper wasp *Polistes dominulus*. From http://www.coopext.colostate.edu/TRA/PLANTS.

building agents is high enough such that all material components can be manipulated simultaneously, then they can be equivalent). Thus, computational models of wasp nest construction [18, 110] served as direct inspiration for a class of approaches to self-assembly, where an agent/component chooses to deposit itself at a given location (adjacent to the growing structure) based on the local conditions (i.e., the pattern) around that location. Importantly, it was shown that in order to build *arbitrary* structures, this pattern must be more than just geometric; specifically, it may be necessary to differentiate between otherwise identical components that are already part of the structure by color or by the value of some symbolic, numerical variable (which can be stored within each component and somehow made locally readable to other components) [2, 48, 49]. Methods were also developed for generating a set of patterns (encoded as rules) that enable the assembly of any connected, discrete arrangement of such components [2]. However, they were limited in that they could only apply in very simple environments with idealized conditions. Typically, such an environment is a two-dimensional or three-dimensional cellular grid, where the agents/components move randomly from cell to cell, with no constraints on their motion; thus, it is very far removed from the real world, which is continuous and heavily constrained. Furthermore, the number of generated rules was typically equal to the number of self-assembling agents; in other words, these rules were unable to *parsimoniously* capture any inherent order within a structure.

For continuous environments, other approaches have been developed. Some involve defining a set of local "forces" between the individual agents that guide them into a desired structure [35, 104]. To some extent, these studies take inspiration from another class of swarm intelligence techniques, which were originally developed to model moving groups of animals such as flocks of birds or schools of fish [91]. These models have shown that if each agent in a group tends to move in the same direction as its neighbors and to stay close to them while avoiding collisions (these tendencies can be implemented as force equations in a computational setting), then the collective tends to form a moving flock. The concept has been applied in an attempt to solve the self-assembly problem; however, the "tuning" of forces to achieve some desired structure (whose specification is typically much more precise than that of a flock) was found to be a difficult endeavor, and for all but very simple structures, the approach was inadequate, on its own.

As the next chapter will discuss, a variety of other self-assembly methods emerged, some of them conceived as hybrids of existing techniques from swarm intelligence as well as other fields. These methods have been developed for a variety of different environments, which range from aforementioned cellular spaces to the real world, where self-assembly is accomplished by physical robots [14, 80, 81, 120, 121]. However, a general trend remains: *the complexity of the environment E greatly affects the range and sophistication of structures S that can be assembled.* Oftentimes, the same methodology can produce a diverse set of structures within a theoretical context, or in an idealized simulation, but when faced with the physical world, or even with non-trivial simulated conditions, the structures become limited to very simple arrangements of just a few components.

In particular, a limitation exists in the methodologies for dealing with environmental constraints on the agents' motion. As Chapter 3 shall demonstrate, such constraints translate into ordering restrictions on the self-assembly process. For example, consider the self-assembly of a building that contains internal columns surrounded by walls. If the assembly of the walls is begun prior to the completion of the columns, it may become impossible for components (which, in any realistic scenario, are impenetrable) to finish them. The self-assembly process must therefore be conducted over a number of stages. In the work reviewed, there is a lack of well-defined and general approaches for determining what those stages are, and how to employ local behaviors in achieving the appropriate, global sequencing between them.

This dissertation seeks to overcome these limitations by achieving the self-assembly of a wide variety of prespecified structures in a continuous environment that simulates important physical constraints that one would expect to encounter in real world scenarios. In particular, the agents are embodied as different-sized blocks that are impenetrable (this is ensured via collision detection), and have very restricted vertical motion, as if in the presence of gravity. The latter constraint necessitates the assembly and subsequent disassembly of temporary structures such as staircases, at appropriate points in the process. Given such constraints, the environment considered here presents significantly greater complexity than environments where the self-assembly of a broad range of structures (albeit, structures not very realistic in appearance) has been achieved in the past [2, 48, 55, 57]. Nonetheless, I hypothesize that a large class of interesting structures resembling real world objects (e.g., building, pyramid, etc.) can indeed self-assemble under these conditions through appropriately specified local behaviors, and moreover, that this specification can be done *automatically*: in other words, given a structure, appropriate control rules (to be followed by blocks) can be generated algorithmically, without human intervention.

In testing this hypothesis, I essentially seek to understand the aforementioned correspondence between local behaviors and global structures. In a continuous, constrained environment, the relationship between the two is more complex than in environments where rules for a variety of structures have been generated automatically in past work [2, 48, 55, 57], because the individual behaviors must involve non-trivial movement capabilities and lead to the emergence of explicit, higher-level coordination over a number of process stages. Studying this relationship leads to two broad goals. From a purely scientific perspective, the first goal is to gain a better understanding of the processes that underlie self-organization in general. Such processes are fundamental to natural systems [4, 24], and a study of their dynamics may help us to obtain a better sense of our surrounding universe. From a more pragmatic point of view, the second goal is to achieve the self-assembly of arbitrary prespecified structures in the environment presented here. This will bring us closer to the development of a methodology for assembling such structures in physical environments, and ultimately, to the aforementioned possibility of applying self-assembly as a useful construction tool.

To achieve these goals, a self-assembly methodology is developed by extending and integrating existing swarm intelligence approaches in order to address their individual deficiencies. Specifically, stigmergic pattern matching (which can be generalized to a wide range of structures, but has originally been developed for discrete environments) is combined with "force"-based movement control (which is limited in the structures that it can form, but is useful for guiding agents with local sensing capabilities through continuous space). In addition, the agents' behavior is enhanced with the use of internal state variables, which can be shared between nearby agents to achieve simple but explicit communication. The use of memory and communication has recently made its way into swarm intelligence systems, allowing them to perform more complex tasks [94, 95]; within the system presented here, it gives rise to higher-level coordination in the self-assembly process, allowing it to complete in spite of the aforementioned environmental constraints, namely gravity and impenetrability.

Importantly, these various techniques are applied in a local, distributed fashion (i.e., each block governs its own behavior based on limited information), allowing the agents to retain the self-organized (rather than centrally controlled) nature of swarm intelligence systems. However, in the vein of other recent work [94, 95], the methodology is *adapted* towards the specific task at hand. The self-assembly problem, as studied in this dissertation, requires structures to be formed precisely, up to the global, block-by-block specification (which is not given to the individual blocks). Thus, while general capabilities such as pattern matching, movement control and communication remain the same between different target structures, they make use of a distinct set of distributed control rules for each structure. These rules encode the stigmergic patterns to be matched as well as the changes made to a block's internal memory in response to locally detected events, and control the system's behavior in a "bottom-up" fashion. Initially, the rules were designed by hand; however, the later chapters of this dissertation present a methodology for generating them automatically, given a specification of the desired structure. The procedure determines when the different parts of the structure must be built, and produces rules that not only yield the self-assembly of these substructures, but also enforce the overall sequencing of the process through its various stages. Importantly, for some target structures, it was possible to generate sets of rules that parsimoniously describe the required agent-level behaviors, allowing the agents to be simpler in terms of their knowledge and memory requirements. This serves as an additional improvement over past rule generation procedures [2, 48, 55, 57], where parsimony was not achieved.

The research presented in this dissertation combines both theoretical and experimental approaches. The former involve an effort to build abstract, high-level, environment-independent models of the self-assembly process. These models do not take into account details such as block geometries and the non-linear interactions between moving blocks; due to such interactions, it is generally infeasible to guarantee that the collection of blocks is *always* guaranteed to converge to the target structure. However, the models can be used to formally show that under reasonable assumptions, certain aspects of the self-assembly process (in particular, those relating to its sequencing through the various high-level stages) are correct. These assumptions are then verified experimentally by performing repeated simulations, where the methodology is applied towards the self-assembly of a diverse set of target structures, under various conditions and with different streams of random numbers. (The self-assembly of one such structure is illustrated in a video that can be downloaded at http://www.cs.umd.edu/˜reggia/grushin.html). These experiments not only demonstrated the effectiveness of developed techniques, but also yielded important insights into the complex dynamics of self-assembly.

1.2 Contributions

The five key contributions of the work presented in this dissertation are as follows:

- Three distinct swarm intelligence techniques are integrated into a unified, generalized methodology for the self-assembly of prespecified artificial structures in a continuous and constrained environment. These techniques are stigmergic pattern matching, force-based movement control, and coordination through a limited amount of memory and simple (but explicit) communication. Importantly, a combination of these techniques is able to handle a significant degree of both structural and environmental complexity.

- A procedure is designed and implemented that automatically specifies individual, local control rules for the self-assembly of a given target structure. This procedure allows the adaptation of swarm intelligence techniques described above to perform precise self-assembly in an environment that models gravity and component impenetrability. These constraints exist in a variety of real world environments, and therefore, the approach presents a step towards the goal of using self-assembly to automate practical construction.

- A formal, abstract model is developed of environmental constraints and the requirements that these constraints place upon the overall sequencing of the self-assembly process (where different parts of the structure must be built in different stages). Within the context of this model, it is proved that under reasonable assumptions, the control rules generated by the automated procedure ensure that the resulting process correctly follows these stages, allowing a target structure to self-assemble in spite of the constraints.

- It is empirically demonstrated that the developed methodology effectively achieves the self-assembly of a broad range of structures. Through repeated experiments, the dynamics of the self-assembly processes are studied in detail. In particular, observations are made of the complex interplay between the need to make blocks available in regions where they are required and to reduce interference between them, and it is shown how various aspects of block control can affect the satisfaction of these objectives.

- A stigmergic framework is created that enables some structures to self-assemble using relatively parsimonious sets of rules, allowing for simpler agents. An individual rule is specified such that it can apply at multiple locations within a target structure. By taking advantage of local and global structural patterns, the automated procedure is able to generate such rules.

1.3 Overview

The rest of the dissertation is structured as follows. The next chapter presents an introduction to the field of swarm intelligence (which has inspired much of this work), and reviews past research in self-assembly. Directed by the limitations in the current literature, Chapter 3 begins by defining a non-trivial simulated environment, along with a diverse set of target structures that will be used as test cases throughout the dissertation. It then presents an integrated methodology for controlling self-assembly within this environment, and demonstrates that a hand-generated set of control rules leads to the correct self-assembly of an interesting target structure. The self-assembly process is studied in detail, through repeated experiments under various conditions. Chapter 4 then presents a procedure that generates control rules automatically, given a target structure. It states theorems which show that certain formal aspects of the rule generation methodology are correct (the proofs are given in Appendix A), and experimentally demonstrates that this methodology is successful for all the target structures presented earlier. A limitation is that while the generated rule sets are correct, they are quite large (particularly, as compared with sets of rules designed by hand). Thus, Chapter 5 presents a modified procedure, which uses more sophisticated pattern matching to produce rule sets that are considerably smaller, but still achieve effective self-assembly. Finally, Chapter 6 discusses the results of preceding chapters, and lays out a number of directions for future research.

Chapter 2

Background

As suggested by its title, this dissertation deals with the application of swarm intelligence techniques to the control of self-assembly processes. Thus, the first section of this chapter presents an overview of swarm intelligence, with a focus on those aspects of the field that are particularly relevant to the endeavor. When it benefits the discussion, references are made to related disciplines, such as cellular automata, neural computation, evolutionary algorithms and robotics. The second section will outline existing approaches to self-assembly and certain related problems. The chapter will conclude with a discussion of limitations in the current methodologies, setting the context for the work presented here.

2.1 Swarm Intelligence and Related Disciplines

Since its first appearance in a 1989 paper on the control of cellular robots [13], the term "swarm intelligence" has been used to describe a variety of distributed (i.e., multi-agent) computational systems [17, 53], which solve problems in domains ranging from computer animation [91] to network routing [29]. The diversity suggests that beneath these applications is a powerful computational paradigm; however, it also makes this paradigm somewhat difficult to define and characterize. Before discussing its various aspects in greater detail, let us attempt to determine what sets swarm intelligence apart as an emerging field.

As the name itself suggests, the field derives inspiration from collections of social insects such as wasps, ants and termites (and, more generally, other animal groups such as flocks of birds or schools of fish). Although each member of an insect swarm is believed to have relatively simple behavior [51], groups of such insects have been known to achieve tasks of great complexity, from the computation of shortest paths by ants [41] to the construction of elaborate nests by paper wasps [17, 18, 50, 51, 110]. Presumably, such intricate system-level behavior arises as a result of appropriately-evolved local interactions between the individual organisms within a swarm. Thus, a key idea behind swarm intelligence is that complex, organized behavior should not be programmed to be executed by some monolithic entity, but rather, should arise as a result of interactions between a number of relatively simple agents. At a given point in time, an agent can observe only a limited subset of its environment, and can interact with only a few other agents. The interactions are typically simple, and in some cases, are limited to observing the results of another agent's actions, rather than explicitly communicating with said agent.

Given the local and distributed nature of the computations within an artificial swarm, what differentiates it from other systems, such as neural networks [73] or cellular automata [100], where such computations also take place? Here, I argue that in a swarm intelligence system, it is natural to view the behavior of agents (at least, in part) in terms of *motion* through a *space*; in other words, the agents operate in an inherently spatial environment. As the subsequent examples will show, the underlying space may be Euclidean or graphical, discrete or continuous, high-dimensional or low-dimensional. The dichotomy between spatial and non-spatial environments is not rigorously defined, but rather, depends on the conceptual framework that is most convenient for reasoning about a system. For example, consider a *neural network*, which is a topologically connected set of simple nodes, where each node has an activation level (which abstractly models the firing rate of a biological neuron), and influences the activation values of its neighbors through excitatory or inhibitory connections (synapses) [73]. The entire network can be viewed as a *single* agent (rather than a collection of agents) moving along some trajectory through a high-dimensional space, where each dimension is an individual activation level. Conversely, the individual nodes in the network can be thought to operate in a one-dimensional, continuous space of possible activation levels; however, this is typically not the best way of thinking about the network's dynamics.

A related (albeit typically discrete) computational device is a *cellular automaton*, which consists of cells on a grid, where each cell repeatedly updates its internal state as a function of its previous state and the states of its neighbors [100]. A well-known example is the two-dimensional Game of Life (invented by John Conway; first introduced in [36]), where each cell has has the state of being either "alive" or "dead". A dead cell becomes alive if precisely three of its eight adjacent neighbors are alive, but subsequently dies if fewer than five or more than six of its neighbors are dead. These very simple rules lead to highly complex behavior and the formation of patterns that appear to "move" across the grid. However, this "movement" is in fact an emergent property of *stationary* agents that become alive and die; for this reason, I do not apply the term "swarm intelligence" to such cellular automata systems. It is the underlying notion of spatial movement that makes swarm intelligence a particularly well-suited paradigm for the control of self-assembly processes, where individual components must obviously traverse space in order to form a structure.

In summary, I propose the following definition:

Swarm intelligence *is the study of computational systems consisting of relatively simple agents, which move and interact locally within a simulated space to achieve complex behavior and self-organization at the system level, and whose design is (typically) inspired by the ethology of social animals.*

The following will give a more detailed discussion of swarm intelligence systems that are particularly relevant to this work.

2.1.1 Stigmergy

As stated above, the agents in a swarm intelligence system (a.k.a. *swarm*) are generally simple in terms of their structure, behavior and interactions. In some cases, these agents are in fact unable to communicate directly (e.g., by sending messages), but only leave each other clues by making local environmental modifications. This form of implicit communication through the environment is known as *stigmergy* [111]. The term was originally formed as a compounding of "stigma" and "ergon" (Greek for "mark" and "work", respectively) by the zoologist Pierre-Paul Grassé in 1959 to denote the two-way relationship between the agents' actions and their environment [42]. Subsequently, it became an inspiration for models of construction and other behaviors by insects such as wasps [17, 18, 50, 51, 110], ants [28, 41] and termites [16]. Because of the inherent similarity between this *collective construction* and self-assembly [4], some of these models served as inspiration for a number of self-assembly approaches (e.g. [48]), including the one presented here.

An insect such as the paper wasp (e.g. *Polistes dominulus*) is not believed to "know" the global properties of the nest that it builds (see Figure 1.2), but rather engages in reactive, stimulus-response type behaviors, which were presumably developed over long periods of time through evolution. Here, the stimulus consists of local observations of discrete arrangements/patterns (called *stimulating configurations* in [110], and *stigmergic rules* in this dissertation) of existing nest elements, and the response, which may be probabilistic [50], is the deposition of a new element in a particular location. As wasps thus build, the structure of the nest changes: new arrangements are created, triggering construction at new locations. The process is self-coordinating in that by the results of its actions, one wasp gives others clues about what should be done in the future.

A computational model of wasp nest construction appears in [18, 110], where agents representing "wasps" move randomly in a discrete, 3D lattice, and match stigmergic rules when they are close enough to detect some part of the existing structure. Initially, this structure consists simply of one block (the word "brick" is used by the authors), which is called a *seed*. When a rule matches, an agent deposits a block (which is either grey or black) in a location that corresponds to the center of the encoded pattern, as shown in Figure 2.1a. The authors suggest that no interesting structures can be produced when all blocks are identical; this claim

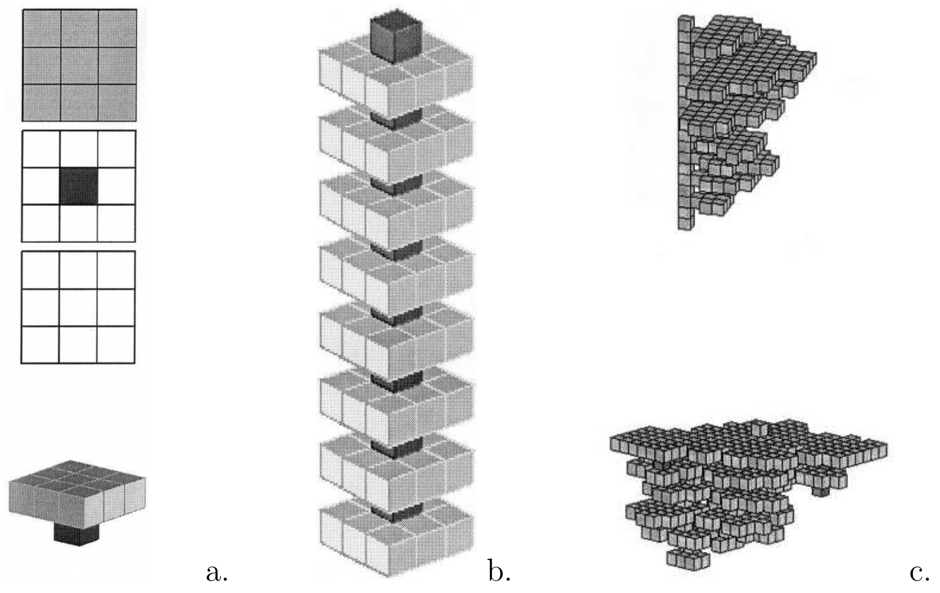


Figure 2.1: A computational model of wasp nest construction. In (a), a simple stigmergic rule is illustrated schematically (top), along with the result of its application (bottom). The rule, drawn as three horizontal "slices", states that a black block (denoted by the black square in the center) should be deposited at a location that has nine grey blocks above it (grey squares), but nothing around or below it (white squares). Combined with other rules, it results in the structure depicted in (b), which grows indefinitely, from top to bottom. In (c), two other structures (constructed via other sets of rules) are shown; these bear resemblance to wasp nests encounted in nature. From [18].

is further examined in [2]. On the other hand, when two block colors are employed, a number of interesting structures have resulted (e.g. Figure 2.1b), some of them bearing a considerable resemblance to wasp nests found in nature (e.g., Figure 2.1c).

In [110], the stigmergic rule sets were designed and fine-tuned by hand to yield these structures. However, in the later study [18], they were evolved via a *genetic algorithm* [75]. Sets of rules were represented via an encoding that could be modified via "genetic operators" such as mutation and recombination. These operators were more likely to be applied to sets with a higher "fitness", which was affected by a number of factors, such as the proportion of stigmergic rules actually used during construction, and the modularity of the resulting structure. Through a "survival of the fittest" process, stigmergic rules eventually emerged that were able to build novel and interesting structures.

A central theme of this dissertation is the relationship between local rules and global structural properties. In [110], the authors made an attempt to analyze this relationship within the context of their simple, stigmergic model, with an emphasis on outlining conditions under which emergent structures have coherent features. A key result is that coherence is typically achieved by sets of stigmergic rules that can be partitioned into non-overlapping subsets, where each subset corresponds to a disjoint stage in time, as well as a modular substructure. This requirement severely constrains the range of "interesting" structures that can be built, and therefore, the methodology of [18, 110] is primarily useful as a computational model of nest formation, rather than as a general method for constructing arbitrary prespecified structures. However, as discussed later in this chapter, their model has inspired extensions that are substantially more powerful.

It is worth noting that the focus here is primarily on *qualitative* stigmergy (i.e., where the stimulus is qualitative in nature), of the kind found among wasps. This can be contrasted with *quantitative* stigmergy [4], which is observed in the pillar-building behavior of termites [16] and trail-following behavior of ants [41] that react to pheromone concentrations. As ants collectively test various paths to a food source, they deposit pheromones along each path, and the shorter path tends to accumulate a greater concentration of these pheromones, because ants are able to return to it faster. In turn, the pheromones attract additional ants, and often allow the entire colony to converge on the shorter path, in a process of *positive feedback* [4]. This basic principle has been successfully applied to solve computational problems such as the traveling salesman problem (TSP) [30] and network routing [29].

Another application of stigmergy includes the clustering and sorting of objects on a two-dimensional surface [28], inspired by corpse clustering and brood sorting behaviors among ants. An agent is more likely to pick up an object if it is different from those that it has detected in the recent past, and conversely, will put it down sooner if it is similar to past observations. Through these indirect interactions, agents are able to separate and place different objects into separate clusters. It is worth noting that this type of stigmergy is somewhat difficult to classify as qualitative or quantitative [4], because the stimulus (i.e., the number of objects in a local region of space) is a numerical, yet discrete quantity. Subsequently, methods were developed for clustering via physical robots [12, 70, 74]. More complex stimuli are used in an approach to the modeling of city growth, where agents interact directly with the environment (but not each other) by placing roads where they appear to be needed, according to observations of local environmental characteristics, such as road density [65]. Finally, it is worth mentioning that methods somewhat analogous to stigmergy are used in synthesizing textures for computer graphics, where an image with some texture is "grown" from a seed pixel, and the choice of a pixel to be drawn at some location in the image depends on nearby pixels that have already been determined [33].

2.1.2 Collective Motion

Recall from the beginning of this chapter that swarm intelligence systems are characterized by a well-defined notion of movement through space. The ability of agents within a swarm to effectively engage in collective motion behaviors is thus critical to their proper functioning. The general nature of movement control depends on the agents' environment and the problem to be solved by the swarm. In the examples given previously (except robotic implementations [12, 70, 74]), the agents typically operate either in a discrete, cellular lattice (in the case of qualitative stigmergy and collective construction) or a (non-Euclidean) graph/network (in a number of quantitative stigmergy applications). In the latter case, the probability of moving from one node to the next is affected by the pheromone concentration on the edge between the two nodes, but in the former case, the agents typically engage in very simple wandering, by moving randomly from cell to cell. In a more realistic environment, such motion can be ineffective [15, 81], due to the continuous nature of space and the presence of various physical constraints. The following is an overview of movement control techniques for operating in various continuous environments.

In 1987, even before the term "swarm intelligence" came into use, a novel approach was presented for the computer animation of social animals that display collective movements in continuous space [91]. It was shown that without the use of scripting or a group leader, remarkably realistic flock-like behaviors can be achieved (as illustrated in Figure 2.2) through an appropriate combination of three simple behaviors:

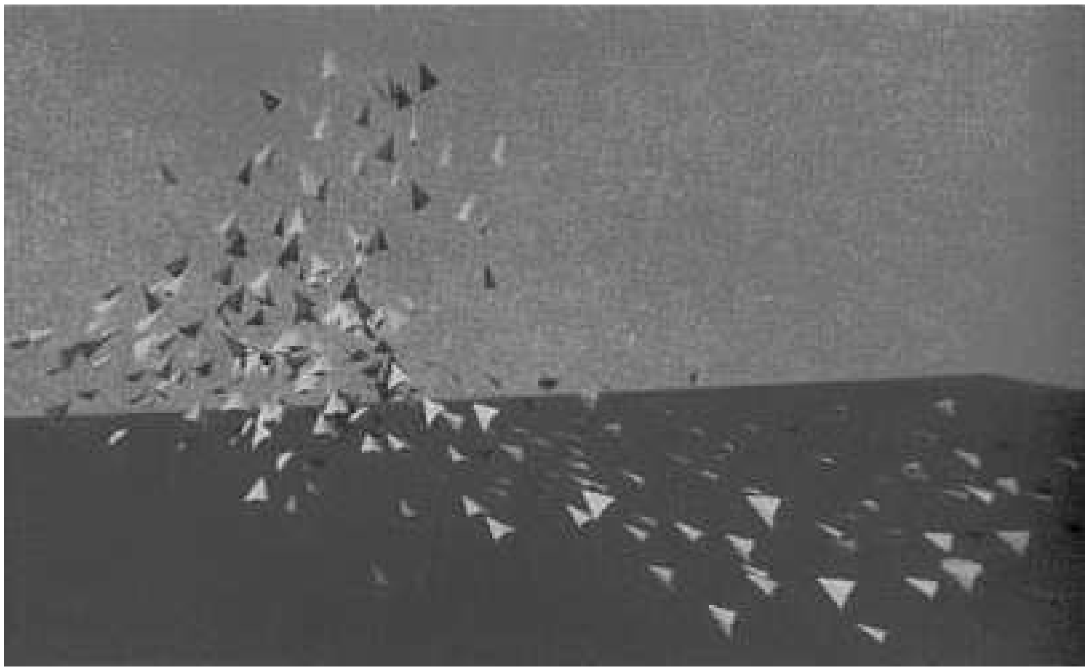


Figure 2.2: Simulated flocking behavior, emerging through simple, local interactions. From [91].

1. Direct movement towards the center of neighboring agents (cohesion).
2. Attempt to match the velocity of neighboring agents (alignment).
3. Avoid collisions with neighboring agents (avoidance).

These behaviors are based strictly on the positions and velocities of nearby agents, which are referred to as *boids* (for “bird-oid” objects), allowing for purely local and distributed control. In fact, locality is essential to the model, whereas global perception would have undesirable consequences, such as the rapid convergence of the entire flock towards its center. A theoretical treatment of flocking is given in [54], which rigorously analyzes the effects of attractive and repulsive forces between agents.

In [92], additional behaviors are outlined, such as avoiding stationary obstacles, pursuing moving targets and goal locations, and wandering/exploration. The behaviors can be implemented as local “forces”, where a particular stimulus, such as the presence of an obstacle, results in the computation of a force. These forces do not actually exist within the environment, but rather, serve as internal influences that guide an agent’s motion. It is emphasized that a linear combination of the forces can be inadequate, because there are situations where some behaviors (such as avoiding a nearby obstacle) must take precedence over others [91, 92]. Thus, *prioritized* combination approaches are proposed, where low-priority behaviors may be inhibited, or ignored completely. The net (combined) force $\mathbf{F}$ directly affects the agent’s acceleration, by Newton’s second law: $\mathbf{F} = m\mathbf{a}$.

Originally intended for use in computer animation, the boids model has been extended for solving a number of other problems. In [94], it was shown that flocking is beneficial in expediting the performance of a search-and-collect task as compared with the use of independently-moving agents. In the reported experiments, a “pull effect” was observed, where an agent that discovers a deposit of material tends to draw the rest of the flock towards this deposit. This implicit sharing of goal information has been further studied in [26], where it was shown that a large group can be led by a small number of informed individuals, even without having knowledge of who these individuals are. Another study demonstrated the application of the force-based approach to the collective transport problem, where heavy objects must be transported by multiple agents [95]. Once again, it was shown that a collective dynamic is beneficial, for example, in effectively steering an object around obstacles.

As early as 1990, [22] suggested the use of these sorts of interactive behaviors in robots that would perform construction atop the lunar surface. However, it is known that methods which work well in simulation can fail when confronted with the control of physical devices [23]. Still, the individual behaviors can be realized via other control mechanisms, such as *motor schemas* [8, 9, 10], where tendencies such as goal pursuit and obstacle avoidance are used to govern motor velocity or direction of travel, rather than acceleration. "Flocking" behaviors similar to [91] have been realized in [72] by a group of 20 physical robots. In general, robot control techniques inspired by animal behavior are covered in [9]. For combining behaviors in the physical environment, the *subsumption architecture* [21] can be quite relevant. Rather than producing a classical decomposition of the controller into functional units for world modeling, planning, execution, etc., this architecture proposes a behavioral partition, where each unit is by itself capable of a small set of behaviors. These units can be viewed as "levels of competence", where lower levels correspond to more primitive behaviors (e.g. obstacle avoidance), but with a potentially greater priority, while higher levels (e.g. exploration of the environment) can use the data and enhance the functioning of the levels below them.

While the original, simulated approach to multi-agent motion has inspired mechanisms for the control of multiple moving robots in a very concrete, physical environment, it has also been extended to high-dimensional, abstract spaces. A well-known example is the *particle swarm optimization* (PSO) model, where agents swarm within a space of decision variables, in order to optimize some continuous function of these variables [52, 53]. Each agent tends to accelerate towards the best solution (i.e., point in space representing a combination of variables) observed thus far, but is also attracted to the best solution found by agents within its neighborhood. Unlike in studies mentioned earlier, this neighborhood is not spatial (i.e., neighboring agents may in fact be occupying very different regions of the space) but topological, where each agent's set of neighbors remains fixed. Analogous to the pull effect observed in [94], this allows an agent to gradually draw other members of the swarm towards a more optimal region of the search space. The approach has been quite effective at optimizing a variety of functions. Recently, swarm intelligence has also been extended towards the high-dimensional problem of abductive diagnosis, where agents that represent manifestations (e.g. symptoms) move towards regions of space containing stationary agents that correspond to the disorders (e.g. diseases) that are likely to have caused them [63]. Possible causal links between disorders and manifestations are represented as topological connections between corresponding agents, allowing them to influence each other. While these abstract applications are not as closely related to the problem of controlling self-assembly (which takes place in a constrained, low-dimensional environment), they illustrate that the solution to a wide range of problems can be viewed in terms of collective motion through space.

2.1.3 Memory and Communication

Early swarm intelligence systems such as the boids model [91] demonstrated the emergence of complex, life-like behavior through the use of very simple interactions. However, the application of such systems towards performing concrete tasks is not trivial, if the system's low-level behaviors are to vary over time, in response to a change in the environment, or the completion of a particular subtask. For example, when a grazing herd of animals encounters a predator, it must escape to a safer location (which obviously requires a very distinct movement dynamic); subsequently, it may return to feeding. It can be argued that in order to thus coordinate their behavior over a large temporal scale, the agents should carry an internal model of the current state of the process, and to store this model, they must be endowed with *memory*.

The original boids model has been thus extended in [93], by implementing a finite state machine (FSM) within each agent. In the presented example, the agents model a flock of pigeons, and the FSM can be in one of two states: "walk" or "fly", each corresponding to a different set of low-level movement behaviors. A user can interact with the system and frighten the walking pigeons, causing one or more of them to take off the ground, and enter the "fly" state for a period of time. A pigeon also enters the "fly" state if the percentage of nearby pigeons that have recently taken off exceeds some threshold; thus, a frightening stimulus can cause a large part of the flock to lift into the air, even if it is detected (locally) by only a few of the pigeons. This example, though very simple, illustrates two important concepts, namely: (a) the state of a pigeon's internal memory can represent aspects of the global state of the world that it cannot detect via local perception (i.e., it knows the presence of danger without detecting it directly); and (b) this state can be communicated to other pigeons, even if it is done in an indirect fashion (i.e., when a pigeon takes off, others may do so as well).

Similar concepts have found application in the aforementioned study of a

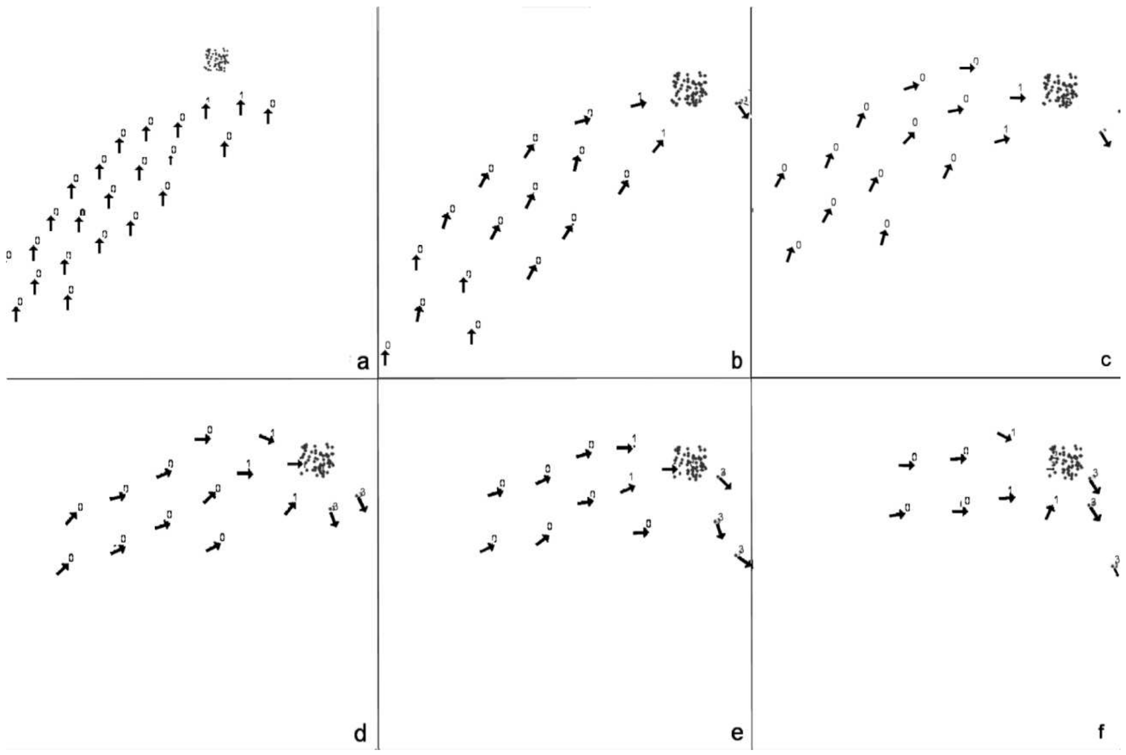


Figure 2.3: The indirect communication of state via the pull effect. Each agent is shown with its state value (“0” for “spreading” and “1” for “seeking”). The first agents to discover a deposit of material (top right hand corner of each frame) “pull” the rest of the flock with them. As subsequent agents see the material, they switch their state from “spreading” to “seeking”, which allows them to pick up the material. From [94].

search-and-collect task [94], where the FSM includes (as states) the high-level behaviors “spreading”, “seeking”, “caravan” and “guarding”. Each state is associated with a distinct combination of low-level behaviors (cohesion, alignment, etc., as discussed earlier). State changes occur in response to locally detected events: for example, when an agent detects a deposit of material, it switches from the “spreading” state (which is best for searching) to “seeking”, which allows it to pick up the material. State changes can propagate to other agents via indirect communication through movement (specifically, the aforementioned pull effect), as illustrated in Figure 2.3. In another study, it was shown that agents (which did not implement an FSM) moving through an environment were better able to avoid difficult terrain by maintaining a limited memory of environmental observations [123]. Each agent attempts to avoid the few areas of the environment that it recalls as undesirable, and through movement dynamics, influences nearby agents (whose memories may be quite different from its own) to avoid these areas as well.

Apart from this implicit sharing of information, the agents in [94] were also capable of directly reading each others' memory contents when in close proximity. This allowed an agent to determine whether or not another agent was guarding a deposit of material, or whether it was simply moving nearby. In the subsequent work on collective transport [95], further use was made of this explicit communication between agents. Specifically, when items of a product must be transported to several destinations, a more even distribution of the items amongst these destinations is achieved when agents share with each other the number of items seen at destinations encountered in the past. The use of memory sharing thus allows agents to communicate more abstract information that what can be conveyed with simple signaling via stigmergy (i.e., a modification to the environment) or movement behavior (e.g. goal pursuit), and is particularly relevant to the work of this dissertation.

While memory sharing as a model of communication has only recently made its way into swarm intelligence systems [94, 95], it is by no means new. In particular, it serves as the basis for cellular automata models, where, as discussed earlier, the change in the state of any cell depends on the states of neighboring cells. Given an appropriate (even if simple) set of rules, the cells in a cellular automaton can exhibit highly coordinated behavior, and transmit information over large distances [62] (this is illustrated by the apparently "moving" patterns in the Game of Life [36]). These ideas have found application in the control of *self-reconfigurable (metamorphic) robots* [15, 46, 60, 106], which consist of multiple modules that can rearrange themselves (provided that connectivity between modules is preserved), and where these modules can share their state with each other. Similarly, memory sharing is used within the *amorphous computing* paradigm [1, 79], which studies collections of spatially distributed, locally interacting agents, but with a much more limited range of motion than swarm intelligence systems.

On a final note, it is worth mentioning that depending on an agent's local neighborhood of communication (whether fixed or changing over time), information can spread very rapidly through a system. This is observed in social networks, where a link indicates that two people know each other directly, and the path between arbitrary pairs of individuals typically contains only a few links (this is known as the *small world phenomenon*) [82]. While communication in these networks typically occurs through directed message passing, rather than memory sharing, the latter means of communication similarly has the potential for quickly transmitting information over large distances via local means. This allows the overall system to repond promptly to locally detected events, which, as shown later in this dissertation, can be quite useful in the control of complex self-assembly processes.

2.2 Self-Assembly, Collective Construction and Related Problems

This section will discuss a variety of proposed solutions to the self-assembly problem, as defined in the previous chapter, as well as the closely related problem of collective construction (where there is a distinction between the agents and the building material). Researchers who studied these problems have used different target structures as test cases (examples are shown in Figure 2.4), and have conducted their experiments in distinct environments that range from idealized spaces to the real world. The developed control methods are therefore very diverse. In the following, an attempt will be made to classify methods according to the set of swarm intelligence techniques (as given earlier) that they most closely resemble, namely, stigmergy (generalized for arbitrary structures), force-based movement or communication of state. Subsequently, the discussion will include approaches that are either hybrid in nature, or make use of techniques that are sufficiently distinct from those mentioned previously.

2.2.1 Stigmergic Approaches

As discussed in Section 2.1.1, computational models of qualitative stigmergy have been developed to explain how complex wasp nests are constructed through primitive, local behaviors [18, 110]. While the emergent structures often exhibited complex patterns, and sometimes resembled natural wasp nests, the range of such structures was relatively narrow, in the sense that the model was not powerful enough to describe (let alone construct) most arbitrary structures. This can

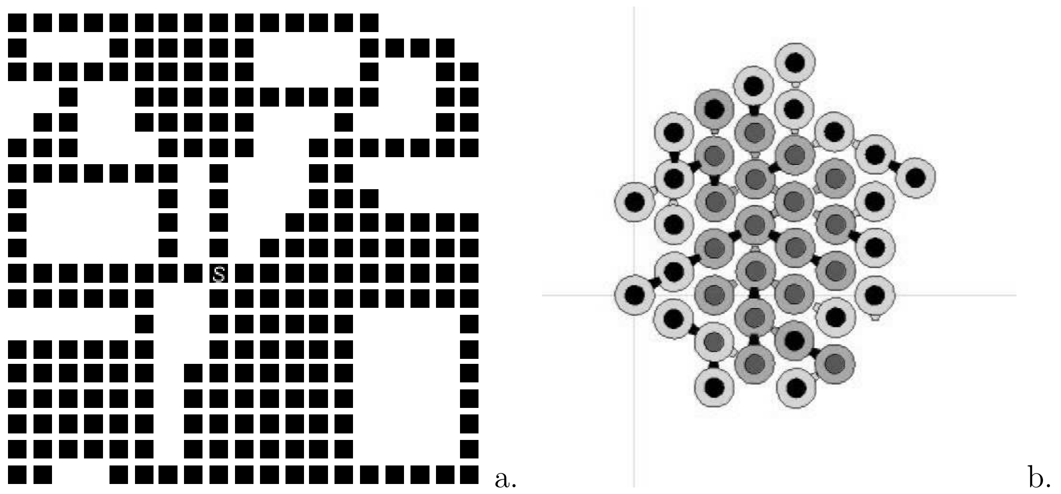


Figure 2.4: Examples of structures assembled in past work. In (a), a somewhat arbitrary structure self-assembled from square agents in a discrete environment; the seed agent (labeled "S") is in the center. In (b), a cluster of robots formed in a simulated, but somewhat more realistic environment, with some distinction between the robots at the periphery vs. the center. From [48] (a) and [99] (b).

be shown formally with a simple counting argument: there is a finite number of stigmergic rules (such as the one shown in Figure 2.1a), because each location in a rule is either empty, or specifies a block of one of two colors. Thus, the number of possible rule sets is also finite, even if very large, whereas the number of possible structures is infinite. During the construction process, the pattern surrounding a desirable location (where a block should deposit itself) may be identical to a pattern that surrounds an undesirable one; a rule that fills the former location may fill the latter one as well, resulting in an incorrectly built structure. To extend stigmergy to arbitrary, *prespecified* structures, either the agents or the material components (which are one and the same in self-assembly) must be enhanced with additional information, typically encoded in the form of *state variables* (while still stigmergic, the resulting model thus incorporates certain features discussed in Section 2.1.3).

To my knowledge, the first such enhancement has been described only a few years ago [48]. The agents, embodied as squares, move randomly on a 2D lattice, and assemble into structures such as the one shown in Figure 2.4a. While the term “stigmergy” is not used by the authors, the process is inherently stigmergic, as an agent decides to deposit itself at some location if the pattern around it is matched by a pre-stored rule. However, this rule specifies not only the presence of (already stationary) agents adjacent to this location, but also, the values of integer state variables within the memories of these agents. As in the work presented in this dissertation, agents do not have prespecified positions within the structure; thus, the rule also specifies the state value that the matching agent sets when it deposits itself. During the self-assembly process (which begins with a seed agent as in [18, 110], with state value 0), the presence of state variables thus allows agents to differentiate between otherwise identical structural patterns.

This study further presents a simple rule compiler, which automatically generates rules, given a structure [48]. Each location in the structure (except for the one occupied by the seed) is assigned a separate rule, which assumes that certain nearby locations have already been filled. This places an implicit, local ordering constraint on the deposition of agents; through such constraints, the generated rules ensure that the order of self-assembly does not result in the development of internal holes, which cannot later be filled (if the agents are assumed to be impenetrable). However, because of this restriction, there exist certain structures that are not accepted by the compiler.

In the following year (2005), an entire Ph.D. dissertation was devoted to this generalized model of stigmergy [2], where agents build structures from multi-colored blocks. It was shown that rules can be automatically generated to describe any discrete, connected structure, provided that there is a supply of blocks with an arbitrary number of colors (essentially, colors encode variable values). As in [48], the number of generated rules is generally the same as the number of deposited blocks, although the number of necessary colors has been optimized somewhat via a genetic algorithm [75]. Assuming an idealized environment, where there are no physical constraints on the agents' motion (such as the impenetrability of blocks), the rules can be used to construct the given structure. By contrast, the methods presented in this dissertation can achieve self-assembly in a constrained environment.

Although it is quite different from the above approaches, it is worth mentioning a system where quantitative, rather than qualitative stigmergy was employed [71]. A swarm of brick-laying agents builds chain-like and arc-like shapes in a 2D environment. The agents' actions are guided by simulated pheromone concentrations, and in turn, can affect these concentrations through further depositions of pheromones. Although [71] did not directly inspire the research of this dissertation, it helps to illustrate the variety of stigmergic approaches that can be used in controlling self-assembly and collective construction.

2.2.2 Force-Based Approaches

An early attempt to control self-assembly was presented in a 1992 paper, describing a system where *potential methods* are used to guide a set of spheres into simple, 2D arrangements, such as a 15-sphere equilateral triangle [119]. A *potential* *function* defines a scalar field over the space of all possible sphere arrangements (this is called a *configuration space* [64]), and each sphere moves along the negative gradient in this field, with an attractive tendency towards its goal location, and a repulsive tendency to avoid collisions. This early study presented an important step away from the traditional manufacturing paradigm, where all components are manipulated by a centralized entity; however, it was limited in that the agents require access to to the global state of the environment and have prespecified positions within the structure.

Some later approaches overcame these limitations by defining movement forces locally, as in the boids model [91], without a explicitly computing a potential field. For example, in [35], these forces are modeled as virtual "springs" between memoriless agents. When two agents are within a radius of visibility, a spring can "appear" between them, with a certain probability. The spring impacts the motion of the agents on both ends, and different agents may have a different number of springs attached. Simples structures such as triangles, ladders and hexagons have been assembled, although the tuning of parameters such as spring lengths and constants can be difficult. A different study defined forces somewhat akin to gravity (but with a localized range, and repulsive at short distances) to guide agents (called "particles") into hexagonal and square lattices [104]. Each particle may have a binary "spin" property, such that particles of like spin have stronger repulsion between them; it was discovered that randomly "flipping" spins under appropriate conditions helped to repair flaws in the lattice.

2.2.3 Communication-Based Approaches

A different approach to self-assembly was taken in [6, 7] where FSM-controlled agents engage in random (rather than "force"-guided) movement and communicate explicitly by sending messages with hop counts. For example, a row can be built by propagating "build a row" messages across the existing structure as agents attach themselves, and terminating the assembly once the hop count on the messages reaches a certain threshold, which corresponds to the desired length [6]. (This model of communication is thus somewhat different from the memory sharing discussed in Section 2.1.3; however, memory sharing could, in principle, be used to convey information in a similar manner). The authors extend their approach towards the simple repair of structures, by giving agents the ability to fill cavities within a structure [7]. The assembled structures are fairly simple shapes, such as 2D squares. However, in some cases, these structures are not specified in detail, but rather must adhere to certain higher-level requirements; for example, "roads" are built to maintain connectivity between two regions, without prespecifying their exact path [6]. These roads are constructed by extending rows of assembling agents in various directions, until the regions are connected. Subsequently, the unnecessary rows (called *sacrificial components* in [6] and *temporary substructures* in this dissertation) disassemble. The use of a FSM, along with message passing, allows for this sort of higher-level sequencing through time, as discussed in Section 2.1.3.

2.2.4 Other Approaches

Over the years, a variety of other approaches have been developed for solving certain variants of the self-assembly and collective construction problems. Most of these make use of a combination of techniques, some of which include those mentioned earlier. For example, the authors of [48] have further developed an approach to collective construction, where agents deposit multi-colored blocks to build a simple structure [49]. Here, color information is used in the specification of patterns (as in [2]); additionally, the agents themselves now store a state variable. A rule applies if a particular stigmergic pattern is observed *and* if the matching agent is in the appropriate state; the consequent of the rule specifies a state change and/or a block deposition. Although state is not communicated between agents within the simple context of the problem, it allows for greater control over the sequencing of the construction process.

In another study, stigmergic rules are purely geometric in nature (as in [18, 110], but they also allow blocks to be placed at non-integral positions relative to each other), but an attempt is made to *approximately* build prespecified structures in continuous space [113]. This is done by evolving rule sets along with movement dynamics, which are inspired by the boids model [91]. As the fitness function depends on how accurately a target structure is built, the agents evolve to quickly move away from the structure once it has been roughly approximated, to avoid placing blocks at locations where they do not belong. This stands in contrast to the other stigmergic approaches discussed earlier (as well as the approach presented in this dissertation), where the system *does not* rely on movement dynamics to prevent the incorrect placement of blocks.

For more complex environments, another set of methods for collective construction has been recently developed [116, 117, 118]. These methods are classified as stigmergic by the authors, since an agent determines whether or not a block should be attached at a particular location within the growing structure by the local conditions around it; however, each agent or each block stores a *global* map of the entire target structure, rather than a set of local rules. Assembled structures are two-dimensional; agents generally traverse the perimeter of the structure as it is built, and attach blocks in a way that will prevent the development of internal holes. However, unlike in [48], it is assumed that such holes are never desirable; thus, a structure such as the one in Figure 2.4a cannot be built. Four different methods for determining block placement have been developed and compared. According to the simplest method, agents determine their location by measuring distances as they move around the structure; this is done by counting the number of stationary blocks passed on the way from the seed block (it is ensured that this block is at the edge of the structure), which serves as a point of reference. The internally stored map of the structure can then be used to determine whether or not a block should be placed at a particular location. Another method makes use of static (but distinct) block *labels*, where agents store information regarding the labels of blocks deposited in particular locations. This increases the efficiency of the process, because an agent is no longer required to find the seed block prior to determining its position; rather, it can measure distances from the nearest known label. In the third method, the labels are writable (much like the variables used in [48]), and allow a block to store its position within the structure. Finally, a distinct method is presented, where the task of determining whether or not a block should be deposited is shifted almost entirely from robots to blocks that are already part of the structure. A robot simply queries the blocks surrounding a potential goal location, which determine whether the location should be filled, possibly communicating with other blocks in the process. This last method was found to be most efficient (via experiments [117] and through analysis [118]), both in terms of the overall distance that must be traveled by robots, and in terms of the number of locations that can (potentially) be filled simultaneously. However, the first three methods are easier to implement via physical devices; in fact, a prototype system is presented in [117], where a single robot builds simple structures consisting of a few blocks.

Systems of multiple, self-assembling physical robots are presented in [120, 121]; these robots are moved by external, environmental forces, and thus do not require independent actuators. In [120], three robots move on an oscillating air table. Computer simulations were also performed with a larger number of square-shaped components. While the free-floating components are largely passive, components that are already part of the assembling structure can attract them by activating "bonding sites" on their edges. Stationary components are also able to exchange state information, in order to determine whether or not to activate a bond. In a subsequent study [121], the methodology has been extended to 3D structures, and (once again) tested both in simulation and via physical robots. Stationary block-shaped modules are situated in a fluid (thus, the effects of gravity are negligible), and attract free-floating modules via short-range magnetic forces or by pumping the surrounding fluid; modules can also be released. When a module is attached, it is communicated a piece of state information by the neighboring modules; based on this information (which allows a module to identify its role in the structure), it performs specific actions, such as attempting to capture another module. Furthermore, communication between modules allows the process to occur over a number of disjoint stages, where no modules are attached (or released) in a given stage until the previous stage has been completed. The approach potentially allows the formation of a wide range of 3D shapes, although only relatively simple structures (for example, consisting of 13 modules in simulation and two to four modules in physical experiments) have been presented thus far.

Another class of approaches is based on *graph grammars* [14, 55, 57, 58, 80]. Abstractly, a graph grammar consists of a set of rules, where each rule specifies how a collection of labeled nodes can change by developing new edges, breaking old edges, or modifying their labels [57]. In the context of self-assembly, the nodes, labels and edges correspond to agents, states and topological connections, respectively. Each agent stores a representation of the grammar, and attempts to apply rules based on local observations. To "connect", two agents must simply maintain close proximity to each other, and need not come into physical contact, which simplifies the problem substantially. Structures are, in fact, free to move through space, so long as the proximity constraints are preserved. The individual agents can govern their movement through attractive and repulsive forces [55], as in some of the aforementioned systems [35, 104, 119]; however, they can also engage in more random motion [14, 57, 80], as if situated in a stirred fluid (e.g., [120, 121]). In [55], a method is given for generating a set of rules that allows disk-shaped components to form various tree-like structures in a 2D continuous space, although the approach has somewhat high memory and preprocessing requirements. A theoretical extension to arbitrary graphs (rather than only trees) is proposed in [57], but it is shown that this requires non-trivial communication schemes, where each part must have a *unique* identifier (in addition to modifiable state). Further theoretical treatment of graph grammar behaviors is given in [58].

Notably, the graph grammar approach is often applied to the assembly of *multiple* topologically identical structures from a large number of agents, much like a chemical reaction tends to form a number of identical molecules. In [80], a methodology is presented for approximating the rate at which a particular grammar will yield these structures. In addition to simulations, this paper presents experiments with ten physical devices known as *programmable parts* (discussed further in [14]). An individual programmable part is triangular, and is represented by three nodes/agents in a graph grammar rule, each corresponding to one of its three latches. The approach succeeded in forming very simple structures from these devices, such as a chain of four parts [80] or a hexagon of six parts [14]. As in [120], the self-assembly takes place upon an air table, but the otherwise static parts are moved by air currents generated by fans, rather than by the table's vibrations.

The vast majority of work in self-assembly involves components that are identical and interchangeable. Some degree of heterogeneity is found in [99], where a set of simulated robots (known as *s-bots* individually and as the *swarm-bot* collectively;

the concept is further described in [78]) can assemble, via simple attractive and repulsive behaviors, into clusters where the robots at the periphery are somewhat distinct from those at the center, as shown in Figure 2.4b. In a more complex collective construction system mentioned earlier [116], simulated robots build structures from blocks of different types. Notably, the notion of *type* used here is somewhat different from the notion of *color* in [2]: whereas a given location within the target structure can, in principle, be filled with a block of any color (colors are merely used as qualitative variable values when defining stigmergic rules), the system of [116] places constraints on the type of a block necessary at some location. These constraints may be somewhat abstract, such as "two blocks of a given type must not be adjacent". While the system is unable to handle all such constraints, it has successfully assembled a number of structures with interesting type patterns, by determining whether or not any given constraint is satisfied locally.

A few additional studies should be mentioned. Of historical relevance is a 1990 paper, where simulated robots are able to form circles or simple polygons in the plane by engaging in simple motion behaviors relative to other robots [107]. This approach was not fully automated, in that it required some robots (e.g., those representing the vertices of a polygon) to be moved manually; furthermore, each robot made use of global information, such as the relative position of the farthest robot. A somewhat similar direction is pursued in [115], where simulated robots follow behaviors (encoded via an FSM) to arrange material blocks into outlines of 2D shapes, where multiple corners are connected with straight or curved lines. The system is fully automated, but is limited in that the robots rely on a globally detected beacon, which serves as a reference point for each robot; furthermore, the robots are assumed to be capable of measuring distances. The work of [44] presents a system of self-assembling squares, where each of a square's four sides has a positive, negative or neutral polarity, which affects whether or not two squares can become adjacent. This system bears some resemblance to the aforementioned work of [120], but is much simpler, because the squares do not communicate with each other. However, the polarity can change in response to local events (an implementation of a somewhat similar device is proposed in [56]), according to a set of rules; the possible number of such rule sets is finite (as in [18, 110]), and therefore, the approach cannot be used to assemble arbitrary structures. Still, the rule space has been explored via a genetic algorithm, and some interesting structures were produced. In [114], a 1D barrier is constructed in a 2D environment by a single physical robot, or a team of simulated robots. Each robot is capable of simple, reactive behaviors such as wandering, picking up or depositing a block, etc. Finally, [81] discusses the development of a system of self-assembling, helium-filled blimps for artistic and research purposes. In the coming years, we can almost certainly expect to see an even greater diversity of approaches for the control of self-assembly and collective construction in various situations.

2.2.5 Related Problems

As mentioned in the previous chapter, the goal of earlier research in self-assembly and collective construction was oftentimes not to control the production of prespecified structures, but rather, to model processes that occur in nature. Examples of computational models of collective construction among wasps [18, 110] and termites [16] have been given earlier, in Section 2.1.1. Chapter 1 also made brief mention of a study where the self-assembly of a bacteriophage was modeled via *movable finite automata* (MFA) [112], an extension of the cellular automata model [100] that incorporates biological functions such as movement. Another relevant example is a model of protocell structure formation, where the simulated particles have hydrophilic heads and hydrophobic tails, and their self-assembly is driven by simulated inter-particle forces [32], somewhat similar to the later work by [35, 104].

In an effort to make advances within the field of nanotechnology [31], extensive research is being conducted in physical and molecular self-assembly of small-scale objects [19, 38, 39, 40, 87, 97]. At present, the computational capabilities of individual components in such systems are somewhat limited; however, approaches are proposed for the development of nanoscale processors that will allow these components to autonomously control their own behavior [31, 38, 40]. Conversely, it is interesting that self-assembly can in fact be used to used to perform universal computation, as investigated in [96] and demonstrated in [97], where self-assembly in two dimensions simulates the operation of a one-dimensional cellular automaton (with one of the spatial dimensions representing time).

Related to self-assembly and collective construction is the process of *self-replication*, where a structure generates a copy of itself; the relationship between self-assembly and self-replication is discussed in [103]. The latter has been studied within the context of computer simulations in [86], where cellular automata rules are evolved to generate self-replicating patterns, and in [103], where self-replication takes place in a simulated continuous space. It was also implemented in simple physical systems of both passive (i.e., moved by forces in the environment) [20, 43] and active (moved via actuators) [125] components.

Also relevant are various studies of pattern formation in agent swarms [3, 102]; while the emerging patterns are not specified precisely a priori, they are oftentimes quite intricate. On the other hand, in the aforementioned field of *amorphous computing* [1, 79], there is (in some cases) a desired global pattern. Specifically, in a paper by [79] (who, in fact, uses the term "self-assembly"), a 2D "sheet" of agents is able to fold itself into a variety of shapes through local interactions. The desired shape is specified imperatively by the user, as a series of global folding actions necessary to produce it; subsequently, a rule compiler generates the local actions necessary on the part of individual agents. While the approach is related to the work of this dissertation, the underlying dynamics are quite different, because the agents within a sheet always remain connected to each other. On the other hand, in the self-assembly systems discussed earlier, individual components are initially separate, even if they must eventually form and maintain physical contact (although in some systems such as [35, 55, 104], they must simply stay in close proximity to each other).

Within the context of robotic systems (whether real or simulated), it is thus suggested by [99] that self-assembly falls between the related areas of *self-reconfigurable robotics*, where (as discussed in Section 2.1.3) multiple modules must typically maintain connectivity at all times [15, 46, 60, 106], and *collective robotics*, which is an umbrella term that refers to systems of multiple robots that may interact with each other, but that generally do not come into prolonged physical contact. (Of course, this terminology is not consistent between all researchers; for example, [120] uses the term "self-reconfigurable" to refer to self-assembling structures). Collective robotics has been studied in the context of a variety of problems. For example, the problem of *formation control* requires multiple robots to move in an organized pattern [10, 88, 89]; this pattern can be thought of as a moving structure, although its specification is generally not nearly as strict as the specification of structures for self-assembly problems. Further, physical robots have been used to perform the clustering of objects [12, 70, 74], which can be viewed as a relaxed form of collective construction. These various examples demonstrate that self-assembly is not an isolated problem of study, but one that bears much relation to research in a variety of other areas.

2.3 Discussion

This chapter began by defining swarm intelligence as a growing field of study, and discussing three classes of techniques from this field that are of particular inspiration to the work of this dissertation. Since the underlying goal here is that of controlling self-assembly, the next section gave an overview of past solutions to this problem, as well as the related problem of collective construction. In the presentation, parallels were drawn between existing control methodologies and the swarm intelligence models that inspired them, either directly or indirectly. Self-assembly was also placed in a wider context by discussing related problems, which differ either in the underlying research goal (e.g., building a prespecified structure vs. modeling a natural process) or the nature of the task (e.g., self-assembly vs. self-replication).

An analysis of the reviewed literature on self-assembly and related problems revealed a very important trend: the difficulty of controlling self-assembly is strongly correlated with the complexity of the environment in which it takes place. Given idealized environmental conditions, where the assembling agents can move in an unconstrained and primarily random fashion, some of the existing approaches are able to achieve the self-assembly of a wide range of structures, such as the one shown in Figure 2.4a. Most notably, the stigmergic models of wasp nest construction have been enhanced with qualitative variables (e.g., "colors") to assemble blocks into any discrete, connected arrangement [2]. Even so, the resulting structures would typically consist of geometrically identical components, and bear little resemblance to real world objects. In more complex environments, the diversity of assembled structures is far more restricted. In a continuous (even if generally unobstructed) space, if self-assembly is driven primarily by inter-agent forces [35, 104, 119], then these structures are limited to very simple shapes (such as triangles or hexagons), which may be repeated in a lattice. More sophisticated methodologies are able to extend the range and complexity of structures, but once physical constraints are introduced into the system, these once again become limited to simple formations (e.g. Figure 2.4b). For example, while graph grammars can be specified for a wide variety of tree-like structures [55, 57], their application to robotic self-assembly has thus far yielded structures of only a few components [14, 80]. Thus, a key limitation of the existing approaches is their inability to tackle structural and environmental complexity at the same time.

This limitation is particularly pronounced when an attempt is made to generate agent-level behaviors automatically, given a desired structure to be assembled. Once again, the automatic specification of these behaviors has typically been accomplished only for very simple environments [2, 48, 55, 57] (there have also been some attempts to use evolutionary computation for the endeavor [44, 113]). The underlying assumption of these methods is that the generated, low-level behaviors can be applied fairly indiscriminately over the course of the self-assembly process. (A partial exception is found in [49], but only the construction of a very simple structure in a predefined, block-by-block sequence is demonstrated; relevant work on rule generation in other problem domains includes [79] for amorphous computing and [106] for self-reconfigurable robotics). Conversely, systems exist where the process can be explicitly sequenced through a number of stages, with different low-level behaviors [6, 121]. However, it remains up to the human designer to determine what these stages are, and to hand-design the coordination schemes for ensuring that one stage is not begun until the previous one has been completed.

The automated design of individual control rules for achieving complex, coordinated behavior remains a difficult, and largely unsolved problem not only within the context of self-assembly, but in swarm intelligence systems at large. The problem becomes yet more difficult if the agent-level behaviors are to be encoded parsimoniously. In the stigmergic rule generation procedures, the number of rules is equal to the number of deposited components [2, 48]; for graph grammars, the representation complexity can be even greater [55, 57]. The procedures are thus unable to take advantage of the order that may be inherent within a given target structure.

Taking these various limitations as a starting point, the following chapters will take the self-assembly problem along a number of interesting research directions.

Chapter 3

Controlling Self-Assembly through Swarm Intelligence

As discussed in the previous chapter, the problem of controlling self-assembly processes has received considerable attention in recent years. A number of interesting methodologies have been developed; some of them are quite general, in that they can (in theory) assemble a very wide range of target structures, whereas others are much more specialized. However, it is apparent that no past approach is capable of building a diverse range of complex structures if faced with a non-idealized environment that presents a continuous, constrained space. There appears to be a fundamental tradeoff between structural and environmental complexity.

Given these limitations, I set out to design a distributed control methodology that will succeed at assembling a substantial variety of interesting target structures in a non-trivial environment. Although it is simulated, this environment makes the endeavor quite challenging, having a 3D continuous space with a number of constraints on the motion of the agents, which are embodied as different-sized blocks. From an initially random distribution on a 2D surface, these blocks must collectively assemble themselves into various 3D structures while subject to a gravity-like restriction on vertical movement (a block can climb onto/off another block by coming into contact with it, but cannot scale a stack of several blocks), collision detection (which ensures that blocks cannot simply "pass through" each other), and other constraints, such as limitations on the velocity and acceleration. The fundamental problem addressed here is how to design distributed control mechanisms based on solely local interactions that will effectively accomplish the self-assembly of structures in such an environment. The goal is to better understand the underlying principles and algorithms that this entails, and not to implement a veridical model of physical robotic construction.

The approach given here modifies the stigmergic models described in [2, 48, 49] to allow prespecified structures to be built with a parsimonious set of rules, where possible. I hypothesize that these stigmergic mechanisms would be able to provide an effective solution to the problem if they are integrated with continuous movement control techniques [35, 104, 119] and memory-based temporal coordination [6, 7]. In this chapter, the hypothesis is tested by applying the developed methodology to the self-assembly of a non-trivial, 3D "building" having internal columns, a door and a roof, and requiring temporary staircases (such that a block is able to move vertically, one "step" at a time). The self-assembly of this structure is studied in some detail, investigating the dynamics of the system both qualitatively and quantitatively. In particular, it is shown that the use of a stochastic component in determining acceleration can benefit the process by mitigating situations of persistent interference between blocks, and that flock-like behaviors [91] can improve efficiency by increasing the availability of blocks in the regions where they are needed.

3.1 The Problem

The generalized self-assembly problem given in Chapter 1 is too abstract to guide a specific research endeavor; thus, we begin by defining a more concrete version. To this end, it is necessary to specify the target structures S that will self-assemble, as well as the environment E where the self-assembly must take place.

3.1.1 Target Structures

The basic components of the self-assembly processes studied here are small ($1 \times 1 \times 1$ units), medium ($2 \times 1 \times 1$) and large ($4 \times 1 \times 1$) blocks, which can be conceptually partitioned into 1, 2 or 4 cubic *sections*, respectively. For simplicity, it is assumed that the structures that self-assemble from these blocks are discrete (the external face of any cubic section of any block either completely overlaps some other such face of another block, or overlaps nothing), connected (a path can be traced between any two blocks through face adjacencies) and stable (each block is supported underneath by either the ground or at least one other block). Under this definition, if a structure is connected, and at least one block is touching the ground, then it is considered stable. Figure 3.1 shows six example target structures that are used as test cases for the methodology developed in this dissertation (although this chapter will focus specifically on just the third structure, to illustrate the concepts in detail). These structures are:

1. A *barrier* is constructed primarily from medium blocks, as shown in Figure 3.1a,b. Each of the three holes in the front wall (which is three layers in

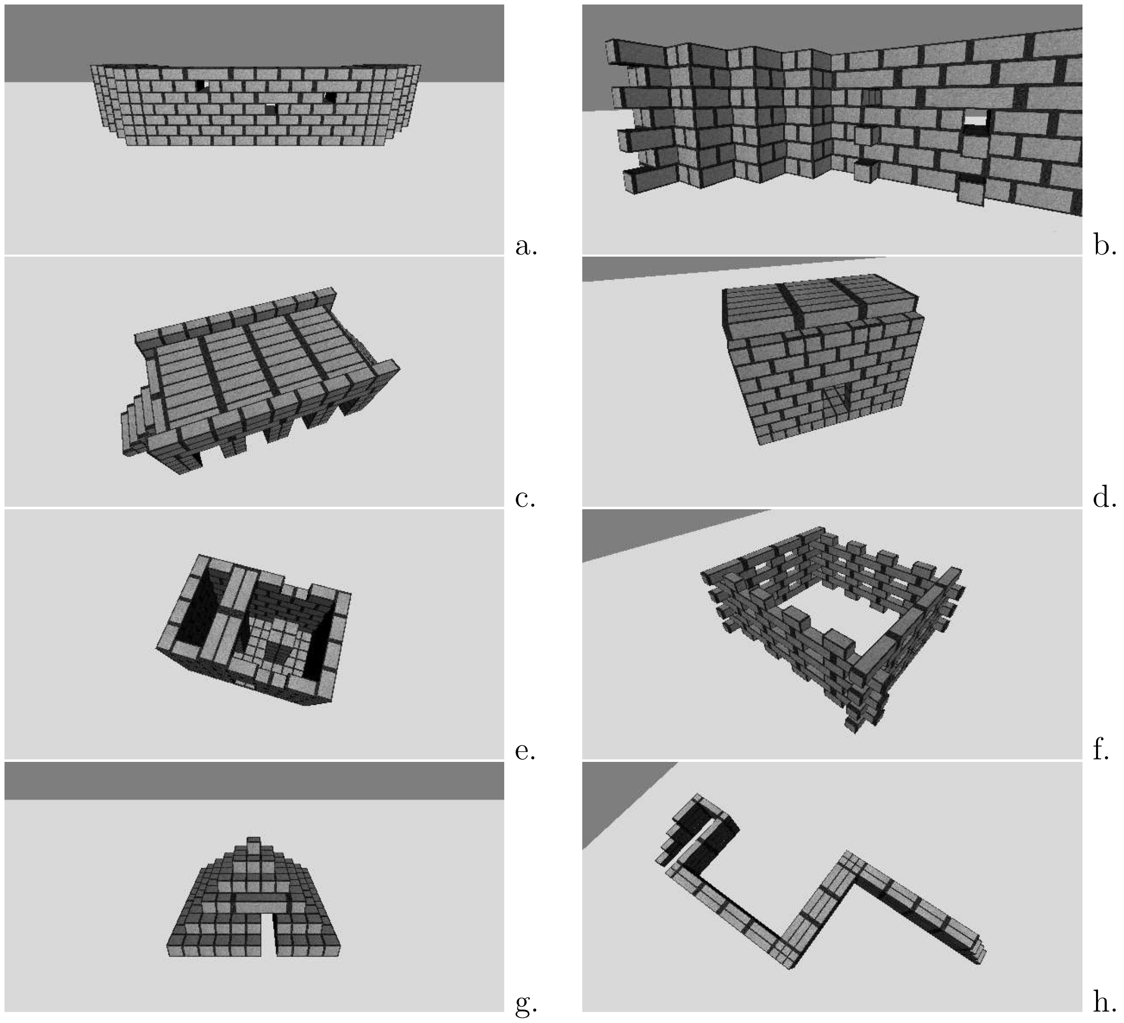


Figure 3.1: Example target structures: a barrier, shown (a) from the outside and (b) from the inside; (c) a bridge; a building, shown (d) whole and (e) with most of the roof removed, to reveal one of the two inner supporting columns; (f) a fence; (g) a pyramid; and (h) a road.

thickness) is made by using (for each layer of the wall) a small block instead of a medium block, and covering it above with a large block. A set of protuberances leads to each hole, formed by medium blocks. The side walls of the barrier are formed by a zigzag pattern of medium blocks. Additional small blocks fill in the "joints" between the front wall and the side walls.

2. A *bridge* is supported by 15 columns that stand directly on the ground, and

consist of small blocks (Figure 3.1c). Its surface is composed of large blocks, and has "guard rails" assembled from medium blocks. Large blocks form permanent staircases that allow access to both ends of the bridge.

3. A *building* has two small block columns standing upon a small block floor, walls formed from medium blocks laid out in a "brick-like" pattern, and a roof consisting of large blocks (Figure 3.1d,e). There is a door in the front wall of the building, its frame formed from two medium blocks and one small block on each side, as well as a single large block above.

4. A *fence* consists of medium and large blocks, with the latter forming interlocking "beams", and the former in place for support (Figure 3.1f).

5. A *pyramid* is formed out of progressively decreasing square layers of small blocks, and has an internal passage (of height 2 and depth 5), covered with large blocks (Figure 3.1g).

6. A *road* is elevated to a height of 4 units, and contains a series of segments of varying lengths and widths, formed from large blocks (Figure 3.1h). The "joints" between the segments are built from small blocks. At either end of the road, medium blocks are arranged as permanent staircases.

Structures self-assemble from an initially unstructured arrangement, where blocks are scattered on the ground with random positions and orientations. The only initially present structure is a *seed*, which consists of four stationary blocks located at the center of the world (in subsequent chapters, the seed will consist of just one block). This seed “grows” into the structure via blocks arriving and coming to rest adjacent to blocks that are already stationary. Individual blocks have no prespecified locations within a structure, and one block can be used in place of any other block of the same size.

3.1.2 Environmental Features

The self-assembly process takes place in a simulated, continuous environment with a number of constraints on the blocks' movements. This environment is not designed to be an accurate representation of reality, and it is not intended that the developed control mechanisms be directly applicable in the physical world without significant extension; such an extension is almost never trivial [23]. Rather, by incorporating simplified models of certain physical phenomena, an attempt is made to uncover fundamental (i.e., independent of how the models are implemented) issues that will need to be addressed when the self-assembly of complex objects is attempted in the real world.

For concreteness, the agents are embodied as blocks of three different sizes and operate in a world with dimensions $200 \times 100 \times 20$. At any point in time, each block is marked as either *stationary* (it is part of the emerging structure and thus immobile) or *non-stationary* (it is free to move). Non-stationary blocks move either on the ground (at height 0), or atop a horizontal surface that consists of one or more stationary blocks. A block's acceleration $\mathbf{a}^t$ at time step t affects its velocity $\mathbf{v}^t$ and position $\mathbf{p}^t$, which are updated via Euler step integration: $\mathbf{v}^t = \mathbf{v}^{t-1} + \epsilon_t \mathbf{a}^t$

and $\mathbf{p}^t = \mathbf{p}^{t-1} + \epsilon_t \mathbf{v}^t$, where $\epsilon_t = 0.1$ is the duration of the time step. Prior to these computations, the acceleration vector is truncated, if necessary, such that its magnitude does not exceed $a_{max} = 10.0$, and $\|\mathbf{v}^t\| \leq v_{max} = 4.9$ is also enforced. Somewhat like a wheeled vehicle, each block is always oriented (with a real-valued orientation ω^t) such that $\mathbf{v}^t$ is orthogonal to its *leading side*, which is the forward side relative to the block's motion. However, this can be any of a block's four sides, and it can switch between them, by abruptly changing its velocity such that $\mathbf{v}^{t-1} \cdot \mathbf{v}^t \leq 0$ holds. This is only possible at low velocities, due to a_{max}.

An important constraint on the motion of blocks is their impenetrability: at each time step, all collisions are resolved by returning any inter-penetrating blocks to their previous positions $\mathbf{p}^{t-1}$ and orientations ω^{t-1}, and setting their velocities to $\mathbf{0}$; collisions are thus completely inelastic. Furthermore, a gravity-like constraint is used, but it is not realized as a force (in fact, the vectors $\mathbf{a}^t$ and $\mathbf{v}^t$ are two-dimensional); rather, it is implemented as a restriction on vertical movement. The vertical dimension is essentially discrete, and a block can increase its vertical position by a single unit by climbing onto a stationary block. For simplicity, the change in the vertical position is made automatically, when the former block comes into contact with the latter, provided that it does not collide with any other block(s) at its new location. In a practical setting, this could be implemented, for example, via physical robots that are able to lift each other [46]. Similarly, a block can descend from a stationary block, so long as it is still supported below at its new position; thus, a block cannot "jump off" a structure that is more than 1 unit in height; such attempts are treated as collisions. To greatly simplify the implementation, the area of contact between the supported block and the supporting block(s) is allowed to be arbitrarily small, and there is no notion of “center of gravity”. Even so, as demonstrated below, the problem is made quite challenging by the existing constraints.

Apart from these constraints, which are physical in nature, there are also limitations on the blocks’ sensing capabilities. A controller for a block b can detect other blocks b' if their centers, denoted by $c(b')$, fall within a $15 \times 15 \times 4$ neighborhood centered at $c(b)$ (i.e., b can “see” 7.5 units in each of the four horizontal directions, and 2 units above and below). While a detailed discussion of possible physical implementations is beyond the scope of this dissertation, one can imagine that in addition to being able to detect each others’ relative positions, blocks would also be capable of engaging in short-range communication in order to exchange information about their velocities, orientations, etc., share their memory contents, and determine the positions of nearby blocks that may be obstructed from direct view. A block can also detect the boundaries of the world, if they fall within its neighborhood. For convenience, in the simulations reported below, the neighborhood of visibility is always aligned with these boundaries, irrespective of b’s orientation. The same holds for the initial seed, and therefore, for the assembling structure. However, blocks do not fundamentally require an absolute sense of direction, since the direction is encoded by the orientations of the seed blocks, and can propagate to additional blocks as they align themselves with the existing structure and become stationary. In some of the experiments, blocks are allowed access to a single piece of global information, which is the relative direction of the center of the world; as Section 3.4.3 will show, this expedites the self-assembly process, but is by no means necessary for its completion. Otherwise, anything that lies beyond a block's local neighborhood remains undetected.

3.2 An Integrated Approach to Distributed Control

Since any given block has no direct representation of the global structure being built, no preassigned location in this structure, and can only perceive and act within a local neighborhood, it seems improbable upon first consideration that a collection of such blocks could be made to self-organize into a relatively complex predefined structure like those shown in Figure 3.1. However, as discussed in Chapter 2, recent computational models provide evidence that wasp nests are constructed in this fashion [18, 110], and their extension to other geometric structures assembled by randomly moving agents or components in discrete cellular spaces [2, 48] suggest the feasibility of such an approach. Still, generalizing the control mechanisms of these past models to work for predefined structures consisting of different-sized components, to support movement through a continuous space, to avoid collisions between blocks as they move, to construct temporary staircases for access to higher level parts of a structure, and in general to sequence events properly (e.g., columns should be built before the walls and the roof) remains a formidable task, whose resolution requires a diverse set of methods. The control mechanisms used by individual blocks in the given approach are described below, and can be characterized as distributed (each block has its own control program), localized (blocks can only perceive and act locally), and homogeneous (the same control algorithm is used by

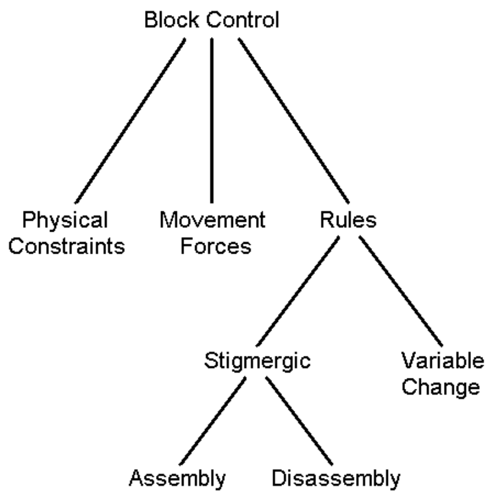


Figure 3.2: Influences on block behavior. The physical constraints are provided by the environment, while the remaining influences are internal to the block.

each block, although there are three physical block types).

3.2.1 Top-Level Block Behavior

The different factors that influence a block b's behavior, outlined in Figure 3.2, include the physical constraints mentioned earlier, "forces" that direct a block's movements through physical space, and a variety of if-then rules that assign it goals, change internal state variables, and trigger various actions. As is generally the case in agent swarm systems such as [92, 94, 123], the term "force" here refers to internally generated influences on movement and not to an external physical force that acts on the block. For simplicity, it is assumed that each block has a mass of 1 unit (i.e., $\mathbf{F} = \mathbf{a}$), so the terms "force", "acceleration" and "influence" are used interchangeably. The resultant force on block b is computed based on the characteristics of the environment within b's local neighborhood. For example, local influences may

If *b* is non-stationary:

- Consider stigmergic assembly rules that are applicable in the current mode, computing a goal location where *b* could settle to become part of the emerging structure.
- If some other non-stationary nearby block of the same size as *b* is closer to the goal, or if there was difficulty pursuing goals in previous time steps, then ignore the goal.
- If no goal has been found, or if the goal is ignored, then compute the resultant influence **F** to perform the following possible behaviors (in accordance with the current mode):
  - Avoid obstacles.
  - Climb up or down staircases.
  - Move with other blocks in a collective fashion.
  - Engage in random motion.
- Otherwise, if *b* is not properly oriented as it is pursuing the goal, then change the leading side.
- Otherwise, if *b* is at the goal, then stop, and become stationary.
- Otherwise, if *b* is close enough to the goal to reach it within the next time step, then do so.
- Otherwise, compute **F** to perform the following possible behaviors:
  - Pursue the goal.
  - Avoid obstacles.
  - Engage in random motion.

If *b* is stationary:

- Consider stigmergic disassembly rules that are applicable in the current mode, determining whether local conditions warrant disassembly.
- If a rule is matched and *b* is not supporting other blocks above it and there are no non-stationary blocks nearby, then become non-stationary and begin moving in an unobstructed direction.

Test variable change rule antecedents against local conditions and modify memory variables as appropriate.

Figure 3.3: Overview of a single iteration of block *b*'s control loop.

attempt to direct a block away from nearby obstacles and towards a specific location, such as a goal. Goals are determined via the use of *stigmergic rules* [2, 17, 18, 110] whose antecedents are matched against local structural patterns (i.e., local arrangements of stationary blocks already part of the emerging structure). While no block has an explicit representation of the desired structure that is to be built, this structure is implicitly encoded by these rules. If a part of the emerging structure falls within b's local neighborhood, b will match it against the antecedents of stigmergic *assembly rules*, to determine whether there is some adjacent location where b itself can belong. This goal can change between time steps. The disassembly of sacrificial components (i.e., the staircases) is controlled by stigmergic *disassembly rules*; similar to the assembly rules, they specify the local conditions around a stationary block that cause it to begin moving again (see Figure 3.2).

At any given point in time, the specific stigmergic rules that are applied when attempting to match a pattern, as well as the parameters used for force computation (given later, in Table 3.3), are determined by a block's *mode*, which is a variable in its memory. Typically, in some modes, a block attempts to climb up staircases and follow assembly rules, while in others, it tends to move down and away from structures. Mode changes generally occur in response to locally detected events, such as the completion of some part of the structure. These events are communicated via one or more *message* variables: when a block sees that a message variable of a nearby block is set to some particular value, it may choose to set its own message variable to that value; in this fashion, a message can propagate through the system, through memory sharing. Such modifications to a block's memory are governed by a third, distinct set of *variable change rules*.

One iteration of the control loop (as performed at each time step) is sketched in Figure 3.3, which shows how the distinct mechanisms are combined. It should be noted that the integration of these methods is not trivial. For example, even if a goal location is found by the stigmergic algorithm, it is sometimes best to temporarily ignore it when computing the force vector **F**, as stated in Figure 3.3 (a similar strategy is reported in [115]); otherwise, a block might spend a great deal of time in attempting to pursue a goal that is currently difficult to reach, and may repeatedly collide with other blocks in the process.

3.2.2 Stigmergic Goal Recognition

A block is not only limited to local perception, but also holds no explicit or direct global representation of the desired final structure. How, then, is it to decide where it belongs? In simulations of wasp construction behavior reviewed in Chapter 2, structures were encoded via sets of *stigmergic rules*, where each rule defines a goal location (i.e., a location where a block should be deposited) by the local arrangement around it [18, 110]. Subsequent studies have extended the model to a much wider class of structures by using blocks with an arbitrary number of colors (i.e., qualitative variable values) [2, 48, 49]. However, apart from the assumption of a very simple environment, a disadvantage of this approach is that the generated rule sets are unable to compactly capture large-scale patterns: for example, to build a 1D row of n blocks, at least $\frac{n}{2}$ rules are required in order to be able to "measure" the distance from the center of the row to its end points, and to ensure that the construction terminates there [2]. By contrast, in the approach given here, an integer variable is used, so a *single* rule can state that a block b can be placed adjacent to the right of another block b' if the value of this variable for b' is less than n. Unlike the set of colors, the set of integers is ordered; thus, relations such as $<$ are defined. When b is placed, it sets its own variable to the value stored in b' plus 1. Thus, while blocks have no global coordinate representation of the world around them, they can incrementally learn their positions relative to each other through local interactions. For three-dimensional construction, stigmergy is enhanced by endowing the memory of each block b with three positional variables x^b, y^b and z^b. The four initial seed blocks (Section 3.1.1) are assigned arbitrarily chosen constants (x_o, y_o, z_o), $(x_o, y_o{+}1, z_o)$, $(x_o{+}1, y_o, z_o)$, and $(x_o{+}1, y_o{+}1, z_o)$ as respective values. As another block b deposits itself adjacent to the emerging structure, it reads the memory of a nearby stationary block b' and sets its own variables to the correct relative position within the overall structure. This is somewhat similar to an approach given in [117, 118] (discussed briefly in Chapter 2), although the methodology presented here was developed independently, and differs in that blocks store a set of stigmergic rules, rather than a global blueprint of the structure.

Here, the notion of a stigmergic rule is presented through somewhat informal descriptions and examples; a more precise and general definition is provided in Appendix B. Each stigmergic rule is structured as an antecedent-consequent pair, specifying a target *goal location* g for a block when its antecedents match a specific local arrangement (pattern) of stationary blocks around g. Blocks of different sizes follow different rules. In general, a rule's antecedents can be represented by a three-dimensional grid of *sites*, where the inner sites correspond to a potential position for block b, and the outer sites must cover all locations *adjacent* to this position (see Figure 3.4). Thus, for a small block, the antecedent grid's typical dimensions are $3 \times 3 \times 3$; for a medium block, they are either $4 \times 3 \times 3$ or $3 \times 4 \times 3$ (depending on the required orientation of the block in the goal position); and for large blocks, they are $6 \times 3 \times 3$ or $3 \times 6 \times 3$. However, in some cases, the dimensions may be somewhat higher; for example, if a rule specifies the placement of a small block adjacent to a large block (the latter is 4 units in length), it may be necessary to use the dimensions $5 \times 5 \times 3$, in order to ensure that the adjacency occurs where it should. Each grid site in a rule is either *empty*, indicating that there must be no block at that location; *wildcard*, allowing either the presence or the absence of a block; or *full*, denoting the presence of a stationary block. A full grid site can specify the dimensions of the block at the corresponding location; if the dimensions are not given, then the site simply states that there should be some block section present at that location. The site can also impose certain *memory conditions* on the memory of the corresponding block b'. These memory conditions have the form $a^{b'} < k$, $a^{b'} > k$, $a^{b'} = k$ or $a^{b'} \neq k$, where $a^{b'}$ is a variable and k is an integer value, and all such conditions must hold if the rule is to be applicable. As in the simple, 1D example given above, these conditions are useful for assembling structures with prescribed dimensions, preventing, for example, the floor of the building from growing beyond 12×8 blocks.

An example of a specific stigmergic assembly rule is given in Figure 3.4a, which

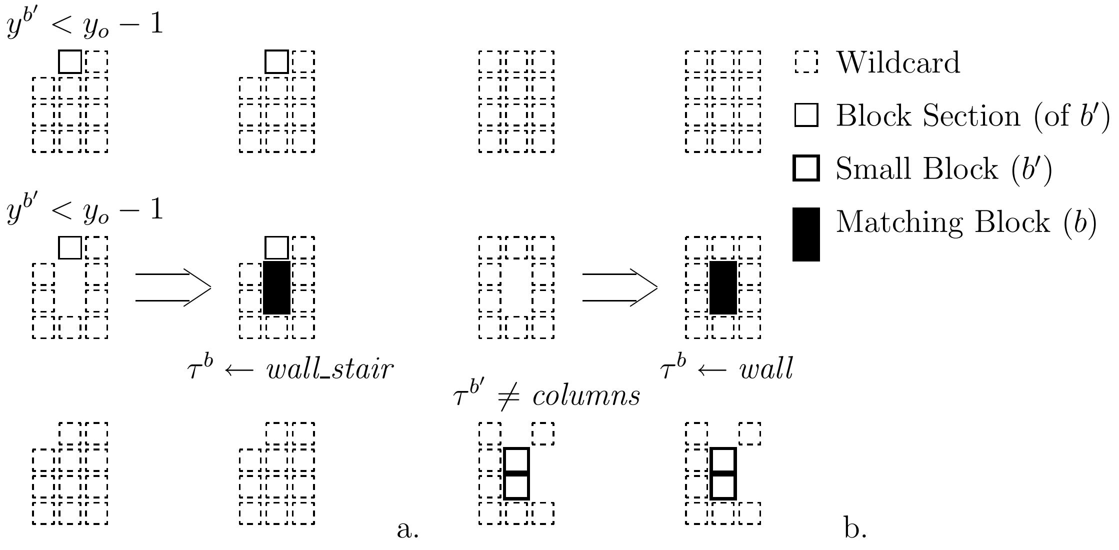


Figure 3.4: A graphical representation of two stigmergic rules used by medium size blocks for (a) constructing the wall staircase and (b) initiating the walls. The antecedents (to the left of the arrow) show the arrangement around a potential goal location before the matching block deposits itself; the consequents (to the right of the arrow) show the arrangement immediately after. The rules consist of three horizontal cross sections above (top), at the same level as (middle), and below (bottom) the goal location. In each rule's memory conditions and assignments, which are explained in the text, b' refers to a matched block that is already present (open square), and b refers to the matching block.

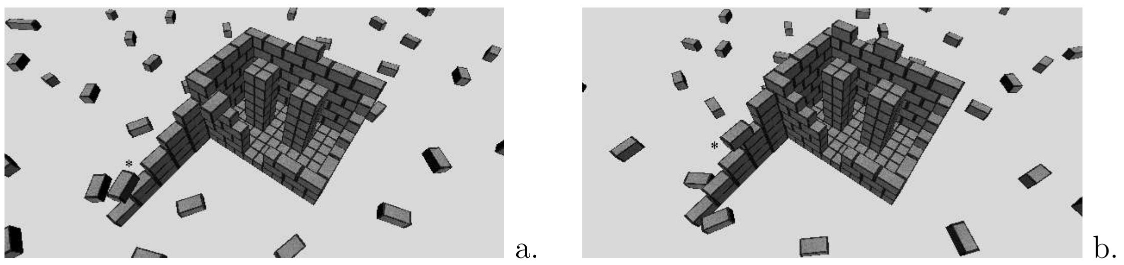


Figure 3.5: The construction of the wall staircase at a point where the top step is required (a), and subsequently when the next block is about to be deposited (b). Climbing blocks are marked by an asterisk (*).

(using notation somewhat similar to Figure 2.1a) shows a rule used by medium blocks that results in the construction of the temporary staircase (Figure 3.5) employed during the assembly of the walls, and built concurrently with them. In the antecedents, three horizontal slices are given, specifying the arrangement of blocks above a potential goal location (top), around it (middle), and below it (bottom). Wildcard sites are denoted by dashed lines, while the sites that must be empty are unmarked; since a goal location must leave room for a block, the two inner sites are empty in the antecedents. Solid lines mark the two full sites, which specify that some block sections must be present just "behind" the goal location, both at the same level and above. This rule creates a natural procedure (somewhat similar to [46]) for stair assembly where the steps are deposited as needed: since a barrier that is two or more blocks in height cannot be scaled, a block deposits itself just in front of that barrier, creating a step that other blocks can climb. Other steps will be deposited using this same rule, and in this way a new, diagonal "layer" of the staircase is laid down, beginning with the top step and propagating down the emerging staircase, as shown in Figure 3.5. Irrespective of its height, the assembly of a staircase can thus be accomplished in a very parsimonious fashion, with a single rule. It should be noted that the three vertical grid sites to the upper left of the goal location are empty to prevent the staircase from being built at a wall location other than what is shown in Figure 3.5. Even with this restriction, it is still possible that blocks could begin to assemble staircases adjacent to the columns after they are completed; this is prevented by the memory condition $y^{b'} < y_o - 1$ (Figure 3.4a), since column blocks b' have values of $y^{b'}$ that are either y_o or $y_o + 1$. Finally, the consequent states that when a block reaches a goal location that was matched by this rule, and becomes stationary, it sets the value of the *substructure type* variable τ^b to *wall_stair* (the value is actually an integer, but is provided mnemonically for convenience). This allows a block to determine the substructure to which another block belongs, which can play a role in force computation, as described in Section 3.2.3.

An additional example of a stigmergic assembly rule is shown in Figure 3.4b, in this case for initiating the walls. At most vertical levels, the first wall block is placed in the far right corner of the structure; subsequently, the wall layer expands as other blocks become adjacent. The rule in Figure 3.4b exists specifically to begin the very first layer. The full sites are emphasized with thick lines, to indicate that the blocks below the goal location must be of a small size (i.e., floor blocks), with the empty sites to the top and to the right denoting the far right corner of the floor. The memory condition $\tau^{b'} \neq$ *columns* ensures that a medium block is placed on the floor, and not on top of a column; thus, the rule is further restricted to one specific location within the entire structure. Such rules are sometimes necessary: not all features of a structure can be encoded parsimoniously; in fact, by a Kolmogorov complexity argument [67], the vast majority of possible structures cannot be described compactly. Structures can encode strings of information (DNA is a quintessential example), and it is known that most strings have a high Kolmogorov complexity, meaning that the string cannot be generated by a program whose size is much smaller than the length of the string itself. However, it is expected that many "useful" or "interesting" structures will have repeating patterns, allowing the parsimony that is potentially allowed by the approach to be exploited.

In contrast to *assembly rules*, such as the ones discussed above, there are also *disassembly rules* that are used to dismantle parts of the structure that are no longer needed. The two types of rules are structured in a similar way, the difference being that the former specifies where a block *should be*, while the latter describes the conditions around the block where it *already is*. The block begins to move again only if these conditions hold and if it is safe to do so. If a stationary block begins moving, $x^b = y^b = z^b = \tau^b = \mathit{null}$ is set, as these variables are only relevant for stationary blocks.

At every time step, each block b scans its neighborhood for existing parts of the structure to determine whether any rules are matched. If some rule is matched, the algorithm produces a vector $\mathbf{g} = \mathbf{p}(c(b), g)$ to the goal location g, relative to $c(b)$. While structures are discrete, the components of $\mathbf{g}$ are real-valued because the current position of b relative to any block within the structure need not be integral. For a given rule, it is possible that several locations may be matched; in this case, preference is given to locations at the same level as b. If there are several locations that are vertically closest to b, then the one with the minimal horizontal distance to $c(b)$ is chosen. Locations that are not supported immediately below (by other stationary blocks, or the ground) are ignored. If several different rules match valid locations, then the actual goal location g to be pursued is chosen such that $\|\mathbf{g}\|$ is minimized. For disassembly rules, $\mathbf{g} = \mathbf{0}$.

3.2.3 Force-Based Movement Control

While destination goals are determined by stigmergic rules, reaching these goals can be a difficult task due to the continuous nature of space, coupled with block impenetrability. In fact, many classes of motion planning problems with multiple impenetrable components are known to be PSPACE-hard [64]. Rather than resorting to computationally expensive approaches such as path planning [64], the control of a block's movements (regardless of whether or not it currently has a target destination) is force-based, along the lines of systems discussed in Section 2.1.2. At any given point in time, a non-stationary block is influenced by a number of forces that combine in a non-linear fashion into a resultant force $\mathbf{F}$. As in many swarm intelligence systems such as [92, 94, 123], these forces do not actually exist in the simulated environment; rather, they are influences computed internally that allow the block to determine its movement. Most individual force definitions are given in Table 3.1. The formulas, which are explained below, were typically developed empirically, through an extensive trial-and-error process, with past studies serving as guidelines.

If a goal location g has been found by the stigmergic matching procedure, the *goal approach* force $\mathbf{F}_{ga}$ guides a block towards it [92]. (Note that the subscript ga here stands for "goal approach", and is *not* a product of g and a). This force acts in the direction $\mathbf{g} = \mathbf{p}(c(b), g)$ of the goal g relative to a block's center $c(b)$, and is offset by b's current velocity. Without the addition of $\epsilon_g \frac{\mathbf{g}}{\|\mathbf{g}\|}$, goal approach would be asymptotic and the goal would never be reached, so the block "aims" slightly

Table 3.1: Forces Influencing a Block's Movements [*]

Force	Description	Formula
$\mathbf{F}_{ga}$	Goal Approach	$w_g\left(\mathbf{g}+\epsilon_g\frac{\mathbf{g}}{\lVert\mathbf{g}\rVert}\right)-w_v\mathbf{v}\left(b\right)$
$\mathbf{F}_{su}$	Step Up	$V\left(b\right)\cdot\frac{\mathbf{p}(c(b),s(T_{su},0))}{\lVert\mathbf{p}(c(b),s(T_{su},0))\rVert}$
$\mathbf{F}_{sd}$	Step Down	$V\left(b\right)\cdot\frac{\mathbf{p}(c(b),s(T_{sd},-2))}{\lVert\mathbf{p}(c(b),s(T_{sd},-2))\rVert}$
$\mathbf{F}_{rm}$	Random Motion	$\mathbf{N}\left(0,1\right)$
$\mathbf{F}_{gc}$	Global Centering	$\frac{\mathbf{p}(c(b),c_{world})}{\lVert\mathbf{p}(c(b),c_{world})\rVert}$
$\mathbf{F}_{nc}$	Neighbor Cohesion	$\frac{V(b)}{\left\lvert B_f\right\rvert^2\cdot\sqrt{N_x^2+N_y^2}}\cdot\left\lVert\sum_{b'\in B_f}\mathbf{p}\left(c\left(b\right),c\left(b'\right)\right)\right\rVert\cdot\sum_{b'\in B_f}\mathbf{p}\left(c\left(b\right),c\left(b'\right)\right)$
$\mathbf{F}_{na}$	Neighbor Alignment	$\frac{V(b)}{\left\lvert B_f\right\rvert^2\cdot v_{max}}\cdot\left\lVert\sum_{b'\in B_f}\mathbf{v}\left(b'\right)\right\rVert\cdot\sum_{b'\in B_f}\mathbf{v}\left(b'\right)$
$\mathbf{R}\left(p_b,p_o\right)$	Repulsion	$\frac{1}{(\lVert\mathbf{p}(p_b,p_o)\rVert-\delta)^2}\cdot\frac{-\mathbf{p}(p_b,p_o)}{\lVert\mathbf{p}(p_b,p_o)\rVert}$

[*] Vector $\mathbf{p}(x,y)$ is the position of y relative to x; p_b and p_o are reference points on the block and the obstacle, respectively.

past the goal, to the extent determined by the parameter ϵ_g. Because a block's orientation depends directly on its velocity (Section 3.1.2), it is often necessary to first approach a temporary location near the actual goal, chosen such that when the goal is subsequently reached, $\mathbf{F}_{ga}$ results in an appropriate orientation (details are omitted for brevity).

If no goal has been detected but the block is near some part of the existing structure, this structure may offer clues as to where assembly takes place. In particular, if a block detects a staircase, it may choose to ascend or descend it depending on the stage of the assembly process. This is accomplished for block b via the *step up* and *step down* forces $\mathbf{F}_{su}$ and $\mathbf{F}_{sd}$, which are based on the nearest cubic section of a block $s(T, z)$ that supports no stationary block above it, is at level z relative to b's current position, and has substructure type $\tau^{s(T,z)} \in T$. The set T can be specified, for example, such that in computing these forces, staircase blocks are considered, but all other blocks are ignored. For $\mathbf{F}_{su}$, cubic sections on the same level (which can be climbed) are considered; similarly, $\mathbf{F}_{sd}$ considers block sections two levels below, which can be descended upon. The magnitude of these forces is not dependent on distance, but is proportional to the volume (size) $V(b)$ of the block.

To prevent a block from becoming "stuck" in a local attractor, it is potentially useful to add a degree of probabilistic behavior to its motion. This is achieved via the *random motion* force $\mathbf{F}_{rm}$ whose components are independent, normally distributed random variables. The *global centering* force $\mathbf{F}_{gc}$, similar to what is used in [35], is based on the only piece of global information known to the block controllers, namely the normalized relative direction of the world's center c_{world}, where the construction takes place. One can imagine a long-range beacon existing at the center of the world that all blocks can detect and have a tendency to move towards, although the global position of the beacon is not known and the force is distance-independent. This exception to the principle of locality [88] is made for efficiency and because one can conceptually separate the task of finding the assembled structure from the task of finding and reaching goal locations within that structure. The former task has been studied in earlier related work, such as [94]. Here, the primary focus is on the more guided (by elements of existing structures) yet more constrained (by obstacles) search that is specific to self-assembly; thus, in the simulations of this chapter, the process of reaching the regions near the structure is typically expedited by incorporating $\mathbf{F}_{gc}$.

The obstacles that must be avoided during movement may consist of other moving blocks, stationary structures, or world boundaries. Thus, b computes a *block avoidance* force $\mathbf{F}_{ba}$, an *obstacle avoidance* force $\mathbf{F}_{oa}$ and a *limit (boundary) avoidance* force $\mathbf{F}_{la}$, respectively. Furthermore, an edge avoidance force $\mathbf{F}_{ea}$ applies to blocks that are near the edge of a high structure. It is implemented by defining a "virtual obstacle" that pushes the block away from the edge. The formulas for these forces are somewhat detailed, and thus omitted for brevity, but all are based on the inverse square law $\mathbf{R}$, given in the last row of Table 3.1. The parameter δ is an offset which is sometimes used in computing $\mathbf{F}_{oa}$ (it is then called δ_{oa}) for intensifying the force near obstacles. Several vectors $\mathbf{R}$ are sometimes combined for a given force: for example, when b detects another moving block b', it computes $\mathbf{F}_{ba}$ by summing $\mathbf{R}$ over (possibly) multiple pairs (p_b, p_o), where p_b is the center of a $1 \times 1 \times 1$ cubic section of b, and p_o is the center of a cubic section of b'.

The force $\mathbf{F}_{ba}$ is an *interactive* force in that it creates a mutual repelling influence between non-stationary blocks. Two other interactions between blocks are defined here, and they are referred to as the *neighbor cohesion* force $\mathbf{F}_{nc}$, which attempts to move an agent towards the center of the group of neighboring blocks, and the *neighbor alignment* force $\mathbf{F}_{na}$, which attempts to match their velocity. As discussed in Section 2.1.2, an appropriate combination of cohesion, alignment, and avoidance forces can lead to realistic flock-like (or herd-like, school-like, etc.) collective movements. The forces $\mathbf{F}_{nc}$ and $\mathbf{F}_{na}$ *increase* quadratically with the average distance and velocity of neighbors, respectively, as shown in the table, where $N_x = N_y = 7.5$ is the maximum range of visibility along the x or the y-axis. Since blocks of different sizes are used for different purposes, the set B_f includes only blocks that are of the same size as b. (Chapter 5 will present experiments where this restriction is removed). This causes the emergence of collectively moving groups of like-sized blocks.

At every moment in time, each block computes a resultant force $\mathbf{F}$ acting on it using:

$$\mathbf{F} = \mathbf{f}\left(w_{ga}, \mathbf{F}_{ga}, w_{su}, \mathbf{F}_{su}, \cdots, w_{na}, \mathbf{F}_{na}\right)$$

where $\mathbf{f}$ is a *prioritized* non-linear vector sum of the weighted individual forces. Non-linearity arises in that certain forces can sometimes be ignored to prevent them from interfering with forces of greater importance [91, 92, 94]. For example, the collective forces $\mathbf{F}_{nc}$ and $\mathbf{F}_{na}$ are suppressed when either $\mathbf{F}_{su}$ or $\mathbf{F}_{sd}$ is non-zero, so that a block that wishes to climb a staircase is not pulled away from it by nearby blocks. The computation of $\mathbf{F}$ also varies depending on the situation that a block is currently facing, as has been done in past collective movement systems such as [94, 95]. In particular, when a block is searching for goals, all forces can come into play with the exception of $\mathbf{F}_{ga}$; on the other hand, when a block is actively pursuing a goal, $\mathbf{F}_{ba}$, $\mathbf{F}_{su}$, $\mathbf{F}_{sd}$, $\mathbf{F}_{nc}$, $\mathbf{F}_{na}$ and $\mathbf{F}_{gc}$ are not used. Furthermore, parameters (such as the weights w_i) depend on the block's current mode, as discussed below.

3.2.4 Temporal Coordination

Environmental features such as gravity and impenetrability can impose high-level ordering constraints on the self-assembly process: it may be necessary to complete the assembly or disassembly of certain parts of the structure before commencing the assembly/disassembly of others. Stigmergy is sometimes insufficient for this purpose: when deciding whether to deposit itself at a particular location, a block that only uses stigmergy may not be able to detect whether particular substructures have been completed, due to a limited neighborhood of perception. As a result, some parts of the structure (such as the outer walls of the building) may begin to self-assemble before others (such as the inner columns) are completed, causing a problem later in the process. At the same time, it is also possible that blocks (which are initially on the ground) will be unaware that a remote part of the structure (such as the roof) must now be assembled, and will not assume appropriate movement dynamics (e.g., allowing them to climb a staircase). Therefore, certain information must somehow be communicated throughout the system, allowing blocks to coordinate their behavior over time [6, 94, 95, 121], which makes the design of control methods more challenging.

To address this issue, self-assembling blocks can display simple messages (signals) to neighboring blocks and thus affect each others' mode of operation, which permits the temporal sequencing of top-level events. A block's *mode* variable m^b determines both the subset of stigmergic rules that are considered by a block during pattern matching (somewhat akin to [49]) and its movement dynamics (as in [93, 94, 95]). The modes used during the various stages of assembling the building are given mnemonically by the entries in Table 3.2. Modes *regular_assembly* and *regular_disassembly* are used for general assembly and disassembly tasks. They have distinct associated movement dynamics: for example, with respect to staircases, in the former mode blocks attempt to climb up, while in the latter they climb down. The *frame_assembly* mode is associated with rules that are used specifically for placing the lower parts of a door frame (Figure 3.1d). To prevent the premature disassembly of the wall staircase (Figure 3.5), an additional *wall_stair_disassembly* mode is used. Stages which are not separated by a horizontal line in Table 3.2 are not necessarily disjoint in time: for example, the staircase of a finished column may be disassembled while the other column and its staircase are still being completed. Furthermore, a block may switch from its typical mode into a different mode if conditions warrant. This can be useful in situations when some blocks wish to climb the wall staircase while others try to descend it. To avoid conflicts, descending blocks are given a "right of way": when an ascending block b detects a nearby block b' such that $m^{b'} = \textit{regular\_disassembly}$ (i.e., b' is trying to descend), it temporarily switches

Table 3.2: The Stages of Assembling a Building

Stage	Small	Medium	Large
Assembly of Floor and Columns	*regular_assembly*	*null*	*null*
Disassembly of Column Staircases	*regular_disassembly*	*null*	*null*
Assembly of Lower Door Frame	*frame_assembly*	*frame_assembly*	*null*
Assembly of Walls	*regular_disassembly*	*regular_assembly*	*null*
Placement of Door Frame Top	*regular_disassembly*	*regular_assembly*	*regular_assembly*
Assembly of Roof	*regular_disassembly*	*regular_disassembly*	*regular_assembly*
Disassembly of Wall Staircase	*regular_disassembly*	*wall_stair_disassembly*	*regular_disassembly*

IF b is non-stationary AND $m^b = \textit{regular\_assembly}$ AND
    $\exists b'$ within a $5 \times 5 \times 3$ neighborhood of b such that $z^{b'} > 0$ AND $\tau^{b'} = \textit{wall\_stair}$
THEN $m^b \leftarrow \textit{on\_wall\_stair}$

Figure 3.6: A variable change rule for changing modes when block b is on top of the wall staircase.

to mode *regular_disassembly* as well. However, this behavior is only desired when b is actually on the staircase. Thus, when it detects a wall staircase block, it first temporarily switches to the special *on_wall_stair* mode (not given in Table 3.2), and the switch to *regular_disassembly* can only be made from this mode.

State variable changes like the above are primarily specified via non-stigmergic *variable change rules*, such as the one given in Figure 3.6. As stated more formally in Appendix B, such rules can impose memory conditions (as well as size and whether or not it is stationary) on the block b itself as well as other nearby blocks b' within a prescribed neighborhood. For a given rule, when all internal conditions are satisfied and a certain number (no lower than some given threshold) of nearby blocks satisfy the external conditions, some variable in memory is set to a given value. For example, the rule in Figure 3.6 states that if a block b detects at least one other block b' such that $\tau^{b'} = \textit{wall\_stair}$, and b is currently in mode $m^b = \textit{regular\_assembly}$, then b should switch to mode $m^b = \textit{on\_wall\_stair}$. It is specified that only blocks within a $5 \times 5 \times 3$ (rather than $15 \times 15 \times 4$) neighborhood should be considered; together with the memory condition $z^{b'} > 0$, this restriction prevents the rule from applying unless a block is very close to (or on) the staircase. If a rule is matched, the corresponding variable change does not take effect until the next time step. Sometimes, several rules may be applicable for a given variable; in this case, only the last such rule will have an effect, so rule order provides a prioritization mechanism.

Finally, one or more *message variables* μ^b are used to properly transition between modes. Typically, a block sets μ^b to some specific value when it detects that a particular part of the overall structure is complete. Through variable change rules, other nearby blocks take on this message variable value, which thus propagates through the system. In assembling the building, a single message variable is sufficient, though it takes on multiple values. For example, when the left column staircase is completely disassembled, the last block of the staircase to commence moving sets the message $\mu^b = \textit{left\_column\_done}$. The message *right_column_done* is used similarly to signify the completion of the right column. The first of these messages to be produced propagates through the rest of the system via another rule. When the second message appears, the two combine and a third message, *columns_done*, is produced. This in turn causes small and medium blocks to switch to mode *frame_assembly*, and to begin assembling the door frame. In the general case of detecting the completion of n substructures, where the order of completion can be arbitrary, it is preferable to use n message variables (as shown in later chapters); otherwise, the number of rules necessary to combine the messages (which can be produced in any order) becomes intractable, along with the number of variable values that are used to represent intermediate combinations.

3.3 Experimental Methods

Thus far, the self-assembly process was presented from the "microscopic" viewpoint of an individual block. At a more "macroscopic" level, which views the entire collection of agents (blocks) interacting with each other and with their environment, we are dealing with a complex dynamical system whose behavior is difficult to predict. The existence of mutual influences between blocks gives rise to an n-body problem and chaotic effects, such that a slight change in a local variable (e.g., the position of a block) can send the system on a distinct trajectory and result in different global properties (such as the number of time steps to completion). It is possible, however, to gain insight into the *typical* collective functioning of the system through repeated empirical observations under various conditions.

In the experiments discussed below, the focus is specifically on the self-assembly of the building illustrated in Figure 3.1d,e, which consists of 146 small blocks, 116 medium blocks, and 27 large blocks. Furthermore, 10 small blocks are used to build each temporary column staircase, while the temporary wall staircase is assembled from 28 medium blocks. In practice, additional small blocks and medium blocks can be necessary. If the number of small blocks is too low, then a deadlock situation is possible, where both columns are partially completed, but neither can be finished, because all small blocks that are not part of the floor or the columns are currently assembled as column staircases. Deadlock can also potentially occur if all medium blocks are on top of the walls, but the wall staircase is not high enough for them to climb down and continue its assembly, although this situation was never actually observed. In the experiments, block numbers are generally kept small: there are 164 small blocks, 150 medium blocks, and 27 large blocks, except where indicated. The movement control parameter values (Section 3.2.3) used in the experiments are listed in Table 3.3, unless explicitly noted otherwise. For each experiment described below, 30 trials are performed, each trial making use of a distinct stream of (pseudo-) random numbers, and beginning with a different initial arrangement of blocks.

In any trial, the system's *progress* $P(t)$ is defined as the number of assembly and disassembly steps in the construction process that have been completed by time t, measured as the total number of times (thus far) that some non-stationary block became stationary, plus the number of times that some stationary block became non-stationary. For example, the completion of the building floor and column assembly (including column staircase disassembly) and lower door frame assembly occurs at $P(t) = 184$, the roof assembly at $P(t) = 353$, and the task is complete at $P(t) = 381$, which is equal to $146+116+27 = 289$ (the number of blocks in the target structure) plus 2 times $(10 + 10 + 28) = 48$ (the number of staircase blocks) minus 4 (the number of seed blocks). At this point, it is verified that the structure adheres to all specifications (i.e., it is ensured that no stationary blocks are missing, or occupying incorrect locations), and the completion time is noted. Also recorded is the number of collisions that occurred during a trial, which is defined as the number of times that some block had to be forcefully stopped and returned to its previous position and orientation, as described in Section 3.1.2. If the self-assembly task is not complete, but progress $P(t)$ remains unchanged for more than 30000 time steps, then the

Table 3.3: Movement Parameters for Experiments

Parameter	Value(s)
T_{su}	$\{$*floor*, *column_stair*, *wall_stair*$\}$ if b is small; else $\{$*column_stair*, *wall_stair*$\}$
T_{sd}	$\{$*floor*, *column_stair*, *wall_stair*$\}$
w_{ga}	1.0
ϵ_g	0.5
w_g	6.0
w_v	5.0
w_{ea}	20.0
w_{la}	45.0
w_{oa}	5.0 if $m^b =$ *regular_disassembly*; else 20.0
δ_{oa}	0.75 if $m^b =$ *regular_disassembly*; else 0.0
w_{gc}	2.0 if $m^b \in \{$*regular_assembly*, *frame_assembly*, *on_wall_stair*$\}$; else 0.0
w_{su}	17.0 if $m^b \in \{$*regular_assembly*, *on_wall_stair*$\}$; 3.0 if $m^b =$ *frame_assembly*; else -5.0
w_{sd}	22.0 if $m^b \in \{$*regular_disassembly*, *wall_stair_disassembly*, *null*$\}$; else -5.0

trial is terminated and considered to be unsuccessful (all trials reported below were successful, unless explicitly stated otherwise). However, quantities measured up to termination time in unsuccessful runs are included in the computed statistics for the set of trials. These statistics are the average completion time (μ_t), the standard deviation of completion times (σ_t), the average number of collisions per time step ($\mu_{c/t}$), and the standard deviation of collisions per time step ($\sigma_{c/t}$). In comparing differences between means, all reported p-values are determined via a two-sample, two-tail t-test assuming unequal variances.

While metrics such as μ_t give an overall idea of the system's performance, they do not show where time was lost or gained when comparing sets of trials. An indication of the performance at various points in time can be given by the *progress curve*, which plots $P(t)$ over t. Since the completion time varies between trials, a given value t could correspond to distinct stages of the self-assembly process. Due to this non-uniformity between time scales, it is difficult to present an entirely meaningful average progress curve. For this reason, rather than computing the average progress at some time t, it is instead asked: for a given progress level P, how many time steps pass before the next progress level is reached (i.e., before some non-stationary block is deposited, or some stationary block begins to move again)? For a given trial, the *progress level duration* $d(P)$ is defined as $d(P) = |\{t \in [1, \infty] : P(t) = P\}|$. A fixed level of progress still does not necessarily correspond to some specific stage of the assembly process, as the sequence of assembly is only partially ordered. However, because the number of times that a non-stationary block becomes stationary and vice-versa is fixed between trials, one can take the average value of $d(P)$ for any

$P \in [0, 380]$. Furthermore, $d(P)$ allows the total completion time, which can be expressed as $1 + \sum_{P=0}^{380} d(P)$, to be broken down into components (it is necessary to add 1 to the sum in order to account for the final time step at $P = 381$). A plot of $d(P)$ over the range of progress values is typically very "jagged" in appearance, with sharp spikes, and comparing two or more such plots is therefore difficult. Thus, it is typically beneficial to plot a *smoothed* duration curve:

$$d_L(P) = \frac{1}{L} \cdot \sum_{p=P-\frac{(L-1)}{2}}^{P+\frac{(L-1)}{2}} d(p)$$

where L (an odd integer) is the *smoothing level*, which is 5 in all given plots unless otherwise indicated.

The environment simulator and block controllers are are implemented in Java$^{\text{TM}}$. In the experiments discussed in this chapter, two sets of 30 trials typically require about 9 hours on a Dell Precision WS machine running Linux, with two 3.2 GHz Intel Xeon processors and 1.0 GB of memory. Each block makes use of its own instance of the random number generator (RNG) provided by the standard Java$^{\text{TM}}$ libraries. (In experiments not reported here, a more sophisticated RNG implementation was used [69], but this yielded qualitatively similar results). Another RNG instance is employed to determine the initial arrangement of the blocks. To generate seeds for these RNGs, random numbers are first obtained from a "master" RNG instance, which is seeded with the current time. The Virtual Reality Modeling Language [45] is sometimes used to visualize the simulation through real-time animation.

3.4 Experimental Results

This section describes a number of experiments performed to evaluate and analyze the performance of the system during the self-assembly of a building (Figure 3.1d,e) under various conditions.

3.4.1 Self-Assembly at a Glance

Initial experiments focused on the question of whether the self-assembly process outlined above, based on self-directed block movements through continuous space guided by local forces and rules, could reliably and repeatedly construct the specific building structure shown in Figure 3.1d,e. The force definitions, parameters, rules, etc. described above were initially established through a trial-and-error process, guided by observing the required structural and behavioral patterns; this process required a few person-months of effort. The correctness of the control methods was verified by performing a large number of simulations with distinct random number streams. A key experimental result is that the building structure could successfully self-assemble, even in the absence of any global information ($w_{gc} = 0$). It was found that this can be accomplished with 66 stigmergic rules (6 of which are used for disassembly) and 77 variable change rules.

The progress through time of a typical and representative self-assembly process is depicted in Figure 3.7. (A short video at http://www.cs.umd.edu/~reggia/grushin.html provides an animation of some of its stages). Self-assembly begins with a seed consisting of four blocks, while

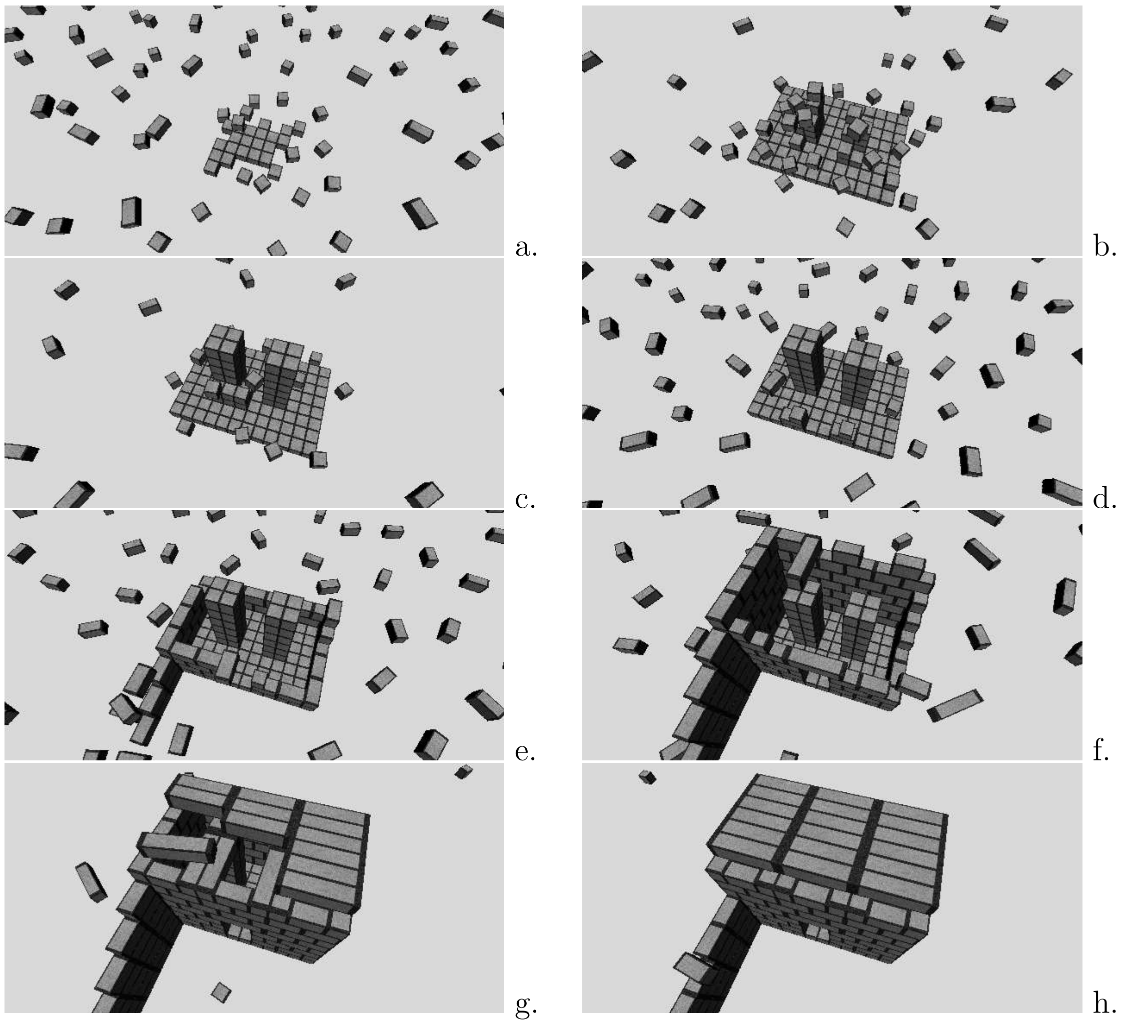


Figure 3.7: The self-assembly of a building at various points in time: (a) $P(151) = 15$ (i.e., at $t = 151$, 15 blocks have been added): floor assembling; (b) $P(919) = 121$: columns assembling on completed floor; (c) $P(2253) = 173$: columns complete and left column staircase disassembling; (d) $P(2727) = 183$: lower parts of door frame almost complete; (e) $P(4273) = 223$: walls assembling (note staircase on lower left); (f) $P(7914) = 296$: wall assembly continues with a large block just laid over the door and another already in place for the roof; (g) $P(12205) = 346$: roof assembling; (h) $P(14383) = 357$: wall staircase disassembling. The completed building is seen in Figure 3.1d.

remaining blocks are randomly distributed on the ground. In Figure 3.7a, the seed "grows" to become the floor, as incoming blocks position themselves adjacent to seed blocks and to one another. Two columns then begin emerging from specific locations in the floor, as shown in Figure 3.7b. As these columns grow in height, some small blocks assemble into staircases that partially wind around the columns in order to allow small blocks to reach higher levels (recall that blocks can move at most one vertical level at a given time). Each diagonal layer of steps on these staircases is deposited "backwards", from top to bottom. Once a column is complete, its staircase disassembles (Figure 3.7c); as in assembly, this disassembly is also done layer by layer, in a top to bottom fashion.

After the floor and the columns are complete, the door is marked by depositing the lower sides of its frame, consisting of a small and a medium block each as shown in Figure 3.7d. The walls, which consist of medium blocks, are then begun along with the wall staircase, which allows other blocks to reach higher levels on top of the growing structure (Figure 3.7e). As these walls reach an appropriate height, large blocks begin to climb the staircase. One such block eventually places itself over the door, to form the top part of the door frame (Figure 3.7f). In some trials, roof assembly is not begun until after the top of the door frame is in place, while in others, a number of large blocks deposit themselves as part of the roof (in this particular trial, only one such block is observed) before one of them finds the door frame and deposits itself over it. Once the door frame is covered, any excess non-stationary large blocks remaining on top of the structure climb down the staircase, allowing the walls to be completed without excessive interference between large

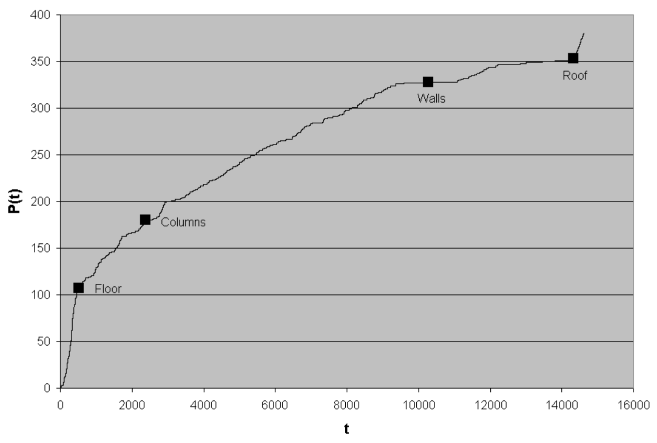


Figure 3.8: Progress curve $P(t)$ for the simulation shown in Figure 3.7. Labeled points denote the completion of the major substructures.

and medium blocks. When the walls are finished, large blocks begin to climb the staircase once again to build the two-layer roof. The second roof layer (Figure 3.7g) forms perpendicularly to the first layer; it begins near the edges farthest from the staircase and extends towards the staircase. When the roof is complete and there are no blocks left moving above it (variable change rules are in place to ensure that this is the case), the wall staircase disassembles (Figure 3.7h), resulting in the final structure (Figure 3.1d,e).

Let us now analyze the self-assembly process quantitatively. Figure 3.8 depicts the progress curve for the same simulation, plotting the values $P(t)$ versus t. This curve is particularly steep for the first 500 time steps or so while the floor is assembling. This indicates a high level of efficiency, which arises from the inherent parallelism and accessibility of floor assembly: at any given point in time, there exist a relatively large number of goal locations that are easy to reach, as these locations are on the ground and unobstructed by stationary obstacles. Subsequently, the task becomes more difficult, and the curve is typically considerably less steep: fewer goals are available and they are less accessible. Minor periods of efficiency are observed in places, but only at the very end of the assembly does the curve develop a high rate of change again for a substantial amount of time. This corresponds to the disassembly of the wall staircase, which is quite efficient, with little interference between blocks.

Flat regions in the curve that extend over many time steps indicate periods of particular difficulty where the system struggles to accomplish something useful. For example, at progress level $P(t) = 327$ (about $t = 10000$), the walls are almost built with the exception of one location and the remaining medium blocks have some difficulty finding this location. Subsequently, at $P(t) = 328$ (just after the completion of the walls), there is a period of time when the large blocks switch back to the *regular_assembly* mode. Because the number of large blocks is relatively small and their density is low, it takes some time for them to make this transition and find the staircase. Progress is also slow around $347 \leq P(t) \leq 352$ when the roof is almost complete and there are few large blocks left to find the remaining goal locations and deposit themselves. Once again, the inefficiency is caused by a lack of parallelism, which is characterized both by the low density of blocks and the low number of goals.

Is the completion time dominated by such periods of difficulty, or does it mainly consist of shorter durations of many progress levels? In order to answer this question, the (unsmoothed) $d(P)$ values for a *set* of 30 trials (the trial depicted in Figures 3.7 and 3.8 is 16th best of this set, in terms of completion time) are analyzed,

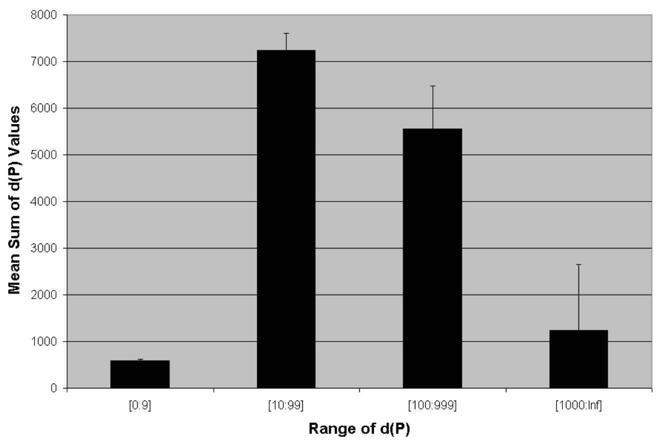
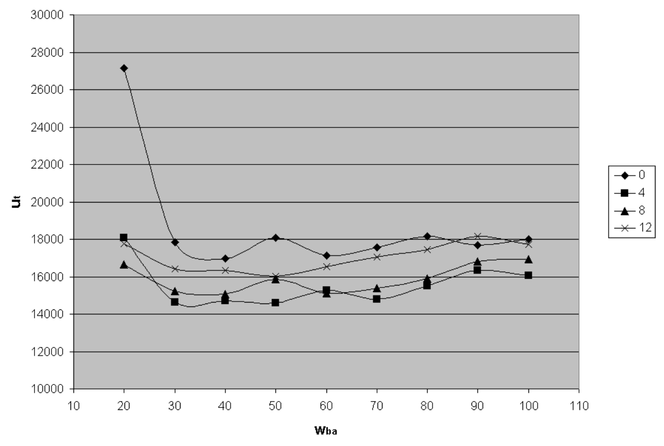


Figure 3.9: The distribution of progress level durations $d(P)$ for the baseline trials.

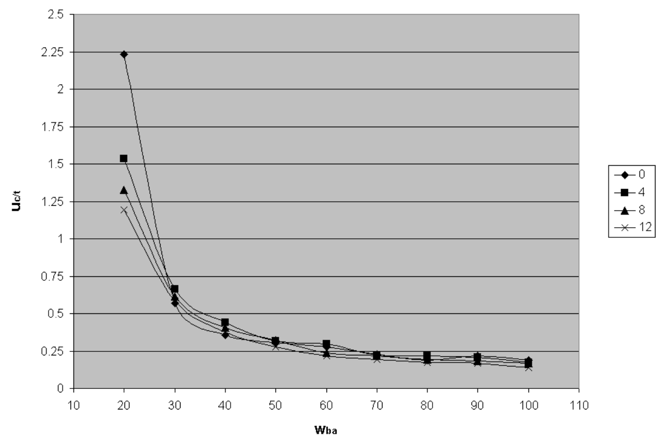
where each trial used different random numbers. The sums of all durations that lie within the intervals $[0, 9]$, $[10, 99]$, $[100, 999]$, and $[1000, \infty]$ are averaged over the 30 trials, and their averages are depicted in Figure 3.9, with error bars denoting the standard deviations. The distribution suggests that the greatest portion of the time is spent by the system in achieving goals of medium difficulty, rather than many goals that are simple, or a few goals that each take a large number of time steps to complete. Furthermore, the error bars show that between trials, there is a progressively greater variability in the amount of time spent as goal difficulty increases.

3.4.2 Repulsion and Randomness

Here, the effects of the block avoidance force $\mathbf{F}_{ba}$ and the random motion force $\mathbf{F}_{rm}$ are investigated without the use of collective movement influences (neighbor cohesion $w_{nc} = 0$ and neighbor alignment $w_{na} = 0$). This allows the effects of these forces to be observed within a simpler dynamical context and establishes baseline



a.



b.

Figure 3.10: The effects of w_{ba} (the block avoidance coefficient) upon (a) the average completion time μ_t and (b) the average collision rate $\mu_{c/t}$. Each curve corresponds to a distinct value $w_{rm} \in \{0, 4, 8, 12\}$, as shown in the legend.

values for their relative weights for use in later experiments.

Figure 3.10 shows the average number of time steps μ_t necessary to complete a trial, as well as the average number of collisions per time step $\mu_{c/t}$. (The values σ_t and $\sigma_{c/t}$ for the given data points fall in $[1241.0, 8790.0]$ and $[0.02, 1.87]$, respectively). As would be expected, $\mu_{c/t}$ decreases with larger values of w_{ba}, although in the higher range of avoidance coefficients the rate of change is quite small. The effect on completion time is slightly more complex with the relationship being flatter (which indicates that this variable is fairly insensitive to parameter changes over a broad range), but somewhat non-monotonic. If block avoidance is too weak ($w_{ba} = 20$), performance is rather poor, especially for low noise levels. For $w_{rm} = 0$ and $w_{rm} = 4$, of the 30 trials, eleven and two were unsuccessful, respectively, with such weak repulsion. In one trial, there was difficulty in completing a corner section of the walls due to a large number of blocks interfering with each other. Another trial did not complete because the last non-stationary large block moved on top of the roof in a limit cycle, bouncing between two edges and unable to break the trajectory, due to a complete lack of noise. (The same issue occurred in a different trial, where $w_{rm} = 0$ and $w_{ba} = 50$; apart from this, all trials were successful for $w_{ba} > 20$). In the remaining unsuccessful trials, there was too much interference between blocks to finish assembling the left column staircase. For w_{rm} values 8 and 12 the effects of a low avoidance weight were less drastic: all trials ran to completion and the average collision rates for these trials were also considerably better at $w_{ba} = 20$. Thus, it appears that a higher degree of noise can compensate somewhat for low avoidance, allowing the system to resolve difficult situations (i.e., blocks repeatedly colliding, or otherwise interfering with each other) that inevitably arise when there is a lack of sufficient repulsive influences between blocks. It should also be stated that for $w_{rm} = 0$, the standard deviation values σ_t were particularly high. Thus, noise is also beneficial in that it provides a greater degree of reliability, "smoothing" the differences between individual trials.

Performance thus tends to be optimized in the moderate range of w_{ba} and w_{rm} values. While small random forces can be beneficial in certain situations (in Figure 3.10a, the curves for $w_{rm} = 4$ and $w_{rm} = 8$ generally appear below the other two curves), excessive noise does not help and appears to interfere with the functioning of the system, increasing completion time somewhat even if all trials successfully finished. For increasing values of w_{ba}, a gradual increase in the average completion time is also observed in all curves, except for $w_{rm} = 0$. The effect is not very severe because $\mathbf{F}_{ba}$ is deactivated during goal pursuit.

As seen in Figure 3.10a, the 30 trials with the best observed μ_t resulted from the values $w_{ba} = 50$ and $w_{rm} = 4$. These are henceforth referred to as the *baseline trials* and are, in fact, the trials discussed in Section 3.4.1. These trials were also characterized by absent collective influences ($w_{nc} = 0$ and $w_{na} = 0$) and 164 small, 150 medium and 27 large blocks, while other parameters were set in the course of system development in order to achieve qualitatively reasonable performance and are listed in Table 3.3. The parameter values used for the baseline trials are referred to below as *baseline values*.

3.4.3 Collective vs. Independent Movements

Experiments are now presented that examine the effects on self-assembly of the neighbor cohesion force $\mathbf{F}_{nc}$ and the neighbor alignment force $\mathbf{F}_{na}$, which are responsible for making blocks move in a collective fashion (i.e., like a bird flock, fish school or animal herd). Earlier work has shown that in some applications where multiple agents have a limited range of vision, the use of such forces can improve the performance of agents through a "pull effect" [94], where the first agents to detect some target location pull the rest of the group towards it, as shown in Figure 2.3.

Table 3.4: Collective vs. Independent Movements

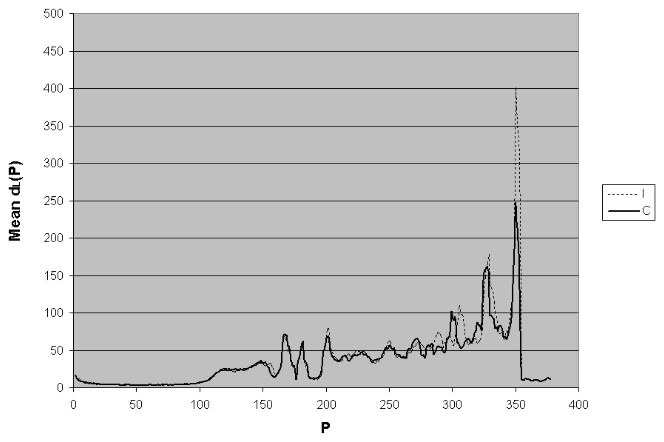
Measure	Centering			No Centering		
	I	SC	C	I	SC	C
μ_t	14604.1	13575.9	13461.0	67666.5	45781.3	43338.4
σ_t	1586.5	1317.5	2101.5	7982.7	7528.8	6443.7
$\mu_{c/t}$	0.32	0.35	0.37	0.03	0.07	0.07
$\sigma_{c/t}$	0.06	0.04	0.08	0.01	0.02	0.02

To determine whether this benefit applies in the context of self-assembly, a set of 30 trials was executed with the neighbor cohesion weight $w_{nc} = 0.6$ and the neighbor alignment weight $w_{na} = 0.6$, the remaining parameters and block quantities set at baseline values. Under the presence of these forces, moving blocks form into "flocks", which can break apart and reform with relative ease, given the relatively low coefficient values.

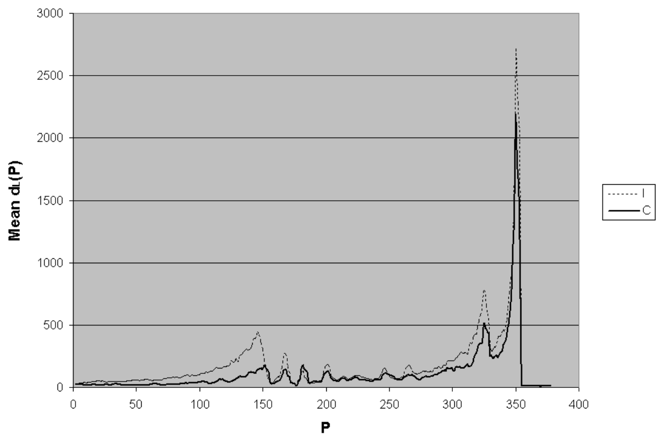
Table 3.4 (left half) gives the difference between the average completion time for the baseline trials, where blocks move independently ("I"), and the collective trials, where blocks move in a coordinated, flock-like fashion ("C"). A two-sample, two-tail t-test assuming unequal variances showed that this difference is statistically significant ($p < 0.05$). This indicates that collective forces benefit the efficiency of the process. Interestingly, this benefit occurs in spite of a slight increase in the collision rate between blocks. This difference between the respective collision rates is also statistically significant ($p < 0.01$).

The experiments described in [94] were set in the context of a search-and-collect task, which is quite different from the problem of locating and reaching specific, dynamically changing goal locations in a 3D structure during self-assembly. Accordingly, it is of interest to determine whether the collective movement forces only help blocks when they are performing the relatively unguided and unconstrained search for the central region of the world, or when they are in the vicinity of the structure as well. So, the experiments were repeated with collective forces applied ($w_{nc} = w_{na} = 0.6$) only when no parts of the existing structure are available for guidance, and $w_{nc} = w_{na} = 0$ otherwise. Denoted as *semi-collective* ("SC"), these trials performed similarly to the collective trials, with no statistically significant differences in μ_t and $\mu_{c/t}$ (left half of Table 3.4). This suggests that cohesion and alignment are primarily useful for guiding blocks towards the structure, and do not appear to have a significant impact on blocks that are at or near the structure. For further exploration, and to determine whether the task can be accomplished in the complete absence of any global information, three additional experiments were performed with independent, semi-collective and collective movements, but with the global centering force always turned off ($w_{gc} = 0$). Table 3.4 (right half) shows that the absence of this force greatly slows down the completion of the process (even though the collision rate is also significantly reduced); however, self-assembly is still ultimately successful in all trials. The use of collective forces, whether at all times or only when no existing parts of the structure are visible (the difference in performance between the collective and the semi-collective trials is not significant) partially yet significantly compensates for the lack of global guidance.

The effects of collective movements can be examined more closely by a comparison of the smoothed average progress level duration curves $d_L(P)$. Figure 3.11a reveals that when the global centering force is present, the chief gain in performance from moving collectively takes place in the completion of the roof, which corresponds



a.



b.

Figure 3.11: Smoothed average progress level duration curves, for independent (I) and collective (C) trials, with (a) and without (b) a global centering force. Note the difference in the vertical scales.

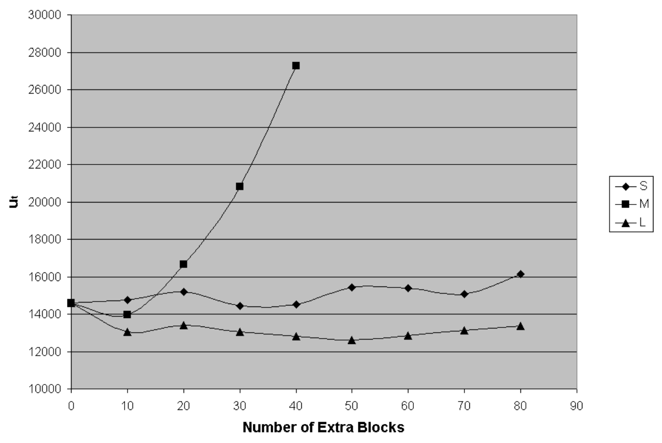
to the largest spike in the progress level duration curve for the baseline trials, appearing around $P = 350$, where the inefficiency results from the low number of large blocks available to find the few remaining goals. Here, collective movements help guide these blocks towards the staircase and the goal locations. On the other hand, when the global force is not available, gains can also be observed at several other stages of the self-assembly process (Figure 3.11b). This can be explained by the fact that there are many points in time when the system suffers from low block availability if the global centering force is not present.

3.4.4 Effects of Extra Blocks

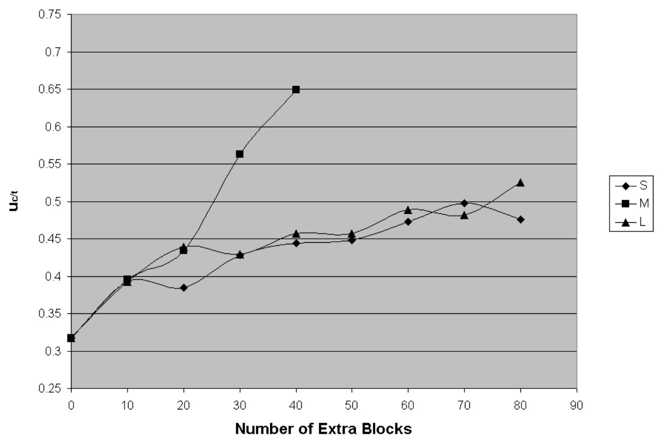
Is there a benefit to the use of extra blocks, beyond the minimal or nearly-minimal required numbers? With the baseline coefficient values established in Section 3.4.2 (no collective forces were used), three sets of experiments were conducted, systematically increasing the quantity of small, medium or large blocks between experiments, while keeping the number of blocks of the remaining two sizes at the original (baseline) values given in Section 3.3. Blocks of different sizes were varied independently in order to reduce the confounding of multiple effects. The initial expectation was that, up to a point, increased numbers of blocks would facilitate the self-assembly process until collisions become excessive.

From Figure 3.12, which shows the average completion times and collision rates for each experiment (σ_t and $\sigma_{c/t}$ for all experiments are in $[1168.3, 23513.4]$ and $[0.06, 0.24]$, respectively), it is immediately apparent that the introduction of extra blocks can have very different effects depending on their size, although in all three cases there is an expected increase in the collision rates. Raising the number of medium blocks has a strong negative effect on the performance of the system: μ_t and $\mu_{c/t}$, as well as σ_t and $\sigma_{c/t}$ (not shown) rise quite rapidly. On the other hand, the introduction of even just 10 additional large blocks has a small but significant positive impact. The effect of the number of small blocks is less pronounced. Insight into why such different effects occur based on block size can be obtained as follows.

Figure 3.13a shows that at progress levels that correspond to the placement of the lower parts of the door frame ($180 \leq P \leq 183$), increasing the number



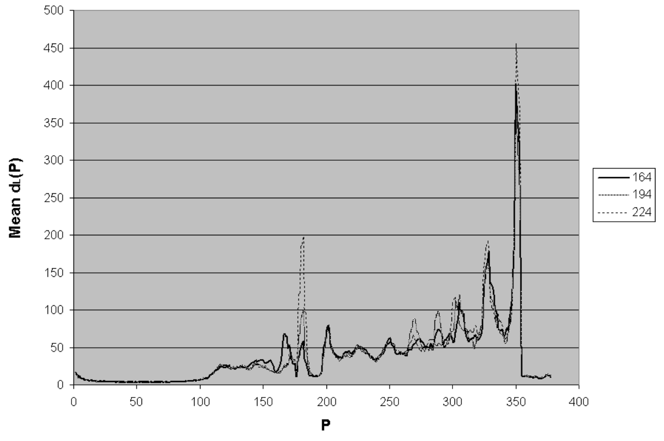
a.



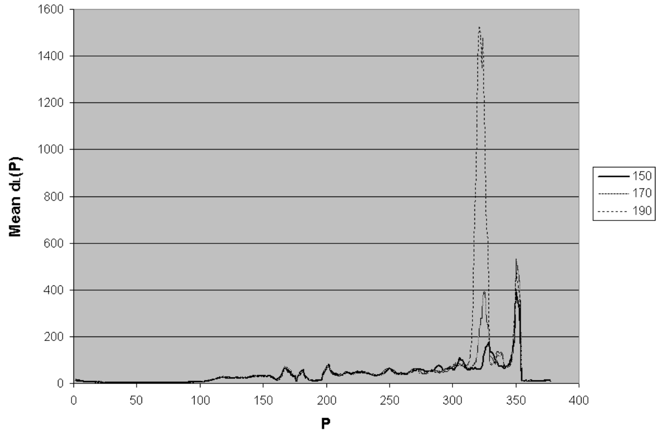
b.

Figure 3.12: The effects of numbers of extra blocks upon (a) μ_t and (b) $\mu_{c/t}$ for small (S), medium (M) and large (L) blocks. To preserve scale, only 5 data points are shown for medium block variations, but there were additional experiments with 50 ($\mu_t = 31116.7$, $\mu_{c/t} = 0.68$) and 60 ($\mu_t = 49497.4$, $\mu_{c/t} = 0.84$) extra blocks.

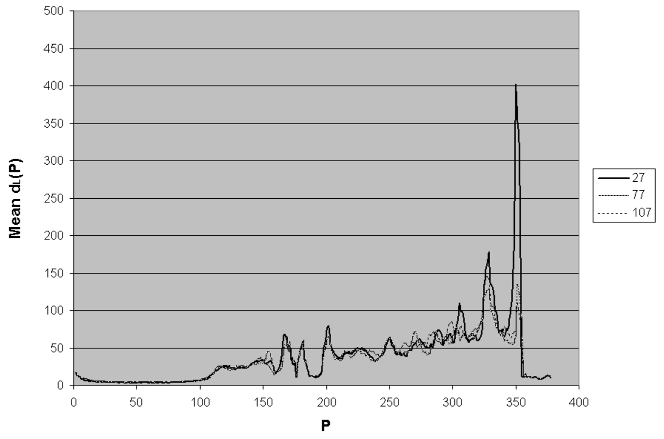
of small blocks causes greater interference with the medium blocks, as is evident in the growing curve peaks. On the other hand, at slightly lower progress levels ($165 \leq P \leq 170$), the baseline number of small blocks (164) does not allow for a very efficient completion of the columns, as there are few available blocks left, and the presence of additional blocks expedites the process. These two trends oppose each other, so for the most part, the mean completion times do not vary significantly. In the later stages of the process, small blocks are not needed, and the effect of



a.



b.



c.

Figure 3.13: Smoothed average progress level duration curves for select quantities (given in the legend for each graph) of (a) small, (b) medium and (c) large blocks. Note that the vertical scale in (b) is much greater than in (a) and (c).

their quantity is not well-pronounced. There can be some interference between these blocks and blocks that are still actively assembling, but it is generally very mild because most of the former blocks are typically not near the region where the assembly takes place. The small differences between the small block curves in the higher range of P can thus be largely explained by random variation.

Figure 3.13b reveals that the inefficiency arising from increases in the number of medium blocks can be primarily localized to the placement of a large block over the door frame, which appears as a major spike around $315 \leq P \leq 330$. For clarity, only three curves are plotted, but they are representative of the typical trend - the spike becomes more pronounced for progressively higher quantities of medium blocks. This can be attributed to a greater interference with the relatively small number of large blocks, delaying the completion of the door frame. In fact, when 60 additional (210 total) medium blocks were used, two trials did not complete because a large block was unable to reach the top of the door frame before the trial was terminated. At the same time, extra medium blocks do not appear to provide any significant benefit at other stages of the process. The only observed exception to this trend is a set of trials with 160 medium blocks (only 10 extra), where the $d_L(P)$ curve (not shown) is rather similar to the curve for the baseline trials, although the completion of the roof is actually slightly more efficient. However, the difference between the mean completion times for these two neighboring points is not statistically significant, and may once again be attributed to chance.

Finally, Figure 3.13c shows that the introduction of additional large blocks considerably reduces the time necessary to complete the roof (around $P = 350$): as with the use of collective forces, block availability is improved. Further increases in the quantity of these blocks appear to have a marginal benefit at first, with 164 small, 150 medium and 77 large blocks giving the best performance of all combinations attempted.

3.5 Discussion

In this chapter, a methodology was presented for the distributed, local control of complex self-assembly tasks in a continuous environment with various constraints, and using multiple-sized blocks. The approach is based on extensions to existing stigmergic construction methods [2, 48, 49], which allow certain structural features to be expressed in a parsimonious fashion. Because stigmergic rules alone (whether parsimonious or not) are insufficient for self-assembly in non-trivial environments such as the one considered here, the stigmergic model was integrated with a force-based movement control scheme, and a coordination mechanism based on local communication between agents that allows the system to follow high-level ordering constraints on block placement. The effectiveness of the approach was verified on an interesting target structure (a building), and a number of experiments were performed to study the system's dynamics.

Through an analysis of qualitative observations and quantitative performance data (such as duration curves), it was possible to infer how the attributes of the current situation, such as block availability and interference, affect the system's progress within any given trial. Between sets of trials, it was shown how particular parameter settings can affect these attributes. Specifically, it was found that the use of a stochastic component in computing acceleration helps to mitigate situations where blocks persistently interfere with each other. Further, it was observed that coordinated, flock-like motion (which has been shown in the past to improve agents' ability to find goals [94, 95]) also has benefits in the context of self-assembly, by drawing more blocks to the general region where the construction takes place. Finally, by varying the quantities of blocks used, it was discovered that the presence of multiple-sized blocks creates additional challenges, because the interference between blocks of different sizes can be much more severe than the typical saturation effect in the context of identical components [8, 12, 70, 114, 115]. With the exception of a study where simple, 2D center-periphery patterns (e.g., Figure 2.4b) were formed from robots that had preassigned roles (i.e., center vs. periphery) with distinct control mechanisms [99], related past work on self-assembly has generally assumed not only simpler target structures and/or environment, but also, identical components; thus, heterogeneity issues in self-assembly have largely not been addressed. (Another study that dealt with some degree of heterogeneity is [116], but it examined the problem of collective construction rather than self-assembly, and also made use of a simpler, 2D environment). In the system presented here, it proved possible to alleviate these issues through high-level coordination. For example, in delaying the assembly of the building's walls until after the lower parts of the door frame have been placed, interference between small and medium blocks is prevented. However, this comes at a price, because the potential for parallelism in construction is reduced. On the other hand, by allowing wall construction to continue while a large block attempts to cover the door frame, the degree of parallelism is augmented, but at the same time, significant interference is created between medium and large blocks, and this becomes progressively worse if the number of medium blocks is increased. Thus, in designing control methods for self-assembly, it is imperative to consider the tradeoff between maximizing block availability and minimizing interference.

Ultimately, this chapter showed that in spite of the issues raised by environmental complexity and physical differences between the self-assembling components, it is possible to self-assemble a non-trivial structure while using strictly local information. However, there is, at present, no guarantee that the developed methodology will extend to other target structures. Determining whether or not it is generalizable would involve the design of stigmergic and variable change rules for structures besides the building, along with possible modifications to the movement control mechanisms. Based on past experience, this would likely require a substantial amount of additional human effort. Naturally, an important question arises: Is it possible to *automate* the process of generating control mechanisms, given some desired target structure? The next chapter will demonstrate that this can indeed be done, even for the non-trivial environment presented here.

Chapter 4

Automating the Design of Rule Sets

In the previous chapter, control rules were designed manually to guide the self-assembly of a non-trivial, 3D structure (specifically, the focus was on the building in Figure 3.1d,e) from blocks of different sizes, in a continuous environment with constraints such as gravity and block impenetrability. The approach incorporated several distinct techniques from the field of swarm intelligence [17, 53], namely stigmergic pattern recognition, force-based movement control, and higher-level coordination via the use of a limited amount of memory and local message passing. While this approach was successful, the hand-design of control methods for assembling a specific structure proved to be a time-consuming and error-prone process. This raises the question of whether there is a procedure that will take as input a specification of a target structure, and produce as output a set of control rules which can be used to successfully assemble this structure. As discussed in Chapter 2, such rule generation procedures have been developed in the past to work under simpler, somewhat idealized conditions [2, 48, 55, 57]; however, their extension to the more complex environment simulated here is presently an open-ended problem, due to the difficulty of controlling motion in such an environment and the presence of physical constraints. As argued earlier, these constraints impose higher-level sequencing requirements on the steps of the self-assembly process, and the question remains as to how these requirements can be captured and automatically translated into local, low-level behaviors.

To address these issues, this chapter first develops algorithms for computing a partial *order* on the sequence of block placements, such that the target structure can be successfully assembled in spite of the environmental constraints; to restate the example given earlier, the inner parts of a building must be assembled first, if they are enclosed by other parts (e.g., walls). The ordering step is followed by the generation of rules, some of which enforce the computed order at runtime, through local communication and memory manipulation, while others allow for the assembly and disassembly of the individual parts of the structure. Even though self-assembly takes place in a simulated environment, the system's overall dynamics can be chaotic, and it is unrealistic to expect that a set of local control methods can always be found that is *guaranteed* to produce the target structure. This issue is approached by factoring out those aspects of control that can be dealt with in a formal manner, and proving formal, environment-independent properties. Specifically, it is shown that both order generation and order enforcement are correct under reasonable assumptions that are made explicit. Other aspects of the problem, such as movement control, are handled in a more empirical manner, by testing various possibilities over a large number of independent trials. It is experimentally shown that with a few modifications, the force-based movement control mechanisms that were presented in the previous chapter are general enough to handle a diverse range of structures, when combined with the rules generated by the procedure, thus providing an integrated, fully automated approach.

4.1 Automatic Rule Generation

This section presents a well-defined, fully automated procedure for generating a set of rules for the self-assembly of a given target structure. This procedure operates in two major phases: order computation and rule generation. During the former phase, it computes a partial *order* O on the steps of the self-assembly process, which is imposed by the environmental constraints of gravity and impenetrability, discussed in Section 3.1.2. Notably, the assembly and disassembly of temporary staircases must be determined and represented in O. While more general representations (such as directed, acyclical graphs) are possible, for simplicity, O is a list containing parts of the target structure as well as temporary substructures. The order is still partial, because any particular item O_i in O can self-assemble or disassemble with blocks arriving or departing in arbitrary order or possibly in parallel, provided that O_{i-1} has fully assembled/disassembled. In essence, O can be viewed as a partial order plan, specifying when blocks should deposit/remove themselves into/from particular locations; however, the approach presented here is somewhat different from that of typical planners [37]. Rather than performing a potentially exponential search for a satisfactory plan, the ordering algorithms make use of domain-specific heuristics that are based on the known environmental constraints, and allow O to be determined in a tractable manner. Further, O is never used by the individual blocks, being discarded after the distributed rules governing self-assembly are generated by the procedure.

As discussed in Section 3.2, two distinct types of rules are used in controlling self-assembly. *Variable change rules* ensure that the order that is specified by O is actually followed by the blocks during the self-assembly process. Through the use of internal memory variables, they allow blocks to communicate the completion of the various structures in O_i, and to transition to the next stage of the process (where O_{i+1} is assembled or disassembled) after and only after everything that makes up O_i has been completed. In each stage i, blocks follow a distinct second set of *stigmergic rules*, which allow them to assemble or disassemble the structures present in O_i, as well as detect their completion (as shown later). Presently, each O_i is decomposed into a set of one or more rectangular shapes (where all rectangles $R \subseteq O_i$ can assemble in parallel), and appropriate rules are generated for each rectangle. It should be noted that the automatically generated rules lead to a much more "structured" self-assembly process, when compared to the hand-designed rules described earlier. For example, the rules of this chapter do not allow assembly and disassembly to take place simultaneously, as is done for the building column staircases in Chapter 3 (see Table 3.2); in fact, all stages of self-assembly are now disjoint in time.

A self-assembly process driven by automatically generated rules is illustrated in Figure 4.1, which depicts the self-assembly of the bridge from an initially random collection of blocks (as before), but with a seed that now consists of a single block. Figure 4.1a shows the assembly of the bottom layers of the inner columns, which assemble first, because the remaining columns would otherwise partially obstruct access to them. The seed is contained in the central column; in order to begin the assembly of the other two columns (recall that a block following stigmergic rules

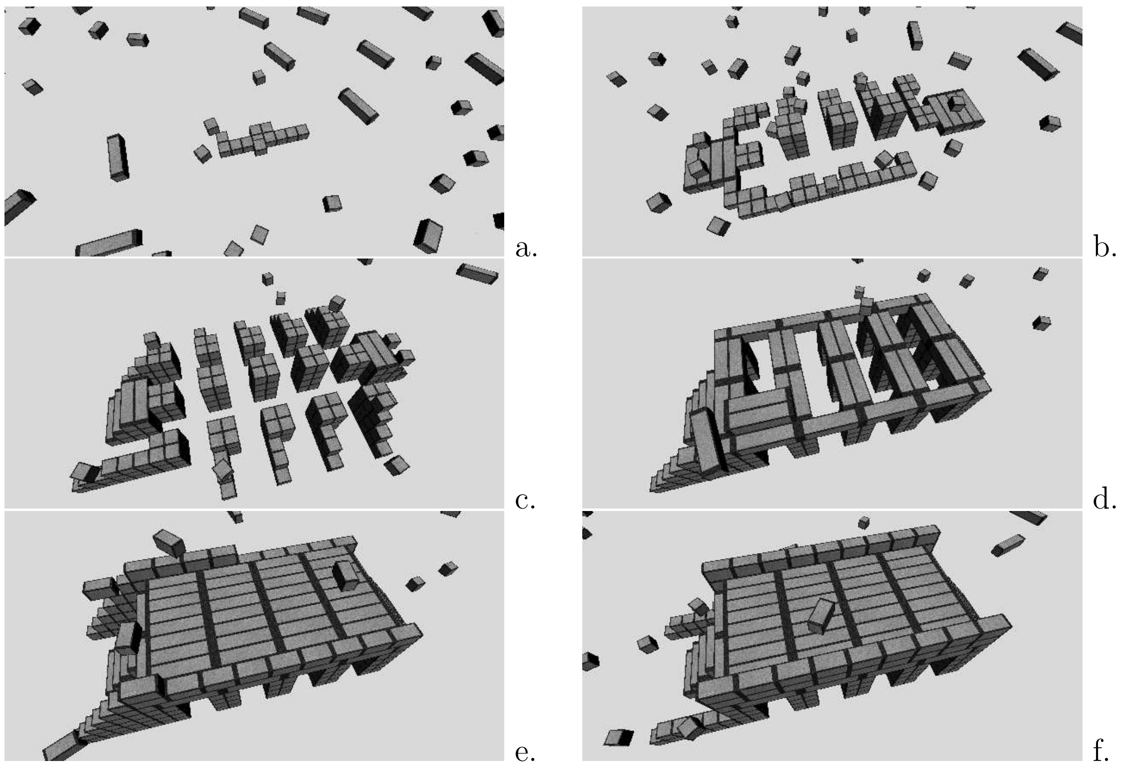


Figure 4.1: The self-assembly of a bridge (Figure 3.1c) at various points in time: (a) $t = 230$: bottom layers of inner columns assembling with connectors; (b) $t = 3664$: bottom layers of outer columns assembling with connectors; (c) $t = 11621$: third layer added to each column staircase; (d) $t = 30346$: bridge surface assembling; (e) $t = 42659$: guard rails assembling; (f) $t = 44455$: guard rail staircases disassembling. Further explanations are given in the text.

cannot place itself in a location that is not adjacent to an already-stationary block), all three are joined by temporary *connectors*, which later disassemble. In Figure 4.1b, connectors similarly join the assembling bottom layers of the outer columns to the completed inner columns and the bottom layers of the permanent staircases (4 large blocks on the left and right). After the connectors disassemble, temporary staircases are incrementally extended in order to allow the columns to grow in height (Figure 4.1c). The completed columns support a frame consisting of large blocks, and additional large blocks deposit themselves over this frame, in order to form the surface of the bridge (Figure 4.1d). The "guard rails" assemble in Figure 4.1e; note that a second staircase has been assembled adjacent to the far left corner of the bridge. The two temporary staircases disassemble in Figure 4.1f, yielding the final target structure (Figure 3.1c).

4.1.1 A Model of Ordering

The following presents a methodology for the problem of computing a partial order on the assembly of a target structure. The goal is to solve the problem such that if this order is followed (via the use of appropriate variable change rules) during the self-assembly process, then the structure can successfully self-assemble in spite of the physical constraints that may exist in the environment. Certainly, the problem is dependent upon specific environmental details, and a solution must therefore be tailored to the environment in question. However, an attempt is made to factor out issues that are not environment-dependent, which allows the creation of a simple mathematical model of ordering in the context of self-assembly, and the design of algorithms with provable properties. At the same time, the presentation will provide less formal, but more concrete explanations of how the model is defined for the specific environment used here (although the proofs are not dependent on these details), and discuss how it is implemented.

For the purposes of this chapter, it is possible to describe a *structure* (such as the *target structures* given in Figure 3.1) as a mathematical set S of *goal locations*. Each goal location $\{(x, y, z), \omega, s\}$ is a triplet that denotes some specific, global position (x, y, z), where a block of size s (small, medium or large), oriented by angle ω is required. It is important to note that there is a distinction between these structural elements and the self-assembling blocks: *any* block of size s can deposit itself at a goal location $\{(x, y, z), \omega, s\}$. A specific set of locations to be occupied by blocks is generally referred to as a *substructure*; examples of encountered substructures include the floor of a building (Figure 3.1e) or a staircase (Figure 4.1c-f). Even though the target structure S is assumed to be connected (in a graph theoretic sense, by location adjacencies), a substructure may consist of several parts that are disconnected from each other; for example, when convenient, the column staircases shown in Figure 4.1c can be viewed as a single substructure. In this case, the substructure is said to consist of several *connected substructures*.

A partial order must be imposed on the locations of S such that during the self-assembly process, when some subset $S' \subseteq S$ of the target structure has already been assembled, it will be possible to eventually assemble the remaining subset $S - S'$ as well. Whether or not this is possible depends on the constraints that are present in the environment. Formally, these constraints are captured by defining the *assembly predicate* $P_A(X, Y)$ over pairs of goal location sets X, Y (where $X \cap Y = \emptyset$), which should evaluate to *true* iff, given that the structure described by Y exists at some point during self-assembly, there is some a priori known order for moving blocks to find and settle into the goal locations defined by X, to form $X \cup Y$ without violating environmental constraints. An order is known a priori if there is an implemented procedure (other than the ordering algorithms discussed below) that is able to automatically recognize this order, or, alternatively, that is able to recognize that X can successfully self-assemble no matter what order is followed during the process. In other words, ordering the assembly of X is a primitive problem, in a problem reduction sense [83]. The *disassembly predicate* $P_D(X, Y)$ is similarly defined, which assumes that $X \subseteq Y$, and which should evaluate to *true* iff there is an a priori known order for blocks that are in locations X to remove themselves from Y, to form $Y - X$.

For the ordering approach to succeed in a concrete environment, it must be possible to efficiently compute these predicates such that their semantics are consistent with the constraints that are present in the given environment, which (in the work presented here) include gravity and block impenetrability. The following environment-specific method is therefore used for computing P_A and P_D:

Method 4.1 (Computation of P_A and P_D) *Given sets of goal locations X and Y, evaluate $P_A(X, Y)$ to true iff (a) X either consists of locations at the same height or is a staircase; (b) every location in X is supported below by another location in X, a location in Y or the ground; (c) no location in X supports a location in Y above it; and (d) a path unobstructed by Y can be found from a remote region of the environment to every location in X (any change in elevation along the path must be 1 unit, since a block cannot ascend or descend a ledge of 2 or more blocks). Evaluate $P_D(X, Y)$ to true iff each connected substructure in X is a staircase and $P_A(X, Y - X) = true$.*

The truth of P_A and P_D thus depends on a number of criteria. Criterion (a) attempts to ensure that an a priori known order exists on X; specifically, it is known that staircases can be assembled layer by layer, from lowest to highest. For arbitrary substructures $X \subseteq S$, the requirement is satisfied if it is known that no matter what order is followed when blocks deposit themselves, X will successfully assemble; i.e., any such order is an a priori known order. It is not required that X can be assembled in *any* order; rather, there should be no situations where it is possible to assemble some subset $X' \subseteq X$, but impossible to subsequently assemble the remaining subset $X - X'$. If X consists of multiple layers, then some orders can lead to these sorts of situations, as illustrated by Figure 4.2a, where a hole of 2 units in depth is developed, and cannot be filled, because a block can only descend one level at a time. For this reason, assembling substructures consist of only one layer. The remaining criteria attempt to further account for environmental constraints. In the implementation, Y is represented by a three-dimensional, grid-like data structure, and algorithms are implemented for determining whether these criteria are satisfied. Criteria (b) and (c) are simple to compute; the purpose of the latter will be explained later in this section. Testing criterion (d) is more difficult, because it requires searching for a path. The problem is simplified by only considering paths that are straight in the horizontal plane. An attempt is made to find such a path from each connected substructure in X to one of the four boundaries of the world.

It is, of course, possible that P_A will return *false* because only "winding" paths exist; however, this is handled in the same manner as situations when P_A cannot be satisfied because no path exists at all. These situations arise very frequently, because blocks are unable to scale sheer vertical surfaces or move, unsupported, through the air. It is nonetheless often possible to assemble a substructure X given Y if Y is modified via the assembly of a staircase Z (Z can also be a group of staircases, if X

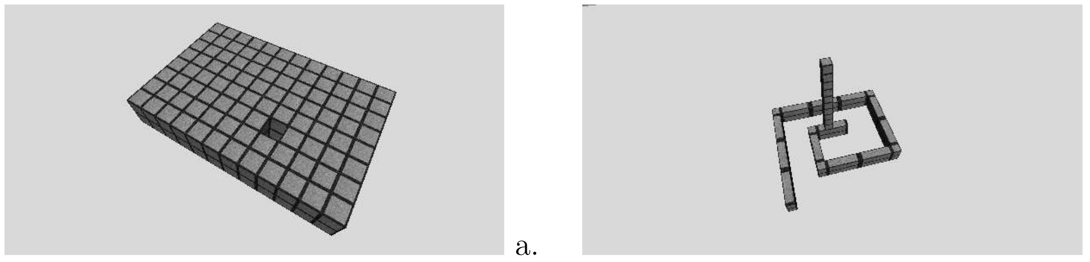


Figure 4.2: Problematic situations: (a) a partially completed structure consisting of two rectangular layers, which has a hole of depth 2 that cannot be filled, because the blocks were not deposited in a correct order (the structure would assemble correctly under some orders, e.g., if the bottom layer is completed prior to starting the top layer); and (b) a structure that cannot be handled by the given implementation of the algorithms, although it is possible to generate rules for it via a more sophisticated implementation, or by hand.

consists of several connected substructures) that is later disassembled. Notably, the self-assembly of temporary substructures for overcoming gravity is observed in living nature: for example, army ants are known to join together to form "ladders", in order to allow the rest of the swarm to safely descend overhanging banks, as reviewed in [5]. Staircases (though not connectors, which are added *after* the order is computed, as discussed in Section 4.1.3) are generated by the *temporary substructure function* $f_T(X, Y, T)$, which is defined as follows:

Definition 4.1 *Given sets of goal locations X, Y and T (assuming that $X \cap Y = \emptyset$), and predicates P_A and P_D, let the temporary substructure function $f_T(X, Y, T)$ be a function that returns a set of locations Z such that $X \cap Z = Y \cap Z = \emptyset$ and $P_A(Z, Y) \wedge P_A(X, Y \cup Z) \wedge P_D(Z, X \cup Y \cup Z) = true$, only if such a set exists, and FAIL otherwise. Furthermore, if $P_A(X, Y) = true$, then $f_T(X, Y, T) = \emptyset$. The argument T is optional.*

Formally, the return value Z is a set of goal locations, which constitute one or more staircases in the implementation. These staircases must satisfy three conditions, expressed as conjuncts of predicate terms. The first conjunct $P_A(Z, Y)$ states that it should be possible to assemble the staircases, given that Y already exists; the second conjunct $P_A(X, Y \cup Z)$ states that once the staircases have been assembled, we should be able to assemble the desired substructure X, and the third conjunct $P_D(Z, X \cup Y \cup Z)$ states that once X has been assembled, it should be possible to disassemble the staircases. The function f_T is required to return $\emptyset$ if the goal locations X can be reached without any staircases. On the other hand, f_T returns *FAIL* to indicate that staircases are necessary, but could not be found. For the moment, the third argument T to f_T is assumed to have the value $\emptyset$, and one can simply write $f_T(X, Y)$. Later, I shall explain how T can contain one or more staircases (generated by earlier calls to f_T), which can be taken into account by the implementation of f_T for a more efficient use of temporary substructures. Ignoring T for the time being, for the environment of Section 3.1.2, f_T can be implemented as follows:

Method 4.2 (Computation of f_T assuming $T = \emptyset$) *Given sets of goal locations X and Y, for each connected substructure of X, extend a staircase from the ground to some location in the substructure, providing a horizontally straight, monotonically ascending path to the substructure, unobstructed by Y. Each staircase is formed as a series of vertical stacks of small block locations, where the top of each stack is a "step". Each stack is supported either by the ground, or by existing locations in Y. The elevation of the surface supporting the staircase must decrease monotonically* *with distance from X. The value Z returned is the union of the generated staircases; if one or more staircases cannot be generated, then FAIL is returned.*

The column staircases depicted assembling in Figure 4.1c are an example of a (disconnected) temporary substructure returned by f_T. As with path finding, the computations are simplified by considering only straight staircases (unlike in Figure 3.7 of the previous chapter, where staircases partially wind around building columns); the implementation of f_T is therefore not complete. For example, in Figure 4.2b, the top of the column could (in principle) be accessed by assembling a staircase that winds with the inner part of the maze, and another staircase (consisting of just one step) that would allow blocks to climb onto the maze (which has height 2); however, since only straight staircases are considered, and the surface beneath a staircase must vary monotonically, *FAIL* would be returned in this case. Still, it is hypothesized that the implementation of f_T is sufficiently good to handle a large class of structures, including the ones depicted in Figure 3.1.

Given a temporary substructure function f_T, Figure 4.3 presents a simple, nondeterministic algorithm for computing an order O on the locations of S, such that if it is followed during self-assembly, then the constraints modeled by P_A and P_D need not be violated. The algorithm, which is given in preliminary form here to illustrate the key ideas (and elaborated in full form subsequently), can be thought to operate in reverse order, relative to an actual self-assembly process and to the output O (which behaves like a stack): it begins with the entire target structure S, and begins to remove subsets from it. At each iteration, some subset (substructure)

```
1. Input $S$.
2. Initialize an empty stack of substructures $O = \emptyset$.
3. While $S \neq \emptyset$:
    (a) Remove some substructure $C$ from $S$ such that $Z = f_T(C, S-C) \neq FAIL$.
        If no such substructure exists, then return $FAIL$.
    (b) If $Z \neq \emptyset$, then push $Z$ onto $O$ as a disassembly substructure.
    (c) Push $C$ onto $O$ as a regular substructure.
    (d) If $Z \neq \emptyset$, then push $Z$ onto $O$ as a temporary substructure.
4. Return $O$.
```

Figure 4.3: Ordering algorithm (preliminary version). An environment-independent, preliminary version of the ordering algorithm for the locations of a target structure S. As given, Step 3a is nondeterministic; in the implementation, it is performed in a heuristic fashion, as explained in the text.

is selected, and removed only if it can be assembled back, either directly or via the addition of a staircase (i.e., if f_T returns a non-empty value). The subset of S removed is called a *regular substructure*, while the generated staircase is called a *temporary substructure* to denote its assembly in the order O, and a *disassembly substructure* to denote its disassembly (the two are identical in a set theoretic sense, but the latter is distinguished form the former in the implementation by a special tag).

Because there are $2^{|S|} - 1$ possible substructures that can be considered for selection from S, Step 3a of the algorithm is performed heuristically, by selecting substructures from a precomputed, unordered partition U of S. To compute U, S is first decomposed into horizontal slices of 1 unit in thickness, in accordance with criterion (a) of Method 4.1. Subsequently, each such slice is further partitioned into its connected substructures. This is done because each separated substructure typically requires its own staircase, as stated in Method 4.2. In executing Step 3a, a single pass is made through U. If a substructure $C \in U$ is reachable directly (i.e., $P_A(C, S - C) = \mathit{true}$) or via the addition of a staircase, then it is selected and removed from U. The substructure that is added to O is the union of substructures removed from U during a given iteration of the preliminary ordering algorithm, and an attempt is made to ensure that these disconnected substructures can be assembled in any order. Given that the implementation of f_T is not complete, failure is a possibility under this heuristic. For example, consider once again the structure depicted in Figure 4.2b. It would still be possible to generate an order for this structure, if both layers of the maze were partitioned such that the inner parts and the column could be built first, without the outer parts preventing the construction of a straight staircase. However, because each layer of the maze is a connected substructure, it is not decomposed further, so the maze has to be built in its entirety, before the column is begun; the implementation of the algorithm would thus fail on this structure. More sophisticated heuristics, which would selectively decompose certain connected substructures further, could mitigate this issue (as would a better implementation of f_T), but they are not explored here.

Does the success of the algorithm depend on the specific choice made by Step 3a? It is argued that due to criterion (c) in Method 4.1, this choice does not affect whether the algorithm will return an order or fail, because the removal of a substructure from S will not cause other substructures in S to lose support. (The issue of nondeterministic failure is examined more formally in Section 4.3.1). In other words, any legal (according to P_A and P_D) choice of substructure C to be removed from S can be made, without causing a problem in later iterations. The algorithm therefore does not perform backtracking during its search, and its environment-specific implementation executes in polynomial time with respect to the size $|S|$ of the target structure.

To further illustrate the algorithm's operation, consider its application to the building shown in Figure 3.1d,e. The top roof layer is removed in the first iteration, because it is the only part of the structure not currently supporting anything above itself; however, due to gravity, a staircase must be generated for it. Subsequently, wall substructures are removed, layer by layer, starting at the top and working downwards; note that this automatically takes care of door placement. Once these are gone, column substructures can be accessed, and are removed in a similar manner. The floor is removed in the last iteration, and so becomes the first substructure in the stack O, with no staircase being necessary. For the other substructures, the assembly and subsequent disassembly of staircases is specified. In general, the following property will always hold:

Property 4.1 *Let $O = (C_1, C_2, \ldots, C_n)$ be an order returned by the preliminary ordering algorithm. Then, for $1 \leq i \leq n$, if C_i is a temporary substructure, then C_{i+1} is a regular substructure, and $C_{i+2} = C_i$ is a disassembly substructure. Likewise, if C_i is a disassembly substructure, then C_{i-1} is a regular substructure, and $C_{i-2} = C_i$ is a temporary substructure. Furthermore, for $1 \leq i < n$, $C_i \cap C_{i+1} = \emptyset$, except*

```
1. Input $S$.
2. Initialize an empty stack of substructures $O = \emptyset$.
3. Initialize an empty temporary substructure $T = \emptyset$.
4. While $S \neq \emptyset$:
    (a) Remove some substructure $C$ from $S$ such that $Z = f_T(C, S - C, T) \neq$
        $FAIL$. If no such substructure exists, then return $FAIL$.
    (b) Choose a subset $W \subseteq T \cap Z$ such that $P_D(Z - W, S \cup Z)$ and $P_A(T -$
        $W, S \cup W)$.
    (c) If $T - W \neq \emptyset$, then push $T - W$ onto $O$ as a temporary substructure.
    (d) If $Z - W \neq \emptyset$, then push $Z - W$ onto $O$ as a disassembly substructure.
    (e) Push $C$ onto $O$ as a regular substructure.
    (f) Set $T = Z$.
5. If $T \neq \emptyset$, then place $T$ as a temporary substructure at the beginning of $O$.
6. Return $O$.
```

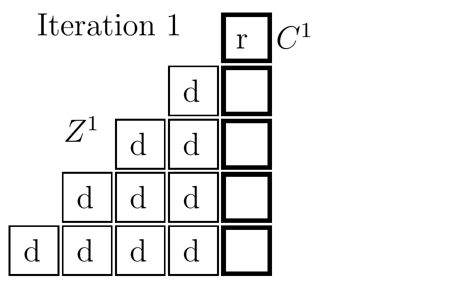
Figure 4.4: Ordering algorithm (full version). The full ordering algorithm for the locations of a target structure S is more involved than the preliminary version, but makes a more efficient use of temporary substructures. Under the given implementation, W can be set equal to $T \cap Z$ in Step 4b.

(possibly) if C_i is a disassembly substructure, and C_{i+1} is a temporary substructure.

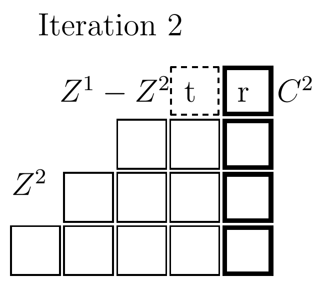
The following section presents a more sophisticated ordering algorithm, which allows parts of staircases to be reused between stages of the self-assembly process.

4.1.2 The Reuse of Temporary Substructures

Orders produced by the algorithm given in the previous section have an inherent inefficiency: any temporary substructure is completely disassembled after



$O^1 = (\text{assemble } C^1,\ \text{disassemble } Z^1)$



$O^2 = (\text{assemble } C^2,\ \text{assemble } Z^1 - Z^2,\ \text{assemble } C^1,\ \text{disassemble } Z^1)$

Figure 4.5: A graphical representation of the substructures handled by the full ordering algorithm during its first two iterations, when applied to a column structure. Locations belonging to regular, temporary and disassembly substructures are marked with letters r, t and d, respectively. Superscripts indicate iteration number. The order computed by the end of an iteration is given at the bottom.

the assembly of a regular substructure, even if it can be reused (at least partially) for the placement of the next regular substructure. For example, suppose that the target structure is a single column of some height. Progressively higher staircases would be assembled and disassembled for each layer of the column (except the bottommost), if we were to use the algorithm of Figure 4.3. It would presumably be much more efficient during self-assembly to incrementally add layers to a single staircase, as needed, and not disassemble it until the column is completed. The full ordering algorithm, given in Figure 4.4, allows this approach. The key idea behind its operation is that when a temporary substructure is generated in a particular iteration, the decision regarding when parts of this substructure should be assembled is postponed until later iterations, which correspond to earlier stages of the self-assembly process. Conversely, in a given iteration, the temporary substructure that is generated may depend on the temporary substructures that were generated in earlier iterations. For this reason, the variable T is used to record the output of the temporary substructure function f_T from the *previous* iteration, and is passed to f_T in the current iteration. While Definition 4.1 and Method 4.2 ignore T, the method can be extended to return a value Z that has many locations in common with T, with the idea that the blocks at these locations can be reused without being disassembled and subsequently reassembled. The implementation of f_T used by the full ordering algorithm can be described as follows:

Method 4.3 (Computation of f_T with optional argument T) *Given sets of goal locations X, Y and T, for each connected substructure of X, determine whether it is possible to reach this substructure if the top horizontal layer is removed from some staircase in T. If so, then use the staircase obtained by removing the top horizontal layer; if not, then generate a staircase as described in Method 4.2, ensuring that there is no overlap with any staircase in T. The value Z returned is the union of the generated staircases; if one or more staircases cannot be generated, then FAIL is returned.*

To illustrate the operation of the full ordering algorithm, its application to a simple column structure is presented, as an example. The column is formed via 5 blocks that are stacked on top of one another (unlike the columns of the bridge and the building in Figure 3.1, where each layer of the column consists of multiple blocks). The column blocks are denoted with bold lines in Figure 4.5, which illustrates the first two iterations of the algorithm:

Iteration 1: C^1 is the top layer of the column (location r in the left side of Figure 4.5); T^1 is initially $\emptyset$; Z^1 is a staircase with 4 layers (d in Figure 4.5), created in Step 4a to reach C^1; and W^1 is $\emptyset$ in Step 4b. Since $T^1 - W^1 = \emptyset$, the algorithm does not specify the assembly of the staircase just yet (i.e., a temporary substructure is not placed in O). On the other hand, in Step 4d, $Z^1 - W^1 = Z^1$, so the entire staircase Z^1 used to reach the top of the column is placed as a disassembly substructure in O. In Step 4e, the top layer C^1 of the column is added to O.

Iteration 2: C^2 is layer 4 of the column; $T^2 = Z^1$ is the staircase generated in the previous iteration; Z^2 is computed by f_T in Step 4a to be just like Z^1, but without the top horizontal layer (i.e., Z^2 contains 3 layers); and W^2 is set in Step 4b to $T^2 \cap Z^2 = Z^2$. The top horizontal layer of the staircase Z^1 from the previous iteration is $T^2 - W^2 = Z^1 - Z^2$, and Step 4c of the algorithm places it as a temporary substructure in O; Step 4b ensures that this substructure can be assembled, as claimed formally later in this chapter. Because $Z^2 - W^2 = \emptyset$, no disassembly substructure is specified. Finally, in Step 4e, C^2 is added to O.

In this fashion, further iterations of the algorithm will specify the placement of successively lower horizontal layers of the staircase, along with lower layers of the column, in earlier stages of the assembly process. In the last (fifth) iteration, f_T will return $\emptyset$, as the bottom layer of the column can be deposited without a staircase. In fact, in the environment considered here, the condition of Step 5 will never hold, because the regular substructure removed from S in the last iteration of the algorithm (and thus the first regular substructure to be assembled) will always be a ground substructure, which requires no staircase; thus, Z will be $\emptyset$ in Step 4a, and $T = Z = \emptyset$ will hold after Step 4f.

In general, if some regular substructure C^{i-1} is removed by the full ordering algorithm in the iteration just prior to the current one, then the algorithm commits to having $T^i = Z^{i-1}$ as a temporary substructure for C^{i-1}, and current and future iterations must honor this commitment. In the current iteration, another regular substructure C^i is removed, and $f_T(C^i, S^i - C^i, T^i)$ generates a new temporary substructure Z^i. Because the algorithm operates "in reverse", C^{i-1} will be assembled after C^i; therefore, after C^i is assembled, it is necessary to transform the temporary substructure Z^i into T^i. To do this, the algorithm selects a subset $W^i \subseteq T^i \cap Z^i$, which will consist of locations that are common to both temporary substructures, and can remain unchanged. The idea is to first disassemble $Z^i - W^i$, given $S^i \cup Z^i$, and then assemble $T^i - W^i$, given $S^i \cup W^i$, which will result in $S^i \cup T^i$. The terms $P_D(Z - W, S \cup Z)$ and $P_A(T - W, S \cup W)$ in Step 4b ensure that this will be possible to do during self-assembly. The trivial value $W^i = \emptyset$ is always guaranteed to work; however, with an empty W^i, the full ordering algorithm essentially behaves like the preliminary ordering algorithm. Ideally, $W^i = T^i \cap Z^i$, which allows all locations in common to both temporary substructures to remain in place. Theoretically, this is not always possible to achieve, and it may be necessary to use a W^i with a smaller (possibly zero) cardinality. However, these situations generally do not arise in the given implementation: as outlined in Method 4.3, f_T is implemented in such a way that each staircase in Z^i either has no locations in common with any staircase in T^i, or is just like some staircase in T^i but without the top layer, and it is typically straightforward to assemble this top layer, given the rest of the staircase. Due to the incremental assembly of staircases, Property 4.1 does not extend to the full ordering algorithm, but a weaker version exists:

Property 4.2 *Let* $O = (C_1, C_2, \ldots, C_n)$ *be an order returned by the full ordering algorithm. Then, for* $1 \leq i \leq n$*, if* C_i *is a temporary substructure, then* C_{i+1} *is a regular substructure. Likewise, if* C_i *is a disassembly substructure, then* C_{i-1} *is a regular substructure. Furthermore, for* $1 \leq i < n$*,* $C_i \cap C_{i+1} = \emptyset$*, except (possibly) if* C_i *is a disassembly substructure, and* C_{i+1} *is a temporary substructure.*

4.1.3 Postprocessing

For convenience of implementation, the predicates P_A and P_D (as computed via Method 4.1) and thus the ordering algorithms do not take into account every detail of the self-assembly process, such as the interference that can occur between blocks of different sizes (Section 3.4.4), or the assumption (underlying stigmergic rules) that substructures are topologically connected. Thus, the order O returned by an ordering algorithm undergoes additional refinement over a number of postprocessing steps. Like the algorithms themselves, these steps are implemented to execute in time that is polynomial in the size $|S|$ of the target structure.

First, each staircase (or group of staircases) that was generated during an algorithm's operation is decomposed further into horizontal layers. The layers are ordered in O to be assembled from lowest to highest, and disassembled from highest to lowest. For the full ordering algorithm, the temporary substructures oftentimes consist of single layers, and need not be decomposed further. For example, the 4 layers of the staircase Z^1 in Figure 4.5 appear individually as separate temporary substructures in the final order O. However, Z^1 also appears as a single disassembly substructure at the end of O, and is replaced with 4 disassembly substructures, each corresponding to a distinct layer, which is assembled or disassembled in a separate stage. Since staircases thus have an a priori known order, they satisfy P_A and P_D.

Subsequently, any substructure in O (except the first) that consists of locations of different sizes (recall that there are small, medium and large blocks) is partitioned into two or three substructures, such that each substructure contains only like-sized locations. In the previous chapter, it was shown that in some situations, severe interference can occur between blocks of different sizes; for example, in assembling the building (Figure 3.1d,e), medium blocks can make it difficult for a large block to place itself over the door frame. The incidence of such problems can be significantly reduced by ensuring that blocks of different sizes are active in different stages.

An exception to this separation by size is made for the very first stage of the self-assembly process, because all substructures assembled at this point in time are on the ground, and movement is still relatively unrestricted. This exception is made for an additional reason: as discussed in the previous chapter, the stigmergic mechanisms that drive the self-assembly process require the presence of blocks that are already stationary; hence, construction begins with a seed, which now consists of a single block. If only blocks of a particular size can assemble in the first stage, then it is possible that there will be a substructure that cannot be commenced because it is disconnected from the seed. For some target structures, this situation is unavoidable even if blocks of all sizes are active simultaneously; for example, recall that the bottom layers of the columns of the bridge (Figure 4.1a-b) are disconnected from each other, and only one of them contains the seed. To allow the stigmergic self-assembly of such structures, an additional postprocessing step is necessary, where another type of temporary substructure, called the *connector*, is created, and added to the order O. Each connector is either a row of small blocks, or two rows of blocks joined together in an "L"-shape, and is able to connect two regular substructures, as shown in Figure 4.1a-b, thus enforcing their relative alignment. Somewhat analogous behavior is observed in comb construction by honeybees, which use their own bodies to form chains between combs, in order to ensure that they are parallel to each other (reviewed in [5]). If the first substructure in O consists of several separated substructures (e.g., Figure 4.1a), then they are assembled concurrently with connectors that join them all to form one connected structure, which can stigmergically self-assemble, with every block depositing itself adjacent to a block that is already stationary. The connectors disassemble after the substructure has been built. Sometimes, as Figure 4.1b illustrates, it is necessary to use connectors in later stages of the process as well, to connect separated ground substructures to the parts of the structure that appear earlier in O, and have therefore already been built. Substructures that are above ground level never require connectors, because they are always adjacent to the locations that support them.

A final postprocessing step is the decomposition of all substructures $C_i \in O$ into rectangular shapes that consist of locations that have the same orientation. As with the addition of connectors, this step is necessary not due to the constraints that exist in the environment, but rather, due to the stigmergic nature of the available control mechanisms. As discussed later, in Section 4.1.5, it is straightforward to produce a set of stigmergic rules for rectangles of arbitrary lengths and widths (since all substructures in O are now 1 unit in thickness, the thickness of each rectangle is also 1), and the decomposition thus prepares the procedure for the rule generation phase. In performing the decomposition, a greedy approach is taken: at every iteration of its outer loop, the decomposition algorithm selects and removes the largest rectangle that it can find within the given substructure C_i, until the substructure is empty. While this method can in some cases be suboptimal, in that it is not guaranteed to return a minimal set of rectangles (an optimal strategy is given in [85]), it is used for ease of implementation. Furthermore, an empirical study [77] of a non-greedy, but otherwise somewhat similar strategy suggests that the decomposition may be close to optimal.

Given the computed order $O = (C_1, C_2, \ldots, C_n)$, it is now possible to describe methods for generating distributed rules that enforce this order during the self-assembly process, and that allow the assembly and disassembly of individual substructures. These tasks depend on the presence of appropriately generated variable change rules and stigmergic rules, respectively. As before, variable change rules represent "control knowledge" about the order in which substructures should assemble or disassemble, and thus coordinate the self-assembly process; stigmergic rules represent patterns in a partially assembled structure, and indicate that a block should deposit itself in a particular location. The following two sections describe the correspondence between the rules and the order O, and outline their use at runtime.

4.1.4 Variable Change Rule Generation

The enforcement of an order $O = (C_1, C_2, \ldots, C_n)$ essentially requires detecting the completion of all rectangles C_{ij} within each C_i (recall that C_i is decomposed into rectangles that can assemble in parallel), communicating this information among blocks, and assuming an appropriate set of low-level behaviors (such as movement dynamics, stigmergic pattern matching, etc.) for assembling or disassembling the next union of rectangles C_{i+1}. The coordination of such tasks is accomplished through the manipulation of variables internal to a block via variable change rules, and this section will give a description of these rules and the variables that they affect. Later, I will present formal assumptions about the nature of the self-assembly process in the environment, and prove that under these assumptions, the coordination scheme is correct in the sense that it allows the eventual completion of the target structure S.

Here, the set of available blocks is denoted by B. In a physical sense, some blocks in B will eventually correspond to the elements of S (i.e., specific goal locations within the target structure); however, the blocks in B are active agents with some degree of autonomy over their behavior, and they have no preassigned final locations in S. For blocks $a, b \in B$, the predicate $P_C(a, b, t)$ is defined to be *true* iff a and b are able to communicate during (discrete) time step t. In the system studied here, this means that a and b are proximate enough to be in each others' local neighborhoods of visibility, and can thus read each others' memory. It is assumed that P_C is commutative; i.e., $P_C(a, b, t) = P_C(b, a, t)$.

For each substructure $C_i \in O$, a set of events E_i is now defined, where each event $e_j \in E_i$ is a mathematical abstraction that denotes the completion (i.e., full assembly or full disassembly) of the j^{th} rectangle C_{ij} within substructure C_i. Through stigmergic interactions, which are discussed in more detail in the next section, a "signal" propagates through C_{ij} from one of its corners, and the block b at the opposite corner will only receive this signal once all blocks are present in C_{ij}; i.e., once C_{ij} is completed. The predicate $P_E(b, e, t)$ is defined to be *true* iff e has been detected by block $b \in B$ prior to or at time t, so if $P_E(b, e, t) = \mathit{true}$, then $(\forall t' \geq t)\ P_E(b, e, t') = \mathit{true}$. It is also assumed that $(\forall a, b \in B)\ P_E(a, e, t) \wedge P_E(b, e, t) \rightarrow a = b$; in other words, a particular event is detected by a single block, as described in the next section. The full list of event sets is denoted by $E = (E_1, E_2, \ldots, E_n)$.

Since self-assembly is inherently a collective endeavor, an individual block b does not generally have control over whether or not some rectangle C_{ij} is completed, and whether or not it detects the corresponding event $e_j \in E_i$. On the other hand, it does have control over its internal state, which consists of memory variables (Section 3.2.4). Recall that among these is a mode variable $m^b(t)$, which has a direct effect on a block's low-level behaviors; importantly, each mode is associated with a distinct subset of stigmergic rules. (The parameter t is used to denote the value of a variable at a particular time step t). The mode value $m^b = i$ for block b corresponds to the stage of the self-assembly process where substructure C_i is assembled or disassembled, and events in E_i are detected. Since these events are detected locally by a single block, but mode changes must occur throughout B, each block b also

```
1. Input $O = (C_1, C_2, \ldots, C_n)$.
2. Construct a list of event sets $E = (E_1, E_2, \ldots, E_n)$ based on $O$.
3. Initialize empty rule sets $R_E = R_C = R_M = \emptyset$.
4. For all $i$ $(1 \leq i \leq n)$:
   (a) For all $j$ $(1 \leq j \leq |E_i|)$:
       i. Add the following event detection rule to $R_E$:
          IF $m^b = i$ AND $P_E(b, e_j, t) = \mathit{true}$ for $e_j \in E_i$ THEN $\mu_j^b = i$.
          // Set message $\mu_j^b$ to the current mode $i$, if detected event $e_j \in E_i$.
       ii. Add the following communication rule to $R_C$:
          IF $m^b = i$ AND $(\exists b' \in B)$ $P_C(b, b', t) = \mathit{true} \wedge \mu_j^{b'} \geq i$ THEN $\mu_j^b = i$.
          // Set message $\mu_j^b$ to the current mode $i$, if a nearby block $b'$
          // knows of event $e_j \in E_i$.
   (b) Add the following mode change rule to $R_M$:
       IF $\forall j$ $(1 \leq j \leq |E_i|)$ $\mu_j^b = i$ THEN $m^b = i + 1$.
       // Transition to the next mode $i+1$, if all events in $E_i$ have been detected.
5. Return $R_E \cup R_C \cup R_M$.
```

Figure 4.6: A procedure for generating variable change rules for a block $b \in B$. Lines beginning with "//" are comments.

has a set of message variables $\mu_1^b, \mu_2^b, \ldots, \mu_l^b$, where $l = \max_{i=1}^{n} |E_i|$, which are used to communicate event detection. Messages do not directly affect a block's low-level behavior, but they act as an intermediate representation between sets of events and mode values.

Communication and mode changes are accomplished via variable change rules, which are easily generated based on the given order O, due to the correspondence between substructures C_i and event sets E_i. The generation procedure, outlined in Figure 4.6, takes as input the order O, and uses its rectangular decomposition to determine the events E that must be detected. It then generates a set of rules $R_E \cup R_C \cup R_M$ that update the variables μ_j^b and m^b at each time step. An *event detection rule* in R_E causes a block to locally signal to others that it detected the completion of some particular rectangle. This signal propagates to the rest of B via some *communication rule* in R_C, as the message value is adopted by neighboring blocks. Unlike the rule in Figure 3.6, generated communication rules make use of the full neighborhood of visibility ($15 \times 15 \times 4$) to find an acceptable block b'. It should be noted that the '$\geq$' in the condition $\mu_j^{b'} \geq i$ within rules in R_C (Step 4(a)ii in Figure 4.6) is necessary for correctness: for example, if there is a block that is more than one mode value "behind" the remaining blocks (this can occur if it has been isolated from the rest of B for a sufficient period of time), then it will be able to eventually "catch up". On the other hand, the condition $\mu_j^{b'} = i$ would not allow this. Finally, once the completion of all rectangles in some substructure C_i has been detected, a mode change is accomplished via a *mode change rule* in R_M. In total, there are $2(\sum_{i=1}^{n} |E_i|) + n$ rules (specific rule numbers for the different target structures are reported in Section 4.4.1), although the n rules in R_M are not constant in size, because each such rule must take into account a variable number of messages; the total size of rules in R_M is proportional to $\sum_{i=1}^{n} |E_i|$. The antecedents and consequents of rules are such that if two rules fire in the same time step, then they will either modify two different variables, or assign the same value to the same variable (the latter can only happen if one of the rules is from R_E and the other from R_C); thus, the prioritization of rules is not an issue, unlike in the previous chapter, where they were hand-designed.

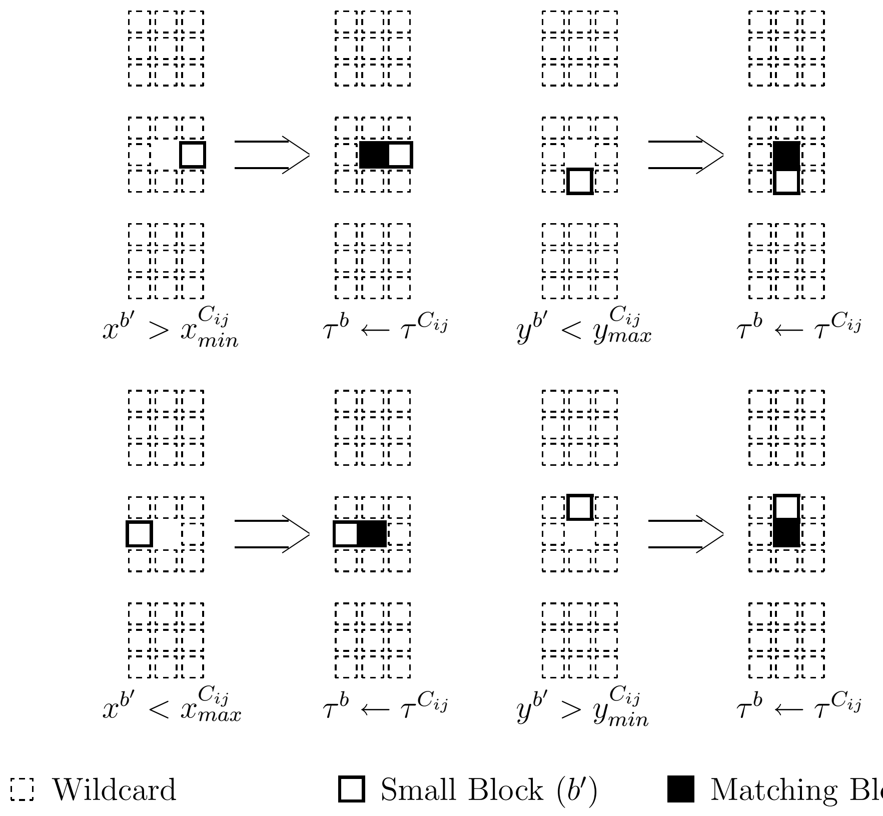

$x^{b'} > x^{C_{ij}}_{min}$
$\tau^b \leftarrow \tau^{C_{ij}}$
$y^{b'} < y^{C_{ij}}_{max}$
$\tau^b \leftarrow \tau^{C_{ij}}$
$x^{b'} < x^{C_{ij}}_{max}$
$\tau^b \leftarrow \tau^{C_{ij}}$
$y^{b'} > y^{C_{ij}}_{min}$
$\tau^b \leftarrow \tau^{C_{ij}}$
Wildcard
Small Block (b')
Matching Block (b)

Figure 4.7: A graphical representation of four stigmergic rules for assembling a rectangle C_{ij} of arbitrary size consisting of small blocks. As in Figure 3.4, b' refers to a matched block that is already present, and b refers to the matching block.

4.1.5 Stigmergic Rule Generation

We now turn to the generation of stigmergic rules governing how the individual substructures are assembled and disassembled, and how the completion of their assembly/disassembly can be detected. Given that all substructures in the order O are further decomposed into rectangles consisting of like-sized locations with identical orientations, the problem is reduced to generating rules that lead to the assembly of individual rectangles in appropriate positions relative to the rest of the structure. For simplicity, the discussion is focused primarily on the generation of rules for small blocks, but the methodology is similar for medium and large blocks as well.

Figure 4.7 shows a set of four *assembly rules* that can be used to build an individual rectangle $C_{ij} \subseteq C_i \in O$ consisting of small blocks; these rules only apply when a block is in mode i. The notation is similar to what is used in Figure 3.4. Most of the rule sites are “wildcards”, which match either the presence or the absence of another block; however, one site in each rule specifies that some block b' must be adjacent to the goal location. Geometrically, the four rules could be viewed as a single rule (by rotational symmetry), which states that in order for a location to be a goal location, there must be a block adjacent directly to its left, to its right, in front, or behind. However, if interpreted in purely geometric terms, the rules would cause unbounded growth; thus, to give the rectangle its required dimensions, memory conditions are placed on the variables $x^{b'}$ and $y^{b'}$ of the already-stationary blocks b' that are adjacent to the goal location, and a rule applies only if the matched block satisfies the memory conditions, as in Chapter 3. The rule generation procedure creates these conditions automatically by translating between the global positions of locations within the structure (which are unknown to any individual block, but are known to the rule generation procedure) and the corresponding coordinate values $x^{b'}$ and $y^{b'}$, which are produced incrementally during the self-assembly process (see Section 3.2.2). The constants $x^{C_{ij}}_{min}$, $x^{C_{ij}}_{max}$, $y^{C_{ij}}_{min}$ and $y^{C_{ij}}_{max}$ correspond to coordinate values for blocks that are on the rectangle's boundary. Each of the four rules counts as a separate rule, as it has a distinct memory condition.

As in Chapter 3, when a block b becomes stationary, its substructure type variable τ^b is also set. The value $\tau^{C_{ij}}$ is determined by the stigmergic rule that b followed to the goal location, and is unique for every rectangle C_{ij}. A memory condition (not shown in Figure 4.7) is also imposed on $\tau^{b'}$ for the matched block b', to ensure that a rule that is defined for one specific rectangle is not accidentally applied towards assembling another. Because each rectangle has a unique value $\tau^{C_{ij}}$ and consists of blocks that are on the same level, a memory condition on $z^{b'}$ is not necessary.

Because the rules generated for a particular rectangle assume the presence of at least one stationary block that belongs to this rectangle, we are faced with the problem of initiating the construction. At the beginning of the self-assembly process, the starting block for one of the rectangles is provided by the seed. For every other rectangle $C_{ij} \subseteq C_i \in O$, a connectivity analysis is performed to determine which rectangles $C_{i'j'} \subseteq C_{i'} \in O$ are directly adjacent to C_{ij}, where $i' < i$ and C_i is a regular substructure or a temporary substructure that has not been disassembled yet. One such rectangle is selected, and a *starter rule* is generated, which places some block b in C_{ij} adjacent to a block b' in $C_{i'j'}$ (which is specified with appropriate memory conditions on $x^{b'}$, $y^{b'}$ and $z^{b'}$). For rectangles in C_1 and certain other substructures, such as the bottom layers of the outer columns of a bridge (Figure 4.1b), no adjacent rectangle can be found among those assembled earlier in the process. In such cases, the procedure searches for adjacent rectangles among those in C_i (i.e., among those assembled in the *same* stage as C_{ij}). A rule is generated for *each* adjacent rectangle, to ensure that stage i completes regardless of the order in which the rectangles are assembled. Adjacent rectangles are guaranteed to be found even for substructures that are disconnected from the rest of the existing structure, because such substructures are joined to the structure via temporary connectors, as

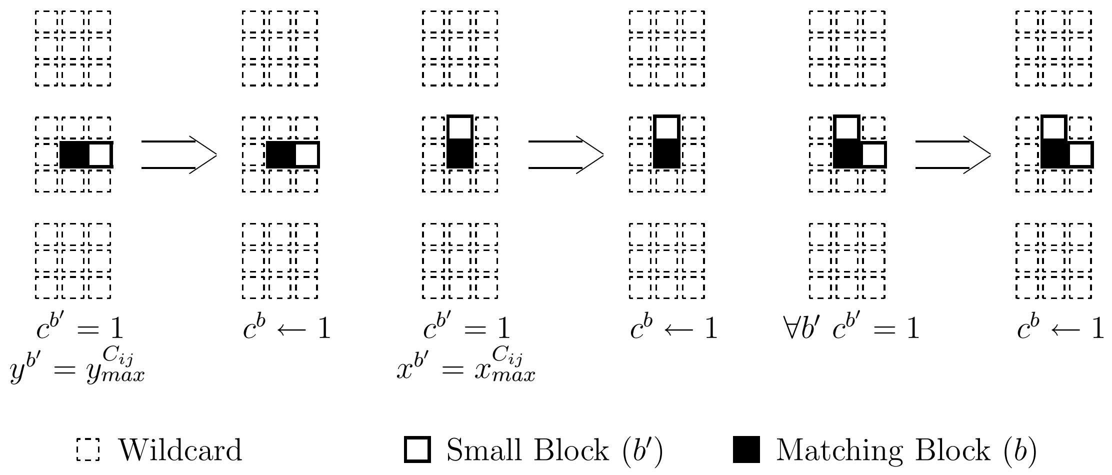


Figure 4.8: A graphical representation of three stigmergic rules for detecting the completion of a rectangle consisting of small blocks. The value $c^b = 1$ propagates from matched block(s) b' to the matching block b, which is already at its goal location.

discussed earlier.

The nature of the generated assembly rules is such that once the first block of a rectangle C_{ij} is deposited via a starter rule, the rest of the rectangle can "grow" from it in a fairly unstructured manner. Presumably, this is beneficial, because fewer ordering constraints are imposed on the placement of individual blocks within a rectangle, and there is a greater potential for parallelism. The downside is that it is not trivial to detect that a rectangle has actually been completed, which is necessary in determining when the collection of blocks is ready to transition to the next building stage. For this purpose, a set of local *completion rules* is generated for each rectangle. These rules were not used in Chapter 3, where the completion of most substructures of the building (Figure 3.1d,e) was always marked with the deposition of a block (or a few blocks) in the same location(s), and could thus be detected more easily. Unlike assembly rules, completion rules are applied by blocks that are already stationary, but unlike disassembly rules, they do not cause a block

b to begin moving again. Rather, when a completion rule applies, it simply changes the value of the *completion flag* c^b from 0 to 1 in the block's memory. Figure 4.8 illustrates that $c^b = 1$ propagates from the far right corner block of a rectangle, towards the near left corner block. The completion detection process is initiated by a variable change rule (not discussed previously), which sets $c^b = 1$ if $x^b = x^{C_{ij}}_{max}$, $y^b = y^{C_{ij}}_{max}$ and $\tau^b = \tau^{C_{ij}}$ (i.e., if b is indeed the far right corner block of the rectangle C_{ij}). The rules in Figure 4.8 are structured such that at any location, if a block b' has $c^{b'} = 0$, or if it is missing entirely, then the blocks immediately to the left and to the front of that location will not set $c^b = 1$. Applying this argument recursively, it is apparent that all other blocks that are some distance to the left and to the front of b' will also have $c^b = 0$. Thus, any "holes" in the rectangle will prevent the value 1 from reaching the near left corner block, and $c^b = 1$ will be set for this block only when the rectangle C_{ij} is indeed completed. This is precisely the concrete manifestation of the event $e_j \in E_i$ (i.e., $P_E(b, e_j, t)$ becomes *true*), and will cause a rule in R_E to fire for b, as described in Section 4.1.4, which will subsequently enable the message $\mu^b_j = i$ to propagate to other blocks as well via rules in R_C.

Detecting the disassembly of a substructure is more straightforward, because all disassembly substructures can be viewed as consisting of $k \times 1$ or $1 \times k$ rectangles (i.e., rows of blocks) that are subsets of staircases or connectors. Such a row is always disassembled from one end to the other, and a single rule in R_E is applied by the last block to begin moving. The disassembly itself is accomplished via two *disassembly rules* (not shown for brevity), one of which causes the leftmost (or nearmost) block to become non-stationary, while the other allows a block to begin moving again if there is no block to its left (or just in front of it). As before, even if a disassembly rule matches, a block does not typically begin to move if there are other blocks moving nearby, thus attempting to avoid colliding with them (see bottom of Figure 3.3), but may do so with a small probability, to avoid waiting for an indefinite period of time.

4.2 Coordination of Movement

Thus far, the focus of this chapter has been on generating stigmergic and variable change rules for the assembly of prespecified target structures, ignoring (until now) the other crucial aspect of force-based movement control. It was found that the exact same set of movement control mechanisms could be reused for a wide variety of structures, as shown experimentally later in the chapter. Essentially, the approach of Section 3.2.3 was employed, with minor modifications. For example, the global centering force $\mathbf{F}_{gc}$ is no longer used; in fact, it was found that for some structures such as the road (Figure 3.1h), which extends far from the center of the world, this force interferes with the construction process. It was also discovered that the movement of blocks atop staircases does not usually benefit from the step down force $\mathbf{F}_{sd}$; so, this force was eliminated as well, and blocks now use only $\mathbf{F}_{su}$ for ascending and descending.

Now, recall from Section 3.2.4 that stair-climbing forces are affected by a block's mode. In Chapter 3, the modes were specified by hand, and each was associated with a particular set of force coefficients (Table 3.3). (Note that in the experiments below, $w_{oa} = 5.0$ and $\delta_{oa} = 0.75$ for all modes; mode-independent parameters remain the same; also, $w_{ba} = 50$, $w_{rm} = 4$ and $w_{nc} = w_{na} = 0.6$). In this chapter, where the modes are specified automatically, a simple mechanism is used for determining the coefficient w_{su} for the force $\mathbf{F}_{su}$. In general, if there are no assembly rules that are applicable under the current mode, then $w_{su} = -5.0$ (thus, a block attempts to descend staircases); otherwise, w_{su} is usually 17.0. However, with a small probability, it can be set to -5.0 again for a period of time, to ensure that non-stationary blocks do not remain on some part of the structure (such as a column) indefinitely, and thus create a more even distribution of blocks atop columns built in parallel. Also, the weight is temporarily set to -5.0 if a block experiences a collision. This mechanism is useful in resolving situations where some blocks wish to climb the staircase, while others attempt to descend it. As ascending blocks and descending blocks collide, the former begin to descend the staircase as well; descending blocks are thus given a "right of way", as before. Unlike in Chapter 3, no special modes such as *on_wall_stair* (Section 3.2.4) are necessary.

Finally, movement control is further simplified by applying $\mathbf{F}_{su}$ indiscriminately; i.e., $T_{su} = \mathbb{Z}$ (where $\mathbb{Z}$ is the set of integers) is used, allowing a stationary block with any value of τ^b to potentially be treated as a "step" to be climbed or descended. Given the aforementioned mechanisms for changing the value of w_{su} at runtime, this was found to be sufficient for assembling a wide range of structures, as verified via experiments.

4.3 Theoretical Results

The rule generation methodology presented in Section 4.1 builds upon earlier stigmergy-based procedures [2, 48] in that it computes and incorporates higher level ordering constraints into a set of distributed rules that govern the self-assembly process. It is relatively straightforward to see how the generated stigmergic rules will, given appropriate movement dynamics, result in the assembly, disassembly or completion detection of specific rectangles. On the other hand, the correctness of the order computation and order enforcement schemes is less obvious. This section presents formal statements about these schemes, deferring proofs to Appendix A.

4.3.1 Correctness of Order Computation

It is shown that under specific assumptions, the ordering algorithms presented earlier are both correct (i.e., if the algorithms do not fail, then the generated order does not violate P_A and P_D) and complete (the algorithms will not fail, if a valid order exists). It should be noted that the derivations do not depend on environmental details, and hold for any predicates P_A and P_D, so long as they satisfy the following condition:

Condition 4.1 *Given sets of goal locations X and Y, the assembly predicate $P_A(X,Y)$ is true if $X = \emptyset$, and is false if $X \cap Y \neq \emptyset$. The disassembly predicate P_D is true if $X = \emptyset$, and is false if $X \not\subseteq Y$.*

This places very mild restrictions on the predicates, allowing their specific definition to depend on the environment in question. The following additional notation is used:

Definition 4.2 *Given $O = (C_1, C_2, \ldots, C_n)$, let the list O^+ be defined such that for $1 \leq i \leq n$, $O_i^+ = C_i$ if C_i is not a disassembly substructure, and $O_i^+ = \emptyset$ otherwise. Similarly, define O^- as the set where $O_i^- = C_i$ if C_i is a disassembly substructure, and $O_i^- = \emptyset$ otherwise. Finally, define $S^{(0)} = \emptyset$, and $S^{(i)} = \left(\bigcup_{j=1}^{i} O_j^+\right) - \left(\bigcup_{j=1}^{i} O_j^-\right)$ for $1 \leq i \leq n$.*

Essentially, $S^{(i)}$ represents the structure that exists in the environment at the end of stage i of the self-assembly process, if O is correctly followed at runtime. The following then holds:

Theorem 4.1 *Let $O = (C_1, C_2, \ldots, C_n)$ be an order returned by the preliminary or the full ordering algorithm, and define $S^{(i)}$ as in Definition 4.2. Then, $S^{(n)} = S$.*

Proof See Appendix A. ■

This theorem states that if O is followed during the self-assembly process, then the structure that exists after the final stage is indeed the target structure S. Thus, the algorithms are said to be correct if O can be followed without violating the constraints that are modeled by P_A and P_D, which is shown to be the case:

Theorem 4.2 (Correctness) *Assuming that the ordering algorithm (preliminary or full) does not fail, let $O = (C_1, C_2, \ldots, C_n)$ be the returned order. Then, $\forall i$ $(1 \leq i \leq n)$ $P_A(C_i, S^{(i-1)}) \vee P_D(C_i, S^{(i-1)}) = true$.*

We now turn to the issue of completeness. First, a simple result is given, which states that Step 4b of the full ordering algorithm (Figure 4.4) cannot fail, because the subset W of a temporary substructure that can be reused in a different stage of the self-assembly process can be set to $\emptyset$ in the worst case (i.e., temporary substructures need not be reused if P_A or P_D do not allow it).

Theorem 4.3 *For any iteration of any run of the full ordering algorithm, the value $W = \emptyset$ will satisfy the terms $P_D(Z - W, S \cup Z)$ and $P_A(T - W, S \cup W)$ in Step 4b.*

This isolates the point of failure to the selection of a substructure C from S (Step 3a in Figure 4.3 and Step 4a in Figure 4.4). Unfortunately, it can be shown that without further assumptions, a satisfactory substructure will not always be found, even if a valid order exists. For instance, the removal of some substructure from S can prevent the removal of another substructure in a later iteration. This can cause the algorithms to fail nondeterministically on some given input S, depending on the specific choice of C that they make during selection. The following theorem gives a sufficient condition for guaranteeing deterministic success (which is a prerequisite for completeness) in the algorithms:

Theorem 4.4 (Determinism) *For any T, any subsets $V' \subseteq V \subseteq S$ and any location $b \in V \cap V'$, assume the following: $[(\exists C \subseteq V)\ b \in C \wedge f_T(C, V - C, T) \neq \mathit{FAIL}] \rightarrow [(\exists C' \subseteq V')\ b \in C' \wedge f_T(C', V' - C', T) \neq \mathit{FAIL}]$. Suppose that some run of the ordering algorithm (preliminary or full) produces an order O for the given input S. Then, there exists no possible run of the algorithm where it will fail on S.*

It should be noted that the assumption of Theorem 4.4 need not always hold to guarantee determinism. Recall that in Method 4.1, it was stated that $P_A(X, Y) = \mathit{true}$ only if the goal locations for X both have support underneath them and support nothing above them. Because of the dependency on supporting locations (i.e., because of criterion (b) in Method 4.1), this formulation does not satisfy the premise of the theorem. However, it can be argued that determinism still holds, because a substructure will never be removed while it is supporting something; in fact, criterion (b) is not necessary for the algorithms' proper operation. On the other hand, if we were only to require that X have support beneath it (i.e., if criterion (c) was not enforced), then the algorithm would (nondeterministically) fail if a substructure is removed from beneath another substructure, and the latter is left without support.

It can be shown that if the preliminary ordering algorithm is deterministic with respect to whether or not it succeeds (as guaranteed, for example, if the assumptions of Theorem 4.4 hold), if f_T is complete, and if there is a valid order on S, then some order will be found:

Theorem 4.5 (Completeness) *Assume that if the preliminary ordering algorithm returns an order O on some input S in some run, then it will return some order on S in any run. Further, assume that the function f_T will return a value other than FAIL whenever one exists (i.e., replace "only if" with "if" in Definition 4.1). Suppose that there exists an order $O = (C_1, C_2, \ldots, C_n)$ on the locations of S such that $\forall i$ $(1 \leq i \leq n)$ $P_A(C_i, S^{(i-1)}) \vee P_D(C_i, S^{(i-1)}) = true$, such that $S^{(n)} = S$, and such that Property 4.1 holds. Then the algorithm will return an order (i.e., will not fail).*

Rather than showing the completeness of the full ordering algorithm directly, the proof is accomplished by making a formal comparison between the two algorithms, and stating that under the appropriate assumptions, they are equivalent in power.

Theorem 4.6 (Equivalence) *Assume that if the algorithm (preliminary or full) returns an order O on some input S in some run, then it will return some order on S in any run. Further, assume that the function f_T will return a value other than FAIL whenever one exists. Then, given a target structure S, the preliminary ordering algorithm will return an order iff the full ordering algorithm will return an order.*

Because the assumptions of Theorem 4.6 include the assumptions of Theorem 4.5, the full ordering algorithm is said to be complete under these assumptions. It should be noted, however, that for both algorithms, completeness is defined in terms of the existence of a valid (in the sense that it does not violate P_A and P_D) order O that satisfies Property 4.1. Theoretically, it is possible to have an environment and a target structure for which valid orders exist, but none of them satisfy Property 4.1; the orders might have temporary substructures that can be assembled or disassembled, but only incrementally. Such an order will not be found by either algorithm, because the conjuncts $P_A(Z, Y)$ and $P_D(Z, X \cup Y \cup Z) = \mathit{true}$ in Definition 4.1 imply the ability to assemble and disassemble (respectively) temporary substructures all at once. In the specific formulation given here, this is not an issue: a whole staircase (or a group of staircases) can satisfy P_A and P_D by criterion (a) in Method 4.1; it is not until the postprocessing stage (Section 4.1.3) that they are further decomposed into layers.

4.3.2 Correctness of Order Enforcement

Given the order O, the coordination scheme implemented by local variable change rules (Section 4.1.4) is correct if it can guarantee that the target structure S will be completed, assuming that O does not violate any environmental constraints, and that other control mechanisms (such as stigmergic pattern matching and the computation of the movement force $\mathbf{F}$) are effective. In order to show correctness, these various assumptions must be formally captured. First, the following is assumed about block communication:

Condition 4.2 (Communication Model) *At some time t, define the sets $H = \{b\}$, where b is some arbitrary block in B, and $G = B - H$. At any time $t' \geq t$, if $P_C(g, h, t') = true$ for blocks $g \in G$ and $h \in H$, then set $G \leftarrow G - \{g\}$ and $H \leftarrow H \cup \{g\}$ (i.e., transfer g from G to H). Then, there exists a time $t'' > t$ where $G = \emptyset$ and $H = B$.*

This condition does not imply that a block b will (at some point in time) be able to directly communicate with any other block in B. However, it implies that if b gains some piece of information, then through appropriate communication between pairs of blocks, this information can eventually be known by all blocks. This is a reasonable assumption, because stationary blocks are usually part of a connected structure, which means that information can propagate to all other stationary blocks, whereas blocks that are non-stationary are expected to eventually move within the vicinity of the structure or each other, and will thus receive information as well, albeit perhaps with considerable delay.

Now, the relationships between events, messages, and modes (Section 4.1.4) are formally captured, as they would be if we assume that the generated order O and the low-level control mechanisms are correct, with respect to the given environment. The following condition is defined:

Condition 4.3 *For all $b \in B$:*

$$(\forall i)(\forall e \in E_i)\ \neg P_E(b, e, 0) \wedge (\forall j)\ \mu_j^b(0) = 0 \wedge m^b(0) = 1$$

So, initially (at $t = 0$), no events are detected; all message values are 0, and all blocks are in mode 1, which corresponds to the assembly of the first substructure C_1 in O. The key assumption is that during the self-assembly process, the low-level behaviors (which are not explicitly defined in the given model) will allow the effective assembly or disassembly of the necessary rectangles, provided that blocks are in the appropriate modes at the appropriate times:

Condition 4.4 *For any $t \geq 0$, for all i $(1 \leq i \leq n)$:*

$$[(\forall i' < i)(\forall e \in E_{i'})(\exists b \in B)\ P_E(b, e, t) \wedge (\forall b \in B)\ m^b(t) = i \wedge (\forall t' \leq t)(\forall b \in B)\ m^b(t') \leq i]$$

$$\rightarrow [(\forall e \in E_i)(\exists b \in B)\ Pr\{P_E(b, e, t)\} > 0 \wedge (\forall i' > i)(\forall e \in E_{i'})(\forall b \in B)\ Pr\{P_E(b, e, t)\} = 0]$$

Informally, this means that events in E_i will be detected with a non-zero probability (i.e., in some finite amount of time) if (a) all earlier events (i.e., events in $E_{i'}$ where $i' < i$) have been detected; (b) all blocks are in the appropriate mode i, and (c) no block has ever been in a mode greater than i. Furthermore, while these conditions hold, no later events ($i' > i$) in E will be detected. We want to show that these conditions can be satisfied such that the self-assembly process can progress through its various stages, and achieve the target structure S. The main result is as follows:

Theorem 4.7 *Given R_E, R_C and R_M, the following holds:*

$$(\exists t)(\forall i)(\forall e \in E_i)(\exists b \in B)\ P_E(b, e, t)$$

In other words, given the rule sets defined in Figure 4.6, at some time t, all substructures are completed. The implicit assumption is that if this holds, then no extra substructures (which are not part of O) are built or disassembled; the rules described in Section 4.1.5 ensure that this is the case. Other assumptions include Conditions 4.2, 4.3 and 4.4. The following section gives empirical evidence that the assumptions are valid for a variety of different structures.

4.4 Experimental Results

The previous section presented theoretical results which indicate that the rule generation procedure, as described, should produce a correct set of rules for assembling a given target structure. Can we therefore be certain that this is indeed the case? The theorems were derived in the context of abstract, mathematical models, and even though the self-assembly processes discussed in this dissertation take place in simulation (rather than the physical world), there exists a gap between these models and the simulated environment. Specifically, during self-assembly, the collection of blocks constitutes a complex dynamical system whose behavior is difficult to predict, as discussed in Section 3.3. The goal of the experiments reported in this section is to show that the assumptions that underly the theoretical results are reasonable enough to effectively achieve the self-assembly of a variety of interesting structures, and also to analyze the generated rules and resulting self-assembly processes. Some of the assumptions appear intuitively correct: for example, while it is known that the given implementation of the temporary substructure function f_T is not complete (as demonstrated in Section 4.1.1 via Figure 4.2b), it appears to be sufficiently powerful for computing an order on the structures given in Figure 3.1; even so, it is prudent verify this. On the other hand, while it was shown in Section 4.3.1 that the order (if any) computed on the target structure will be correct with respect to P_A and P_D, one cannot state with certainty that these predicates are correctly defined with respect to the given environment. Given a partially assembled structure Y and some substructure X, one might say that $P_A(X, Y) = \textit{true}$ if it seems possible to assemble X onto Y (Section 4.1.1), but even if this is indeed possible, it is not known whether the schemes used for computing the movement force (Sections 3.2.3 and 4.2) $\mathbf{F}$ will allow it.

The primary result of the experiments discussed below is that the automatic rule generation methods presented here work effectively for a significant range of non-trivial structures (barrier, bridge, building, etc.). More specifically, prior to performing the experiments, I had in mind the following hypotheses: (a) given the implementation of f_T and the rule generation procedure in general, rules will be successfully generated for a large class of structures; (b) there exists a method for computing the resultant local force $\mathbf{F}$ on blocks that is general enough to successfully assemble a large class of structures, given the automatically generated rules;

and (c) the rules generated based on the order computed by the full ordering algorithm (Figure 4.4) will allow for a significantly more efficient self-assembly process than those generated from the order returned by the preliminary ordering algorithm (Figure 4.3). The following two sections deal with these hypotheses by presenting relevant results.

4.4.1 Rule Analysis

Let us begin by analyzing the generated rules. The rule generation procedure was invoked on the six target structures given in Section 3.1, using both the preliminary and the full ordering algorithm. In each case, sets of rules were produced and (as shown later) resulted in successful self-assembly. Thus, the implementation of the procedure described earlier is sufficiently powerful to generate rules for these structures. Further, for structures such as the one shown in Figure 4.2b, the algorithms fail cleanly, without returning an incorrect order. To generate a set of rules for a structure, the procedure typically required only a second or so of processing time on a machine such as the Dell Precision WS mentioned at the end of Section 3.3. Table 4.1 characterizes the order O and the resulting rules produced as output by the algorithms. When different values are reported for the two different ordering algorithms, they are given as $\langle$value for the preliminary ordering algorithm$\rangle$ / $\langle$value for the full ordering algorithm$\rangle$. The column labeled "Number of Blocks" provides the quantity of each type of block present in the target structures. The value outside parentheses counts only blocks in the (permanent) target structure itself, whereas the value(s)

Table 4.1: Static Performance of the Rule Generation Procedure

Structure	Number of Blocks			Number of	Stages	Msg.	Rules Generated	
	S	M	L	Rectangles		Vars.	Stig.	V. C.
Barrier	70 (94 / 99)	468	9	340 (372 / 351)	56 / 35	57	949 / 841	1236 / 1110
Bridge	244 (300 / 309)	36	80	84 (190 / 148)	56 / 46	24	1097 / 926	838 / 618
Building	146 (180)	116	27	55 (126 / 81)	124 / 58	8	622 / 400	644 / 353
Fence	0 (210)	50	62	64 (275 / 221)	58 / 56	25	1216 / 954	1305 / 1033
Pyramid	265 (265)	0	3	13 (13)	7	3	82	46
Road	64 (64)	12	116	50 (50)	9	13	237	159

Table 4.2: Dynamic Performance of the Generated Rule Sets

Structure	Completion Time		Collision Rate	
	Mean	Std. Dev.	Mean	Std. Dev.
Barrier	95163.7 / 72445.5	7750.3 / 5776.4	0.25 / 0.22	0.16 / 0.14
Bridge	49798.8 / 45059.9	6767.0 / 7126.5	0.17 / 0.18	0.09 / 0.11
Building	101087.4 / 82151.8	7582.1 / 7553.9	0.06 / 0.06	0.01 / 0.01
Fence	60719.0 / 57420.0	6990.6 / 6735.4	0.42 / 0.38	0.17 / 0.12
Pyramid	6583.0	837.6	0.04	0.01
Road	33312.7	4406.8	0.31	0.06

in parentheses include the absolute minimum numbers of small blocks necessary to build not only the target structure, but also the temporary substructures (staircases and connectors) produced during order computation (recall that temporary substructures are always constructed from small blocks). This absolute minimum is computed as the *maximum* number (taken over all stages of the self-assembly process) of small blocks that must be simultaneously stationary (i.e., part of regular and temporary substructures). For example, in assembling the barrier (which contains 70 small blocks), 94 small blocks will need to be stationary at some point during the self-assembly process if the order computed by the preliminary ordering algorithm is followed, and 99 blocks will need to be stationary if the full ordering algorithm is used. Thus, the full algorithm can actually have a small disadvantage in the sense that slightly larger numbers of blocks are sometimes needed. This is a consequence of the reuse of staircases: a typical staircase generated strictly for a substructure C of height $h+1$ will contain $h+(h-1)+\ldots+1=\frac{1}{2}h(h+1)$ blocks (summing the number of blocks in layers, from lowest to highest). However, the full ordering algorithm may use the first h layers of a staircase that is generated for a substructure C' of height $h'+1$ (where $h'>h$); thus, the number of blocks in the staircase for C will usually be $h'+(h'-1)+\ldots+(h'-h+1)=\frac{1}{2}h(2h'-h+1)>\frac{1}{2}h(h+1)$. An example of this is observed in Figure 4.1c, where the staircase for the near left column is made longer than the other staircases, so that it can be reused in later stages (Figure 4.1d,e). For the bridge, as well as the barrier, this makes a difference in the stage with the maximum number of stationary small blocks. It should be noted that if the number of available blocks is limited, then the ordering algorithms can be modified such that fewer temporary substructures are assembled in a given stage, although this will reduce the system's potential for parallelism.

The remaining columns of Table 4.1 show that otherwise, the full ordering algorithm has advantages, as predicted. In the "Rectangles" column, the total numbers of generated rectangles are given, for the target structure only and (in parentheses) for the target structure plus any temporary substructures. For all four structures where temporary substructures are necessary, the latter number is smaller if the full ordering algorithm is used. Because a certain number of rules is generated for each rectangle (Section 4.1.5), the full algorithm results in smaller rule sets, as indicated in the last two columns of the table. Still, the only target structure where the rule sets can be said to be *parsimonious* is the pyramid, where the total number of rules is significantly smaller than the size of the structure itself. Both the pyramid and the road require no temporary substructures, and the former can be decomposed into only 13 rectangles, which translate into low numbers of rules.

The "Msg. Vars." column gives the number of message variables μ_j^b necessary during the assembly of the target structure; it is the maximum number of rectangles that are assembled or disassembled concurrently in any stage of the self-assembly process. This number is generally unaffected by the choice of ordering algorithm, since the number of staircases that must be assembled, extended, or disassembled in a given stage is no greater than the number of connected substructures that must be assembled in the previous or next stage. However, the number of stages (this is one less than the number of mode values, since an extra mode value is used to denote the completion of the process) defined by the full algorithm is somewhat smaller. For the fence, the difference is very small: at least some (though not all, as the case would be with the preliminary ordering algorithm) of the staircases are usually fully assembled/disassembled before/after depositing regular substructures, because of the “alternating” pattern of the layers of the fence (Figure 3.1f). For the other structures, particularly the building, the difference is greater, and the following section tests the hypothesis that this allows for a reduction in the overall completion time.

4.4.2 Rule-Directed Performance during Self-Assembly

For each rule set (two for each of the first four structures, and one for each of the last two, where the choice of ordering algorithm makes no difference, due to the absence of temporary substructures), 30 independent trials were performed with distinct random number streams, simulating the self-assembly process. For any block size, the number of blocks available is the minimum number reported in Table 4.1 plus 10: for example, 109 small blocks are used in assembling the barrier with rules resulting from the full ordering algorithm. Random numbers are generated as reported in Section 3.3. Also, as in the previous chapter, if a trial makes no useful progress (i.e., a moving block becoming stationary or a stationary block commencing movement again) for more than 30000 time steps, then the trial is terminated, and considered unsuccessful, but this never occurred in the experiments reported below. A successful trial ends when some block adopts the final mode value, which happens very soon after the assembly or disassembly of the final substructure in O is completed. An identical set of movement control mechanisms (for computing the movement force $\mathbf{F}$) was used for each structure; these mechanisms are similar to those described in the previous chapter, although some minor modifications were made, as discussed in Section 4.2. Also, recall that the global centering force $\mathbf{F}_{gc}$ was never used, and that collective influences were in effect, with $w_{nc} = w_{na} = 0.6$.

For every rule set, each of the 30 trials successfully completed and resulted in the desired target structure, with no temporary substructures remaining. Table 4.2 presents the average self-assembly completion time for each set of rules, as well as the standard deviation; average collision rates (i.e., number of collisions per time step) and corresponding standard deviations are also given. It is evident that there is a great deal of variation in these statistics between the different structures: some are obviously much more difficult to assemble than others, and somewhat independently of this, the process for some involves considerably greater interference between blocks. For the first four structures, where two different ordering algorithms were used, it is apparent that rules that are based on the order returned by the full algorithm outperform, in terms of efficiency, the rules generated if the preliminary algorithm is used. A two-sample, two-tail t-test assuming unequal variances showed that the difference in the average completion time is statistically significant ($p < 0.05$) for the barrier, the bridge and the building, though not for the fence. This is not surprising, considering that for the fence, the reuse of temporary substructures reduces the number of process stages by only 2 (see Table 4.1). The differences in average completion times are especially considerable for the barrier and the building, where the number of stages is substantially reduced. On the other hand, the reuse of temporary substructures does not have a statistically significant effect on collision rates for any of the target structures.

A possible confounding factor in these experiments arises from the difference in the minimum necessary number of small blocks for the barrier and the bridge, depending on which ordering algorithm is employed (Table 4.1). As mentioned earlier in this section, the quantity of blocks actually used during the self-assembly process is that number plus 10, which results in different initial block quantities. Another possible, somewhat complementary method for comparison is to use the same initial quantities of blocks for the two algorithms. So, 30 additional trials per structure were performed with rules based on the preliminary ordering algorithm, but with block quantities used for the full ordering algorithm (i.e., $99 + 10 = 109$ for the barrier and $309 + 10 = 319$ for the bridge). This yielded mean completion times of 90624.1 (standard deviation 10440.7) for the barrier and 42497.1 (std. dev. 3437.5) for the bridge; average collision rates were 0.31 (std. dev. 0.20) for the barrier and 0.16 (std. dev. 0.02) for the bridge.

In relating these latter numbers with those given in Table 4.2, it is important to note that this method of comparison can be somewhat biased: the trials may have an advantage because 15 extra (above the minimum) small blocks are used for the barrier and 19 extra for the bridge, rather than just 10 extra, as in the original trials. As shown in Section 3.4.4, the presence of these additional blocks can make a difference, particularly in stages where most small blocks are stationary (as part of temporary and regular substructures), and relatively few are available to find the goal locations. Further t-tests were performed between sets of trials that use the same block quantities, but different ordering algorithms, and sets of trials that use the same, preliminary ordering algorithm, but different block quantities. Differences in collision rates were not significant (as before); however, for the barrier, the former test shows that the use of the full ordering algorithm still leads to significantly lower completion times, even in spite of a possible bias arising from the use of 5 additional blocks. The latter test does not show statistical significance in completion times ($p > 0.05$), which suggests that the bias is probably not too great. For the bridge, an opposite trend is observed: the addition of 9 blocks seems to have a strong impact, leading to a significant reduction in the mean completion time for the preliminary ordering algorithm. The value is reduced even somewhat below the mean completion time for the full ordering algorithm, but the difference between the two algorithms is not statistically significant. Whether or not the full ordering algorithm actually yields better unbiased performance for the bridge depends on whether it is fairer, when making the comparison, to use the same difference (i.e., 10) between the number of blocks used and the minimum number of blocks necessary, or to use the same absolute number of blocks. In summary, the results show that the reuse of temporary substructures yields significant gains in the system's performance for at least two (possibly three) of the four target structures that require them.

4.5 Discussion

This chapter has focused on presenting and evaluating a fully automated approach to generating local control mechanisms for the self-assembly of prespecified target structures in a continuous and constrained environment. While the complexity of motion in such an environment makes infeasible a proof that the generated mechanisms will always cause the elements of the system (i.e., the blocks) to converge to the desired structure, it was nonetheless possible to formally verify that certain major aspects of the methodology are correct. Specifically, it was proved that if the environmental constraints are appropriately modeled, then it is possible to generate a partial order on the target structure (if one exists), such that if this order is followed during the self-assembly process, then the structure can successfully assemble in spite of these constraints. Furthermore, it was shown how this order can be correctly enforced at runtime, using simple, local communication. Finally, it was demonstrated how stigmergic rules can be generated to direct the self-assembly of the various parts of a structure, detect their completion, and disassemble temporary substructures such as staircases. In order to verify aspects of control that were not handled in a formal manner and to observe the resulting self-assembly processes, experiments were conducted where the rule generation procedure was applied to six distinct target structures, and the generated rules were coupled with a preexisting set of continuous movement control schemes in a large number of independent simulated trials. The results from these experiments provide support for the hypothesis that not only can the generated rules be successfully applied in the given environment, but also that the developed movement control methodology (modified only slightly from Chapter 3) is generic enough to be reused for assembling a variety of structures. Finally, the experiments demonstrated the advantages of reusing parts of temporary substructures between stages of the self-assembly process, as allowed by one of the ordering algorithms invoked prior to rule generation. While a slightly higher number of blocks may be necessary for constructing these reusable substructures, both the number of generated rules and the overall completion time can be reduced.

As discussed earlier, there have been other studies of automatic rule generation for the control of self-assembly processes, some of them based on stigmergy [2, 48]. The key contribution of this chapter lies in extending these approaches to a considerably more complex environment, whose constraints on movement are addressed by generating rules for a higher-level sequencing of the process, and whose continuous nature is handled via the integration of generated rules with generic continuous movement control schemes. This approach is verified by demonstrating the self-assembly of a variety of interesting structures, which resemble objects in the real world. Still, there remain open research questions. In the previous chapter, the reported average completion time for the self-assembly of the building shown in Figure 3.1d,e is 43338.4 time steps (given hand-designed rules), in the absence of any global information, and with collective, flock-like movements (Table 3.4). This is considerably lower than the times reported in Table 4.2. While many factors could have made an impact on the difference in the means (for example, different temporary substructures were used in the experiments of Chapter 3), it is suspected that a major reason for the slowdown is that the coordination of movement over time (Section 4.2) is no longer structure-specific, whereas before, the computation of the movement force was hand-tuned for every stage of self-assembly (the stages were also hand-specified). More dramatically, for the building, $400+353 = 753$ rules were generated even when the full ordering algorithm was used, whereas Chapter 3 showed that its self-assembly could be accomplished with only 143 (66 stigmergic and 77 variable change) rules, which were hand-designed. In fact, the number of rules can be made slightly smaller if moving blocks are allowed to remain atop the structure at termination time, which was prevented in Chapter 3 (see Section 3.4.1), but not in the experiments of this chapter. Thus, while the stigmergic methodology developed in Section 3.2.2 allows for the encoding of certain structures via compact rule sets, the rule generation procedure often has trouble taking advantage of this. In the following chapter, an attempt will be made to reduce the sizes of the rule sets by decomposing the target structure into shapes other than two-dimensional rectangles, and generating rules that can efficiently and safely handle these shapes.

Chapter 5

Improving Rule Parsimony

In Chapter 3, we saw how earlier, qualitative stigmergic methods [2, 48, 49] can be enhanced with integer variables, creating a model that can potentially describe a variety of interesting structures in a *parsimonious* fashion. However, the automated rule generation procedures developed in Chapter 4 were typically unable to fulfill this potential: while correct rule sets were produced, many more rules were generated than necessary. For example, the automatically generated set of rules for the building in Figure 3.1d,e was more than five times larger than the hand-designed set of rules! Before attempting to correct this shortcoming, it is useful to ask: why is a parsimonious set of rules desirable? Certainly, such a rule set requires less storage, which may be quite beneficial in practice, if self-assembly is to be achieved with physical devices, which might have a limited supply of on-board memory; less processing time is also required for matching the entire set of rules to some visible part of the structure. However, small rule sets (and the ability to generate them) are of interest from a theoretical point of view as well. An agent using few rules is arguably simpler than one that uses many, and when simple agents are able to produce a complex structure, a stronger argument can be made for true self-organization. (An analogy from machine learning [76, 108] is to favor simpler hypotheses to account for a large number of observations). Here, the term *com-*

plex is used in the sense that the structure contains non-trivial patterns, reflecting the notion of complexity that is used when describing the formation of intricate spatiotemporal patterns in systems such as cellular automata [62]. This is distinct from Kolmogorov complexity [67], where most highly random, disordered strings (or structures) are considered complex, in the sense that they cannot be compactly described. This presents a theoretical barrier to the existence of highly parsimonious rule sets for most structures, as argued in Section 3.2.2. However, it is hypothesized that certain "interesting" (i.e., complex in the former sense of the word) structures can be assembled with a relatively small set of rules, and that such rule sets can, in some cases, be automatically discovered.

This chapter presents modifications to the methodology developed earlier, which are made in an effort to reduce the size of the generated rule sets while maintaining their correctness. The order on the self-assembly process (as computed in Sections 4.1.1-4.1.3) is modified, and a more sophisticated pattern-based technique is used for determining the rules necessary to correctly assemble substructures within this order. Further, it is established that the problem of generating a minimal set of rules is NP-hard, even if the environment contains no physical constraints (which necessitate additional rules for higher-level coordination). In spite of this, it is shown experimentally that the new methodology results in significant rule set size reductions for all of the target structures considered (Figure 3.1), and that for some of these structures, the number of rules is reduced below the number of blocks within the structure itself. At the same time, the simulations undertaken here illustrate that a tradeoff exists between the parsimony of rule sets used and the efficiency of the resulting self-assembly process. Upon examining some of the causes behind the loss in efficiency, further modifications are made that yield improved completion times (some of which are comparable to the times reported in Chapter 4), without a significant sacrifice to parsimony.

5.1 An Approach to Parsimony

Before discussing modifications made to the rule generation procedure of the previous chapter, let us first examine the reasons behind the lack of parsimony in the rules that it generates. Recall that in its first phase, the procedure computes an order $O = (C_1, C_2, \ldots, C_n)$, and that rules are then generated for each substructure $C_i \in O$, which are used to assemble or disassemble this substructure, detect its completion, ensure that other blocks are notified that it is completed, and cause the collection of blocks to eventually enter mode $i + 1$ and begin the next stage of the process. For some target structures, $|O|$ can be quite large, as indicated in the "Stages" column of Table 4.1. In particular, because each substructure is only one layer in thickness, $|O|$ depends on the height of the structure. Even for a simple structure, such as a tower where all layers are identical, a separate subset of rules is generated for each layer, in order to avoid the development of unreachable holes, such as the one shown in Figure 4.2a. This can result in a large set of rules, which does not capture the inherent repetition.

Additionally, given the methods described in Chapter 4, each substructure $C_i \in O$ is decomposed into a set of rectangular shapes, and a certain number of rules is generated for each rectangle. In many cases, the number of rectangles, and thus the number of rules generated for C_i is large (see Table 4.1). Viewing a structure simply in terms of rectangles fails to capture its more complex yet repeating patterns.

In the rest of this section, both deficiencies are addressed by presenting a modified version of the rule generation procedure. First, it is shown how the order O can be further transformed by collapsing the small, single-layer substructures C_i into larger, multi-layer ones. Then, a new method is described for generating rules for safely assembling these substructures, and detecting their completion. Finally, the correctness of this method is critically examined.

5.1.1 Collapsing the Order

Given the order O that results from the application of the full ordering algorithm after the postprocessing discussed in Section 4.1.3, we can apply the procedure given in Figure 5.1 to generate a more compact order O'. The procedure illustrated in the figure operates by combining multiple consecutive substructures C_i into a larger substructure $C'_{i'}$, when appropriate. Statements 5a and 5b ensure that the assembly and disassembly (respectively) of substructures takes place in separate stages. As $C'_{i'}$ is formed as a union of successive regular/temporary substructures C_i (one per iteration of the main loop), a disassembly substructure encountered in O will cause Statement 5c to execute, and create a new substructure $C'_{i'+1}$. Further, Statement 5(a)iii helps to keep in separate stages the assembly of substructures that

```
1. Input $O = (C_1, C_2, \ldots, C_n)$.
2. Initialize an empty collapsed order $O' = \emptyset$.
3. Initialize $i' = 1$.
4. Initialize a substructure $C'_{i'} = C_1$.
5. For all $i$ $(1 < i \leq n)$:
    (a) If the following three conditions hold:
        i. $C'_{i'}$ contains only regular or temporary substructures.
        ii. $C_i$ is a regular or temporary substructure.
        iii. When $C_{i-1} \subseteq C'_{i'}$, the height (i.e., vertical position) of $C_{i-1}$ is no
             greater than the height of $C_i$.
        Then set $C'_{i'} \leftarrow C'_{i'} \cup C_i$.
    (b) Otherwise, if the following three conditions hold:
        i. $C'_{i'}$ contains only disassembly substructures.
        ii. $C_i$ is a disassembly substructure.
        iii. When $C_{i-1} \subseteq C'_{i'}$, $C_{i-1}$ has at least one location that is supported
             by a location in $C_i$.
        Then set $C'_{i'} \leftarrow C'_{i'} \cup C_i$.
    (c) Otherwise, set $O' \leftarrow O' \cup (C'_{i'})$, $i' \leftarrow i' + 1$ and $C'_{i'} = C_i$.
6. Return $O' \cup (C'_{i'})$.
```

Figure 5.1: A procedure for collapsing the order O. Recall that a subset of S is called a regular substructure, whereas a staircase or a connector is called a temporary substructure when it assembles, and a disassembly substructure when it disassembles. The operation $O' \cup (C'_{i'})$ appends substructure $C'_{i'}$ to the end of the emerging list O'.

obstruct each other (as illustrated in a subsequent example), while Statement 5(b)iii ensures that if a staircase consists of two disjoint parts at different heights, the lower part will remain in place while the upper part is disassembling, allowing blocks from the latter to descend.

Consider the application of the procedure to the order O generated for the building in Figure 3.1d,e. Then, C'_1 consists of the floor, columns and column staircases. After the top layers of the columns are added to C'_1, a new substructure C'_2 is initialized, and is made to specify the disassembly of the column staircases; subsequently, C'_3 is created as a union of wall layers along with the wall staircase, etc. Now, suppose that the columns had permanent staircases (the floor would have to be larger, in order to accommodate them). The walls would still be assembled in a separate stage: while the top column layers would be immediately followed by the bottom wall layers in the original order O (both substructures being regular), they would remain separate in O', because the former are *higher* than the latter, and Statement 5(a)iii would not hold. (In the current implementation, Statements 5(a)iii and 5(b)iii are not enforced, because they do not play a role for the target structures assembled here; however, they would not be difficult to incorporate).

When O is collapsed, it is "relaxed" in the sense that some of the ordering constraints on the deposition or removal of blocks into/from goal locations are deleted. While O is guaranteed to be correct with respect to the predicates P_A and P_D, as stated in Theorem 4.2, one can generally make no such claims about O'. In the specific environment studied here, a self-assembly process that follows O' may not complete, due to the development of unreachable holes (Figure 4.2a); such situations have been observed experimentally under certain rule sets. The following section presents a rule generation procedure that accounts for these sorts of problems, while attempting to produce rule sets that are parsimonious.

5.1.2 Generating Rules in a Parsimonious Fashion

Given the collapsed order O', we are now faced with the problem of generating stigmergic rules for each individual substructure $C \in O'$. Let us first outline a procedure for the generation of assembly rules. For the purposes of this discussion, S' is defined to be the structure that should exist in the environment just after the assembly of C (thus, $C \subseteq S'$), containing within it a subset of S, along with (possibly) temporary substructures. (Using the more formal notation of Definition 4.2 in Chapter 4, $S' = S^{(i)}$, if C is the i^{th} substructure in O'). Also, a set of rules R for a substructure C is defined to be "parsimonious" if $|R| < |C|$.

Under this definition, it follows that if R is parsimonious, then there are rules in R that apply at more than one location of C. An attempt to discover such a rule set therefore entails searching for local patterns within S' that *repeat*. For each location $c \in C$, it is possible to determine the set $a(c, S')$ of all locations $c' \in S'$ that are adjacent to c by a face, edge or corner (these locations are filled at some point during the self-assembly process). A rule r can then be generated for c, with full sites that correspond to some appropriate subset of $a(c, S')$ (recall from Section 3.2.2 that a rule specifies the required presence of block sections with full sites), whereas the remaining locations in $a(c, S')$ can be wildcards (i.e., they can match either blocks or empty space). Generally, speaking, if more locations in $a(c, S')$ are encoded in r as full sites (as opposed to wildcards), then r becomes more constrained specifically to c, and less likely to apply at other locations $c' \in C$ (unless $a(c', S')$ has a similar structure to $a(c, S')$). Note, however, that while it is desirable to have r apply at multiple locations within C, the rule may also inadvertedly apply at locations where it should not. As before, it then becomes necessary to restrict the application of r to specific regions of space, via the use of memory conditions. In some cases, r must be expanded into several rules that are geometrically identical, but have memory conditions that are distinct.

In summary, two issues must be addressed when generating rules for C: the choice of locations (for all $c \in C$) in $a(c, S')$ that will correspond to full sites in r, and the specification of memory conditions. These shall be discussed in turn.

First, it is important to note that the full sites of a stigmergic rule impose a particular kind of ordering restriction on the self-assembly process; namely, if a goal site of a rule r corresponds to some location $c \in C$, and there is a full site that corresponds to an adjacent location c', then a block following r will be placed at c only after one is placed at c'. Therefore, it is natural to take the approach where all adjacent locations one unit below c are specified as full sites in r. While this will not ensure that a layer of C at height h will always be assembled before any blocks at height $h + 1$ are deposited, it will at least enforce this within a local region of space. As argued in the following section, this is usually sufficient for the prevention of unreachable holes, and allows a multi-layer structure such as the one depicted in Figure 4.2a to always assemble correctly. All other sites within the rule are wildcards (except for the goal site(s), which are empty, since a goal location must leave room for a depositing block). This potentially allows for a single rule to be applicable at a greater number of locations; i.e., one rule is generated for all locations in C that have an identical pattern directly underneath them. Furthermore, because wildcards do not create restrictions on the order in which blocks within a given layer of C can be deposited, the potential for parallelism is increased.

Of course, this approach cannot be utilized for any location that is on the ground, since such a location has no locations below it. For the subset of C that consists of locations at height 0, rules are generated using the method of Section 4.1.5: the subset is decomposed into rectangles, and appropriate rules are produced for each rectangle.

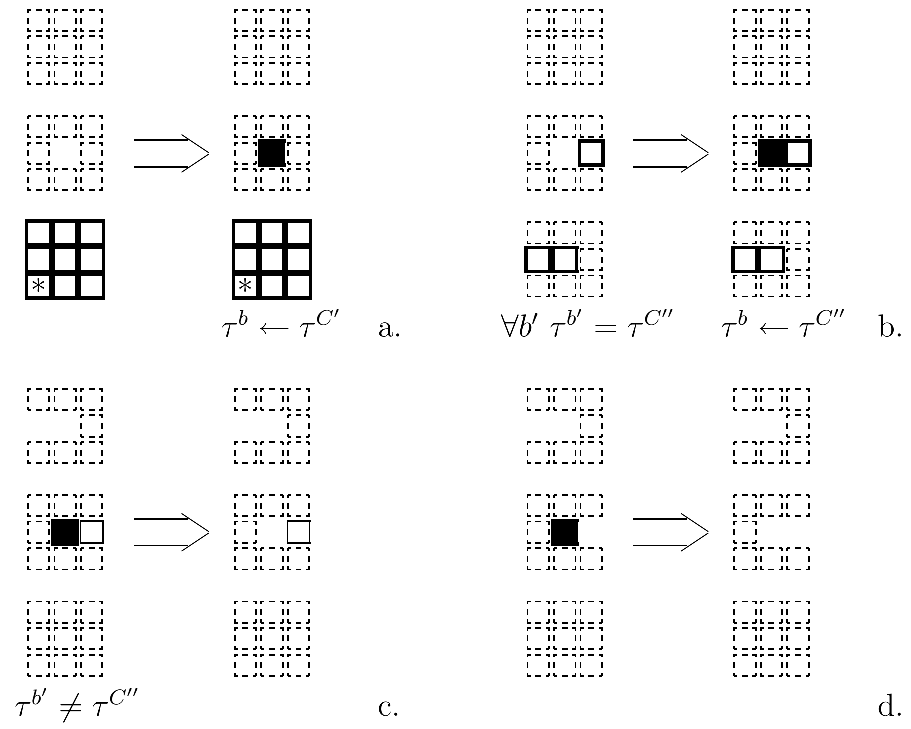
Now, consider all locations where a rule r can potentially match/apply, while $C \subseteq S'$ is assembling. This set can be partitioned into two disjoint subsets $d(r, C, S')$ and $u(r, C, S')$ of desirable and undesirable locations, respectively. The former subset contains only locations that are in C; moreover, it is restricted to locations where r will deposit a block *only* once all adjacent blocks below have been placed. The subset $u(r, C, S')$ contains all other locations where r can potentially apply; importantly, it is undesirable for a rule specifying one full site to apply at a location that actually has two adjacent blocks below it; this may cause a block to be deposited before both of the blocks below have been placed. It is necessary to restrict the application of any rule r generated via the new method to $d(r, C, S')$, ensuring that it does not place blocks elsewhere. To do this, the following procedure is performed for each rule r. First, the structure S' and the surrounding space is scanned to determine any locations where r may apply, but should not; i.e., $u(r, C, S')$ is computed. Subsequently, memory conditions are added to geometrically "separate" $d(r, C, S')$ from $u(r, C, S')$. Suppose that we have a set of six memory conditions, of the following form: $\{x^b \geq x_{min}, x^b \leq x_{max}, y^b \geq y_{min}, y^b \leq y_{max}, z^b \geq z_{min}, z^b \leq z_{max}\}$. Clearly, these memory conditions restrict the application of r to a finite, contiguous, hexahedral region of space, described by the boundaries x_{min}, x_{max}, etc. The problem of computing these boundaries can therefore be phrased as a problem of *hexahedral cover*, where it is necessary to come up with a set of hexahedra, subject to the constraint that each location in $d(r, C, S')$ is covered by at least one hexahedron, while no location in $u(r, C, S')$ is covered by any hexahedron. For each hexahedron, a separate rule r' is generated, with memory conditions that correspond to the hexahedron's boundaries, but with a geometry identical to that of r. These memory conditions are associated with some full site of r', and the computed values x_{min}, x_{max}, etc. are offset by that site's position relative to the goal site(s). The total number of generated rules thus depends directly on the size of the cover.

Can a minimal hexahedral cover be computed efficiently? The 2D version of the problem involves computing a minimal *rectangular cover* on a grid of cells, such that some cells are covered and others are not, and it is known to be NP-complete [25, 27, 61]. The rectangular cover problem readily reduces to the problem of hexahedral cover (one can simply transform the set of cells to be covered into a set of locations that exist in the same horizontal plane, and therefore require hexahedra of 1 unit in thickness); furthermore, given any candidate set of hexahedra, it is possible to determine, in polynomial time, whether or not the cover is valid.

Thus, the hexahedral cover problem is NP-complete as well. As with rectangular decomposition (which is different in that all rectangles are disjoint, and where an efficient, optimal strategy actually does exist [85], as mentioned in Section 4.1.3), a greedy strategy is adopted, repeatedly selecting the hexahedron that covers the greatest number of locations in $d(r, C, S')$ (not counting those already covered by hexahedra selected earlier), and that does not cover any locations in $u(r, C, S')$. This is done until all locations in $d(r, C, S')$ are covered.

The application of the new rule generation procedure is illustrated by Figure 5.2a, which shows the rules that are used for depositing the bottom layers of the columns of a building (Figure 3.1d,e). As explained in the caption, the two rules are geometrically identical, and are drawn as one rule for conciseness. For both rules, all sites below the goal site are full, and correspond to the floor blocks that exist below the column blocks. The memory conditions given in the caption restrict the application of each rule to a 2×2 region of space; two rules are necessary because the two regions where blocks should be placed are disjoint. Note that there is no condition on the variable z^b, because the structural patterns necessary for the rules' application exist only above the floor, and nowhere else within the structure; at no other regions of the building does a small block have 9 other small blocks directly below it.

Let us now discuss the assembly and disassembly of temporary substructures. For connectors, which join assembling substructures that are disconnected from each other, and are always built on the ground (Section 4.1.3), assembly and disassembly rules are generated as in Chapter 4, through rectangular decomposition. The ground


$\tau^b \leftarrow \tau^{C'}$
a.
$\forall b' \ \tau^{b'} = \tau^{C''}$
$\tau^b \leftarrow \tau^{C''}$
b.
$\tau^{b'} \neq \tau^{C''}$
c.
d.

⬚ Wildcard □ Block Section (of b') □ Small Block (b') ■ Matching Block (b)

Figure 5.2: Examples of generated rules. The rule in (a) actually represents two geometrically identical rules for placing the bottom layers of the columns of a building, which are part of a larger substructure consisting of the floor, columns and column staircases (here, the non-temporary parts of this substructure are denoted by C', and blocks within them are assigned the value $\tau^{C'}$). One of these rules has the memory conditions $\{x_o - 4 < x^{b'} < x_o - 1, y_o - 2 < y^{b'} < y_o + 1\}$, where b' is the block associated with the full site labeled with an asterisk (*), and x_o, y_o denote the variable values within the seed block; the other rule has the memory conditions $\{x_o < x^{b'} < x_o + 3, y_o - 2 < y^{b'} < y_o + 1\}$. The rule in (b) places the "interior" blocks of a staircase C'' that extends leftward from the adjacent structure, while the rules in (c) and (d) disassemble this staircase.

layers of staircases are assembled likewise, having no blocks underneath them. In principle, it is possible to assemble the remaining layers by generating rules as is done for non-ground regular substructures. However, it was found to sometimes be more efficient (in terms of completion time) to assemble staircases via rules generated in a slightly different manner, as follows. For staircase locations adjacent to some part of the target structure (e.g., a wall), rules are still generated as before, by specifying full sites for supporting locations and restricting the application of rules by using memory conditions. However, for any "interior" location of a staircase, which has other staircase locations below it and a staircase location adjacent to it on the same level, assembly is accomplished via a single rule, which is depicted in Figure 5.2b. (Note that three other "rotations" of this rule around the z-axis are used, depending on the orientation of the staircase, relative to its adjacent structure). Directed by this rule, a block places itself adjacent to another, already-stationary staircase block on the same level. The memory condition $\tau^{b'} = \tau^{C''}$ is associated with all full sites, to ensure that for any staircase C'', a staircase block is placed only adjacent to blocks already part of C''. This approach allows multiple layers of the staircase to assemble simultaneously (unlike in Chapter 4); however, it is somewhat different from the approach of Chapter 3, where the assembly of a staircase is essentially performed in "diagonal" layers, deposited from top to bottom (Section 3.2.2).

The disassembly of a staircase can also be accomplished in various ways. Under the current implementation, two rules are generated for disassembling each staircase in its entirety. The first rule (Figure 5.2c) causes a block that is adjacent to some (permanent) part of the target structure to begin moving, provided that there are no staircase blocks above; it has the memory condition $\tau^{b'} \neq \tau^{C''}$ associated with the single full site, to prevent any staircase block that has another staircase block to its immediate right (rather than a section of a target structure block) from moving prematurely. The subsequent disassembly of these blocks is governed by the rule in Figure 5.2d. Together, the two rules allow multiple layers of the staircase to disassemble at the same time (with blocks closer to the target structure starting to move earlier), but in a safe manner: the empty sites above the goal location ensure that blocks which are at adjacent levels and horizontally close to each other will not begin moving simultaneously.

Once C has assembled or disassembled, the fact of its completion must be detected and communicated to other blocks. As before, for disassembling substructures, the last block to begin moving (for staircases, this is always the bottom "step" of the staircase) applies a variable change rule to set the appropriate message variable, whose value is later adopted by other blocks, as discussed in Chapter 4. For an assembling substructure, completion detection is somewhat more involved, because the order in which blocks deposit themselves is far less restricted, and there is usually no location that is guaranteed to always be the last filled. However, it is known that certain locations will definitely not be last; specifically, consider any location $c \in C$ with some other location $c' \in C$ that is one level above it, and adjacent to it by a face, edge or corner. According to the rule generation procedure discussed in this section, c will necessarily be filled before c'. Thus, to detect completion, it is necessary to consider locations in C that have no adjacent locations above them, decompose this subset into rectangles, and generate completion rules as outlined in

Section 4.1.5. However, it should be noted that within C, all blocks belonging to regular substructures are assigned the same value of τ^b (each staircase and connector gets a separate value); so, two blocks in separate rectangles are not guaranteed to have distinct values of τ^b. Thus, the memory conditions in completion rules (as well as assembly rules for rectangles) are no longer defined on τ^b, but rather, additional constraints are placed on the positional variables. For example, the first rule in Figure 4.8 has the memory conditions $x_{min}^{C_{ij}} < x^{b'} \leq x_{max}^{C_{ij}}$ and $z^{b'} = z^{C_{ij}}$, in addition to $y^{b'} = y_{max}^{C_{ij}}$. Still, a separate message variable is set for each completed rectangle; these messages propagate to other blocks, and eventually cause them to assume the next mode value, and begin assembling or disassembling the next substructure in O'.

5.1.3 A Discussion of Correctness

Ignoring constraints on the blocks' motion, it is relatively easy to see that the method outlined above correctly generates assembly rules for some substructure C: every location in C is covered by some rule that was generated specifically for the pattern just below it (where the locations within this pattern are filled with blocks earlier in the process); furthermore, memory conditions are added such that no rule applies at any location where it should not apply. When physical constraints specific to the environment (Section 3.1.2) are taken into account, the task of showing correctness becomes much more involved. As stated in Chapter 1, the focus of this dissertation is not on formally proving properties that are specific to the given environment (formal treatment is given only to aspects of the approach that are environment-independent, such as the abstract ordering algorithms of Chapter 4). Instead, a somewhat informal argument for correctness will be presented, including a discussion of problems that can potentially arise, as well as possible approaches for overcoming them.

First, it will be shown that under certain assumptions, unreachable holes (such as the one in Figure 4.2a) will not develop. To aid our reasoning, let us define a *patch* in a partially assembled structure S' as a 1×1 region of space that has a small block, a $1 \times 1 \times 1$ section of a medium or large block, or the ground directly below it, and empty space directly above it. (Only regions of the ground that are in the vicinity of the structure need to be considered). Next, define the *topography* $G^{S'}$ of S' as an undirected graph, where each vertex maps to a patch. Two vertices are connected by an edge if the corresponding patches are horizontally adjacent, and if the difference between their heights is no greater than 1 unit. For example, a typical staircase would be represented as a single path in $G^{S'}$, with the vertices corresponding to the top faces of the steps, and the edges denoting the fact that the difference in the heights of any two consecutive steps is 1 unit. In this context, if an unreachable hole is present, then the topography of S' is a disconnected graph.

Now, assume that:

(a) $G^{S'}$ is connected.

(b) S' has no hollow cavities in it, such as the holes in the barrier (Figure 3.1a,b) or the building's interior (Figure 3.1d,e).

Rephrased more precisely, assumption (b) states that if a vertical line is drawn between any patch in S' and the ground, then that line should not pass through empty space. Consider S' as it is assembling, and let $S'' \subseteq S'$ be the structure currently built. Since S' has no internal cavities, there is a one-to-one correspondence between the vertices of $G^{S''}$ and $G^{S'}$; in essence, both sets of vertices correspond to the aforementioned vertical lines. Now, suppose that there is an edge in $G^{S'}$ that does not exist in $G^{S''}$. Then, the vertical distance between the two patches is at least 2 units. Let b be the block just below the higher of the patches. Because the edge is present in $G^{S'}$, there must be some block b' that is below b and adjacent to it; b' is present in S', but is absent in S''. However, given the stigmergic rules discussed earlier, b would not have deposited itself until b' is in place (the location of b' would be specified as a full site in the rule used by b); this yields a contradiction. Thus, every edge in $G^{S'}$ has a counterpart in $G^{S''}$, and $G^{S''}$ is connected as well. In essence, the generated rules ensure that at any point in space, the topography of S'' does not change abruptly.

Here, it should be mentioned that under the simple construction of $G^{S'}$ given above, an edge in $G^{S'}$ is not equivalent to the assertion that a moving block can necessarily travel directly between the two adjacent patches (especially, if the edge represents a corner adjacency), since blocks (particularly, medium and large blocks) require a certain amount of open space to move freely, but conversely, are able to cross narrow gaps between blocks. To ensure that a connected $G^{S'}$ truly implies that every location atop S' can be reached, it is best to perform a more detailed analysis, which takes full consideration of the blocks' geometries. Still, the preceding argument shows that if S' has a "smooth" topography (where no substructures have precipices of height 2 or more on all sides), then $S'' \subseteq S'$ does as well, and locations atop such a structure are typically accessible during self-assembly.

Now, we turn to a discussion of situations where assumptions (a) and (b) do not hold for the structure S' that exists at the end of some assembly stage. In fact, assumption (b) is not valid for any of the target structures considered (with the exception of the road), due to the presence of hollow cavities. However, in all cases, this does not pose a problem, as verified experimentally in later sections. Assumption (b) is in place to account for a situation such as the following: let S' be a structure (not shown) that consists of two columns, which have a large block placed just above them, and a staircase adjacent to one of the columns. Then, $G^{S'}$ is a connected graph. Now, let S'' be a structure that exists at an earlier point in time, before the large block has been placed; $G^{S''}$ is disconnected, and the top of the column which has no staircase cannot be accessed (in fact, at most two layers of this column can be built). However, given the ordering algorithms discussed in the previous chapter, this issue will not arise, because these algorithms will generate a staircase for the other column as well.

For the barrier and the fence, the presence of holes violates not only assumption (b), but assumption (a) as well, because the patches on the bottom surfaces of these holes are disconnected from the rest of the structure. Once again, this does not present a problem, because the holes are quite small, and blocks that form the bottom surface of any hole will be deposited before blocks on the sides or the top of the hole. The generated rules are such that the side blocks will not be placed

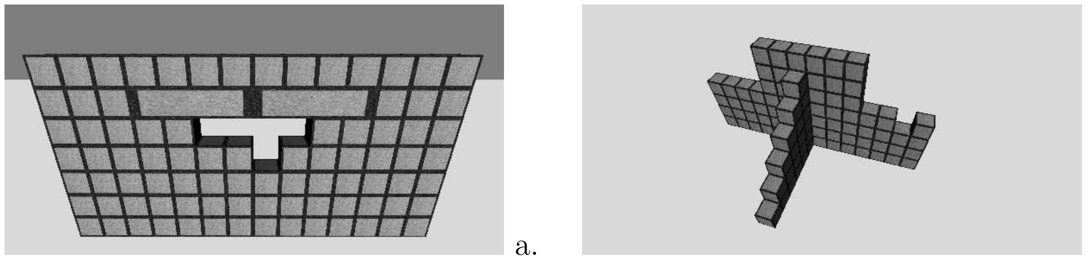

a. b.

Figure 5.3: Examples of structures with unreachable, unfilled locations: (a) the bottom surface of a large hole is missing a block, as a result of large blocks covering the hole too soon; and (b) the surface of the lower wall is missing a block, since the block's location cannot be reached from the precipice formed by the upper wall.

prior to the bottom blocks (to which they are adjacent), and the top blocks will not be placed until after the side blocks have deposited themselves. However, consider the hole shown in Figure 5.3a, which has small blocks forming the bottom surface, and two large blocks forming the top surface. One of the small blocks has not been deposited in time, because the hole was covered with large blocks prior to its placement. To avoid this problem, a procedure can be implemented to analyze a structure for the presence of large holes, and modify O' to ensure that anything below the hole is assembled in a separate stage.

Another potential difficulty can arise in the situation depicted in Figure 5.3b, which shows a target structure (consisting of a base wall and a narrower wall above it), along with a staircase. This structure has an inherently "rough" topography, with a precipice formed by the upper wall. While this wall is itself reachable via the staircase, the surface of the lower wall cannot be accessed, and the location that is missing a block (on the right side) cannot be filled. Once again, it is possible to implement a procedure which would detect the fact that the topography is disconnected, and ensure that the assembly of the lower wall and the upper wall takes place in separate stages.

For an example of yet another, related situation, consider a structure (not shown) that is a simplified version of the building (Figure 3.1d,e), which consists of only the floor and the walls, with no columns, roof or door frame. In this case, the collapsed order O' (produced according to the procedure in Figure 5.1) would consist of a single substructure, which will not be able to fully assemble if walls completely surround the floor, before said floor is completed. This can be mitigated in a similar manner, by assembling the floor and the walls (which are disconnected from each other in $G^{S'}$) in separate stages. Note, however, that this separation is already made in O' for the building in Figure 3.1d,e, ensuring that the floor and columns assemble prior to the obstructing walls. Importantly, O' still serves a purpose, by taking into account such obstructions.

In summary, while the collapsed order is not safe for certain structures, procedures exist for detecting potential problems such as large holes or tall precipices, and changing O' accordingly. In all of the examples discussed, the procedures essentially involve computing the connected components of $G^{S'}$, ordering these components by the height of the topmost patch in each component, and modifying O' such that locations corresponding to lower patches are guaranteed to be filled before locations above them. Because O' suffices for the target structures studied here (and many other structures as well) without modification, these procedures are presently not implemented; however, this section shows that it can be done, if necessary.

5.2 Complexity Considerations

As noted earlier, the intent of this chapter is to produce sets of rules that are not only correct, but parsimonious as well. The following is a discussion of theoretical limitations to this endeavor. First, as stated in Section 3.2.2, by a Kolmogorov (descriptive) complexity argument [67], there are limitations on our ability to compactly describe structures via any means, because structures are able to encode information. Still, it is suspected that parsimonious rule sets exist for some of the more "interesting" target structures, which have repeating patterns. An essential question is then as follows: given a target structure, how difficult is it to generate an optimally parsimonious (minimal) set of stigmergic rules necessary to assemble it? Here, it is argued that the problem of finding such a rule set is NP-hard, meaning that any problem in NP can be reduced to it in polynomial time. The argument is presented somewhat informally, since doing otherwise would require building a formal, low-level model of stigmergy that would precisely capture the preconditions and effects of a rule application (the development of this model could certainly be a subject of future work). However, if the reader has a clear conceptual understanding of how a rule is applied to place a block, then the argument should be convincing.

Recall from Section 5.1.2 that the modified rule generation procedure repeatedly computes a hexahedral cover, in order to generate memory conditions. As stated therein, producing a cover with the minimal number of hexahedra is an NP-complete problem (thus, an approximation is used), since the problem of rectangular cover (which is known to be NP-complete [25, 27, 61]) can be reduced to it. Here, the problem of rectangular cover is reduced to the problem of generating a minimal set of rules; in other words, the latter problem is shown to be at least as difficult as the former. An instance of the former problem can be presented as a set of squares $\mathfrak{B}$ (called *board* in [25]) on a grid, such as the one shown on the left side of Figure 5.4. Rectangles must be defined such that each square is covered by some rectangle, and such that no rectangle covers any empty cell on the grid. This board can be converted, in polynomial time, into a target structure $S_{\mathfrak{B}}$, such as the one depicted on the right side of Figure 5.4, which consists of a set of intersecting rows. Each intersection that corresponds to a square has a single small block above it (any such block shall be called a *square block*), while the intersections corresponding to empty areas of the grid have nothing above them. The square blocks in $S_{\mathfrak{B}}$ must be separated from each other, so that each square block has an essentially *identical* set of adjacent blocks (five blocks below, and no blocks at the same level or above). The following decision problem is now posed: given that the rows at level 0 (i.e., on the ground) already exist, can the placement of the square blocks be accomplished with k stigmergic rules? The following will show that the original board has a cover of k rectangles iff the answer to this question is *true*.

First, suppose that there exists a cover of k rectangles for $\mathfrak{B}$. Clearly, each rectangle (which contains a certain number of squares in $\mathfrak{B}$) can be translated into a set of memory conditions on the variables x^b and y^b defining a rectangular region of space, which encloses the square block locations. These memory conditions can be associated with a rule that specifies five full sites below the goal site (capturing the

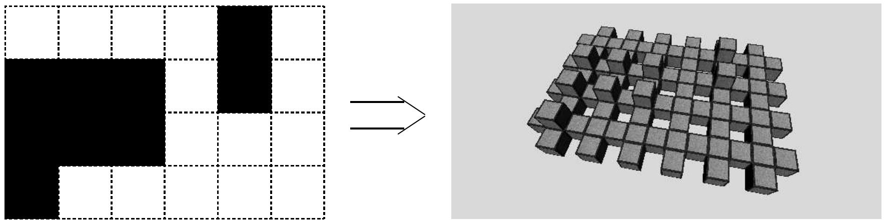


Figure 5.4: An example polynomial time reduction from a board $\mathfrak{B}$, depicted as a set of filled squares on a grid (left side), to a corresponding target structure $S_{\mathfrak{B}}$. Each square in $\mathfrak{B}$ corresponds to a non-ground block in $S_{\mathfrak{B}}$; each "cross" pattern (consisting of five blocks) on the ground corresponds to a cell on the grid.

"cross" pattern). This geometry will prevent the rule from applying at any location that is not above an intersection of two rows, and the memory conditions will prevent it from applying above intersections where it should not apply. Generating a rule for each rectangle in this fashion will yield k rules that correctly place the square blocks.

Now, suppose that there exists a set of rules R for placing the square blocks in $S_{\mathfrak{B}}$, where $|R| = k$. Consider the memory conditions within some rule in R. Note that assembly rules do not use memory conditions with inequalities ('$\neq$'), because the possible number of such memory conditions grows with S, and the size of an assembly rule would no longer be bounded by a constant. (A single condition with an inequality is used in some disassembly rules, as shown in Figure 5.2c, and applies to the variable τ^b). Then, any set of memory conditions on variables x^b, y^b and z^b restricts the application of a rule to a finite, contiguous, hexahedral region of space. (Clearly, this is the case if there is a set of six memory conditions $\{x^b \geq x_{min}, x^b \leq x_{max}, y^b \geq y_{min}, y^b \leq y_{max}, z^b \geq z_{min}, z^b \leq z_{max}\}$. If there are fewer than six memory conditions, then the region of space can be said to extend somewhere past the boundary of $S_{\mathfrak{B}}$. Any other condition in addition to the six creates a redundancy). Within the hexahedral region, there is some number of goal locations for the square blocks. Because each goal location has an identical pattern of locations adjacent to it, if the rule applies at one of these locations, then it applies at all goal locations within the region. Thus, the rule corresponds to some rectangle that encloses the corresponding squares in $\mathfrak{B}$. (If the rule is useless, in that it applies at no goal location, then a redundant rectangle can be defined). This completes the argument, and shows that there exists a polynomial time reduction from the problem of rectangular cover to the problem of generating a minimal set of rules.

It is thus shown that optimally parsimonious rule generation is NP-hard. Whether or not it is NP-complete (i.e., whether or not it is in NP) depends on the ability to efficiently test whether or not a candidate set of rules R will correctly assemble a structure. Given the presence of physical constraints, this may be quite difficult, because it is necessary to prove that R will yield a correct structure regardless of the order in which blocks are deposited, and the number of possible orders is generally intractable. A further characterization of the complexity of parsimonious rule generation can be a subject of future investigation.

In conclusion, assuming that P $\neq$ NP, we cannot hope to produce a minimal set of stigmergic rules for arbitrary target structures, due to the complexity of computing optimal sets of memory conditions. It should also be noted that computing an optimal geometric structure for the rules presents a hurdle as well: while the number of possible rule geometries is finite (since each rule has a finite number of sites, and each site is either empty, corresponds to a small, medium or large block or a block section, or is a wildcard), this number is clearly very large. As discussed in Section 5.1.2, memory conditions are computed via a greedy approximation to an optimal hexahedral cover, and full sites are specified based on a heuristic that takes into account the constraint that a goal location should not be filled until all adjacent goal locations that are below it are filled. In the following section, the effectiveness of the approach is evaluated.

5.3 Experimental Results

Here, the performance of the methodology developed in Section 5.1 is examined, both in terms of the parsimony of the generated rules, and the efficiency of the self-assembly processes that result from their application.

5.3.1 Rule Set Size Reduction

As in Chapter 4, the rules output by the (modified) rule generation procedure are first analyzed, ignoring (for the moment) their use in simulations. The procedure was applied to each of the six target structures (Figure 3.1), successfully yielding sets of rules. This version of the procedure required substantially more processing time, compared with the version of Chapter 4; in particular, the determination of all locations where any given rule (without memory conditions) may apply but should not (i.e., the set $u(r, C, S')$ in Section 5.1.2) is somewhat computationally intensive. However, the generation of rules could still be accomplished in around 1 to 4 minutes

Table 5.1: Static Performance of the New Rule Generation Procedure

Structure	Blocks	Stages		Msg. Vars.		Rules Generated					
						Stig.		V. C.		Total	
		Qt.	Rt.	Qt.	Rt.	Qt.	Rt.	Qt.	Rt.	Qt.	Rt.
Barrier	547 (576)	4	0.11	50	0.88	469	0.56	292	0.26	761	0.39
Bridge	360 (425)	12	0.26	24	1	448	0.48	278	0.45	726	0.47
Building	289 (323)	6	0.10	11	1.38	149	0.37	80	0.23	229	0.30
Fence	112 (322)	14	0.25	25	1	516	0.54	551	0.53	1067	0.54
Pyramid	268 (268)	1	0.14	1	0.33	43	0.52	4	0.09	47	0.37
Road	192 (192)	1	0.11	11	0.85	114	0.48	34	0.21	148	0.37

per structure, on an aforementioned Dell Precision WS machine. The properties of the resulting rule sets are given in Table 5.1. The entry in the “Blocks” column lists the total number of blocks in the given structure; this corresponds exactly to the sum of the counts of small, medium and large blocks in Table 4.1. As before, the quantity outside parenthesis provides the total number of blocks in the target structure itself, whereas the number within parenthesis includes the lower bound on the number of blocks necessary to also build the temporary substructures. For an easy comparison with the results of the previous chapter, each entry in the remaining columns is provided as the actual quantity measured (“Qt.” subcolumn) and the ratio of that quantity to the corresponding quantity in Table 4.1 (“Rt.”). Recall that in Table 4.1, each entry is given as ⟨value for the preliminary ordering algorithm⟩ / ⟨value for the full ordering algorithm⟩, unless both algorithms produced the same result. All comparisons reported here are based on the latter values, because the modified version of the rule generation procedure is based on the full ordering algorithm, which reuses temporary substructures.

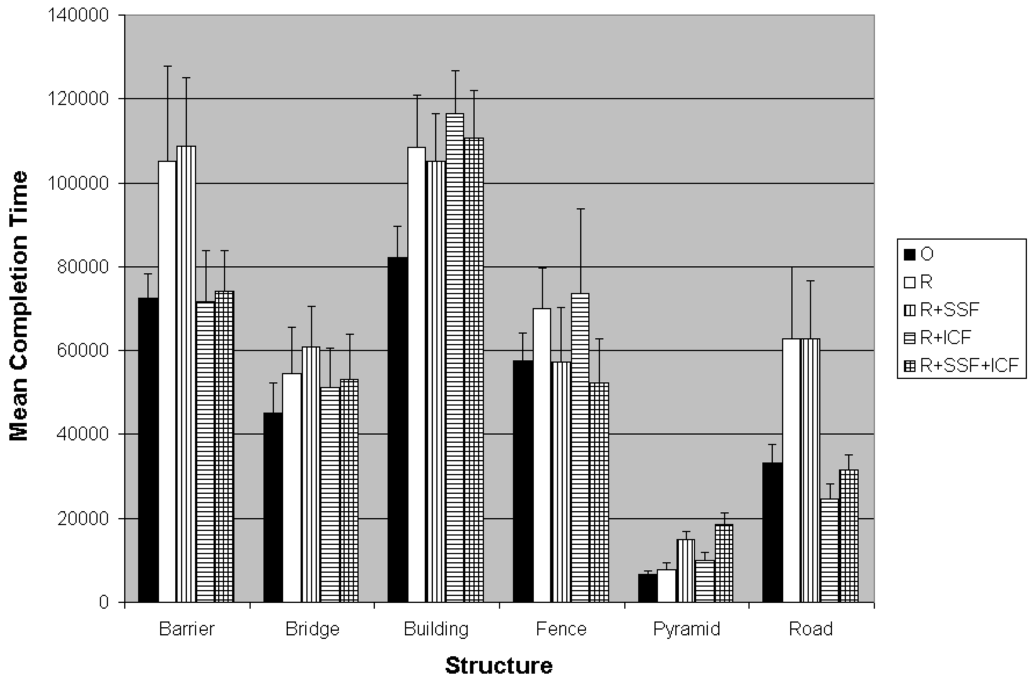
For all six target structures, the “Stages” column of Table 5.1 depicts a marked reduction in the number of assembly/disassembly stages that take place before the structure is complete. This is a direct consequence of the collapse procedure (Figure 5.1), which modifies the original order to allow multiple layers of structure to assemble/disassemble simultaneously. Interestingly, this has a varying effect on the number of message variables needed to communicate the completion of all substructures within a given stage, as shown in the “Msg. Vars.” column. In some cases, this number is reduced: for example, for the pyramid, the first and second layers (as well as the part of the third layer which consists of small blocks) are each decomposed into 3 rectangles, whose completion must be detected, under the old methodology, before the subsequent layer is deposited. Under the methods of this chapter, the pyramid is assembled in a single stage, and the completion of the structure is detected by simply noting the placement of the topmost block. For the building, an opposite effect is observed: 11 message variables are used to detect the completion of the stage wherein the walls and the first roof layer are deposited. Under the old methodology, this number is 8, because blocks of different sizes are deposited in different stages (Section 4.1.3), and there are fewer rectangles to detect per stage. Still, the overall trend (among the six target structures) is for a decrease in the number of message variables.

The remaining columns of the table depict rule counts (and corresponding reductions) for the different target structures. Both stigmergic and variable change rule counts decreased substantially, with the reduction in the latter being more pronounced. This can be attributed to the fact that the collapsed order O' specifies a low number of stages, and thus entails considerably less high-level sequencing. Overall, the parsimonious rule generation procedure typically resulted in sets of rules with a size that is less than one half of what is reported in Chapter 4. The fence is the only structure where the ratio is slightly greater than 0.5, and as before, it resulted in the largest set of rules. This can be attributed to the presence of many disjoint substructures within each layer, and the "alternating" nature of these layers. The ordering algorithm, whether preliminary or full, specifies the assembly and disassembly of a number of different staircases after each layer is deposited. The reduction in rules arises from the fact that the staircases are no longer assembled layer by layer, and further, from the ability to detect repeating patterns within the substructures, which cannot always be efficiently described as sets of rectangles. Ultimately, this sort of pattern extraction typically proved itself to be more parsimonious than rectangular decomposition: for all six structures, a reduction was observed in the number of generated assembly rules, which helped to reduce the overall number of stigmergic rules. As discussed in Section 5.1.2, rectangular decomposition still plays a role in the production of rules for the assembly of ground substructures and for completion detection. This adds a significant number of rules for the bridge (which has many ground substructures, as shown in Figure 4.1) and the barrier (where the top surface of the side walls consists of medium blocks that are laid out in a zigzag pattern, as best illustrated in the subsequent Figure 5.6a, such that each block constitutes a separate rectangle).

Can we consider any of the generated rule sets to be truly parsimonious? As discussed earlier (beginning of Section 5.1.2), the convention taken is that parsimony is achieved when the size of the rule set $|R|$ is smaller than the size of the structure $|S|$. According to Table 5.1, this is satisfied for the building, pyramid and road, even if one does not count the blocks necessary for building temporary substructures when computing $|S|$. (By contrast, the methods of Chapter 4 were only able to generate a parsimonious rule set for the pyramid). However, recall from Section 4.1.4 that the rules governing mode changes are not constant in size; more precisely, the number of memory conditions within each mode change rule is equal to the number of messages used in the corresponding stage. The total size of these rules is thus proportional to the total number of message variable-value pairs used over the entire self-assembly process. The number of such pairs for the building, pyramid and road is 26, 1 and 11, respectively. If we subtract the number of mode change rules (which is equal to the number of stages) from the total rule count, and add the number of message variable-value pairs used (to attempt to account for the total size of these rules), we obtain values of 249, 47 and 158. These values, even if treated as rule counts, are still smaller than total block quantities within the corresponding target structures. Thus, even under this stricter definition of parsimony, the new version of the procedure was able to generate parsimonious sets of rules for half of the target structures tested.

5.3.2 Effects on Performance during Self-Assembly

Now, the performance of the reduced rule sets is discussed, as evaluated via simulations. The experimental methodology was essentially unchanged from the previous chapter: for each rule set, 30 trials were performed, with distinct streams of random numbers. One difference is that in the experiments discussed below, there was no enforcement of the termination criterion, where a trial is terminated if no useful progress (i.e., a moving block becoming stationary or a stationary block commencing movement again) is made for more than 30000 time steps. Some of the experiments yielded a few such trials (the discussion below makes note of them), but the period of “no progress” was 51609 time steps in the worst case, and the completion of all trials could still be observed in a tractable amount of time.



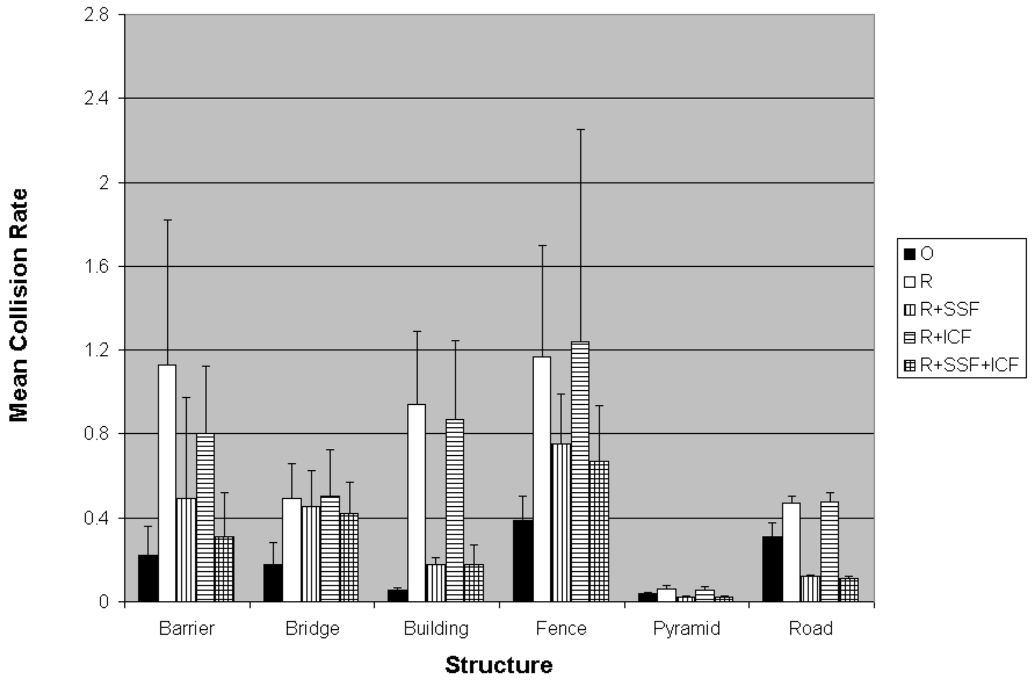


Figure 5.5: Mean completion times (top) and collision rates (bottom) for original (O) and reduced (R) rule sets, with possible modifications to the movement control methods: Selective Step Force (SSF) or Intergroup Collective Force (ICF), which are explained in the text. Error bars denote standard deviations.

For an initial set of experiments, the generated rule sets were tested with movement control mechanisms that were unchanged from those used in the previous chapter; i.e., the trials were essentially different only in the rules that were used to govern self-assembly. All trials eventually ran to completion, correctly yielding the final target structures; however, four trials involving the self-assembly of the barrier "stalled" for more than 30000 time steps (51609 time steps in the worst case), whereas no such trials were observed in the experiments of Chapter 4. A side-by-side comparison of the performance of the reduced rule sets ("R") (from this chapter) and the original rule sets ("O") (summarized in Table 4.2) is provided by the first two bars in Figure 5.5, for each target structure. (Subsequent bars correspond to experiments that involved modifications to the movement dynamics, and will be discussed later). In all cases, the substitution of rule sets from Chapter 4 with the reduced rule sets yielded higher completion times and collision rates, on average.

In order to determine whether the differences are statistically significant, two-tail t-tests (assuming unequal variances) were used to compare means. The results of these comparisons are shown in Table 5.2 for completion times, and in Table 5.3 for collision rates. The second column of the tables compares the performance of the reduced vs. original rule sets ("R vs. O"), under an identical force computation method. The '$-$' signs for all entries indicate that the reduced rule sets resulted in significantly ($p < 0.05$) worse runtime performance (i.e., higher completion times and collision rates) for all target structures.

Thus, it is apparent that the parsimony gained in the newly generated rule sets comes at the price of reduced runtime efficiency and greater interference during construction. Let us now attempt to examine the causes of this tradeoff. First, recall that in the computation of the original order, it is typically ensured that substructures consisting of small, medium and large blocks assemble in different stages of the process (Section 4.1.3). The collapse procedure (Figure 5.1) combines these substructures into larger substructures that contain blocks of different sizes. As observed in Chapter 3, significant interference can result when blocks of different sizes simultaneously attempt to find and fill goal locations. A concrete example of this sort of interference is illustrated in Figure 5.6a, which shows a difficult situation that can arise during the construction of the barrier. The barrier itself is almost completed, with the exception of one unfilled location (marked with an arrow on the lower right of the figure), which requires a medium block. While there are medium blocks atop the barrier, they are separated from the goal location by a group of small blocks. These latter blocks wander aimlessly, and remain atop the structure for long periods of time, because they follow assembly rules in the current stage, and their dynamics are typically conducive to ascending, rather than descending staircases, as stated in Section 4.2. While the collisions between the two groups of blocks are not exceptionally frequent, small blocks are quite effective at preventing medium blocks from reaching the goal location, essentially displaying an unwanted "guarding" behavior [94]. In some cases, as stated above, it can take more than 30000 time steps to resolve such a situation.

Given that no special measures were taken to account for the fact that blocks of different sizes now seek goals simultaneously, the decrease in performance comes

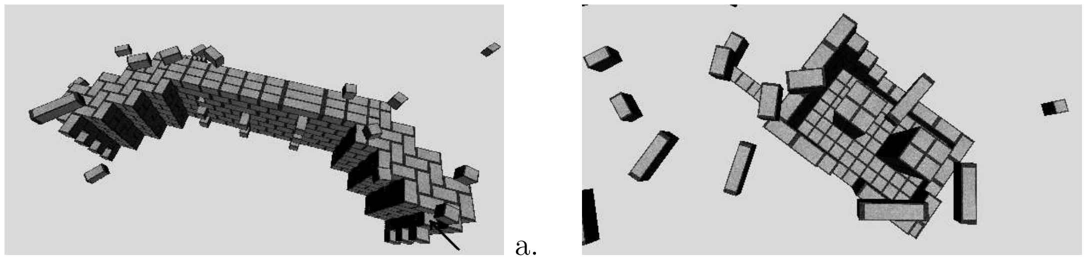

a. b.

Figure 5.6: Examples of potential causes of inefficiency and interference, arising under the reduced rule sets: (a) small blocks prevent a medium block from reaching the final goal location during barrier construction; and (b) the walls of a building develop unevenly, and this can interfere with the motion of blocks atop them, due to the indiscriminate computation of the stair climbing force.

as no surprise. In addition to this, I identified another potential issue that can arise when the collapsed order O' is followed. Notably, because O' allows multiple layers of a structure to assemble simultaneously, its developing topography can vary significantly in height. For example, in Figure 5.6b, the partially completed walls of a building are considerably better developed in regions near the staircase than in the far right corner. The top surface of the walls is thus stair-like in places. Recall from Section 4.2 that for simplicity, blocks following automatically generated rules compute a stair climbing force $\mathbf{F}_{su}$ indiscriminately, typically considering any nearest stationary block with nothing above it as a step to be climbed. In some cases, this can have undesired effects; for example, a medium block may be reluctant to climb down to reach the wall sections where it is needed, because of its tendency to ascend, rather than descend staircases.

The subsequent section discusses modifications to the blocks' movement dynamics, which are aimed at mitigating these two problems.

Table 5.2: Effects upon Completion Time

Structure	R vs. O	R+SSF vs. O	R+ICF vs. O	R+SSF +ICF vs. O	R+SSF vs. R	R+ICF vs. R	R+SSF +ICF vs. R
Barrier	−	−	≈	≈	≈	+	+
Bridge	−	−	−	−	−	≈	≈
Building	−	−	−	−	≈	−	≈
Fence	−	≈	−	+	+	≈	+
Pyramid	−	−	−	−	−	−	−
Road	−	−	+	≈	≈	+	+

Table 5.3: Effects upon Collision Rate

Structure	R vs. O	R+SSF vs. O	R+ICF vs. O	R+SSF +ICF vs. O	R+SSF vs. R	R+ICF vs. R	R+SSF +ICF vs. R
Barrier	−	−	−	≈	+	+	+
Bridge	−	−	−	−	≈	≈	≈
Building	−	−	−	−	+	≈	+
Fence	−	−	−	−	+	≈	+
Pyramid	−	+	−	+	+	≈	+
Road	−	+	−	+	+	≈	+

5.3.3 Improving Performance

Recall that in Chapter 3, the step forces were not applied indiscriminately; rather, they were parametrized by the set T_{su} of substructure types that corresponded to only those substructures that were meant to be climbed (see Section 3.2.3 for details). All other substructures were ignored by $\mathbf{F}_{su}$ (as well as $\mathbf{F}_{sd}$, which is no longer used). As with other aspects of control, the exact choice of substructures for T_{su} was made by hand. Here, an attempt is made to improve the runtime performance of the generated rules by allowing this choice to be made automatically. At the beginning of a simulation, blocks determine, based on existing disassembly rules, the substructure types of all staircases (only staircases and connectors require disassembly rules, and it is possible to differentiate between the two), and these substructure types make up T_{su}. While this is often sufficient, there are target structures such as the pyramid or the road, where temporary (i.e., eventually disassembling) staircases do not exist. For the road in particular, access to higher parts of the structure is provided by two permanent staircases, and if $\mathbf{F}_{su}$ is not activated in the vicinity of these staircases, then it is very unlikely that blocks will climb them by chance alone. To take such substructures into consideration, a slight modification was made to the full ordering algorithm (Figure 4.4), such that when it removes a substructure C from S, it keeps track of locations belonging to S (i.e., any locations not generated as part of a temporary staircase by f_T) that are nonetheless used in accessing C. The set of all such locations is subsequently decomposed into rectangles, and for each rectangle, a variable change rule is generated, causing a depositing block b that becomes part of this rectangle to set the *step flag* s^b (a previously unused variable in its memory) to 1. This modification caused an addition of 6 rules for the bridge, 1 rule for the building, 5 for the pyramid, and 3 for the road; otherwise, the generated rule sets were just as before. When a moving block detects that a stationary block b has $s^b = 1$ set, it takes this stationary block into account when computing the step force, just as if $\tau^b \in T_{su}$ were true.

The effects of this *selective step force* (SSF) modification are illustrated by the third bar (for each target structure) in Figure 5.5. They are also compared statistically against the performance of the original rule sets (Chapter 4) in the third column of Tables 5.2 and 5.3, and to the performance of the reduced rule sets (but without the modification) in the sixth column. A '+' indicates that the first set of trials mentioned at the top of the column (i.e., the reduced set of trials with SSF) performed significantly better than the second set of trials mentioned; a '$\approx$' states that the difference in performance is not statistically significant ($p > 0.05$). It is apparent that SSF has the general effect of lowering collision rates; for the pyramid and the road, they were reduced even below the numbers reported in Chapter 4. As far as completion time is concerned, the effect is less consistent. It is noteworthy that the road has permanent staircases at both ends, but under the current implementation, only one of these is considered by the algorithm as essential for access; thus, blocks in the other staircase have $s^b = 0$, and are not actively climbed. This seems to be counterbalanced by a reduction in the interference between blocks, yielding a similar mean completion time as before. For the bridge (which is the only structure where a significant reduction in collision rates is not observed), $s^b = 1$ holds for the bottom three steps of both permanent staircases (which consist of large blocks), but the top steps and the large blocks just above them (seen clearly on the left side of Figure 4.1d) have $s^b = 0$, which may contribute to the slowing down of the self-assembly process. A similar effect occurs for the pyramid: whereas before, it could be climbed from all directions, with SSF, only some of the blocks have $s^b = 1$ set. On the other hand, SSF allows the fence to assemble in a significantly shorter amount of time, possibly, because it has a rather jagged surface when it is under construction, and the movement force of a block is no longer sensitive to this. For the building, SSF actually does not have a significant effect on completion time, even though collision rates are reduced. The same holds true for the barrier, where the issue depicted in Figure 5.6a still arose, and two trials stalled for more than 30000 time steps (36091 in the more severe case).

How can such interference be mitigated? Figure 5.6a shows that the small and medium blocks essentially behave as competing teams/flocks [94]. This can be attributed to the computation of collective forces $\mathbf{F}_{nc}$ and $\mathbf{F}_{na}$ (Section 3.2.3), which are defined only between blocks that are identical to each other in size. A very simple *intergroup collective force* (ICF) modification removes this restriction, resulting in the emergence of flocks that contain blocks of different sizes. It was observed that this "mixing" of different-sized blocks allowed the situation of Figure 5.6a to resolve in a considerably shorter amount of time, and resulted in a faster assembly of the barrier, with lower collision rates (Tables 5.2 and 5.3, seventh column).

For other target structures, ICF had no significant effect on the collision rates, and a varying effect on completion times. It is difficult to gauge the exact impact of this modification in different situations, but it appears to be beneficial for target structures where many blocks of different sizes are deposited in the same stage, as is the case for the barrier and the road. In fact, with the modification, the road assembles faster than it did in Chapter 4, as indicated in the fourth column of Table 5.2. On the other hand, structures such as the building (where the floor and columns consist of small blocks, the walls consist almost entirely of medium blocks, and the roof consists of large blocks) and the pyramid (where all but three blocks are small) seem to benefit from more competitive intergroup dynamics, as employed previously. For the fence, average-case performance is not significantly affected by ICF; however, one trial was observed with no useful progress for 41609 time steps. The difficulty occurred after the assembly of the fence itself has been almost completed, except for the placement of a single medium block. Because medium blocks are disconnected from each other (Figure 3.1f), a staircase is generated for each one, and a long period of time was needed to finish assembling the staircase for reaching the unfilled location, because blocks would often climb the wrong staircases. A stall in the system's progress was observed in some other trials/experiments around the same point in the self-assembly process, but in all other instances, considerably less time was necessary to resolve the situation. The length of the delay in this particular trial may be partly attributed to chance, but it may have been made more probable by the fact that without the SSF modification, small blocks attempt to climb the medium blocks already deposited atop the fence, and possibly may remain atop the structure longer, before climbing down and attempting to find the correct staircase.

In the final set of experiments, both SSF and ICF were applied simultaneously. As shown in the graphs and the tables, this appears to provide the greatest benefit overall, in terms of improving efficiency and reducing collision rates. Most significantly, in this set of experiments, no trials stalled for more than 30000 time steps. It appears that the two modifications are often able to simultaneously provide their respective benefits to the self-assembly processes. While the runtime performance of the reduced rule sets with both modifications is still overall not quite as good it was under the original rule sets, it is comparable (in terms of completion time) for the barrier and road, and superior for the fence, as indicated in the fifth column of Table 5.2.

5.4 Discussion

This chapter considered the question: is it possible to modify the rule generation procedure presented earlier in order to allow the production of rule sets that *parsimoniously* assemble a given target structure? In the beginning, two key causes of large rule sets were identified: (a) the originally computed order specifies a high number of stages, which necessitates a large number of rules for sequencing from one stage to the next; and (b) rectangular decomposition (wherein a certain number of rules is generated to assemble each rectangle) often fails to capture more intricate patterns that may be present within a target structure. A new procedure (for individual substructures) was developed that generates rules according to patterns existing immediately below each goal location, and possibly restricts their application further with memory conditions. It was argued that this heuristic allows for the correct self-assembly of a large class of structures, even under a more relaxed version of the order: although this order potentially allows certain problematic situations to occur (e.g., Figure 4.2a), the generated stigmergic rules contain implicit constraints that prevent such situations from arising. Furthermore, it was hypothesized that the approach has the potential for effectively capturing various repeating patterns within a structure.

After evaluating the new methodology on the six target structures (Figure 3.1), substantial rule set reductions were observed in all cases, and for three structures, the resultant rule sets can be considered parsimonious. These results were found to be quite encouraging, in light of the fact that parsimonious rule generation is an inherently difficult problem (as argued in Section 5.2) even without the presence of physical constraints, whereas the procedure developed here was required to generate rules not only for assembling a structure, but for ensuring (through higher-level coordination) that correct self-assembly can take place in spite of these constraints. It is inferred that as far as assembly rules are concerned, parsimony is chiefly achieved in two ways. First, the geometry of a generated rule is sometimes able to effectively capture local patterns that repeat, as in traditional stigmergic models [18, 110]. On a more global scale, while memory conditions serve to restrict the application of rules, they can still allow a rule to be applied within a potentially large region of space, because they are defined over integer variables, rather than qualitative variables (such as colors) [2, 48, 49]. Given a constrained environment, it was found that while stigmergic rules are not able to capture arbitrary ordering constraints (see Section 3.2.4 for a further discussion), using them in an appropriate fashion (e.g., such that unreachable holes are not developed) can substantially decrease the number of rules needed for higher-level sequencing. The prevention of locally unreachable holes has also been addressed in [48, 116, 117, 118, 120, 121], although reported methods differ from the ones developed in this dissertation.

Although the rule set size reductions were considerable, it was also found that the runtime performance of the reduced rule sets was less efficient than the performance of rule sets from Chapter 4, and generally resulted in higher collision rates between blocks. This once again illustrates the complex interplay between rules and movement dynamics, which was already discussed in Chapter 3, where the integrated control methodology was first developed. After identifying two potential causes of the parsimony-efficiency tradeoff, it was possible to improve performance significantly for most structures at the expense of very small increases in the number of rules. For a few of the structures, the reduced rule sets (along with the modifications) performed comparably with, or even superior to the original automatically generated rule sets; for others, the original rule sets still had better performance.

Among these latter structures is the building, and as discussed at the end of the previous chapter, even the original automatically generated rule set was unable to achieve the level of runtime efficiency that was gained when rules were designed by hand (Chapter 3), so given the reduced rule set, the gap is even greater. With respect to parsimony, while the reduced rule set is vastly superior to the rule set from Chapter 4 (229 vs. 753 rules), it is still somewhat larger than the hand-designed set, which contains only 143 rules. Thus, room for improvement remains. It is suspected that when designing automatic rule generation procedures and control methods for arbitrary structures, further improving parsimony and efficiency will be a highly challenging problem, due to the theoretical limitations that underlie parsimonious rule generation (Section 5.2), the complexity of the relationship between particular choices of movement control mechanisms and overall performance (recall from Section 5.3.3 that the modifications made to force computation had varying effects, depending on the target structure), and the tradeoff between the two objectives. In the final chapter, I shall outline potential ideas for further improvement, as well as other research directions.

Chapter 6

Discussion

This concluding chapter reviews and critically examines the work presented in the dissertation, summarizing its main contributions, and outlining several possibilities for future research.

6.1 Summary and Limitations

During the last decade, the problem of self-assembly has received an increasing level of attention from researchers in computer science and other fields, in part due to the falling costs of computing hardware and an increased interest in nature-inspired computation in general. These efforts have focused on designing local control mechanisms that individual components can follow, in order to converge to some desired structure, while subject to environmental constraints. As discussed in Chapter 2, a variety of approaches have been developed for achieving this sort of distributed control in various environments, which range from very simple, cellular spaces to the real world. An overview of existing literature revealed that the complexity of the environment plays an enormous role in determining the difficulty of the problem. While methods have been developed for assembling somewhat arbitrary (though generally not very realistic) discrete structures in environments that impose no constraints on the components' motion, the range of structures that have been successfully self-assembled decreases dramatically as the environment becomes more complex. Thus, structural and environmental complexity present two conflicting tradeoffs, which are difficult to address simultaneously.

In this context, I set out to achieve the self-assembly of a broad range of structures (e.g., Figure 3.1) in an environment that, while simulated, presents far more complexity than existing methods (for assembling diverse structures) could handle, if faced with its characteristics. Specifically, this environment is a three-dimensional, continuous world, where the components' horizontal motion is constrained by their impenetrability, and their vertical motion is restricted even further to simulate gravity. The components are embodied as different-sized blocks, introducing into the problem a degree of heterogeneity that has received limited attention in past work on self-assembly. This heterogeneity was useful in the production of structures with a more realistic appearance relative to past studies, which yielded structures that may be somewhat non-trivial, but typically do not resemble real world objects (e.g., Figure 2.4a).

How must blocks behave in order to successfully self-assemble into desired structures within such an environment? In an attempt to answer this question, I turned to the field of swarm intelligence. The choice was motivated by the fact that swarm intelligence studies the dynamics of simple agents that are able to achieve complex, collective behavior and self-organization. Moreover, unlike in other distributed computational paradigms (such as neural networks [73] or cellular automata [100]), swarm intelligence systems have a strong underlying notion of *motion* through a *space*, and the process of self-assembly is inherently one of spatial movement. Finally, swarm intelligence takes inspiration from the collective behavior of animals such as social insects, where certain species have been able to achieve remarkable tasks of self-assembly [5] and collective construction [16, 17, 18, 50, 51, 110], in spite of the relative simplicity in the behavior of each individual organism.

The developed control mechanisms can be viewed as an extension to models of nest building by paper wasps, where an individual wasp uses simple, local pattern matching to determine where to add material (pulp), in a process known as *stigmergy* [111]. In the system under study, where the self-assembling blocks constitute both the agents and the material, this behavior has been enhanced with the use of variables, which exist in the internal memory of each block. A goal location (where a block should deposit itself) is then determined by a *stigmergic rule*, which specifies not only the geometric pattern surrounding this location, but also, the acceptable variable values taken on by neighboring blocks; an appropriate set of such rules can be used to implicitly describe any structure (rather than just nests). In order for blocks to find and fill goal locations that are defined by the rules, while operating in a continuous environment with constrained motion, stigmergic behaviors have been integrated with force-based movement control, where each block determines its acceleration as if it were under the influence of a number of forces. As in many other systems with collectively moving agents (e.g., [92, 94, 123]), these forces do not actually exist in the environment, but rather, serve as internal "influences" that an agent computes to determine where to go; this approach requires neither extensive computational resources nor the availability of global information, in contrast to alternative methodologies such as conventional motion planning techniques [37, 64].

Together, the forces and the stigmergic rules constitute a set of low-level, reactive behaviors that drive the self-assembly process. However, due to the complexity of the environment, these behaviors must not be applied indiscriminately, but rather, must vary through time. Specifically, physical constraints such as gravity and impenetrability translate into ordering constraints on the self-assembly process, where certain parts of the structure must be completed before others are begun, and where temporary substructures (such as staircases) must be assembled and disassembled at appropriate points in time. Early swarm intelligence systems were generally incapable of such higher-level coordination; however, recent work has shown that it can be achieved by endowing agents with internal state [94, 95]. This state determines the exact subset of rules that a block attempts to match, and affects its movement dynamics; in turn, it can be modified in response to local events via *variable change rules*.

Designing an integrated controller for individual blocks proved to be a challenging problem, due to the complex interactions between the individual techniques. For example, situations can arise where a block detects a goal location, but is unable to reach it (perhaps because its path is obstructed by other blocks). In such instances, it is far more prudent to deactivate the goal approach force [115], rather than attempt to pursue the goal for an indefinite amount of time. Given a fixed set of rules for a specific target structure, Chapter 3 examined the effects of different variations on the force computation method. The experiments lead to the conclusion that in attempting to achieve effective self-assembly, there is often a tradeoff between maximizing the availability of blocks in regions where they are needed, and minimizing the interference between these blocks. This tradeoff appears to be fundamental in nature; its analogs are encountered in abstract, parallel computation [47], where the attempt to perform certain operations simultaneously is constrained by the dependencies between these operations.

In the early stages of the research described in this dissertation, the unassisted design of rules by hand was somewhat akin to writing a piece of software prior to the development of structured programming in the 1960s and the 1970s: it was haphazard and error-prone. However, during the course of this effort, the method for rule specification became more well-defined, and the question arose as to whether it could be formulated as an effective procedure, and thus automated. A major obstacle to the development of such a procedure was, once again, the constrained nature of the environment. The presence of gravity and impenetrability imposes ordering constraints on the self-assembly process, and necessitates the use of temporary substructures (such as staircases), whose assembly and disassembly must be specified within the order as well. Thus, algorithms were developed for computing an order that satisfies all physical constraints, and serves as an intermediate representation between global structures and local rules. Further, a methodology was presented for generating rules that enforce this order at runtime, and ensure that the self-assembly process is sequenced appropriately through its various stages. Finally, by decomposing the various substructures into rectangles, and generating appropriate stigmergic rules for each rectangle, it was possible to achieve the automatic specification of pattern matching behaviors necessary for assembly and disassembly. The rule generation procedure thus *adapts* swarm intelligence techniques to the assembly of a prespecified structure.

In a non-trivial environment such as the one used here, the relationship between individual and system-level behavior is highly complex, and subject to chaotic effects that result from non-linear interactions between blocks. To my knowledge, there is no proof that the given set of control mechanisms will guarantee that the system will always converge to the desired structure. However, when environmental details were abstracted away, it was possible to establish and prove a number of important formal properties (such as correctness and completeness) about the methodology, while the validity of the assumptions (upon which the proofs depend) could be verified experimentally. Thus, the work on automated rule generation was essentially conducted on two levels. Abstractly, it was indeed possible to show a formal correspondence between a given structure in some environment, the high-level coordination requirements (represented as a correct order) imposed by the environment on the self-assembly of the structure, and the individual rules (also proved to be correct) that are necessary for achieving this coordination. On a more concrete level, ordering and rule generation procedures were successfully implemented for the specific environment of Section 3.1.2, and it was experimentally shown that the generated rules effectively achieve self-assembly.

While rule generation for self-assembly processes has been achieved in the past, the output rules would generally allow the use of only low-level behaviors (such as the deposition of a block [2, 48] or the joining of two components [55, 57]), and would thus have difficulty achieving self-assembly in a complex environment. At the same time, the adoption of higher-level behaviors (which can allow the system to sequence through various operational stages) has recently made its way into certain self-assembly [6, 121] and swarm intelligence [94, 95] systems, but the agent controllers which allow such behaviors to emerge are typically designed by hand. The result of Chapter 4 is important in that it demonstrates the possibility of automatically discovering local rules that cause the emergence of explicit high-level coordination within the overall system.

While the automated methodology was successful at achieving self-assembly, its drawback (relative to the design of rules by hand) was in the size of generated rule sets. Thus, the later part of this dissertation dealt with the problem of generating rule sets that are not only correct, but parsimonious as well. It was established that finding a minimal set of rules is a computationally difficult problem, and that for many structures, even this minimal set cannot be parsimonious. However, it was found that for the given target structures, which contain repeating patterns and resemble human constructions, it is possible to achieve significant rule reductions, leading to parsimonious sets of rules for three of the six structures. This was done through the use of a simple heuristic, where generated rules are based on the local patterns that exist below goal locations within a structure. In particular, it was shown that because these rules ensure (in an implicit, stigmergic fashion) that a goal location is not filled with a block until all adjacent locations below it are filled, the need for explicit, higher-level sequencing is greatly reduced, which also accounts for a smaller number of rules.

While these results were encouraging, it was discovered, through experiments, that the greater parsimony comes at the price of reduced runtime efficiency and increased collision rates. This once again illustrates the non-trivial nature of the interactions between the different facets of the control methodology (namely, rules and force equations). Certain causes of the tradeoff were identified, and (for at least some of the target structures) the runtime performance of the reduced rule sets was improved to a level that was comparable to the runtime performance of the larger rule sets from Chapter 4. However, it was found that the effects of the modifications varied, depending on the target structure in question, indicating that the problem of making general (i.e., not structure-specific) optimizations to parsimony and efficiency simultaneously is expected to be difficult. Still, this problem may be worthy of further investigation, considering that for the building target structure (Figure 3.1d,e), the hand-designed rules, coupled with force-computation methods (which were initially developed with the building as a test case, and only later applied to other target structures) outperformed, both in terms of efficiency and parsimony, any rule sets that were generated automatically. Thus, the suboptimal (though otherwise effective) nature of the automated approach presents one limitation of this work.

Certainly, the approach presented here has other limitations as well. In particular, while the environment under consideration models a number of physical constraints, it certainly does not take account the full complexity present in the physical world, with its friction, noise and uncertainty. The transfer of control mechanisms from simulated agents to robotic devices is not a trivial endeavor [23], and one cannot expect the methodology to apply in a real world environment without extensive modification. Still, by modeling at least a subset of the constraints that may be present in such an environment, this work attempts to narrow the gap that exists between present-day self-assembly methodologies and the needs posed by practical construction applications [22, 38, 39, 40, 87, 97, 101].

A related issue is that of robustness, which is a critical factor for any system that interacts with the physical world. While the force computation mechanisms developed here are not sensitive to certain types of perturbations, and (under appropriate conditions) actually benefit from a moderate degree of noise (as shown in Chapter 3), we presently cannot expect the system to be robust to errors of a discrete nature, such as deformations in the assembling structure, where a block occupies an incorrect location. Such a deformation can occur due to outside environmental forces, such as high winds or an earthquake; however it may also result from sensor failure that leads to a rule misapplication. In either case, the error can cause further rule misapplications, yielding an incorrectly built structure.

Finally, it is worth mentioning that while the agents (blocks) are adapted *a priori* to assemble a given target structure under the current environmental conditions, their ability for *runtime* adaptation are presently very limited. An agent maintains an internal memory in order to be aware of the current state of the self-assembly process, or to know its relative position within the assembling structure; it may also recall certain events from the recent past, such as decisions to not pursue a goal for a certain number of time steps. Beyond this, agents do not improve their behavior based on past experiences; i.e., true learning does not occur. As discussed at the end of this chapter, endowing agents with the ability to self-adapt may help them to overcome some of the other limitations given here.

6.2 Contributions

Here, I restate and elaborate on the main contributions of this dissertation:

- A methodology was developed for the distributed control of self-assembly processes, with a greater degree of complexity (both structural and environmental), as compared with past approaches. This methodology successfully integrates several distinct techniques from the field of swarm intelligence, namely, stigmergic pattern matching used in collective construction, force-based control of multi-agent motion, and coordination via the use of state (memory) along with basic communication. Stigmergy, enhanced with the use of integer variables, allows blocks to (locally) determine goal locations; force-based movement gives them the ability to find and reach these locations while avoiding obstacles, and higher-level coordination ensures that self-assembly can progress in spite of physical constraints that exist in the environment. The integration of these techniques provides not only a more powerful control methodology, but also, serves as a step in the maturation of swarm intelligence into a more unified field.

- A procedure was given for automatically specifying behaviors that successfully achieve the self-assembly of a broad range of structures in a non-trivial environment. This environment is three-dimensional, continuous, and imposes constraints on the horizontal and vertical motion of the agents, which are embodied as blocks of different sizes (and thus present a degree of heterogeneity). The target structures consist of hundreds of such blocks, arranged in a variety of ways, and (due to the restriction on vertical movement) often require the assembly and subsequent disassembly of temporary substructures, such as staircases. To my knowledge, this work presents the first successful attempt to achieve the self-assembly of such structures in an environment as complex as the one presented here. While this environment is still greatly simplified, as compared with the real world, the presented research potentially carries practical significance, because it addresses issues (such as the presence of physical constraints and the resulting need for higher-level coordination) that are likely to arise when self-assembly is attempted with robotic devices.

- This dissertation developed a formal model of order computation and order enforcement in the context of self-assembly, and proved important properties, such as correctness and completeness. First, algorithms were designed that compute an order on the self-assembly of a given target structure, along with necessary temporary substructures. Subsequently, a methodology was developed for generating a set of local rules, which enforce this order at runtime. Together, these methods formalize the relationship between structures (situated in a constrained environment) and individual, agent-level behaviors. The latter cause the emergence of coordination that allows the overall system to succeed. To feasibly allow the derivation of this relationship, the theoretical model is environment-independent, and therefore, necessarily makes some assumptions regarding what can be achieved in a given environment. However, these assumptions can be verified experimentally.

- Experiments were conducted to demonstrate that the distributed control methods developed here effectively achieve the self-assembly of a diverse set of structures, given either hand-designed or automatically generated rules. This serves as an important complement to the theoretical results, because it takes into account the full complexity of the environment, where blocks engage in non-linear interactions, and where their collection constitutes a complex, dynamical system with chaotic effects. Apart from this verification, the experiments also yielded insight into the effects of various design choices upon overall performance, showing, for example, the benefit of incorporating stochastic and collective forces into the blocks' motion. In particular, the results show that in designing appropriate methods, one is faced with a series of tradeoffs. During the self-assembly process, a tradeoff arises between the benefit of increasing the availability of blocks in regions where they are needed and the necessity of reducing the interference between these blocks; both factors must be taken into account in order to achieve self-assembly efficiently. Further, runtime efficiency trades off with parsimony, where reduced sets of rules may result in slower performance. Finally, a tradeoff exists between both efficiency and parsimony and the use of automation: presently, hand-generated rule sets still outperform automatically generated rule sets with respect to both criteria.

- The final major contribution of this dissertation is in the parsimonious specification of individual self-assembly behaviors (as sets of control rules) for at least a portion of the assembled structures. This was achieved by, first, enhancing existing stigmergic construction models with integer, rather than qualitative variables, potentially allowing for structures to be specified with a more compact set of rules. While the automated rule generation procedure was initially unable (for the most part) to take advantage of this, further modifications (involving a somewhat more sophisticated method for capturing structural patterns) resulted in significant reductions to the number of generated rules. This result was achieved in spite of the fact that parsimonious rule generation is an inherently difficult problem (as was argued), and that the physical constraints present in the environment necessitate the generation of rules not only for the assembly of structures, but also, for coordination. This result is important from a self-organization point of view, in that it allows some structures to self-assemble via individual behaviors that are relatively simple.

6.3 Future Directions

Given the current state of this research, there are a number of worthwhile directions for further exploration. I will close this dissertation with a brief outline of several possibilities for future work.

Through experiments, it was shown that the developed movement force equations are generic enough to allow the self-assembly of a number of different structures. However, I hypothesize that runtime performance could be substantially improved if these forces were to *adapt* over time, in response to positive feedback (such as the successful deposition of a block into a goal location) as well as negative feedback (such as a collision between two blocks). It should be noted that while the approach of this dissertation adapts agent behaviors for the self-assembly of specific target structures (by hand, or via an automated procedure), the term “adaptation” in the following has a somewhat different meaning, referring to the ability of agents to modify their behavior based on past experience. For example, a simple form of machine learning [76, 108] would involve the modification of numerical coefficients that weigh the various components of the net movement force, in response to such events. However, much greater power may potentially be gained from an approach where the equations governing the computation and combination of these forces (Chapter 3) vary over time in their entirety. In order to achieve a more smooth landscape over which learning can take place, it may be of benefit to replace these equations (which are hard-coded) with neural networks, as has been done in past work [34, 59, 66, 89]. The networks can then be trained, possibly via reinforcement learning [108] (applied specifically towards neural networks in [109]), allowing the agents to adapt their behavior to the given structure (or a set of structures), perhaps over a number of simulations, where the set of learned control mechanisms can be transferred from one simulation to the next.

On these longer time scales, it is of benefit to explore adaptation not only through learning, but via evolution as well. Evolutionary computation has emerged as a powerful problem solving paradigm, where the more “fit” solutions in a population (i.e., in a candidate set) are transformed, via modifications that resemble recombination and mutation, in order to eventually produce better solutions through a “survival of the fittest” process [11, 75]. Various techniques can be used, depending on what is evolved: the optimization of numerical coefficients could be accomplished with a genetic algorithm [75] or evolutionary strategies [98]; the modification of hard-coded force equations may be amenable to genetic programming [11], whereas if movement is instead governed by neural networks (as suggested earlier), a variety of neuroevolution approaches exist as well [59]. However, it is important to keep in mind that most evolutionary algorithms typically require repeatedly evaluating the fitness of each solution in a population (which may contain tens or hundreds of solutions) over a number of generations. If each evaluation involves simulating the self-assembly of a target structure, then the computational cost may become excessive, since a single simulation may require hours of CPU time. It may be worthwhile to consider an evolutionary approach where a member of the population is not a collection of blocks (each having an identical set of movement control methods), but rather, an individual block, whose movement behaviors may be different from those of other blocks. Such an approach is inspired by earlier work in evolving populations of agents [90, 105], and carries the advantage of evaluating *multiple* candidate solutions over a single simulation, although it is expected that appropriately assigning fitness to individual blocks will be a challenging problem. If an effective means of fitness assignment is found, then it would be of interest to investigate the possible costs and benefits of heterogeneity in the control methods (rather than just the physical sizes) of different interacting blocks [68, 89].

While learning and/or evolution appear to have the potential for improving the performance of the system under the current environmental conditions, a question of greater practical interest is whether they would allow the current methodology to adapt to more complex environments, and eventually, to the real world. Given the existence of promising results from experiments in evolving or learning control methods for physical robots [66, 72, 84] (see also [23] for a discussion of the subject), I hypothesize that such adaptive approaches may be quite useful in the extension of present methods towards greater environmental complexity, without a significant sacrifice in the range of structures that can self-assemble, or a tremendous amount of human effort. Initial work in this direction could first take place within the context of simulations that incorporate more realistic models of gravity, noise, friction, etc. Subsequently, this can be followed by explorations of robotic self-assembly in the physical world [14, 80, 81, 120, 121].

Thus far, the focus of the discussion has been primarily on the adaptation of movement control mechanisms, either during or after a simulation. However, it may be of interest to further investigate the use of evolutionary computation in the production of rules, as previously explored in [18, 44, 113]. Existing approaches are limited in that the evolved rules are not able to build structures with full precision. The evolution of precisely-tailored sets of rules (as generated here using more conventional algorithmic techniques) is impeded by the fact that it can be difficult to determine whether some arbitrary, candidate set of rules will always correctly assemble a particular structure. However, evolutionary computation may nonetheless assist in further optimizing the parsimony of rules if it is used in combination with existing rule generation procedures. For example, rather than computing memory conditions via a greedy strategy, as is done in Chapter 5, it may be possible to produce better solutions via evolution.

While further improvements to rule parsimony would be well-justified, another property of the stigmergic model, which has not received attention in this work, is its robustness. An inquiry into the conditions that would allow the system to gracefully recover from a variety of errors (such as rule misapplications) would be worthwhile, particularly if the methodology is to be extended to the real world. In the recent past, approaches have been presented for the *self-repair* of structures that develop holes [7, 60, 102, 104]; however, it is similarly necessary to also be able to remove blocks from locations where they have been mistakenly placed, before they result in the incorrect placement of other blocks (this issue is discussed in [116]). A methodology should be developed to allow a block to detect whether it, or a nearby block, is in an incorrect location, and to modify the sequencing of the overall process accordingly (for example, special modes / process stages may be allotted for error recovery). It it is certainly possible that learning and evolution may once again be of assistance in the endeavor, although their role in this context is presently not obvious. Ultimately, the practical goal is to develop a self-assembly methodology that carries the robustness, adaptability, parallelism and parsimony of natural systems, but that can be effectively tailored to artificial structures that meet human needs.

Appendix A

Proofs of Theorems

To derive a correctness result for the ordering algorithms in Sections 4.1.1 and 4.1.2, it is important to capture the relationship that exists between $S^{(i)}$, as given in Definition 4.2, and the values of the variables S and T in the iteration of the algorithm where C_i is added to O. If C_i is a regular substructure that is removed in some iteration of the algorithm (note that this is generally *not* the i^{th} iteration, since the algorithm operates "in reverse"), then let us denote by $S^s(i)$ and $T^s(i)$ the values of the variables S and T, respectively, at the beginning of that iteration, and by $S^f(i)$ and $T^f(i)$ the values of these variables at the end of the iteration.

Lemma 1 *If C_i is a regular substructure, then $S^{(i-1)} = S^s(i) - C_i \cup T^f(i)$.*

Proof Use induction on i. If $i = 1$, then $S^{(i-1)} = S^{(0)} = \emptyset$ by Definition 4.2. Also, C_1 is removed in the last iteration, where $S^s(1) = C_1$, and there is no temporary substructure preceding C_1, which means that $T^f(1) = \emptyset$. Thus, $S^s(i) - C_i \cup T^f(i) = \emptyset$ as well. If $i = 2$, then $S^{(i-1)} = S^{(1)} = C_1$. Also, $S^s(2) = C_2$, and $T^f(2) = C_1$. Thus, $S^s(i) - C_i \cup T^f(i) = C_1$ as well. By Property 4.2, this covers the base cases. Now, assume that the statement holds for all $i' \leq i$. Supposing, without loss of generality, that C_{i+1} is a regular substructure, let us relate C_{i+1} to the regular substructure removed in the subsequent iteration (i.e., the previous regular substructure in O). By Property 4.2, there are four cases to consider, as described below.

If C_i and C_{i+1} are both regular substructures, then in the iteration when C_i is removed, $S^s(i) = S^s(i+1) - C_{i+1}$. Furthermore, since C_{i+1} is neither a temporary nor a disassembly substructure, the conditions of Steps 4c and 4d of the algorithm did not hold, which implies that $T^s(i) = T^f(i)$. Since C_{i+1} is removed one iteration prior to C_i, $T^f(i+1) = T^s(i) = T^f(i)$. By the induction hypothesis, $S^{(i-1)} = S^s(i) - C_i \cup T^f(i) = S^s(i+1) - C_{i+1} - C_i \cup T^f(i) = S^s(i+1) - C_{i+1} - C_i \cup T^f(i+1)$. Then, $S^{(i)} = S^s(i+1) - C_{i+1} \cup T^f(i+1)$, because $S^{(i)} = S^{(i-1)} \cup C_i$ (Definition 4.2).

Now, suppose that C_{i+1} is a regular substructure, C_i is a temporary substructure, and C_{i-1} is also a regular substructure. Then, $S^s(i-1) = S^s(i+1) - C_{i+1}$. Since C_i is added to O as a temporary substructure when C_{i-1} is removed, $T^s(i-1) - W = C_i$ for some $W \subseteq T^s(i-1) \cap T^f(i-1)$ (Step 4c); on the other hand, since no disassembly substructure is added, $T^f(i-1) - W = \emptyset$. This is only possible if $T^s(i-1) = T^f(i+1) = C_i$ and $T^f(i-1) = \emptyset$. By the induction hypothesis, $S^{(i-2)} = S^s(i+1) - C_{i+1} - C_{i-1} \cup \emptyset$; thus, $S^{(i)} = S^s(i+1) - C_{i+1} \cup C_i$ by definition of $S^{(i)} = S^{(i-2)} \cup C_{i-1} \cup C_i$. But since $C_i = T^f(i+1)$, $S^{(i)} = S^s(i+1) - C_{i+1} \cup T^f(i+1)$.

For the third case, suppose that C_{i+1} is a regular substructure, C_{i-1} is also a regular substructure, and C_i is a disassembly substructure generated for C_{i-1}. Once again, $S^s(i-1) = S^s(i+1) - C_{i+1}$. Furthermore, $T^s(i-1) - W = \emptyset$ and $T^f(i-1) - W = C_i$ for some W; thus, $T^s(i-1) = T^f(i+1) = \emptyset$ and $T^f(i-1) = C_i$. By the induction hypothesis, $S^{(i-2)} = S^s(i+1) - C_{i+1} - C_{i-1} \cup C_i$; thus, $S^{(i)} = S^s(i+1) - C_{i+1}$ by definition of $S^{(i)}$. But since $T^f(i+1) = \emptyset$, $S^{(i)}$ can be expressed as $S^s(i+1) - C_{i+1} \cup T^f(i+1)$.

Finally, there is the case where C_{i+1} and C_{i-2} are regular substructures, C_i is a temporary substructure, and C_{i-1} is a disassembly substructure. Then, $S^s(i-2) = S^s(i+1) - C_{i+1}$. Here, $T^s(i-2) - W = C_i$ and $T^f(i-2) - W = C_{i-1}$; thus, $T^s(i-2) = T^f(i+1) = C_i \cup W$ and $T^f(i-2) = C_{i-1} \cup W$. By the induction hypothesis, $S^{(i-3)} = S^s(i+1) - C_{i+1} - C_{i-2} \cup C_{i-1} \cup W$; thus, $S^{(i)} = S^s(i+1) - C_{i+1} \cup C_i \cup W = S^s(i+1) - C_{i+1} \cup T^f(i+1)$. ■

Theorem 4.1 *Let $O = (C_1, C_2, \ldots, C_n)$ be an order returned by the preliminary or the full ordering algorithm, and define $S^{(i)}$ as in Definition 4.2. Then, $S^{(n)} = S$.*

Proof The result holds trivially for the preliminary ordering algorithm. For the full ordering algorithm, if C_n is a regular substructure, then by Lemma 1, $S^{(n-1)} = S^s(n) - C_n \cup T^f(n)$. Because C_n is removed in the first iteration, $S^s(n) = S$; also, because C_n is the last substructure in O, $T^f(n) = \emptyset$; thus, $S^{(n-1)} = S - C_n$. Since $S^{(n)} = S^{(n-1)} \cup C_n$, $S^{(n)} = S$. If C_n is a disassembly substructure, then by Property 4.2, C_{n-1} must be a regular substructure. Then, $S^{(n-2)} = S^s(n-1) - C_{n-1} \cup T^f(n-1)$. Since $S^s(n-1) = S$ and $T^f(n-1) = C_n$ (the latter holds because this is the first iteration, and $T^s(n-1) = \emptyset$; thus, $T^f(n-1)$ is pushed onto O in Step 4d of Algorithm 4.4), $S^{(n-2)} = S - C_{n-1} \cup C_n$. Then, $S^{(n)} = S^{(n-2)} \cup C_{n-1} - C_n = S$. ■

Theorem 4.2 (Correctness) *Assuming that the ordering algorithm (preliminary or full) does not fail, let $O = (C_1, C_2, \ldots, C_n)$ be the returned order. Then, $\forall i$ $(1 \leq i \leq n)$ $P_A(C_i, S^{(i-1)}) \vee P_D(C_i, S^{(i-1)}) = true$.*

Proof I give a proof for the full ordering algorithm; a similar, albeit simpler proof exists for the simple version, but is omitted for brevity. Consider the iteration of the full algorithm when some arbitrary substructure C_i is added to O. Several cases arise, depending on whether C_i is a temporary (here, there are actually multiple subcases), regular, or disassembly substructure. In showing the correctness of each of these cases, I will make use of some specific predicate (P_A or P_D) conjunct either in the definition of the function f_T (Definition 4.1) or in Step 4b of the full ordering algorithm.

First, consider the special case where C_1 is a temporary substructure. Then, by Property 4.2, C_2 must be a regular substructure removed in the last iteration. By the first predicate conjunct in the definition of f_T, $P_A(C_1, S^s(2) - C_2) = P_A(C_1, \emptyset) = \mathit{true}$. Because $S^{(i-1)} = S^{(0)} = \emptyset$, $P_A(C_i, S^{(i-1)}) = \mathit{true}$.

If C_i is a temporary substructure for $i > 1$, then there are two additional possibilities. If C_{i-1} is a regular substructure, then by Lemma 1, $S^{(i-2)} = S^s(i-1) - C_{i-1} \cup T^f(i-1)$; thus, $S^{(i-1)} = S^s(i-1) \cup T^f(i-1)$. Also, $f_T(C_{i-1}, S^s(i-1) - C_{i-1}, T^s(i-1)) = T^f(i-1)$. By the second predicate term in Step 4b, there exists a subset $W \subseteq T^s(i-1) \cap T^f(i-1)$ such that $P_A(T^s(i-1) - W, S^s(i-1) \cup W) = \mathit{true}$. However, since no disassembly substructure was produced in the given iteration, $T^f(i-1) \subseteq W$; from the definition of W, it then follows that $T^f(i-1) = W$. So, $P_A(T^s(i-1) - W, S^s(i-1) \cup T^f(i-1)) = \mathit{true}$. By substituting C_i for $T^s(i-1) - W$ (Step 4c) and $S^{(i-1)}$ for $S^s(i-1) \cup T^f(i-1)$, we obtain $P_A(C_i, S^{(i-1)}) = \mathit{true}$.

For the second possibility that can arise when C_i is a temporary substructure and $i > 1$, suppose that C_{i-1} is a disassembly substructure for the regular substructure C_{i-2}. Then, $S^{(i-1)} = (S^s(i-2) - C_{i-2} \cup T^f(i-2)) \cup C_{i-2} - C_{i-1}$. Because $C_{i-1} = T^f(i-2) - W$ for some W, $S^{(i-1)}$ can be reexpressed as $S^s(i-2) \cup T^f(i-2) -$

$(T^f(i-2)-W) = S^s(i-2) \cup W$. Also, $f_T(C_{i-2}, S^s(i-2)-C_{i-2}, T^s(i-2)) = T^f(i-2)$. Referring to the second term in Step 4b once again, $P_A(T^s(i-2)-W, S^s(i-2) \cup W) = P_A(C_i, S^s(i-2) \cup W) = P_A(C_i, S^{(i-1)}) = \mathit{true}$.

If C_i is a regular substructure, then $S^{(i-1)} = S^s(i) - C_i \cup T^f(i)$ and $f_T(C_i, S^s(i) - C_i, T^s(i)) = T^f(i)$. By the second conjunct in Definition 4.1, $P_A(C_i, S^s(i) - C_i \cup T^f(i)) = \mathit{true}$; by substitution, $P_A(C_i, S^{(i-1)}) = \mathit{true}$.

Finally, if C_i is a disassembly substructure, then, by Property 4.2, C_{i-1} is a regular substructure. Then, $S^{(i-2)} = S^s(i-1) - C_{i-1} \cup T^f(i-1)$, and $S^{(i-1)} = S^s(i-1) \cup T^f(i-1)$. Also, $f_T(C_{i-1}, S^s(i-1) - C_{i-1}, T^s(i-1)) = T^f(i-1)$; thus, by the first term in Step 4b, for some W, $P_D(T^f(i-1) - W, S^s(i-1) \cup T^f(i-1)) = P_D(C_i, S^s(i-1) \cup T^f(i-1)) = \mathit{true}$. Thus, $P_D(C_i, S^{(i-1)}) = \mathit{true}$ once again. ■

Theorem 4.3 *For any iteration of any run of the full ordering algorithm, the value $W = \emptyset$ will satisfy the terms $P_D(Z - W, S \cup Z)$ and $P_A(T - W, S \cup W)$ in Step 4b.*

Proof If $W = \emptyset$, then the first term simplifies to $P_D(Z, S \cup Z)$. Because $Z = f_T(C, S - C, T) \neq \mathit{FAIL}$ in the given iteration, by the third conjunct of Definition 4.1, $P_D(Z, S \cup Z) = \mathit{true}$. The second term simplifies to $P_A(T, S)$. If this is the first iteration of the algorithm, then $T = \emptyset$, and $P_A(T, S)$ holds trivially, by Condition 4.1. Otherwise, T is the value returned by the call $f_T(C', S' - C')$ in the previous iteration, for some regular substructure C' and structure S'. By the first conjunct of Definition 4.1, $P_A(T, S' - C') = \mathit{true}$. Because C' was removed in the previous iteration, $S = S' - C'$, so $P_A(T, S)$ holds once again. ■

Theorem 4.4 (Determinism) *For any T, any subsets $V' \subseteq V \subseteq S$ and any location $b \in V \cap V'$, assume the following: $[(\exists C \subseteq V)\ b \in C \wedge f_T(C, V - C, T) \neq \mathit{FAIL}] \rightarrow [(\exists C' \subseteq V')\ b \in C' \wedge f_T(C', V' - C', T) \neq \mathit{FAIL}]$. Suppose that some run of the ordering algorithm (preliminary or full) produces an order O for the given input S. Then, there exists no possible run of the algorithm where it will fail on S.*

Proof I give a proof for the full ordering algorithm; the proof for the preliminary version is almost identical. The conclusion of the theorem can be rephrased as follows: for any $k \geq 1$, if an arbitrary location b has been successfully removed (as part of some substructure C) in iteration k of some run, then it can be successfully removed in any other run. Now, use induction on k. If $k = 1$, then $f_T(C, S - C, T) \neq \mathit{FAIL}$, which means that C can be removed via the addition of a temporary substructure (which may be an empty set, if C is reachable directly). At the beginning of any iteration of any run of the algorithm, let V' be the value of S at the beginning of that iteration. Clearly, $V' \subseteq S$, and if $b \in V'$ (i.e., if b has not already been removed), then by the premise of the theorem, $(\exists C' \subseteq V')\ b \in C' \wedge f_T(C', V' - C', T) \neq \mathit{FAIL}$. Thus, b can be removed as part of C'. This proves the base case $k = 1$. Now, assume that the statement holds $\forall k' \leq k$. For a run where C (containing b) is removed in iteration $k+1$, let V be the value of S at the beginning of that iteration. Then, $f_T(C, V - C, T) \neq \mathit{FAIL}$. By the induction hypothesis, all locations in $S - V$ (which have been removed within the first k iterations of this run) can be removed in any other run of the algorithm. Consider any such run, and denote by V' the structure that remains after the locations in

$S - V$ (along, possibly, with others) have been removed. Then, $V' \subseteq V$, and thus, $(\exists C' \subseteq V')\ b \in C' \wedge f_T(C', V' - C', T) \neq \mathit{FAIL}$. Once again, b can be removed. ■

Theorem 4.5 (Completeness) *Assume that if the preliminary ordering algorithm returns an order O on some input S in some run, then it will return some order on S in any run. Further, assume that the function f_T will return a value other than FAIL whenever one exists (i.e., replace "only if" with "if" in Definition 4.1). Suppose that there exists an order $O = (C_1, C_2, \ldots, C_n)$ on the locations of S such that $\forall i\ (1 \leq i \leq n)\ P_A(C_i, S^{(i-1)}) \vee P_D(C_i, S^{(i-1)}) = true$, such that $S^{(n)} = S$, and such that Property 4.1 holds. Then the algorithm will return an order (i.e., will not fail).*

Proof First, let us prove the following statement: for all $k \geq 1$, there exists a run of the preliminary algorithm that will successfully remove the *last* k regular substructures of O, in reverse order, during its first k iterations. This statement is proved by induction on k. The very last regular substructure in O is either C_n or C_{n-1}. In the former case, $P_A(C_n, S^{(n-1)}) = P_A(C_n, S - C_n) = \mathit{true}$, so it is possible to remove C_n from S. In the latter case, by Property 4.1, $C_{n-2} = C_n$, the former being a temporary substructure, and the latter a disassembly substructure. So, $S^{(n-3)} = S - C_{n-1}$. Then, $P_A(C_{n-2}, S^{(n-3)}) = P_A(C_{n-2}, S - C_{n-1}) = \mathit{true}$, $P_A(C_{n-1}, S^{(n-2)}) = P_A(C_{n-1}, S - C_{n-1} \cup C_{n-2}) = \mathit{true}$, and $P_D(C_n, S^{(n-1)}) = P_D(C_{n-2}, S - C_{n-1} \cup C_{n-1} \cup C_{n-2}) = \mathit{true}$. By Definition 4.1, C_{n-2} is a possible return value for $f_T(C_{n-1}, S - C_{n-1})$. Because f_T is assumed to be complete, it will return some value other than *FAIL* on $(C_{n-1}, S - C_{n-1})$, which can be used to successfully remove C_{n-1}. This proves the base case. Now, assume that the statement holds for all $k' \leq k$. By this induction hypothesis, there exists a run of the preliminary algorithm that removes the last k regular substructures of O. Let $C_{n'+1}$ be the regular substructure removed in the k^{th} iteration. Then, by Property 4.1, $C_{n'}$ or $C_{n'-1}$ is the $(n-k)^{\text{th}}$ regular substructure in O. Denote by S' the value of the variable S at the beginning of iteration $k+1$. The proof that $C_{n'}$ or $C_{n'-1}$ can be successfully removed from S' in iteration $k+1$ is analogous to the base case: replace n with n', and S with S'. Now, it has been shown that there exists a run of the preliminary algorithm that will successfully remove all regular substructures of O. By the assumption of deterministic success, the preliminary algorithm will return an order in any run. ■

Theorem 4.6 (Equivalence) *Assume that if the algorithm (preliminary or full) returns an order O on some input S in some run, then it will return some order on S in any run. Further, assume that the function f_T will return a value other than FAIL whenever one exists. Then, given a target structure S, the preliminary ordering algorithm will return an order iff the full ordering algorithm will return an order.*

Proof First, suppose that the preliminary algorithm returns an order O, given a target structure S. Let us prove the following statement inductively: for all $k \geq 1$, there exists a run of the full algorithm that will successfully remove the *last* k regular substructures of O, in reverse order, during its first k iterations. If $k = 1$, then let C be the last regular substructure in O. It is evident from Figure 4.3 that since C is removed from S, $f_T(C, S-C) \neq$ *FAIL*. Then, $f_T(C, S-C, T) \neq$

FAIL holds as well, because the formal definition of f_T (Definition 4.1) does not make use of the argument T, and f_T is assumed to return a value other than *FAIL* whenever one exists; thus, C can be removed by the full ordering algorithm as well. Now, suppose that k regular substructures have been thus removed from S by the preliminary algorithm, and denote by S' the structure that remains; let C' be the regular substructure removed by the preliminary algorithm in iteration $k+1$. By the induction hypothesis, there exists a run of the full algorithm that has S' as the value of the variable S at the beginning of the $(k+1)^{\text{st}}$ iteration. Once again, because f_T will return a value (if one exists) regardless of whether the optional argument T is used, C' can be removed from S' by the full algorithm in that iteration. Since the full algorithm returns O in this particular run, it will return some order in any run, by assumption. The proof "in the other direction" is analogous. ■

To show the validity of Theorem 4.7 in Section 4.3.2, we can begin by showing that the mode of any block will not change to a value greater than i unless all events in E_i (or greater) have been detected.

Lemma 2 *Given R_E, R_C and R_M, the following holds at all $t \geq 0$, for all i:*

$$(\forall i' \geq i)(\exists e \in E_{i'})(\forall b \in B)\ \neg P_E(b,e,t) \rightarrow m^b(t) \leq i$$

Proof Suppose, by way of contradiction, that for some i $(1 \leq i \leq n)$, $(\forall i' \geq i)(\exists e \in E_{i'})(\forall b \in B)\ \neg P_E(b,e,t)$ and for some block b, $m^b(t) = i' > i$. Since each block is in mode 1 initially, some rule $r \in R_M$ must have fired to transition b to mode i'. This indicates that $(\forall j, 1 \leq j \leq |E_{i'-1}|)\ \mu^b_j = i' - 1$. Because $\mu^b_1 = \mu^b_2 = \ldots = \mu^b_l = 0$ initially, a rule from R_E or R_C must have fired to set μ^b_j

to $i' - 1$ for each j. This indicates that for each j, either b or some other block b' has detected event $e_j \in E_{i'-1}$. This violates the assumption that $(\forall i' \geq i)(\exists e \in E_{i'})(\forall b \in B)\ \neg P_E(b, e, t)$; thus, the statement of the lemma must hold. ■

Further, the message and mode values are monotonically increasing with time.

Lemma 3 *Given R_E, R_C and R_M, the following holds at all $t \geq 0$, for all i:*

$$(\forall b \in B)(\forall j)\ \mu_j^b(t) = i \rightarrow (\forall t' \geq t)\ \mu_j^b(t') \geq i$$

$$(\forall b \in B)\ m^b(t) = i \rightarrow (\forall t' \geq t)\ m^b(t') \geq i$$

Proof Suppose, by way of contradiction, that $m^b(t) = i$, and that at some later time $t' \geq t$, $m^b(t') = i' < i$. This means that between t and t', some rule in R_M fired to result in this mode change. This indicates that rules in R_E or R_C fired to set $\mu_1^b, \mu_2^b, \ldots, \mu_{|E_{i'-1}|}^b$ to $i' - 1$. But the antecedent of any such rule states that $m^b(t) = i' - 1$ must hold in order for this to happen, which contradicts the premise that $m^b(t) = i$. Thus, mode values cannot decrease. Similarly, suppose that for some j, $\mu_j^b(t) = i$, and that at $t' \geq t$, $\mu_j^b(t') = i' < i$. At time t, $m^b(t) \geq i$, since b must have been in mode i in order for a rule (in R_E or R_C) to set μ_j^b to i, and mode values never decrease, as just shown. Then, at some time t'', $(t \leq t'' < t')$, some rule fired to set $\mu_j^b = i'$. Thus, $m^b(t'') = i' < i$. This is a contradiction, as mode values cannot decrease, and thus, the statement of the lemma must hold. ■

Next, let us show that given sufficient time, all blocks will eventually transition to the mode i that is appropriate for the current building stage (i.e., for completing C_i and detecting events in E_i). Certainly, there are situations where all events in

E_i are detected without all blocks having switched to mode i; as is evident from Condition 4.4 in the text, the latter is a sufficient requirement for the former, not a necessary one. However, in situations where all blocks must be in mode i in order to complete C_i, the system-wide transition to mode i will eventually take place, as follows.

Lemma 4 *Given R_E, R_C and R_M, the following holds at all $t \geq 0$, for all i:*

$$(\forall i' < i)(\forall e \in E_{i'})(\exists b \in B)\ P_E(b, e, t) \rightarrow (\exists t' \geq t)(\forall b \in B)m^b(t') \geq i$$

Proof Use induction on k to prove an equivalent conclusion: For any k, $(1 \leq k \leq i)$, there exists a time $t' \geq t$ where $(\forall b \in B)m^b(t') \geq k$. The base case $k = 1$ holds trivially, because all blocks begin in mode 1, by Condition 4.3, and mode values never decrease, by Lemma 3. Now, suppose that for all $k' \leq k$, the statement holds. Then, eventually, all blocks will be in mode at least k, and it must be shown that after an additional interval of time, they will all be in mode at least $k + 1$ (where $k + 1 \leq i$). Let b be some block that is not yet in mode $k + 1$; i.e., $m^b = k$. Let μ_j^b be some message variable, for $1 \leq j \leq |E_k|$. If b is the block which detected the corresponding event $e_j \in E_k$ (which already took place), then it will set $\mu_j^b = k$, via a rule in R_E. Otherwise, e_j has been detected by some other block b', so $\mu_j^{b'} \geq k$ (by Lemma 3, message variable values never decrease). As defined in the statement of Condition 4.2, suppose that $H = \{b'\}$ and $G = B - H$, and consider the blocks' communications that take place until $G = \emptyset$ and $H = B$. I inductively show that at any point in time, $(\forall h \in H)\ \mu_j^h \geq k$. This is trivially true when $H = \{b'\}$. Assume that it is true for all blocks currently in H, and suppose that $g \in G$ is some block that communicates with a block $h \in H$. If $m^g > k$, then $\mu_j^g \geq k$ (μ_j^g must have had the value k at some point in order for g to transition to mode $k+1$, and by Lemma 3, it could not have decreased). If $m^g = k$, then it follows form the induction hypothesis that $\mu_j^h \geq k$; thus, a rule in R_C will fire and set $\mu_j^g = k$. Thus, $\mu_j^b = k$ for any b. Since j was chosen arbitrarily, this eventually holds for every message variable necessary to transition b to mode $k+1$ by a rule in R_M. In this manner, all blocks will eventually be in mode at least $k+1$. ■

Theorem 4.7 *Given R_E, R_C and R_M, the following holds:*

$$(\exists t)(\forall i)(\forall e \in E_i)(\exists b \in B)\ P_E(b, e, t)$$

Proof In order to allow an inductive (on i) approach, I will prove a somewhat stronger conclusion, which explicitly incorporates the third conjunct of the hypothesis (i.e., the conjunction to the left of the arrow) of Condition 4.4 as an invariant that holds throughout the assembly process:

$$(\exists t)(\forall i)[(\forall e \in E_i)(\exists b \in B)\ P_E(b, e, t) \wedge (\forall t' \leq t)(\forall b \in B)\ m^b(t') \leq i]$$

Suppose that $i = 1$. At time $t = 0$, the hypothesis of Condition 4.4 holds trivially by Condition 4.3. We must show that it will hold for all $t \geq 0$ until all events in E_1 have been detected (this should happen in a finite amount of time, since $(\forall e \in E_i)(\exists b \in B)\ Pr\{P_E(b, e, t)\} > 0$). The first conjunct of the hypothesis of Condition 4.4 holds trivially. The second conjunct of the hypothesis of Condition 4.4 holds by Lemmas 2 and 3. The third conjunct of the hypothesis of Condition 4.4 follows from the second. It should also be noted that at time t_1, when all events in

E_1 have finally been detected, the third conjunct of the hypothesis of Condition 4.4 will hold, because corresponding mode changes occur no earlier than the next time step (rules in R_E and possibly R_C must fire first). Thus, t_1 satisfies the statement of the theorem for $i = 1$, which proves the base case. Now, assume that the statement of the theorem holds for all $i' \leq i$, and consider the state of the system at the time t_i (where $i+1 \leq n$) when all events in E_i have been detected, and the system is finally ready to assemble (or disassemble) C_{i+1}. The first conjunct of the hypothesis of Condition 4.4 holds by the induction hypothesis; furthermore, by definition of P_E, it will hold for all $t \geq t_i$. The third conjunct also holds by the induction hypothesis. Also, by Lemma 2, $m^b \leq i+1$ until after all events in E_{i+1} are detected at some time t_{i+1}; thus, the third conjunct holds through t_{i+1}. On the other hand, the second conjunct does not hold yet: all blocks are in mode i or less. But by Lemma 4, all blocks will eventually transition to mode $i + 1$. Since the hypothesis of Condition 4.4 can thus be satisfied for any necessary period of time, all events in E_{i+1} will eventually be detected at some time t_{i+1}, and no block will be in mode greater than $i + 1$ until after this time. Thus, the statement of the theorem holds. ■

It was thus shown that the presented scheme for communication and mode changes, as encoded by the rules in R_E, R_C and R_M, will correctly enforce the order O during the self-assembly process. It should be noted here that the proofs never made use of the last conjunct in the conclusion of Condition 4.4, which states that until some block(s) have transitioned to mode $i + 1$, events in $E_{i'}$ for $i' > i$ will not be detected, which guarantees that the blocks will not begin to assemble or disassemble C_{i+1} until after the assembly or disassembly of C_i has been completed. If this conjunct was to be removed, then the given model would suggest that in some cases, the assembly/disassembly of C_i and $C_{i'}$ (for $i \neq i'$) may take place simultaneously, which defeats the purpose of computing an order in the first place, and which is certainly not true in the system presented here, because in mode i, only stigmergic rules for substructures in C_i can be applied. However, from the point of view of the proofs, the conclusion of Theorem 4.7 would still be achievable: even if events in $E_{i'}$ were to appear before all events in E_i (where $i < i'$) have been detected, the condition $m^b = i$ in the rules within R_E and R_C would prevent the blocks from reacting to events in $E_{i'}$ until it is appropriate to do so.

Appendix B

General Rule Definitions

Here, I present formal, syntactic definitions of stigmergic and variable change rules. These definitions are not strictly used by the proofs derived above (in Appendix A), because these proofs do not formally consider stigmergic rules (assuming, rather, that they are correct), and only operate upon the somewhat restricted properties of those variable change rules that are computed by the rule generation procedure. However, the definitions are given here in order to show the full expressive power of the rules. Note that in the following, the rules are not explicitly defined as antecedent-consequent pairs, although it us conceptually useful to view them as such.

Formally, a stigmergic rule r is defined as a tuple $\{p, G, L, M, \tau\}$. Here, p denotes the *purpose* of r, and can take on one of three values, namely: "assembly", "disassembly" or "completion". The overall structure of r is specified by G, which is a set of unit cubes called *sites*, whose union is a (local) hexahedral region of space. Each site has a specific integral position (x, y, z) within G, and is formally a tuple $\{(x, y, z), t, C, B\}$, where t denotes the *type* of the site, and can have as values "empty", "wildcard" or "full" (see Section 3.2.2). If $t =$ "full", then the set of possible memory conditions associated with the site is denoted by C, and each one has the form $a^{b'} < k$, $a^{b'} > k$, $a^{b'} = k$ or $a^{b'} \neq k$, for some integer variable-value pair

$(a^{b'}, k)$. Furthermore, $B \subseteq G$ can associate a full site with other adjacent sites, such that B specifies the presence of a medium or a large block (which is 2 or 4 cubic units, respectively), rather than a section of some block. Similarly, the *goal sites* $L \subseteq G$ in r are those sites that correspond to the goal location, where the block that follows r intends to deposit itself (if $p =$ "assembly"), or where it currently resides (if $p =$ "disassembly" or $p =$ "completion"). The set of *applicable modes* M specifies the mode values under which r may apply. Finally, τ specifies the *substructure type* of r. If $p =$ "assembly", then a block b that follows r will set $\tau^b = \tau$, once it deposits itself; otherwise, τ specifies the value of τ^b that a stationary block b must have if it is to apply r.

Similarly, let us define a variable change rule r as a tuple $\{u, I, E, a^b, k\}$. The *usage* u of r can take on either the value "always", denoting that r can potentially apply at any time step, or "disassembly", indicating that r should only be applied by a stationary block at the time step when it begins to move again (this is useful in detecting the disassembly of a temporary substructure). The *internal conditions* I are a tuple $\{s, \mu, C\}$, which specifies restrictions on the block b that may apply r. Specifically, s denotes the size of b; μ is a boolean variable that specifies whether or not b must be stationary, and C is a set of memory conditions on the variables in b. Notably, any of the internal conditions may be unspecified; for example, if s is not specified, then a block of any size may apply r. Similarly, the *external conditions* $E = \{s, \mu, C, N, l\}$ specify the conditions local to b that must hold if b is to apply r. As with I, the variables s, μ and C in E optionally place restrictions on the size, movement status (i.e., stationary or not) and memory of blocks b' in the vicinity of b. Blocks are considered to be "in the vicinity" if they are within a hexahedral neighborhood, centered around b, whose dimensions are specified by N; this neighborhood may be smaller than the full neighborhood of visibility that is allowed for a block ($15 \times 15 \times 4$ units). The minimum number of blocks b' (within the neighborhood defined by N) that must satisfy S, μ and C in E is given by l, which can also take on the special value "majority", indicating that more than half of the blocks b' within the neighborhood must satisfy the conditions, regardless of their total number. Finally, in what can be viewed as r's antecedent, the variable to be changed is specified by a^b, and its value will be set to k, if r applies. It should be noted that in the automatically generated variable change rules (see Figure 4.6), many features, such as smaller neighborhood dimensions N, are not used.

Bibliography

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, Amorphous Computing, *Communications of the ACM*, 43(5):74-82, 2000.

[2] J. Adam, *Designing Emergence: Automatic Extraction of Stigmergic Algorithms from Lattice Structures*, Ph.D. Dissertation, University of Essex, 2005.

[3] A. Adamatzky, O. Holland and C. Melhuish, Laziness + Sensitivity + Mobility = Structure: Emergence of Patterns in Lattice Swarms, *5th European Conference on Advances in Artificial Life*, 432-441, 1999.

[4] C. Anderson, Self-Organization in Relation to Several Similar Concepts: Are the Boundaries to Self-Organization Indistinct?, *The Biological Bulletin*, 202:247-255, 2002.

[5] C. Anderson, G. Theraulaz and J.-L. Deneubourg, Self-Assemblages in Insect Societies, *Insectes Sociaux*, 49:99-110, 2002.

[6] D. Arbuckle and A. Requicha, Active Self-Assembly, *IEEE International Conference on Robotics and Automation*, 896-901, 2004.

[7] D. Arbuckle and A. Requicha, Shape Restoration by Active Self-Assembly, *International Symposium on Robotics and Automation*, 173-177, 2004.

[8] R. Arkin, T. Balch and E. Nitz, Communication of Behavioral State in Multi-Agent Retrieval Tasks, *IEEE International Conference on Robotics and Automation*, 588-594, 1993.

[9] R. Arkin, *Behavior-Based Robotics*, MIT Press, 1998.

[10] T. Balch and M. Hybinette, Social Potentials for Scalable Multi-Robot Formations, *IEEE International Conference on Robotics and Automation*, 73-80, 2000.

[11] W. Banzhaf, P. Nordin, R. Keller and F. Francone, *Genetic Programming*, Morgan Kaufman, 1998.

[12] R. Beckers, O. Holland and J.-L. Deneubourg, From Local Actions to Global Tasks: Stigmergy and Collective Robotics, *Fourth Workshop on Artificial Life*, 181-189, 1994.

[13] G. Beni and J. Wang, Swarm Intelligence, *7th Annual Meeting of the Robotics Society of Japan*, 425-428, 1989.

[14] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp and T. Nguyen, Self-Organizing Programmable Parts, *IEEE International Conference on Intelligent Robots and Systems*, 2005.

[15] H. Bojinov, A. Casal and T. Hogg, Multiagent Control of Self-Reconfigurable Robots, *Artificial Intelligence*, 142:99-120, 2002.

[16] E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, N. Franks, O. Rafelsberger, J.-L. Joly and S. Blanco, A Model for the Emergence of Pillars, Walls and Royal Chambers in Termite Nests, *Philosophical Transactions of the Royal Society of London, B*, 353:1561-1576, 1998.

[17] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, 1999.

[18] E. Bonabeau, S. Guérin, D. Snyers, P. Kuntz and G. Theraulaz, Three-Dimensional Architectures Grown by Simple 'Stigmergic' Agents, *BioSystems*, 56:13-32, 2000.

[19] N. Bowden, A. Terfort, J. Carbeck and G. Whitesides, Self-Assembly of Mesoscale Objects into Ordered Two-Dimensional Arrays, *Science*, 276:233-235, 1997.

[20] J. Breivik, Self-Organization of Template-Replicating Polymers and the Spontaneous Rise of Genetic Information, *Entropy*, 3:273-279, 2001.

[21] R. Brooks, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, 2(1):14-23, 1986.

[22] R. Brooks, P. Maes, M. Matarić and G. More, Lunar Base Construction Robots, *IEEE International Workshop on Intelligent Robots and Systems*, 389-392, 1990.

[23] R. Brooks, Artificial Life and Real Robots, *First European Conference on Artificial Life*, 3-10, 1992.

[24] S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz and E. Bonabeau, *Self-Organization in Biological Systems*, Princeton University Press, 2001.

[25] S. Chaiken, D. Kleitman, M. Saks and J. Shearer, Covering Regions by Rectangles, *SIAM Journal of Algebraic and Discrete Methods*, 2(4):394-410, 1981.

[26] I. Couzin, J. Krause, N. Franks and S. Levin, Effective Leadership and Decision-Making in Animal Groups on the Move, *Nature*, 433:513-516, 2005.

[27] J. Culberson and R. Reckhow, Covering Polygons is Hard, *Journal of Algorithms*, 17:2-44, 1994.

[28] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain and L. Chrétien, The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots, in J.-A. Meyer and S. Wilson (eds.), *From Animals to Animats*, MIT Press, 356-363, 1991.

[29] G. Di Caro and M. Dorigo, AntNet: Distributed Stigmergetic Control for Communications Networks, *Journal of Artificial Intelligence Research*, 9:317-365, 1998.

[30] M. Dorigo and L. Gambardella, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*, 1(1):53-66, 1997.

[31] K. Drexler, *Nanosystems: Molecular Machinery, Manufacturing, and Computation*, John Wiley and Sons, 2001.

[32] L. Edwards, Y. Peng and J. Reggia, Computational Models for the Formation of Protocell Structures, *Artificial Life*, 4:61-77, 1998.

[33] A. Efros and T. Leung, Texture Synthesis by Non-Parametric Sampling, *IEEE International Conference on Computer Vision*, 1033-1038, 1999.

[34] D. Floreano and F. Mondada, Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot, in D. Cliff, P. Husbands, J.-A. Meyer and S. Wilson (eds.), *From Animals to Animats*, MIT Press, 421-430, 1994.

[35] K. Fujibayashi, S. Murata, K. Sugawara, M. Yamamura, Self-Organizing Formation Algorithm for Active Elements, *21st IEEE Symposium on Reliable Distributed Systems*, 416-421, 2002.

[36] M. Gardner, The Fantastic Combinations of John Conway's New Solitaire Game "Life", *Scientific American*, 223:120-123, 1970.

[37] M. Ghallab, D. Nau and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

[38] A. Globus, D. Bailey, J. Han, R. Jaffe, C. Levit, R. Merkle and D. Srivastava, NASA Applications of Molecular Nanotechnology, *The Journal of the British Interplanetary Society*, 51:145-152, 1998.

[39] S. Glotzer, Some Assembly Required, *Science*, 306:419-420, 2004.

[40] S. Goldstein, J. Campbell and T. Mowry, Programmable Matter, *Computer*, 38(6):99-101, 2005.

[41] S. Goss, S. Aron, J.-L. Deneubourg and J. Pasteels, Self-Organized Shortcuts in the Argentine Ant, *Naturwissenschaften*, 76:579-581, 1989.

[42] Grassé, P.-P., La Reconstruction du Nid et Les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La Théorie de la Stigmergie: Essai d'Interprétation du Comportement des Termites Constructeurs, *Insectes Sociaux*, 6:41-81, 1959.

[43] S. Griffith, D. Goldwater and J. Jacobson, Robotics: Self-Replication from Random Parts, *Nature*, 437:636-636, 2005.

[44] Y. Guo, G. Poulton, P. Valencia and G. James, Designing Self-Assembly for 2-Dimensional Building Blocks, in G. Serugendo, A. Karageorgos, O. Rana and F. Zambonelli (eds.), *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering, LNAI 2977*, Springer, 75-89, 2004.

[45] J. Hartman and J. Wernecke, *The VRML 2.0 Handbook: Building Moving Worlds on the Web*, Silicon Graphics, 1996.

[46] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda and I. Endo, Self-Organizing Collective Robots with Morphogenesis in a Vertical Plane, *IEEE International Conference on Robotics and Automation*, 2858-2863, 1998.

[47] J. Jájá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[48] C. Jones and M. Matarić, From Local to Global Behavior in Intelligent Self-Assembly, *IEEE International Conference on Robotics and Automation*, 721-726, 2003.

[49] C. Jones and M. Matarić, The Use of Internal State in Multi-Robot Coordination, *Hawaii International Conference on Computer Sciences*, 27-32, 2004.

[50] I. Karsai and Z. Pénzes, Comb Building in Social Wasps: Self-Organization and Stigmergic Script, *Journal of Theoretical Biology*, 161:505-525, 1993.

[51] I. Karsai, Decentralized Control of Construction Behavior in Paper Wasps: An Overview of the Stigmergy Approach, *Artificial Life*, 5:117-136, 1999.

[52] J. Kennedy, The Particle Swarm: Social Adaptation of Knowledge, *International Conference on Evolutionary Computation*, 303-308, 1997.

[53] J. Kennedy, R. Eberhart and Y. Shi, *Swarm Intelligence*, Morgan Kaufman, 2001.

[54] D. Kim, Self-Organization for Multi-Agent Groups, *International Journal of Control, Automation, and Systems*, 2(3):333-342, 2004.

[55] E. Klavins, Automatic Synthesis of Controllers for Distributed Assembly and Formation Forming, *IEEE International Conference on Robotics and Automation*, 2002.

[56] E. Klavins, Toward the Control of Self-Assembling Systems, in A. Bicchi, H. Christensen and D. Prattichizzo (eds.), *Control Problems in Robotics*, Springer, 153-168, 2003.

[57] E. Klavins, R. Ghrist and D. Lipsky, Graph Grammars for Self-Assembling Robotic Systems, *IEEE International Conference on Robotics and Automation*, 2004.

[58] E. Klavins, Self-Assembly from the Point of View of Its Pieces, *American Control Conference*, 2006.

[59] J. Kodjabachian and J.-A. Meyer, Evolution and Development of Control Architectures in Animats, *Robotics and Autonomous Systems*, 16:161-182, 1995.

[60] K. Kotay and D. Rus, Generic Distributed Assembly and Repair Algorithms for Self-Reconfiguring Robots, *IEEE International Conference on Intelligent Robots and Systems*, 2004.

[61] V. Kumar and H. Ramesh, Covering Rectilinear Polygons with Axis-Parallel Rectangles, *SIAM Journal on Computing*, 32(6):1509-1541, 2003.

[62] C. Langton, Computation at the Edge of Chaos: Phase Transitions and Emergent Computation, *Physica D*, 42:12-37, 1990.

[63] G. Lapizco-Encinas and J. Reggia, Diagnostic Problem Solving using Swarm Intelligence, *IEEE Swarm Intelligence Symposium*, 365-372, 2005.

[64] J. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[65] T. Lechner, B. Watson, U. Wilensky and M. Felsen, Procedural City Modeling, *1st Midwestern Graphics Conference*, 2003.

[66] M. Lewis, A. Solidum and A. Fagg, Genetic Programming Approach to the Construction of a Neural Network for Control of a Walking Robot, *IEEE International Conference on Robotics and Automation*, 2618-2623, 1992.

[67] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, 1997.

[68] S. Luke and L. Spector, Evolving Teamwork and Coordination with Genetic Programming, *Genetic Programming 1996*, 150-156, 1996.

[69] M. Lüscher, A Portable High-Quality Random Number Generator for Lattice Field Theory Simulations, *Computer Physics Communications*, 79:100-110, 1994.

[70] A. Martinoli, A. Ijspeert and L. Gambardella, A Probabilistic Model for Understanding and Comparing Collective Aggregation Mechanisms, *5th European Conference on Advances in Artificial Life*, 575-584, 1999.

[71] Z. Mason, Programming with Stigmergy: Using Swarms for Construction, *8th International Conference on Artificial Life*, 371-374, 2002.

[72] M. Matarić, Issues and Approaches in the Design of Collective Autonomous Agents, *Robotics and Autonomous Systems*, 16:321-331, 1995.

[73] K. Mehrotra, C. Mohan and S. Ranka, *Elements of Artificial Neural Networks*, MIT Press, 2000.

[74] C. Melhuish, Exploiting Domain Physics: Using Stigmergy to Control Cluster Building with Real Robots, *5th European Conference on Advances in Artificial Life*, 585-595, 1999.

[75] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[76] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[77] S. Mohamed and M. Fahmy, Binary Image Compression using Efficient Partitioning into Rectangular Regions, *IEEE Transactions on Communications*, 43(5):1888-1893, 1995.

[78] F. Mondada, G. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg and M. Dorigo, SWARM-BOT: A Swarm of Autonomous Mobile Robots with Self-Assembling Capabilities, *International Workshop on Self-Organisation and Evolution of Social Behaviour*, 11-22, 2002.

[79] R. Nagpal, Programmable Self-Assembly using Biologically-Inspired Multiagent Control, *1st International Conference on Autonomous Agents and Multiagent Systems*, 418-425, 2002.

[80] N. Napp, S. Burden and E. Klavins, The Statistical Dynamics of Programmed Self-Assembly, *IEEE International Conference on Robotics and Automation*, 1469-1476, 2006.

[81] J. Nembrini, N. Reeves, E. Poncet, A. Martinoli and A. Winfield, Mascarillons: Flying Swarm Intelligence for Architectural Research, *IEEE Swarm Intelligence Symposium*, 225-232, 2005.

[82] M. Newman, A.-L. Barabási and D. Watts (eds.), *The Structure and Dynamics of Networks*, Princeton University Press, 2006.

[83] N. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

[84] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press, 2000.

[85] T. Ohtsuki, Minimum Dissection of Rectilinear Regions, *IEEE International Symposium on Circuits and Systems*, 1210-1213, 1982.

[86] Z. Pan and J. Reggia, Artificial Evolution of Arbitrary Self-Replicating Structures, *International Conference on Computational Science*, 404-411, 2005.

[87] S. Park, J.-H. Lim, S.-W. Chung and C. Mirkin, Self-Assembly of Mesoscopic Metal-Polymer Amphiphiles, *Science*, 303:348-351, 2004.

[88] L. Parker, Designing Control Laws for Cooperative Agent Teams, *IEEE International Conference on Robotics and Automation*, 582-587, 1993.

[89] M. Quinn, A Comparison of Approaches to the Evolution of Homogeneous Multi-Robot Teams, *IEEE Congress on Evolutionary Computation*, 128-135, 2001.

[90] J. Reggia, R. Schulz, G. Wilkinson and J. Uriagereka, Conditions Enabling the Evolution of Inter-Agent Signaling in an Artificial World, *Artificial Life*, 7:3-32, 2001.

[91] C. Reynolds, Flocks, Herds and Schools: A Distributed Behavioral Model, *Computer Graphics*, 21(4):25-34, 1987.

[92] C. Reynolds, Steering Behaviors for Autonomous Characters, *Game Developers Conference*, 763-782, 1999.

[93] C. Reynolds, Interactions with Groups of Autonomous Characters, *Game Developers Conference*, 449-466, 2000.

[94] A. Rodríguez and J. Reggia, Extending Self-Organizing Particle Systems to Problem Solving, *Artificial Life*, 10:379-395, 2004.

[95] A. Rodríguez and J. Reggia, Collective-Movement Teams for Cooperative Problem Solving, *Integrated Computer-Aided Engineering*, 12:217-235, 2005.

[96] P. Rothemund and E. Winfree, The Program-Size Complexity of Self-Assembled Squares, *Thirty-Second Annual ACM Symposium on Theory of Computing*, 459-468, 2000.

[97] P. Rothemund, N. Papadakis and E. Winfree, Algorithmic Self-Assembly of DNA Sierpinski Triangles, *PLoS Biology*, 2(12):2041-2053, 2004.

[98] G. Rudolph, Evolution Strategies, in T. Bäck, D. Fogel and Z. Michalewicz (eds.), *Evolutionary Computation 1: Basic Algorithms and Operators*, Institute of Physics, 81-88, 2000.

[99] E. Şahin, T. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi and M. Dorigo, SWARM-BOT: Pattern Formation in a Swarm of Self-Assembling Mobile Robots, *IEEE International Conference on Systems, Man, and Cybernetics*, 2002.

[100] P. Sarkar, A Brief History of Cellular Automata, *ACM Computing Surveys* 32(1):80-107, 2000.

[101] W.-M. Shen, P. Will and B. Khoshnevis, Self-Assembly in Space via Self-Reconfigurable Robots, *IEEE International Conference on Robotics and Automation*, 2516-2521, 2003.

[102] W.-M. Shen, P. Will, A. Galstyan and C.-M. Chuong, Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms, *Autonomous Robots*, 17:93-105, 2004.

[103] A. Smith, P. Turney and R. Ewaschuk, Self-Replicating Machines in Continuous Space with Virtual Physics, *Artificial Life*, 9:21-40, 2003.

[104] W. Spears and D. Gordon, Using Artificial Physics to Control Agents, *IEEE International Conference on Information, Intelligence, and Systems*, 281-288, 1999.

[105] L. Spector, J. Klein, C. Perry and M. Feinstein, Emergence of Collective Behavior in Evolving Populations of Flying Agents, *Genetic and Evolutionary Computation Conference*, 61-73, 2003.

[106] K. Støy, Controlling Self-Reconfiguration using Cellular Automata and Gradients, *8th International Conference on Intelligent Autonomous Systems*, 693-702, 2004.

[107] K. Sugihara and I. Suzuki, Distributed Motion Coordination of Multiple Mobile Robots, *IEEE International Symposium on Intelligent Control*, 138-143, 1990.

[108] R. Sutton and A. Barto, *Reinforcement Learning*, MIT Press, 1998.

[109] G. Tesauro, TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play, *Neural Computation*, 6(2):215-219, 1994.

[110] G. Theraulaz and E. Bonabeau, Coordination in Distributed Building, *Science*, 269:686-688, 1995.

[111] G. Theraulaz and E. Bonabeau, A Brief History of Stigmergy, *Artificial Life*, 5:97-116, 1999.

[112] R. Thompson and N. Goel, Movable Finite Automata (MFA) Models for Biological Systems I: Bacteriophage Assembly and Operation, *Journal of Theoretical Biology*, 131:351-385, 1988.

[113] S. von Mammen, C. Jacob and G. Kokai, Evolving Swarms that Build 3D Structures, *IEEE Congress on Evolutionary Computation*, 1434-1441, 2005.

[114] J. Wawerla, G. Sukhatme and M. Matarić, Collective Construction with Multiple Robots, *IEEE International Conference on Intelligent Robots and Systems*, 2696-2701, 2002.

[115] J. Werfel, Building Blocks for Multi-Agent Construction, *Distributed Autonomous Robotic Systems*, 2004.

[116] J. Werfel, Y. Bar-Yam and R. Nagpal, Building Patterned Structures with Robot Swarms, *Nineteenth International Joint Conference on Artificial Intelligence*, 1495-1502, 2005.

[117] J. Werfel, Y. Bar-Yam, D. Rus and R. Nagpal, Distributed Construction by Mobile Robots with Enhanced Building Blocks, *IEEE International Conference on Robotics and Automation*, 2006.

[118] J. Werfel and R. Nagpal, Extended Stigmergy in Collective Construction, *IEEE Intelligent Systems*, 21(2):20-28, 2006.

[119] L. Whitcomb, D. Koditschek and J. Cabrera, Towards the Automatic Control of Robot Assembly Tasks via Potential Functions: The Case of 2-D Sphere Assemblies, *IEEE International Conference on Robotics and Automation*, 2186-2191, 1992.

[120] P. White, K. Kopanski and H. Lipson, Stochastic Self-Reconfigurable Cellular Robotics, *IEEE International Conference on Robotics and Automation*, 2888-2893, 2004.

[121] P. White, V. Zykov, J. Bongard and H. Lipson, Three Dimensional Stochastic Reconfiguration of Modular Robots, *Robotics: Science and Systems*, 161-168, 2005.

[122] G. Whitesides and B. Grzybowski, Self-Assembly at All Scales, *Science*, 295:2418-2421, 2002.

[123] R. Winder and J. Reggia, Using Distributed Partial Memories to Improve Self-Organizing Collective Movements, *IEEE Transactions on Systems, Man, and Cybernetics - B*, 34(4):1697-1707, 2004.

[124] M. Wooldridge, Intelligent Agents, in G. Weiss (ed.), *Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence*, MIT Press, 27-77, 1999.

[125] V. Zykov, E. Mytilinaios, B. Adams and H. Lipson, Self-Reproducing Machines, *Nature*, 435:163-164, 2005.