Anthills Built to Order: Automating Construction with Artificial Swarms

by

Justin Werfel

S.M. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2001

A.B. Physics
Princeton University, 1999

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author . . . .
Department of Electrical Engineering and Computer Science
May 12, 2006

Certified by . . . .
Radhika Nagpal
Assistant Professor of Computer Science, Harvard University
Thesis Supervisor

Certified by . . . .
H. Sebastian Seung
Professor of Computational Neuroscience
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Anthills Built to Order: Automating Construction with Artificial Swarms

by

Justin Werfel

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

Social insects build large, complex structures, which emerge through the collective actions of many simple agents acting with no centralized control or preplanning. These natural systems motivate investigating the use of artificial swarms to automate construction or fabrication. The goal is to be able to take an unspecified number of simple robots and a supply of building material, give the system a high-level specification for any arbitrary structure desired, and have a guarantee that it will produce that structure without further intervention.

In this thesis I describe such a distributed system for automating construction, in which autonomous mobile robots collectively build user-specified structures from square building blocks. The approach preserves many desirable features of the natural systems, such as considerable parallelism and robustness to factors like robot loss and variable order or timing of actions. Further, unlike insect colonies, it can build particular desired structures according to a high-level design provided by the user.

Robots in this system act without explicit communication or cooperation, instead using the partially completed structure to coordinate their actions. This mechanism is analogous to that of stigmergy used by social insects, in which insects take actions that affect the environment, and the environmental state influences further actions. I introduce a framework of *extended stigmergy* in which building blocks are allowed to store, process or communicate information. Increasing the capabilities of the building material (rather than of the robots) in this way increases the availability of nonlocal structure information. Benefits include significant improvements in construction speed and in ability to take advantage of the parallelism of the swarm.

This dissertation describes system design and control rules for decentralized teams of robots that provably build arbitrary solid structures in two dimensions. I present a hardware prototype, and discuss extensions to more general structures, including those built with multiple block types and in three dimensions.

Thesis Supervisor: Radhika Nagpal
Title: Assistant Professor of Computer Science, Harvard University

Thesis Supervisor: H. Sebastian Seung
Title: Professor of Computational Neuroscience

PHD ZEN
If you want to write a thesis...
...start with the acknowledgments.

Acknowledgments

Mom and Dad come first.

I owe the next debt to the members of my committee: Yaneer Bar-Yam, Radhika Nagpal, Daniela Rus, H. Sebastian Seung, and Gerald Jay Sussman.

Yaneer encouraged me, at a time when I was ready to write up other work for the sake of finishing with graduate school quickly, to switch tracks to go ahead and explore this problem I'd long been most interested in. Without his advice, my research path would have been very different. He's consistently shown interest in my progress and welfare beyond the call of duty.

Radhika took me on as her first graduate student when I was transitioning to this new topic, and has been most centrally involved in directing the course of the research described herein. Her comments were also particularly invaluable in the writing of this thesis.

Daniela has kept me grounded in the robotics side of things—not a bad thing for a thesis purporting to be about robots—and extended numerous considerations despite my not being an official member of her lab.

Sebastian was a significant influence on my decision to come to MIT in the first place; and he's provided my home here, even after my research direction started diverging from his. His flexible advising style made it possible for me to explore a variety of very different topics during my graduate career.

Gerry gave me a place to work in the AI Lab (later CSAIL), and regularly challenged the basis for my thinking by questioning my underlying approaches.

I would also like to thank a number of people who contributed to the work described herein, in a variety of ways. Jason Redi contributed to early discussions about the problem and approach. Jake Beal pointed out the opportunity for specifying functional constraints nested at multiple levels. Quinton Zondervan suggested the use of RFID tags to implement inexpensive labeled blocks. Carrick Detweiler and Iuliu Vasilescu provided much-needed help with simple electronics; Iuliu also provided invaluable help with the power analysis. Keith Kotay offered useful suggestions and discussions. Marty Vona suggested a simplification to the fabrication of the physical blocks.

Thanks go to both the Ericas, who made the journey more pleasant.

I am grateful to webcomics artists Pete Abrams, Rob Balder, Darren Bleuel, Jorge Cham, Shaenon Garrity, Jimmy Maidens, Stephen Notley, Mark Stanley, and Richard

Stevens for allowing me to reproduce their work as chapter headings.

There are countless friends and colleagues who have impacted my life during my time in graduate school. I regret that constraints of time and style prevent me from describing in full the ways you've enriched my life. Many thanks to all of you.

During the period I worked on the project I describe in this thesis, I was supported in part by funds from NSF training grant GM07484, NSF grant EIA-0130391, and a Packard Foundation Fellowship (to H.S. Seung). The ER1 was purchased with funds from NSF grants IIS-0426838, IIS-0513755, and/or ARO award W911NF-0510219.

Contents

List of Figures

List of Tables

List of Algorithms

NOW, LET'S DO INTRODUCTIONS. WE'LL START WITH YOU?
UM, I'M...A NAMELESS DRONE?
SAME HERE.
OK, WELL, TITLES ARE CHEAP. YOU'RE "ED" AND HE'S "MARK." YOU'RE BOTH VICE PRESIDENTS.
ED, HOW'D YOU COME TO WORK HERE?

From *PartiallyClips*
©2005 Rob Balder
www.pclips.com
Used with permission.

Chapter 1

Introduction

Social insects, such as ants and termites, are notable perhaps above all else for their building activities. Their structures can be huge (on the scale of several meters) and functionally complex, with features such as gardens, nurseries, royal chambers, and extensive systems for ventilation and for temperature and atmospheric regulation (Figure 1-1) [10, 13, 29, 61, 62]. All this is accomplished by creatures on the scale of millimeters, with 10^4–10^5 neurons total to handle everything they need to do; they possess only local perception, and have no centralized control to coordinate their actions. From their collective behavior emerge these remarkable structures.

One important question is the scientific one of how these insects collectively build the structures they do. My focus is on the complementary engineering question, of how one could design and program a system in which an artificial swarm collectively builds particular desired structures.

The goal of automating collective construction, as I define it, is to be able to take an unspecified number of robots and a supply of building material, give the system a blueprint or other set of specifications for any arbitrary structure desired, and have a guarantee that the system will produce that structure without further intervention

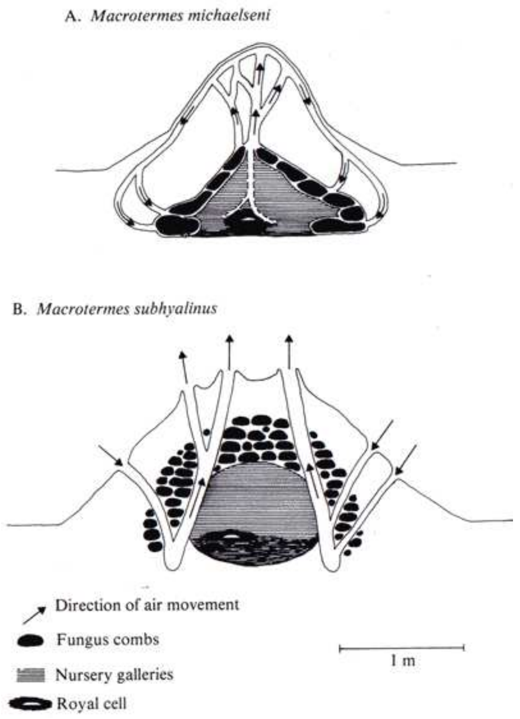

A. Macrotermes michaelseni
B. Macrotermes subhyalinus
Direction of air movement
Fungus combs
Nursery galleries
Royal cell
1 m

Figure 1-1: Left: Cathedral termite mound in Australia's Litchfield National Park. (Photo courtesy Ralf Müller.)
Right: Sketch of internal mound structure for two termite species, indicating air circulation and functional organization of architecture. (Image courtesy Dr. Jo Darlington, from [13].)

(Figure 1-2).

Such systems would have applications to construction projects, as well as to smaller-scale production:

- Construction is the one major human production activity from which automation is still largely absent. Automation could bring significant benefits to traditional construction, for which reported problems include low efficiency, high accident rates, low quality, and skills shortages [52, 66]. It could further be of great use in applications such as facilitating the production of low-cost housing.

- Many reasons automation has not been more widely adopted in construction have to do with adapting robotic elements to traditional sites and techniques [66]. Thus there would particularly be a call for automation in settings difficult or dangerous for humans to work in.

  Such settings include, for instance, other planets. To realize the goal of establishing a human presence on the moon or Mars, it could be enormously helpful to be able to send robots on ahead first, to build a habitat to await the astronauts' later arrival. It might be very useful if space stations, similarly, could be built in an automated way—and outer space would be an especially ap-

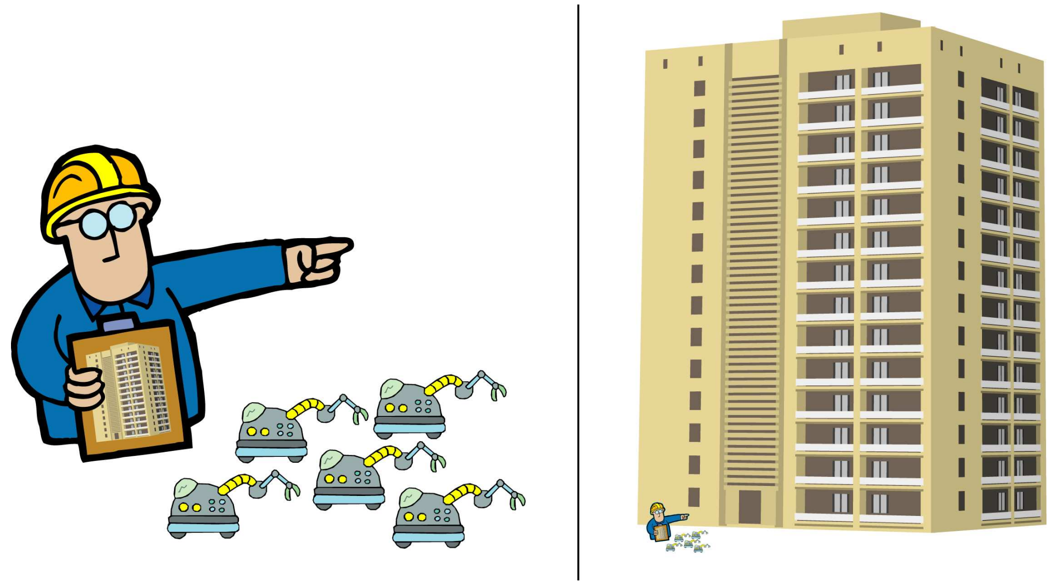


Figure 1-2: The ultimate goal for automated collective construction is to be able to give a set of robots a description of a desired structure (left), and know that it will reliably build that structure without further intervention (right).

propriate environment, due to the opportunity for three-dimensional weightless movement.

Underwater is another setting ripe for automated construction for the same reasons: it's a difficult environment for people to work in, and one which allows weightless movement in three dimensions. Moreover, it's far more accessible than extraterrestrial environments. Potential applications range from oil drilling stations to temporary research bases [1,32,33]; underwater, these would be closer to sites of interest, and insulated from storms and other such disturbances. Conceivably a way to easily build underwater habitats could help open up the oceans for colonization.

- In addition to building structures, mobile robots could act to maintain and repair them during their lifetimes. Further, robots could modify them according to changing circumstances, as termites modify their mounds to maintain internal homeostasis in a variable environment [53, 61, 62].
- On smaller scales, automated construction of desired structures could be applied to everyday artifacts, as for a different approach to rapid prototyping systems.

In this thesis, I show how automating collective construction can be accomplished for solid two-dimensional structures; I demonstrate a hardware prototype, discuss extensions to structures with holes and three-dimensional structures, and introduce a more general principle ("extended stigmergy") for coordinating swarm activities that can confer demonstrated advantages for construction and will likely be useful in other domains.

In the remainder of this chapter, section 1.1 discusses considerations making this a difficult problem, including difficulties associated with swarm systems and mobile robots, and the problem of finding low-level robot actions that will achieve a desired global result. Section 1.2 explains the principle of stigmergy used by social insects, and how it can be extended in artificial systems. Section 1.3 summarizes the approach I take to automating collective construction. Section 1.4 outlines the contributions of the thesis. Section 1.5 lays out the rest of the document.

1.1 Challenges

Automating construction with swarms of mobile robots presents challenges of algorithm design as well as of implementation issues.

1.1.1 Swarms

The philosophy of swarm systems is different from that where one or a few more complicated robots are used. Rather than having a centralized planner that assigns particular robots to perform particular tasks, the idea can be for robots to all take potentially independent actions that end up being consistent with the overall goal. Swarm robots should be simple, interchangeable, and expendable.

These factors have implications for algorithms. Approaches have to be robust to unspecified and potentially variable numbers of robots, which may act with no particular detailed timing or specified order. In a system with large numbers of robots, it is likely that some will break; others may be added to a system mid-task. No central coordination is available to monitor or direct the progress of a global task, or ensure that it progresses in a particular way. Individual robots may not have capabilities available to more complicated, non-swarm robots.

Together with these considerations, swarm systems can have a number of advantages over centralized or few-robot ones.

- Robustness: because no robot is assigned particular tasks, the system is robust to robot loss, typically without a significant effect on task completion.

- Decentralization: no coordination is necessary (or likely possible) between a central planning authority and all the agents, which would require high reliability of (possibly global) communication and/or long delays as information is physically carried over distances.

- Parallelism: if the overall task can be broken up into many independent subtasks, then members of the swarm can work on separate subtasks simultaneously and thus complete the overall task faster than a system forced to work more sequentially.

- Simplicity: simpler robots have fewer components or capabilities subject to potential failure, so any individual robot may be less likely to malfunction than a more complex one. Also, simpler robots are easier to envision scaling down, toward the goal (common among futurists and works of science fiction) of micro- or nanoscale robots for assembly and disassembly.

1.1.2 The global-to-local problem

The forward (local-to-global) problem for swarm construction is this: Given a set of rules for a construction team to follow, what will it end up producing? It may not be possible to predict what the team will generate by looking only at the ruleset itself. (This is the issue of "emergence" in complex systems.) However, it is straightforward to find out what the ruleset will do by having the team go ahead and execute it (potentially in simulation).

Conversely, the inverse (global-to-local) problem is this: Given a desired artifact, what ruleset will end up producing it? This problem, which has not previously been addressed in the construction domain, is the primary focus of this thesis.

1.1.3 Mobile robots

Mobile robots present a number of practical challenges. One of these is the problem of localization. Cues such as odometry and the use of landmarks can help, but establishing pose (position and orientation) with accuracy or certainty is an active research area. For a swarm, that difficulty is compounded: it may be all the more difficult for many robots to establish all their poses, particularly if simplicity considerations limit their capabilities. Thus while it might be useful if all members of the swarm could share a common coordinate system, achieving that goal reliably may be problematic depending on the accuracy desired.

Another problem groups of mobile robots can encounter is with establishing and maintaining ad-hoc communication networks. Passing a message between specific individuals in a network with dynamic topology is nontrivial at best. Further, an approach relying on communication may not scale up well with network size, depending on the details.

Manipulation of the physical world can also be a difficult task for mobile robots (compared, e.g., to assembly-line robots, which are fixed in place and which have items they are expected to manipulate brought to them in predictable locations). When the objects to be manipulated require precise positioning, as is the case in manufacturing or construction, this issue is especially pronounced.

1.2 Tools

One way social insects coordinate their actions is through modification of the shared environment. I use this approach in the algorithms described herein. Moreover, I extend it by increasing the capabilities of environmental elements, and show that so doing can improve the system performance in a variety of ways.

1.2.1 Stigmergy

The term "stigmergy" was coined by French biologist Grassé in a 1959 paper [23]; he intended it to mean "the evocation of work by work"[1]. The idea is that agents (termites, in his case) modify the environment, for instance by depositing pheromones or building material; other agents encountering those environmental modifications take other actions in response. Thus stigmergy describes a way of storing information in the environment, and using it as a means of indirect communication.

A way for agents to use such a tool in building is to take the approach of repeating the following steps:

1. Look at the state of the locally perceivable environment.

2. Based on that state and on the agent's internal state, take an environment-modification action (e.g., deposit or pick up building material), an internal-state-modification action, and a movement action (e.g., take a step in a given direction).

That description is deliberately much like that for a Turing machine. The most significant differences are that there may be restrictions on environmental modification (e.g., conservation of building material); the environment may be two- or three-dimensional rather than a one-dimensional tape[2]; and there are likely to be many agents acting on the environment simultaneously. The actions agents take based on external and internal state may be deterministic or probabilistic.

Limitations

From an engineering point of view, the use of stigmergy as described above has at least two major drawbacks.

One is the matter of specificity of design. Typically a given ruleset, which maps local configurations of building material onto rules for depositing or withholding additional material (or, more generally, onto probabilities of depositing), can give rise to a set of qualitatively similar structures if applied repeatedly to different building sites. However, the structures produced will generally not be identical, as a result of stochastic differences in chosen agent actions, in noisy actions executed by real-world agents, or in initial conditions. In the same way, two termite mounds built by the same species will look qualitatively similar without being identical. In most construction scenarios, by contrast, the goal is to produce a particular desired structure, reliably, independent of initial conditions or randomness.

Rulesets that invariably produce the same structure each time they are applied can be found. However, as noted above, the problem of finding one that produces the structure one wants is unsolved. There has been no general principle described for taking a particular desired structure and coming up with a set of low-level behaviors that building agents can follow to produce it.

[1]See reference [51] for a bibliography of 150 papers that have since used the word, most of them in slightly different ways, with annotations on how many of them have used it.

[2]A term coined by Rudy Rucker for two-dimensional Turing machines is "turmites".

1.2.2 Extended stigmergy

I introduce the term "extended stigmergy" to describe augmenting the basic notion of stigmergy by increasing the capabilities of environmental elements. In this construction setting, those elements are the building blocks; the basic information they carry is the simple fact of their presence at a location [59, 60], and extensions include cases where they can store additional information, perform computations, and/or communicate with physically attached neighbors. Benefits can include increased robustness and faster completion of a desired structure, as will be demonstrated.

This approach to improving the performance of the system, by improving the capabilities of the building material, can be easier and more effective than by improving the capabilities of the robots.

1.3 Scope and overall strategy

The central goal of this thesis is to find solutions to the inverse problem of construction: to start with a desired structure, and come up with a set of low-level agent behaviors that will reliably produce that structure.

That description, and the analogy to insects, might suggest developing a compiler which takes as input a desired high-level structure and generates as output a minimal set of (incomprehensible) low-level rules for agents to follow to produce it. That is not the approach of this work. Instead, the approach is to design fixed sets of rules for agents to follow such that, given an explicit representation of the desired high-level structure, they will produce it by following those rules in conjunction with that explicit representation. The resulting rulesets may not be minimal. However, they have the advantage of being understandable.

The overall strategy is for robots to repeat the following procedure:

1. Bring building material to the structure.
2. Figure out their location in a common structure-based coordinate system.
3. Find an acceptable place to attach material, based on the desired final shape and on where material has already been attached. "Acceptable" involves not only the requirement that the structure ends up having the desired shape, but also that intermediate configurations of material not end up blocking robots from reaching places they need to go to complete the structure.

1.4 Contributions

This thesis makes the following contributions:

- It introduces schemes by which arbitrary structures can be automatically constructed based on a high-level user-specified design. Algorithms presented here apply to solid two-dimensional structures with arbitrary perimeter, to a limited

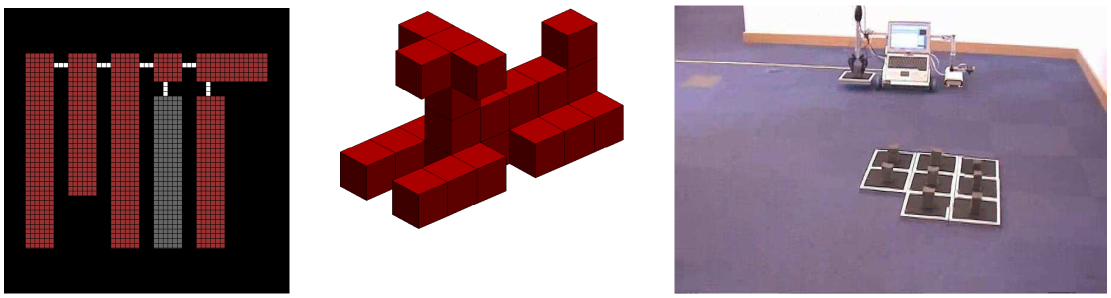


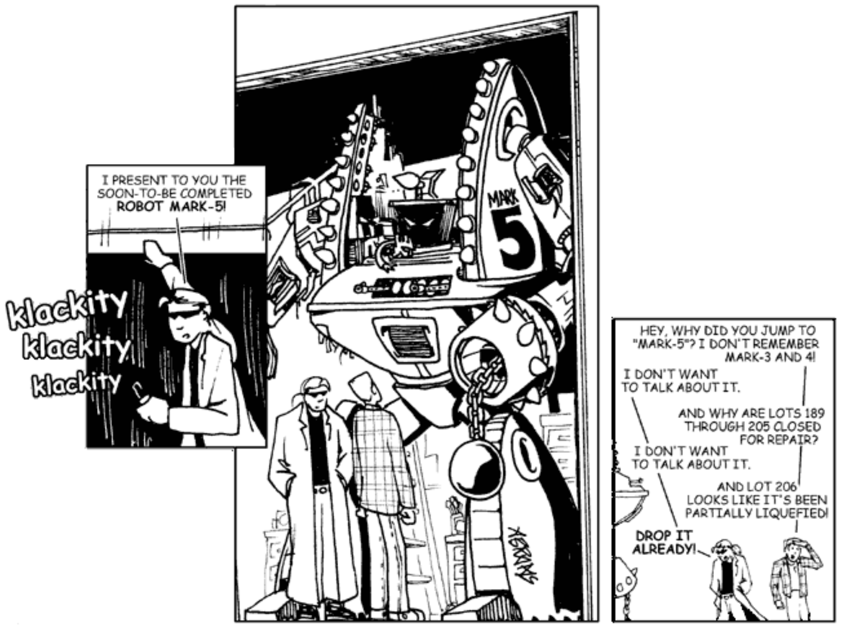
Figure 1-3: Previews of topics to be discussed.
Left: A solid 2-D structure with complex perimeter, built from three types of blocks.
Center: A solid 3-D structure (after [65]).
Right: A physical robot in the process of building a 3×3 square structure.

class of two-dimensional structures with deliberate holes, to structures built from functionally heterogenous blocks, to solid three-dimensional structures, and to structures built from struts assembled into an open latticework.

- It introduces the more general principle of extended stigmergy, and demonstrates the extent of its benefits (and costs) in this construction framework.
- It presents a hardware prototype which can assemble arbitrary solid two-dimensional structures.

1.5 Organization of the rest of this dissertation

Chapter 2 reviews related work. Chapter 3 explains the approach, demonstrates the system for 2-D solid structures, and introduces the use of extended stigmergy. Chapter 4 explores the extent to which that extended stigmergy is useful vs. costly here. Chapter 5 covers the hardware prototype and its performance. Chapter 6 considers functionally heterogeneous building blocks. Chapter 7 describes the extension to three dimensions. Chapter 8 discusses other extensions, including issues such as disassembly and recovery from failures. Chapter 9 concludes and suggests future directions.


I PRESENT TO YOU THE SOON-TO-BE COMPLETED ROBOT MARK-5!
klackity klackity klackity
MARK 5
HEY, WHY DID YOU JUMP TO "MARK-5"? I DON'T REMEMBER MARK-3 AND 4!
I DON'T WANT TO TALK ABOUT IT.
AND WHY ARE LOTS 189 THROUGH 205 CLOSED FOR REPAIR?
I DON'T WANT TO TALK ABOUT IT.
AND LOT 206 LOOKS LIKE IT'S BEEN PARTIALLY LIQUEFIED!
DROP IT ALREADY!

From *Sluggy Freelance*
©2001 Pete Abrams
www.sluggy.com
Used with permission.

Chapter 2

Related Work

Most work in artificial swarm frameworks focuses on tasks like exploration, foraging, and object transportation—all activities that social insects perform, in well-studied ways that often inspire the artificial approaches. However, the relative dearth of work on building is striking, considering that one of the most notable activities of these insects is their construction of extensive and complicated structures.

Section 2.1 reviews work which does explicitly bear on automating construction. The bulk of such work addresses problems other than the one I focus on in this dissertation, that of automatically generating a specific structure starting from a high-level representation. Some studies focus on more purely scientific questions, modeling insect construction behavior or studying how features of low-level behaviors give rise to qualitative high-level results. Most work on artificial swarm construction either does not seek to produce particular structures, or assumes that an appropriate sequence of attachment steps is supplied as an input to the system. Also worth noting is work on automating construction that takes approaches not involving swarms of building agents.

The global-to-local shape-formation problem considered here is also related to analogous problems in a number of other domains, including programmed self-assembly, self-reconfigurable modular robots, and formation control, as described in section 2.2.

Section 2.3 briefly reviews practical issues for localization systems and underwater habitats, both of which bear on realization of construction systems like I describe.

2.1 Construction

Classic studies of insect-inspired construction are primarily concerned with local-to-global problems, starting with low-level rulesets and looking at characteristics of the resulting structures (§2.1.1). Most work on construction with artificial swarms has focused on different aspects of the problem, including communication, debris cleanup, minimalism, or hardware design (§2.1.2). Non-swarm proposals for automating construction have focused on centralized approaches, using one or a small number of active elements (§2.1.3).

2.1.1 Local-to-global models

A number of studies, primarily concerned with scientific questions, take the approach of defining a set of behaviors, simulating their use, and examining the structures that result. The study of this forward (local-to-global) problem has not yet led to solutions for the inverse (global-to-local) problem. Often physical constraints on agent movement are not taken into account in these studies.

Karsai and Pénzes [34] model the nest-building activity of social wasps, using agents with hand-designed behaviors, building combs whose cells are hexagonal cylinders. They show that a multiagent approach based on stigmergy alone, in which agent actions are based on locally perceived features of the structure in progress, can give rise to structures like those produced by real wasps.

Théraulaz and Bonabeau [59,60] consider a three-dimensional lattice-based model, likewise inspired by social wasps, in which stateless agents move randomly on a square grid and have a ruleset inducing them to deposit building material based on locally perceived configurations of existing material. Their main result is the experimental observation that, if a "coherent" structure is to emerge from the agents' collective actions, construction must progress in a sequence of disjoint stages. In any such stage, all local configurations that trigger material deposit must be unique to that stage. Physical movement constraints are not considered; agents can move through one another and through the structure. Like that of Karsai and Pénzes, this approach uses the most basic sense of stigmergy for construction, in which the only information available to agents about the progress of construction is the local presence or absence of material.

They and others [10] have also studied more elaborate models incorporating pheromones, with more complicated features such as air flow and anisotropic agent movement. These mathematical models are deterministic except for stochasticity in initial conditions, and also do not take into account physical constraints such as solid walls.

2.1.2 Collective construction

A number of researchers have studied construction-related issues involving the use of mobile robots and moveable building material. Most have not addressed the issue of automatically generating a particular desired structure, focusing instead on other aspects of the problem. In most cases, either the goal is not to form a particular structure, or the system is given a prespecified sequence of building steps to follow.

Matarić's group at USC focuses on the use of communication in collective tasks involving manipulation of the physical world. These include construction of linear walls of Velcro blocks of alternating polarity [67] or two-dimensional structures of different-colored blocks [31], as well as manipulation tasks requiring the explicit co-operation of multiple robots [19]. In the construction tasks, the requisite sequence of block placements is unique and specified by hand in advance.

Some researchers focus on minimalism, trying to achieve certain outcomes using robots with deliberately limited capabilities. Parker, Zhang and Kube [47] consider the problem of debris cleanup, a prerequisite to construction in littered environments. They demonstrate a collection of robots with only force and collision sensors, inspired by behavior of the ant *Leptothorax tuberointerruptus* [17]. These robots clear spaces in fields of debris by simply moving straight ahead, turning in a random direction when the force opposing their movement exceeds a threshold or when they collide with another robot. Melhuish et al. study minimalism in collective tasks like sorting [44] and construction [45]; in the latter, a collection of robots pushes pucks into a roughly linear wall, using as cues a white line on the floor at one end of their arena and a bank of lights at the other end.

Other researchers who have proposed systems for collective construction in this vein focus on hardware design. Terada and Murata [58] present a hardware system in which a mobile robot assembles structures of cubic blocks. The robot is confined to movement along the structure; blocks are passive, and locked together mechanically upon attachment by connectors manipulated by the robot. Everist et al. [14] describe a hardware system with the aim of eventual use in space, with free-flying robots assembling structures out of beams. Bowyer [11] proposes a scheme for a hardware system of terrestrial robots building with polymer foam, with the goal of building human-scale walls and arches.

Two studies describe simulated systems for swarm construction that do address the global-to-local problem. Both make assumptions likely to make physical implementation problematic, and in both cases, the structures produced only approximate the desired design. Mason [42] describes a system where mobile agents rearrange building material into roughly desired configurations, using artificial 'pheromones'. The shape-encoding scheme is fairly low-level, making specifying desired shapes somewhat involved. Agents are assumed to be able to move through each other and through building material, and to be able to evaluate levels of pheromone with arbitrary precision. I describe elsewhere [68] a system where mobile agents rearrange building material into desired arbitrary configurations, specified in a moderately high-level geometric language. Agents are physically restricted in their movement by the presence of building material and of other agents. Agents move on a discrete grid and are assumed to be able to localize themselves on that grid precisely; I have not shown to what extent those assumptions can be relaxed in that framework.

2.1.3 Non-swarm approaches to automating construction

Not every possible automated construction system must rely on the use of swarms. Other proposed approaches use more centralized methods, with single, more capable robots or other more sophisticated mechanisms. In many cases, devices are not mobile, and may be on a scale larger than that of the building or artifact to be produced.

Khoshnevis et al. [35, 36] propose a "Contour Crafting" approach akin to large-scale rapid prototyping, in which a gantry is erected above a desired building site, and an extrusion nozzle and pair of trowels deposit and shape material, respectively. Extensions involving other manipulation attachments and relatively high-level prefabricated units are proposed to handle features like plumbing and electrical systems.

Soar et al. [12, 52] similarly propose using the equivalent of large-scale rapid prototyping for construction. They further suggest using smaller-scale rapid manufacturing techniques to fabricate panels with desired features (insulation, ventilation, appropriate textures on inner and outer surfaces, etc.), to be assembled on-site by human construction workers.

Another related issue is planning for assembly: e.g., automatic generation of a sequence of actions to put a given set of components together to assemble a given artifact. For instance, Wilson and Latombe [75] discuss issues associated with geometrical constraints on the ordering of operations for assembly or disassembly, and investigate the complexity of assembly sequences.

2.2 Related problems

Several areas of active research involve the global-to-local problem of finding low-level properties or behaviors that will produce a desired high-level configuration of elements. These areas include programmed self-assembly (§2.2.1), self-reconfigurable modular robots (§2.2.2), and formation control (§2.2.3).

In each of these areas, there is typically a single class of elements—tiles, modules, or mobile robots, respectively—all members of which have comparable capabilities.[1] This uniformity distinguishes them from the construction problem, with its two classes of elements, mobile robots and building materials. The latter separation into two classes is an important one, as elaborated in the sections below. However, the global-to-local problem they share is an important similarity.

[1]There may be multiple types of elements within that class, whose detailed properties can differ. For instance, different types of passive tiles may have edges with different binding properties [49], or two different types of modules might have complementary attachment hardware [39]. However, the higher-level capabilities of the different types are not substantially different.

2.2.1 Programmed self-assembly

Programmed self-assembly is the problem of designing a collection of elements with edge binding properties such that, when they mix randomly (e.g., driven by ambient fluid forces), they bind to form desired assemblies. The elements may be homogenous or heterogenous; their binding properties may be fixed or dynamic; and they may have a range of capabilities such as ability to detect binding events or exchange information with neighbors.

This problem is not directly suited to construction as stated, mainly because of the issue of element movement, and associated issues of spatial and temporal scale. Programmed self-assembly assumes that units move at random. Its ultimate concern is typically with the microscale, where ambient fluid forces are sufficient to provide that movement. As a result, applying it to housing-scale construction is unlikely to be straightforward. Moreover, without excess elements in sufficient density and enough agitation, it can take an undesirably long time for elements to reach all necessary sites through random movement.

However, there are important connections to the construction problem. As discussed in Chapter 7, the problem of building a given structure can be decomposed into the subproblems of (1) deciding where units can be attached, and (2) getting them there. While programmed self-assembly's answer to the latter subproblem may not apply well to construction, both frameworks share the same first subproblem. In particular, self-assembling tiles which can change binding properties and communicate with attached neighbors have a great deal in common with the communicating blocks I will describe, where building blocks are equipped with embedded processors. In this respect, solutions to one problem may be applied toward the other. For instance, most systems for programmed self-assembly are subject to crystalline flaws, often as a result of not explicitly considering physical constraints on tile movement. The approach I present, in which a central concern is taking movement constraints into account to prevent such flaws, could then be useful for self-assembly of smaller-scale artifacts.

A number of studies in programmed self-assembly have considered tiles with different levels of capabilities, experimentally or in theory, of which a sampling follows.

Rothemund, Winfree, and others make passive tiles using DNA, with edge binding properties designed so as to stick together in certain patterns when mixed. These tiles may form algorithmically complex patterns such as Sierpinski triangles [49] or binary counting sequences [7].

Guo et al. [26] consider square tiles which can detect binding events, and when triggered by same, can change state and binding properties during the assembly process. They use a genetic algorithm to search for programs for such tiles.

Several researchers consider square tiles which, in addition to detecting binding events and changing state and binding properties, can also communicate state with attached neighbors. Jones and Matarić [30] show how to derive tile programs which will reliably form particular target structures chosen from a certain class of two-dimensional shapes. Arbuckle and Requicha [2] show how to program tiles to form certain simple shapes, and suggest using such a system to build structures whose shape is not fully prespecified but rather satisfies given functional requirements. Griffith et al. [24, 25] focus on the application of self-assembly to self-replication of 'strings' of square base units in two dimensions.

Other studies involve tiles with similar capabilities but other shapes. Klavins et al. describe theory [37] and hardware [8] for primarily triangular tiles, and detail a framework for automatic synthesis of tile programs to generate desired assemblies. White, Lipson, and others describe hardware designs for two-dimensional systems with square or triangular tiles [73], and a three-dimensional system with cubical blocks [74]. They show formation of specific structures, where the sequence of attachments is planned by hand in advance.

2.2.2 Self-reconfigurable modular robots

Self-reconfigurable modular robotics [50] asks the question of how a collection of mobile units can autonomously rearrange itself to achieve a desired configuration. Movement constraints typically allow units to move over a substrate of others, and require that all units be connected at all times. Often units are allowed to move through the structure, an operation that can be difficult to implement mechanically.

One could imagine building structures out of modular robots—starting with a huge collection of modules, directing them to rearrange into the desired structure, and calling the construction problem solved. However, the separation into mobile and structural elements in construction is an important one. Once a structure is in place, it's unnecessary for structural elements to have a capacity for movement. At that point, such a capability becomes a liability: not only would self-mobile blocks be far more expensive and complex than non-mobile ones, but they're likely to be less good at their structural role than dedicated building materials would be—the latter will be more structurally sound, can more easily have insulating and other desired properties, and so on. Optimizing the building materials for their structural properties will make them more effective and cheaper, while mobile robots can be reused for another building project.

A number of researchers have developed self-reconfigurable systems that address global-to-local shape formation problems, and often demonstrate hardware capabilities that would be useful in systems like those I describe, including self-aligning connectors and communication links between physically attached modules.

Rus's group at MIT has developed several modular robotic systems, with accompanying reconfiguration algorithms. The Molecule [39] has planning algorithms to allow its two types of units (male and female) to rearrange into desired structures; units can travel through the structure on a scaffold of other units. The Crystalline robot [65] has cubic modules which can similarly travel through the structure in order to reconfigure into a desired shape. Kotay and Rus [40] have also described hand-designed algorithms for generic reconfigurable robots to perform locomotion and reconfiguration tasks, and shown how a variety of existing modular robots can be made to work with these algorithms.

Støy and Nagpal [55, 56] describe a scheme by which a set of modules can automatically reconfigure into a desired shape whose scale depends on the number of modules available. The approach is for some modules to form a scaffold which other modules move over and through. Information about where additional modules are needed is communicated in the form of a directional gradient that directs modules' movement.

Yim et al. [64, 76] have introduced various self-reconfigurable systems and algorithms, including Telecubes, whose cubic modules can move in three dimensions (and through the structure) to form desired shapes.

Goldstein et al. [21, 22] have proposed a Claytronics system for dynamic three-dimensional object reproduction. The reconfiguration algorithm is based on stochastic motion of "holes" in the structure, which through random movement can transfer modules to desired locations. That approach is simple and decentralized but subject to subsets of modules becoming disconnected from the rest.

2.2.3 Formation control

Of the commonly studied topics using artificial swarms, the closest to construction is formation control. As with self-reconfigurable robots, the goal is for a collection of elements, all mobile, to achieve a desired configuration, sometimes in conjunction with other criteria. Movement constraints typically include maintaining minimum distances between all pairs of elements. Often globally available information or all-to-all communication is assumed. These factors can make it difficult to directly apply studies of formation control of mobile robots to the swarm construction problem. However, the abstract goal of both domains can be seen as similar.

Bahçeci, Soysal and Sahin [4] provide a review of several centralized and distributed methods for formation control.

Formation control and flocking algorithms frequently rely on explicit communication for coordination. Sugihara and Suzuki [57] describe an approach to form shapes such as circles and polygons. User intervention is required to select robots to play special roles (e.g., the corners of the polygon), and every robot is assumed to know the position of every other. Fredslund and Matarić [18] describe a scheme by which a set of robots can form a line of a desired shape, and move in formation, led by a "conductor" robot. Although the approach has robots broadcast identification information globally, robots need not know the position of others. McLurkin [43] describes algorithms for behaviors like clustering and leader-following, for a swarm of robots that use frequent communication.

With appropriate assumptions, analytical tools can be brought to bear on formation control algorithms. Spears and Gordon [54] describe an approach they call artificial physics or physicomimetics, in which agents calculate virtual forces based on their positions relative to others, and move as though acted on by those forces. Like real crystals, the resulting formations are subject to crystalline defects. Ögren, Fiorelli and Leonard [46] similarly use artificial potential functions to maintain formations, along with "virtual leaders", moving reference points used to govern the motion of the actual robots. Their analysis assumes global communication and a shared coordinate system. Feddema and others [15, 16] describe the design of cooperative control algorithms by defining a global performance index, partitioning it so that it reflects only local relationships, and taking the gradient of the result to obtain local control laws. The result may not globally optimize the original performance index, but interactions are local and analysis is tractable.

2.3 Other practical issues

Here I briefly review issues related to localization (§2.3.1) and to underwater structures intended for human habitation (§2.3.2).

2.3.1 Localization systems

As discussed in the previous chapter, localization can be a very difficult task for mobile robots. Approaches that rely solely on the resources of individuals cannot avoid significant uncertainty or errors in position estimates. Odometry is notoriously unreliable. Sufficiently sophisticated inertial navigation systems, such as those used in submarines or strategic missiles, can give better estimates of relative movement [6]. However, the size and expense of such systems is not appropriate for artificial swarms, where individuals are meant to be simple, small, and cheap. Moreover, such dead-reckoning approaches require that the initial position be known with precision, which would interfere with ease of deployment. Perception of passive environmental features can be used to improve estimates, but considerable uncertainty is still typical.

Localization performance can be improved using active external cues, typically artificial landmarks in known locations that communicate their position to the robots. The best-known such example is probably the Global Positioning System (GPS) [20, 28, 48], in which satellites with known orbits transmit signals that a receiver can use to determine position to within about 15 meters. With differential GPS, using another receiver at a known location as an additional reference, the uncertainty can be reduced to less than 5 meters. With still more expensive systems with sufficiently high communication bandwidth broadcast from that fixed receiver, uncertainty in the centimeter range can be achieved.

Trying to use GPS in a swarm construction setting may present a number of problems. One issue is that GPS receivers rely on line-of-sight with the satellites; obstacles (including the building in progress itself) can occlude signals or give multipath reflections, leading to potentially large errors in position estimates. Moreover, while GPS is suited to open terrestrial settings, it will not work underwater, where the satellite signals cannot reach, or in extraterrestrial settings where there are no satellites available. Thus GPS fails in settings arguably most suited for automating construction.

Other systems involving fixed beacons in known locations have been presented for localization in settings where GPS is unavailable, including indoors [48] and underwater [5]. These can suffer from the same problems of occlusion and multipath errors in settings with obstacles, which a construction project will necessarily contain. Also, many existing systems for underwater localization rely on the mobile robot broadcasting a signal to fixed reference units, an approach unlikely to scale up well to handle large numbers of robots.

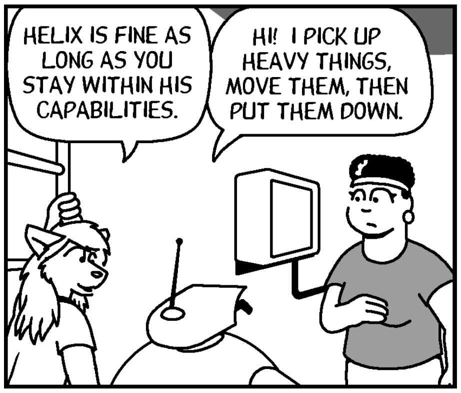
To deal with these problems of position estimation, as I describe in the next chapter, I have robots use the partial structure as a collection of landmarks, by reference to which they can determine their position in a shared discrete coordinate system. The only time robots need to know their location is when along the perimeter of the structure in progress, and then with no greater precision than the size of a block.

2.3.2 Underwater habitats

One potential application for automated construction systems I emphasize is underwater structures, particularly habitats assembled out of high-level prefabricated units each fulfilling a distinct functional role (Chapter 6). Previous studies of undersea habitats can inform future projects such as these, with regard to physical and psychological requirements, and other logistical considerations.

Allmendinger [1] and Jones [32] give extensive studies of manned underwater structures and complexes, primarily for oil production but potentially for other applications such as oceanographic research stations. They discuss spherical and other hull shapes for underwater units; problems manufacturing and transporting large single-unit systems could be addressed with multi-hull systems composed of smaller units—ideal for a system that assembles a large structure out of high-level prefabricated units. Large structures are particularly essential considering the wide range of facilities such a habitat would require, and taking into account the factor of necessary crew sizes for missions of more than a few days—with a crew size of six, the endurance limit is 14 days [1]; for longer missions, crews of 25 or more are preferred, and crews of 50 or more are still better to minimize total cost [32]. These studies also present hardware designs for features such as emergency escape capsules, transfer modules for getting crew in and out, and docking mechanisms.

In a separate study, Jones [33] describes considerations for where an underwater habitat might be located, including factors like currents and temperatures, and how it might be maintained. He also discusses issues important to underwater communication and perception, including optical visibility and the location-dependent effects on it of turbidity, and frequency-dependent considerations for sonic communication and sonar.


HELIX IS FINE AS LONG AS YOU STAY WITHIN HIS CAPABILITIES.
HI! I PICK UP HEAVY THINGS, MOVE THEM, THEN PUT THEM DOWN.

From *Freefall*
©2001 Mark Stanley
freefall.purrsia.com
Used with permission.

Chapter 3

Approach and Variants

This chapter introduces the framework in which I consider the collective construction problem and describes the approach I take to addressing it. Algorithms described in this chapter will let an unspecified number of robots assemble arbitrary solid two-dimensional structures, according to high-level user-specified designs. No explicit communication between robots is necessary; robots use the partial structure for implicit coordination.

The problem framework can be summarized as follows (Figure 3-1). Building blocks are square and can be attached to each other on all four sides. A block-sized marker indicates where construction is supposed to start, acting as a seed around which the structure grows. Robots (in unspecified numbers) fetch blocks from elsewhere in the workspace, bring them to the growing structure, and travel along its perimeter for some distance before attaching their block. The heart of the algorithmic problem is how and where robots decide to attach blocks, such that a desired structure ends up reliably produced.

Section 3.1 describes the assumptions made about robot capabilities and block attachment, and discusses how these assumptions are consistent with reduction to practice in a physical system. Section 3.2 describes the overall approach taken here, covers four specific algorithmic variants which depend on varying capabilities in building blocks, and discusses considerations associated with multiple simultaneously active robots.

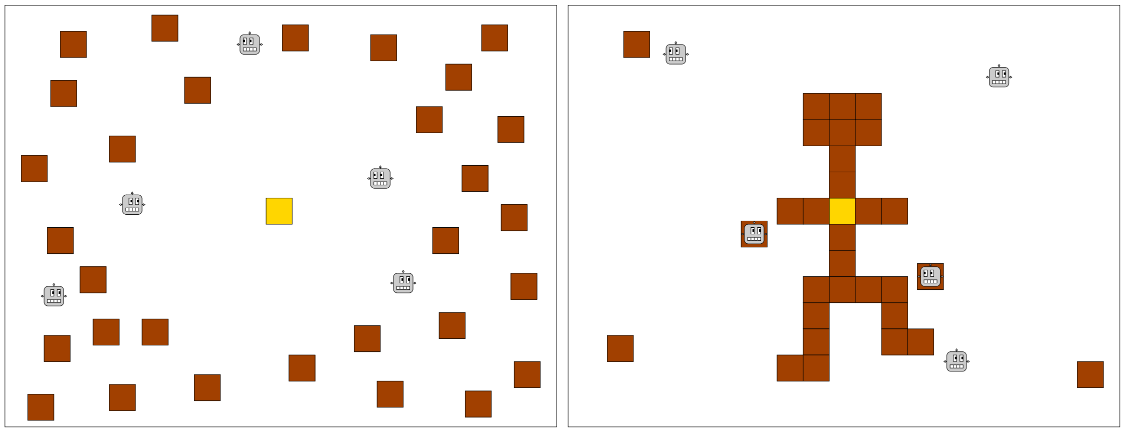


Figure 3-1: Framework for the construction problem as considered here. Left: Elements in the environment are mobile robots (gray), movable building blocks (brown), and a marker where construction is to start (yellow). Right: The object is for robots to fetch blocks and attach them to form some specified structure starting at the marker.

3.1 Assumptions

This section collects the general assumptions I make about the capabilities of robots (§3.1.1) and blocks (§3.1.2). In the variant approaches described in sections 3.2.2–3.2.5, additional capabilities may be assumed; these are described in each case as appropriate. Section 3.1.3 discusses issues related to the realization of a full system for automated construction.

3.1.1 Robots

I assume that robots have the following capabilities:

1. They can move in any direction in the plane, alone or while carrying a block, and avoid collisions.

2. They can follow the perimeter of the partially built structure.

3. They can find both unattached blocks and the structure in progress. (This could be achieved in a number of ways, for instance by following long-range beacons associated with the marker and with supplies of additional blocks; see discussion in §3.1.3 below.)

4. They can pick up unattached blocks once found, one at a time, and attach them to the structure.

The first three of these have repeatedly been demonstrated in a variety of autonomous robotic systems [3].

I do not assume that robots can communicate with one another. While such communication could be helpful, it's preferable not to have to rely on it. In addition to avoiding other problems associated with communication described earlier, that eliminates dependence on teams: robots are capable of building independently, and a single robot could still complete the construction task if the rest of the swarm became disabled.

It can also be useful for robots to have access to aspects of their internal state (to monitor battery power, overall correct functioning, etc.), in order to be able to respond accordingly (seek out a recharging station, leave the building site, travel to a designated location for repairs, etc.).

3.1.2 Block attachment

Materials that social insects use for building include things like mud, pulp, and wax [10, 29, 34, 62]: soft materials that come in variable-sized quantities. This factor contributes to the fact that no two termite mounds are identical, and to the lumpy, organic look of them. In human construction projects, we typically use rigid materials, and build regular shapes with right angles. That's both a limitation and a benefit.

Rigid materials can be more difficult than soft ones to manipulate. Soft materials can be pushed into narrow spaces and plug up gaps, or can be deformed to be pushed out of the way. Rigid materials, by contrast, will bring heavier constraints on where and how they can be attached. For instance, it may be difficult to fit a block into a space exactly one block wide between two other blocks (Figure 3-2A).

In everything that follows, I will make the conservative assumption that a space one block wide between two other blocks is too physically constrained to allow a block to be shoved into position. Disallowing that situation entirely will make it mechanically easier for robots to maneuver and attach their blocks. Another result of that restriction is that if such configurations are avoided, then so are more complicated gaps where a block might have to be moved down a long tunnel, where sites are entirely closed off and inaccessible, etc. (Figure 3-2B, C). The corollary of the restriction is the "separation rule": separated blocks must never be attached in the same row (that is, with the same x- or y-coordinate) if all the sites between them are ultimately meant to be occupied. Otherwise, as blocks are added, eventually an unfillable one-block gap would result (Figure 3-2D).

An advantage of rigid materials is that they can be used to form regular shapes with reliable dimensions. An assembly of square blocks forms a grid with an implicit coordinate system. That coordinate system, and the landmarks represented by block edges, can be used for position reference.

A generally desirable quality in construction, and an important one if the structure is to be used for position reference, is for block alignment to be fairly precise. Ensuring precise alignment can be a nontrivial problem for simple mobile robots, for which manipulation of the physical world can in general be difficult. Rather than having to make robots capable of exact manipulation, a different approach, in keeping with the theme of extended stigmergy, is to improve instead the capabilities of the building material. Blocks can be equipped with self-aligning connectors (which may be either active [39] or passive [58]), so that a robot need only get a block close to an attachment site, and the connectors then ensure fine adjustment.

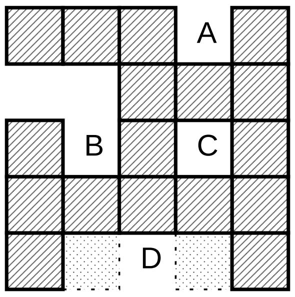


Figure 3-2: Physical constraints on block placement. The desired structure here is a solid 5 × 5 square of blocks (diagonal shading). I assume that a block cannot be physically maneuvered into a constrained site like A. Eliminating such situations also eliminates more complicated problematic cases like those at B and C. To avoid unfillable gaps like A, separated blocks must never be attached in the same row (as they have in the bottom row) if all sites between them are meant to be occupied; otherwise later addition of blocks (light shading) will result in an unfillable gap (D).

3.1.3 Implementation issues

A number of issues fall strictly outside the domain of the algorithmic approach presented here, but would need to be considered in real applications. Here I briefly discuss these issues and ways that they could be approached.

- *Preparing the area for construction:* It may be necessary to start the overall construction process by taking steps to prepare the workspace for construction. On land, this may involve clearing a space of material [47,68] or leveling an area. Some applications may call for robots to maintain models of their environment, in which case an initial stage of construction could involve surveying the area to map the terrain, note the locations of immovable obstacles, choose the best place for the structure, etc. [77, 78], on land or underwater. Here I will not consider these steps, and simply assume that there's initially a clear workspace large enough to build in.

- *Finding building materials:* In some applications, robots may produce their own building material, for instance extruding foam from compressed tanks [11], or collect and use materials already present in the environment. In others, specialized building material may be supplied along with the robots. The latter is the approach taken here.

  Supplied or scavenged building material could be scattered widely [42, 68] or localized to a small number of 'caches' deployed outside the primary workspace [69–71]. Robots may take a variety of approaches to finding materials, e.g., via random walk [42], by searching [68], or in the case of supplied caches, by following a long-range, low-bandwidth signal broadcast by each cache [69–71]. Finding the partial structure under construction may likewise be accomplished in any of these ways. Without restricting the domain to one of these choices, I assume that unattached blocks are located far enough away from the structure so as not to interfere with the construction process, and that robots are capable of finding blocks and returning with them to the structure in progress.

- *Starting the structure:* The marker indicating where construction is supposed to start could be implemented in a number of ways and with a variety of features, depending on the application. It could be sufficient for the marker to be simply a block like any other; there could be more than one such block, if the system is intended to produce multiple objects, as in some self-assembly applications. The marker could be an active element; in such a case, a long-range beacon could be used to help robots find the structure in progress throughout the construction process, as noted above. In weightless settings like outer space and underwater, it could additionally provide an anchor, with potential further benefits such as providing power to active blocks, a repository near the structure for temporarily unwanted blocks, etc.

Realizability

Square blocks, assembled, form a grid. The algorithmic approach I describe takes a step further and abstracts elements to occupy discrete positions in a cellular grid-world. That will be useful for considering algorithmic problems at a higher level of abstraction, and later on, for simplifying simulation experiments.

In making that abstraction from a noisy world with continuous-valued positions, it is crucial to be careful that the system remains physically realizable. Otherwise the value of the abstracted system is questionable at best.

Physical robots in real-world environments are subject to a wide variety of internal and environmental perturbations; the system will need to be robust to such factors. The approach here is designed to minimize the effect or likelihood of problems that robots and blocks might encounter. It relies on simple basic capabilities that can be made robust and self-correcting.

Blocks can be taken to have discrete relative coordinates in the grid's implicit coordinate system; the marker can be taken to be the origin. Robots' precise positions are not important; robots only need to know their position when along the structure perimeter, and then only as precisely as the nearest block coordinate. The physical landmarks represented by blocks and their boundaries make it possible for robots to establish and maintain position estimates to that extent. Self-aligning connectors between blocks ensure the structure conforms to that grid.

These considerations make it safe to make the abstraction to the cellular-world model. Problems, from unexpected environmental influences to breakdown of some robots, need not prevent completion of a structure; temporary failures at any step can be corrected.

Chapter 5 presents a hardware prototype that demonstrates the feasibility of building such a system in the real world.

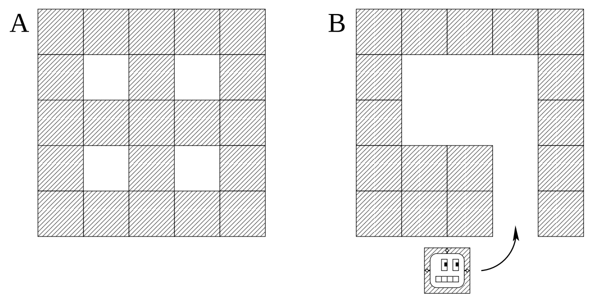


Figure 3-3: Examples of structures not allowed using the approaches described in this chapter. (A) a structure with deliberate holes; (B) a structure with an alley too narrow for perimeter-following.

3.2 Algorithms

In this section I describe four algorithms for the collective construction problem, using different levels of extended stigmergy. Each algorithm is sufficient to reliably produce structures with perimeters of arbitrary shape, specified according to an explicit high-level representation.

This chapter and the next three will consider only two-dimensional structures without deliberate internal holes. Also, because the approach I describe depends on perimeter-following, any desired concavities in a structure must be wide enough for two block-carrying robots to pass in opposite directions while following the perimeter (Figure 3-3). The extent to which the latter consideration limits admissible structures will depend on the implementation.

Section 3.2.1 first outlines the common approach. The next four sections then give the four specific algorithms: using inert, identical blocks (§3.2.2), blocks that are all distinct (§3.2.3), blocks that can store dynamic state (§3.2.4), and blocks that can communicate with physically attached neighbors (§3.2.5). These are introduced assuming a single robot; section 3.2.6 then discusses considerations for many robots operating simultaneously.

3.2.1 General approach

The critical aspect of construction on which I focus is how and when a robot decides to attach a block. It's necessary both that the structure ultimately takes on the desired shape, and that 'dead-end' states are avoided, where sites that are meant to have blocks cannot be physically reached. This section considers these issues in turn.

Specifying the shape

Let the shape to be constructed be given a lattice-based representation, with a coordinate system whose origin is the marker. Each site in this lattice specifies whether or not a block should be attached at the corresponding site in the structure. Call this lattice the *shape map*. Building a prespecified structure then means attaching blocks at all of the sites, and only the sites, specified by the shape map. The shape map can be specified as a binary occupancy matrix, more compactly as a superposition of rectangles [55], etc.

If a robot with the shape map can determine its position in the structure's implicit coordinate system, then it can trivially determine whether or not a given site should be occupied. Because of limitations of position estimation and odometry, establishing position is a very difficult task for mobile robots in general. However, robots can use the structure as a reference to determine their location. Additionally, block edges can act as a collection of features which can be used to correct the effects of odometry errors. How robots agree on a shared coordinate system is described for each variant approach below.

Ordering attachment

The order of attachment must be partially constrained, to avoid dead-end states in which intermediate configurations prevent robots from reaching desired sites, while still allowing parallelism. Full ordering would require that blocks be placed in a particular sequence, in which none could be attached until the previous attachment was complete. Putting restrictions instead on where blocks *may not* be attached, rather than specifying where they *must* be attached, potentially allows many more blocks to be attached at once, and also lends itself more readily to decentralization.

Disallowing attachment at sites that (1) are supposed to be left empty according to the shape map, or (2) break the separation rule, is sufficient for the reliable completion of any solid structure.

Intuitively, the separation rule prevents unfillable gaps, and as a result any other concavities that would prevent robots from following the perimeter. Together with the requirement that planned concavities be wide enough to allow perimeter-following, that ensures that no partial structure will restrict access to sites meant to be occupied.

Further, until the structure is completed, there will always be some site along the perimeter where a block may be attached. Figure 3-4 illustrates: If there were no such site, then it would be the case that for every site along the perimeter meant to be occupied, attaching a block there would break the separation rule. That is, for each such site (e.g., A), there would be a nonadjacent occupied site in the same row (B), with all intermediate sites (shaded) meant to be occupied. In particular, the site adjacent to the occupied site would be meant to be occupied (C). But then one of the following must be true: a block may legally be attached there; a block may be attached at a different site in the perpendicular row (D); or the structure had already violated the separation rule.

Thus it is not possible for structures built according to the above principle for restricting attachment to get stuck in partially complete states where no further block can be added.

It is clear that this is also the least restrictive possible partial ordering: Any sites restricted according to this scheme would lead to a dead-end state if they were occupied following some other scheme.

That principle could be implemented in a straightforward way with a centralized

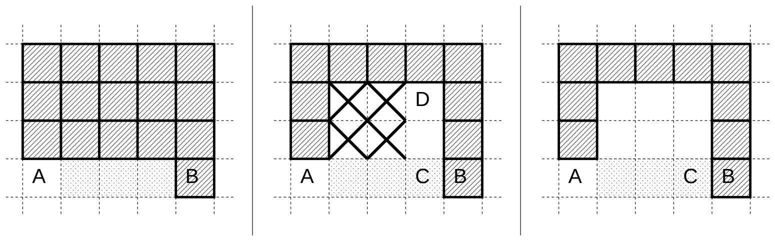


Figure 3-4: Demonstration that building according to the separation rule cannot get stuck in a partial state where no further block can be added. Sites marked with X are meant to be left unoccupied. Suppose attaching at a site A would violate the separation rule. Then there must be a separated block B in the same row, with all intermediate sites (light shading) meant to be occupied. But in that case, either a block can legally be attached at C, the site adjacent to B (left); a block can legally be attached at D, another site in the same row as C (center); or the structure had previously violated the separation rule (right).

approach. However, with a decentralized swarm of robots which have only local perception, some scheme is necessary to acquire the nonlocal information regarding whether distant blocks are attached in a given row.

3.2.2 Inert blocks

In the first variant I assume that blocks are indistinguishable, with no capacity for communication. This approach uses the most basic sense of stigmergy for construction, where the only information blocks carry is the fact of their presence. In this case the robots are responsible for knowing the shape map and ensuring that unwanted gaps are avoided. Algorithm 1 sketches the following approach, consisting of two parts: (1) establish position; (2) find an allowed place to attach a block.

Establishing location: For any fixed perceptual distance, a robot will be able to distinguish only a limited number of distinct local configurations. Solely local observations are then insufficient if the system is to be capable of building arbitrary structures; a robot needs additional state to infer its location. One simple method to do this is to make one edge of the marker distinct, so a robot can recognize it as different from all other block edges, and to specify in the shape map that that distinct edge is to fall along an edge of the desired structure. The marker then serves as a landmark, which will be found by any robot following the perimeter of the structure at any stage of construction. Upon reaching the structure, then, a robot follows the perimeter until reaching that landmark. It then knows its position and orientation, which it updates thereafter by keeping track of the number of blocks it passes and turns it makes. In this way the partial structure provides a source of odometry for the robot.

Avoiding gaps: Algorithm 1 ensures that no separated blocks are attached in the same row if all sites between them should be occupied. The idea is as follows: Every robot follows the perimeter in the same direction (say counterclockwise). To start a

Algorithm 1 Pseudocode procedure for assembly of a structure of inert blocks. An ‘end-of-row’ site is one where the robot is either about to turn a corner to the left, or the site directly ahead is not supposed to have a block according to the structure design.

```
     while structure not complete do
       fetch new block
       go to structure
       while not adjacent to labeled side of marker do
 5:      follow perimeter counterclockwise
       seen-row-start ← false
       while still holding block do
         if (site should have a block) and
             ((site just ahead has a block) or
10:          (seen-row-start and (at end-of-row))) then
           attach block here
         else
           if at end-of-row then
             seen-row-start ← true
15:        follow perimeter counterclockwise
```

new row of blocks, a robot must first check the entire row itself to make sure there are no blocks already present. Once it has traversed a full row along the perimeter and found it empty, then it’s allowed to start that new row by attaching a block at the end. If a robot finds a partially complete row of blocks, it’s allowed to add a block to extend it.

The more detailed operation of Algorithm 1 is as follows. (a) Line 9 allows a block to be attached at a site that has two occupied neighbors, as with either of the sites to the left and right of D in Figure 3-2A. Such attachments correspond to adding new blocks to existing rows, and cannot result in a violation of the separation rule. (b) Line 10 specifies that a new row can be started only if the robot has verified on its tour that no block has been placed earlier in that row, and there are no sites further along in the row where a block might already have been placed. Unfillable gaps are thus avoided.

The algorithm will also result in blocks filling the whole area specified by the structure design, as is straightforward to prove by contradiction:

Define an *inside corner* as an empty site with blocks at two adjacent sites, and an *end-of-row* site as one where a robot there is either about to turn a corner to the left, or the occupancy matrix specifies that the site directly ahead is supposed to be left empty—that is, a site at the counterclockwise end of a row bordering the structure perimeter (Figure 3-6). Attaching a block at an end-of-row site corresponds to starting a new row; attaching at an inside corner extends existing rows.

Suppose that the algorithm can get stuck—that is, that the system can reach a stage of construction where no further blocks can be attached, but there exists at least one site which is empty but supposed to be occupied. That can’t be due to an

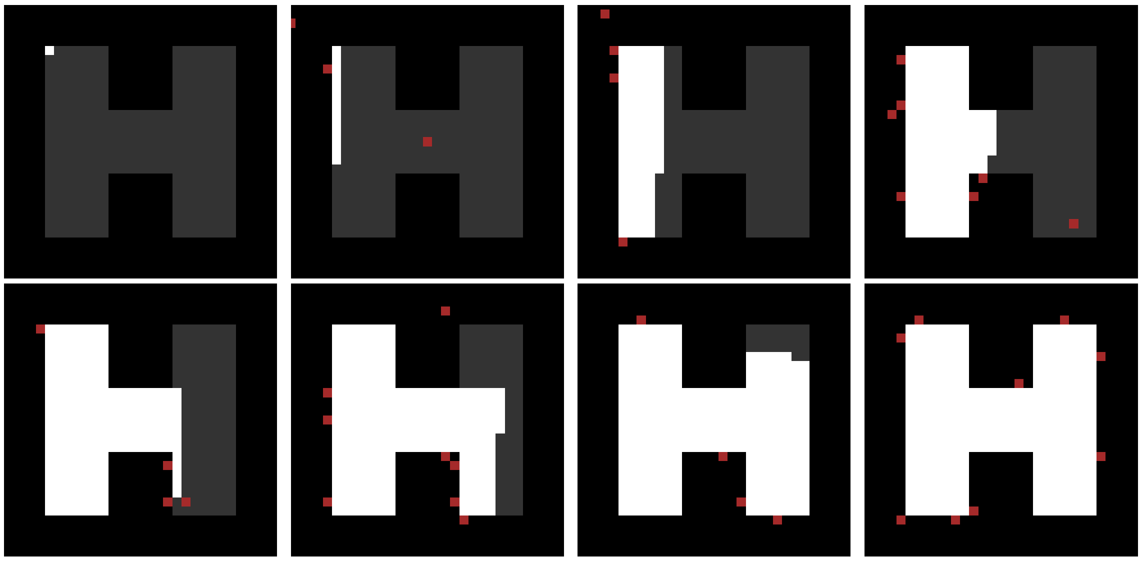


Figure 3-5: Simulated construction of a sample structure of inert blocks, showing successive snapshots during the process of construction by ten robots. White: blocks; brown: robots carrying blocks; gray: empty cells where blocks should be attached; black: empty cells that should be left empty. The marker is in the upper left corner.

unfillable gap in the sense of Figure 3-2A, as shown above.

Now, it must necessarily be the case that along the perimeter of the incomplete structure, either (1) there are supposed to be blocks at all sites, or (2) at least one site is supposed to be left empty.

- In case (1): Consider any closed shape drawn only with right angles, with a robot following its perimeter counterclockwise. Geometrically, there must be a point where the robot comes to two left turns (end-of-row sites) in a row without a right turn (inside corner) in between. But the site just before the latter left turn is an end-of-row site that can be occupied according to the algorithm, a contradiction.

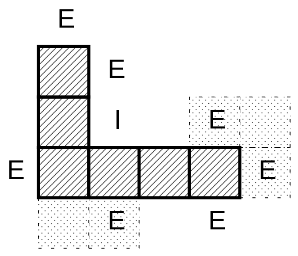


Figure 3-6: A sample structure in progress. Blocks have diagonal shading, empty sites meant to be occupied have light shading. End-of-row sites are marked with E, an inside corner with I.

- In case (2): There can be no inside corners where blocks are supposed to be attached, or the algorithm would be able to attach them there. So somewhere in its perimeter tour, the robot must pass through a site not intended to be occupied. After that it must at some point pass through a site or series of sites meant to be occupied.

  Within that set of sites, it must eventually encounter an end-of-row site. (Since no inside corner is meant to be occupied, the robot will reach an end-of-row before it reaches an inside corner.) If the *seen-row-start* flag has been set to true, then the robot can attach a block there, a contradiction. If the flag is still set to false, then it will be set to true at this point; and in that case, this argument can be applied recursively. At some point in its perimeter tour, the robot will have to reach an end-of-row site with its flag set to true.

In all cases the result is a contradiction, which completes the proof.

Thus Algorithm 1 is sufficient to generate any desired solid structure whose concavities are wide enough to allow perimeter-following.

The memory and communication requirements of the system for assembly with inert blocks are as follows (Table 3.2 summarizes the requirements for all variants considered here):

	Robots	**Blocks**
Static state	shape map	none
Dynamic state	*seen-row-start* (one bit); coordinates (two integers)	none
Communication	none	none

To implement this algorithm, robots must be capable of recognizing block boundaries, end-of-row sites, inside corners, and the distinct edge of the marker.

Indistinguishable marker

The approach just described for inert blocks relies on the marker being distinct from other blocks. It is possible for robots to establish their location even if the marker is a block like all others.

Using the grid formed by the blocks as a guide, a robot can circle the structure, keeping track of its relative movement, and determine when it has returned to a grid space already visited on this tour. (It will eventually return to some previously visited space even if other robots are adding blocks to the structure at the same time, potentially occupying, e.g., the site where this robot first reached the structure.) If it has maintained a record of its full tour, it then knows the shape of the partial structure it has circled. This recorded shape may be out of date, if other robots have meanwhile been attaching blocks; that makes no difference. The important thing is that the shape can be fit into the shape map. If robots can establish their direction (suppose, e.g., each has a compass with accuracy to within better than 45 degrees), and all robots use the same convention to fit the observed shape into the shape map in the same way (e.g., choosing the topmost and leftmost position where it's consistent),

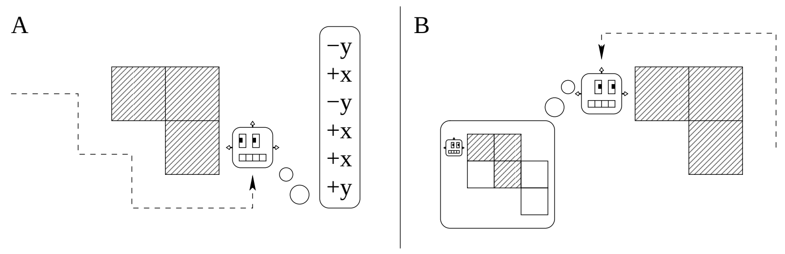


Figure 3-7: Robot procedure for fully indistinguishable blocks. A: As a robot moves along the structure perimeter, it keeps track of each step of its relative movement. B: When it returns to a previously visited grid space, it can determine the shape of the partial structure it circled, fit it into the structure design in a standard way, and thus establish its absolute position in a structure-based coordinate system.

then at this point the robot can determine its absolute position in the shared structure coordinate system (Figure 3-7).

After establishing position in this way, robots can go on to attach blocks at sites consistent with the shape map and the separation rule, exactly as above.

This approach is slower than that where the marker is distinct, because robots have to make a full circuit of the structure, rather than being able to establish their position as soon as they reach the marker. Moreover, it requires more dynamic memory from the robots, since they have to record their whole tour. However, it can allow better use of parallelism by a swarm, because robots may be anywhere along the structure perimeter when they determine their position, rather than only at the single landmark. I will return to this point during the discussion of parallelism in the next chapter. Otherwise, I will not consider this variant in further detail in the remainder of this thesis.

3.2.3 Distinct blocks

Algorithm 1 requires robots to find the marker before they can attach a block, and to keep track of their movement along the structure, with unfavorable implications for construction speed and behavioral robustness (Chapter 4). An alternative is to make all blocks distinct, so that each becomes a potential landmark. Reliably distinguishing an arbitrary number of blocks could be very difficult for robots if it is to be accomplished, e.g., visually. However, it could be done simply and reliably, e.g., by labeling each block with an RFID tag.[1]

Establishing location: With static labels, every block can be made distinct. However, there is no advance information regarding where blocks might end up attached to the structure. A robot must then maintain a dynamic *label map*, storing the labels and locations of all blocks in the structure. The initial label map has the marker at the origin.

[1]Radio-Frequency IDentification tags are circuits that can store information. Passive tags can do so without a power source. When an external transceiver focuses an RF beam at a tag, the current induced in the tag's antenna enables it to transmit a response—e.g., a unique ID code.

Algorithm 2 Pseudocode procedure for assembly of a structure of distinct blocks. The temporary map will only be necessary when multiple robots are active simultaneously (§3.2.6).

```
while structure not complete do
  fetch new block
  go to structure
  while not adjacent to label in label map do
    record label in temporary map
    follow perimeter counterclockwise
  fill in label map from temporary map
  seen-row-start ← false
  while still holding block do
    if (site should have a block) and
        ((site just ahead has a block) or
        (seen-row-start and (at end-of-row))) then
      attach block here
    else
      if at end-of-row then
        seen-row-start ← true
      follow perimeter counterclockwise
      if adjacent labels are not in label map, add them
```

When a robot reaches the structure perimeter, it follows it, keeping track of yet-unknown block labels along the way, until it encounters a block whose position it knows. Disambiguating orientation may be done in a number of ways: the robot may have its own compass; it can go on to find a second known block (or, initially, a distinct edge of the marker as before); or the four sides of each block can be distinct, with robots storing in their label maps the orientation of each block as well. With pose thus established, the robot can fill in its label map with the previously unknown labels it had encountered.

Finding a legal attachment site, so as to avoid unwanted gaps, can then be done as before. Any other unknown labels encountered along the way are added to the label map. Algorithm 2 summarizes this approach.

Construction will proceed faster with this algorithm than with identical blocks. A robot can establish its location more readily, typically not needing to follow the perimeter all the way to the marker to do so. The marker can be located in the middle of the desired structure, rather than along an edge, allowing construction to proceed on all sides.

The memory and communication requirements of the system for assembly with distinct, statically-labeled blocks are these:

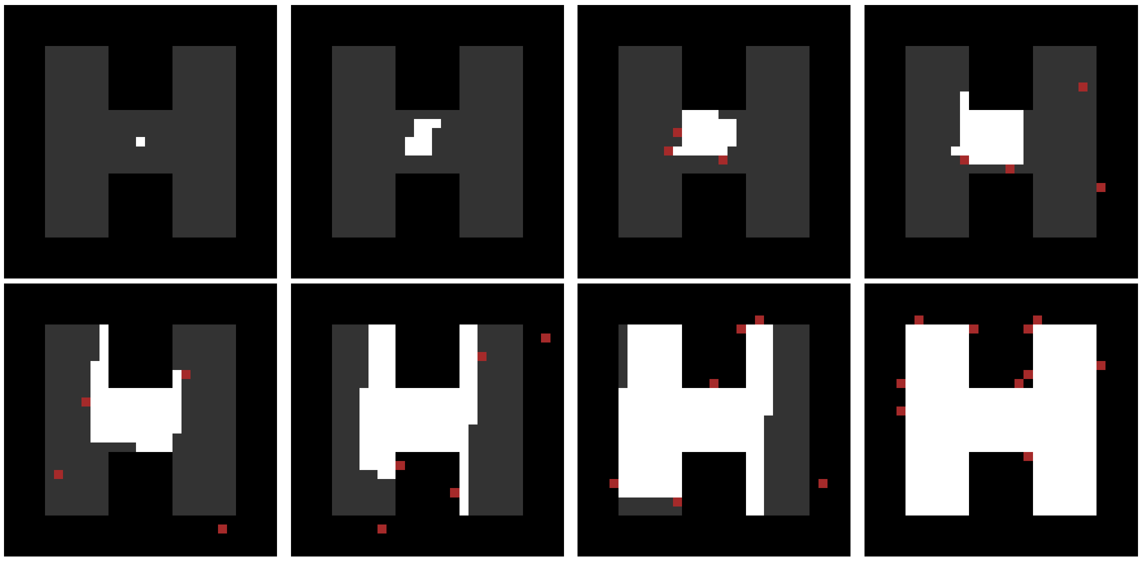


Figure 3-8: Simulated construction of a sample structure of labeled blocks, showing successive snapshots during the process of construction by ten robots. Construction takes place in a qualitatively similar way with both kinds of labeled blocks (distinct and writable). White: blocks; brown: robots carrying blocks; gray: empty cells where blocks should be attached; black: empty cells that should be left empty. The marker is in the center.

	Robots	**Blocks**
Static state	shape map	label ID
Dynamic state	*seen-row-start* (one bit); coordinates (two integers); label map	none
Communication	none	none

Robots using this algorithm must be capable of reading block labels, and of recognizing end-of-row sites and inside corners.

3.2.4 Writable blocks

Robots have access at any time only to local information, and they need nonlocal information to accomplish a global task. That nonlocal information can be stored either in robot state or in the environment; different kinds of information lend themselves to one or the other. Here, the location in the shared coordinate system is global information that it is natural to associate with the blocks rather than the robots.

Suppose robots can change the state of block labels. For instance, some RFID tags are writable, with on the order of 1 kilobyte of memory[2] that any transceiver can write to and any other can read from. In this case every block can store its coordinates explicitly, and thus act as an unambiguous landmark. A robot can quickly establish its

[2]Active tags, containing their own power sources, may have memory in the range of megabytes.

Algorithm 3 Pseudocode procedure for assembly of a structure of writable blocks.

```
    while structure not complete do
      fetch new block
      go to structure
      read position from neighboring label
 5:   seen-row-start ← false
      while still holding block do
        if (site should have a block) and
            ((site just ahead has a block) or
            (seen-row-start and (at end-of-row))) then
10:       attach block here
          write coordinates to that block
        else
          if at end-of-row then
            seen-row-start ← true
15:       follow perimeter counterclockwise
```

position upon reaching the structure, disambiguate orientation as with static labels, and proceed directly to find a legal attachment site, writing coordinates to the new block when it attaches it (Algorithm 3).

Robots need not maintain the (potentially extensive) dynamic memory for a label map; the blocks collectively do that, embodying that information where it is needed. The cost of dynamic rather than static labeling is that it represents a more complicated capability for robots and blocks. However, depending on the implementation (again, as with RFID tags), that additional cost may be small or negligible.

The use of writable labels will generally be unambiguously preferable to that of static ones. Only in circumstances where writable labels are for some reason unfeasible will static labels present a potentially preferable alternative to indistinguishable ones.

The memory and communication requirements of the system for assembly with writable blocks are these:

	Robots	**Blocks**
Static state	shape map	none
Dynamic state	*seen-row-start* (one bit)	coordinates (two integers)
Communication	none	none

Robots using this algorithm must be capable of reading and writing block labels, and of recognizing end-of-row sites and inside corners.

3.2.5 Communicating blocks

Embedding processors in blocks can extend their capabilities to store, process, and communicate information. The increasingly low cost of computing power makes such an approach feasible to consider.

If robots were to communicate explicitly, they would have to use external signaling, which can be unreliable, subject to multipath effects and other complications, and

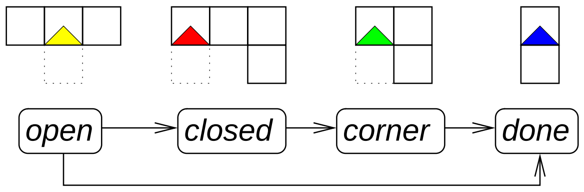


Figure 3-9: Examples of block faces in each of the four possible states, with the finite state machine for a single face. Dotted lines show the potential attachment site associated with a face.

may not scale up well with large numbers of robots. Blocks, by contrast, have a direct physical connection to each other once they are connected to the structure; that link can be the basis for reliable, unambiguous, rapid communication. The structure then becomes a distributed network with nearest-neighbor connectivity.

Sensor nodes, such as the Berkeley motes, have demonstrated low-power, low-cost computing devices [27], while modular robot research has demonstrated designs for reliable communication between physically-connected devices [50, 76].

The responsibility for determining whether or not a site is available for block attachment can then be shifted from the robots to the structure itself. Robots become responsible only for transporting blocks to available sites. Blocks indicate to robots passing by along their perimeter whether attachment there is allowed. That communication need only be low bandwidth and at short range, avoiding problems associated with signaling over distances, interference when many robots are active, and ambiguity in which agents are signaling.

Algorithm 4 outlines control rules for robots and blocks for building a solid structure. Robots simply circle until they find a site that gives them permission to attach. Blocks in the structure have the shape map, and give permission for attachment at a site only if the standard two criteria are satisfied: (1) the shape map specifies a block there; (2) there are no separated blocks in the same row, unless the shape map specifies a deliberate gap between them.

The basic idea for how to guarantee a solid structure remains the same as for noncommunicating blocks, but more sites are now available for potential attachment. The first block added to a given row can be located anywhere along its length, not just at one end. Further construction in that row can proceed in both directions from the first block, not just in the one direction possible with noncommunicating blocks. The active blocks communicate nonlocal structure information and make it available to robots anywhere along the perimeter.

Algorithm 4 refers to knowledge about distant blocks having been attached in the same row, without describing an implementation for how that knowledge can be obtained. When all blocks are identical, the following algorithm lets them ensure the separation rule is obeyed while maintaining only a few bits of state.

Block algorithm: Each block stores the shape map, its own location in the shared coordinate system, and a state for each of its faces. This latter state is associated with where blocks have already been attached in the row the face borders. There are four

Algorithm 4 Pseudocode procedure for assembly of a solid structure of communicating blocks.

A: Blocks

```
loop
  for all sides S do
    if a robot asks to attach a block to S then
      if (design specifies a block there) and
         (no blocks are yet attached in that row) then
        get confirmation from other blocks in this row that they will not allow
            attachment in that one
        allow attachment
      else if (design specifies a block there) and
              (a block is attached in that row adjacent to that site) then
        allow attachment
      else
        forbid attachment
```

B: Robots

```
while structure not complete do
  fetch new block
  go to structure
  while still holding block do
    ask any adjacent structure blocks if block can be attached here
    if all structure blocks answer yes then
      attach block here
    else
      follow perimeter counterclockwise
```

possible states (Figure 3-9): *open*, if no blocks have yet been attached in the adjoining row, so that attaching at that face would start a new row; *closed*, if attaching at that face would put two separated blocks into the same row, or if the shape map specifies that the adjoining site should be left empty; *corner*, if an adjacent block means that the site in question has two neighbors; *done*, if a block has been attached to that face.

New blocks can be attached to *open* or *corner* faces. When a robot is given permission to attach a new block to an *open* face, the structure block sends a message along that row in both directions; each recipient sets its corresponding face, formerly *open*, to *closed*, and passes the message on, thus locking out the rest of the row from attachment. Once a block attachment is completed, the original structure block sets its face to *done*, and passes a message to neighbors on both sides to set their face to *corner*.

Figure 3-10 illustrates an example, focusing on the south faces of the blocks in the lower row. Initially (A) the leftmost is *closed*, because no block is meant to be attached at site 1; the other three are *open*, since no block has yet been attached in the row of sites 2–4. When a robot is given permission to attach at site 4 (B), the

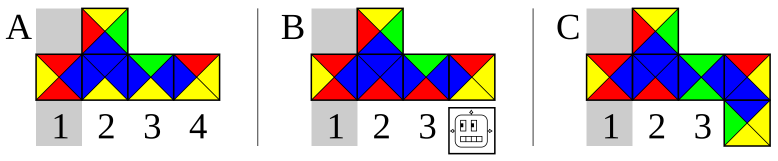


Figure 3-10: Example of communicating block algorithm. The shape map specifies that blocks are to be attached everywhere except at the two shaded sites. See text for details.

adjacent block sends a message along its row, and the blocks bordering sites 2 and 3 set their south faces to *closed*. After attachment (C), the block to which the new one was attached sets its south face to *done*, and sends a message to its neighbor to set its south face to *corner*.

A newly attached block obtains from its neighbors the shape map and its coordinates, and sets the state of its faces as follows. Define a function $f : f(\mathit{done}) = \mathit{corner}, f(\mathit{open}) = \mathit{open}, f(\{\mathit{corner},\ \mathit{closed}\}) = \mathit{closed}$. Any faces x directly attached to the structure are set to *done*. Any faces y adjacent to those faces x are set to $f(y')$, where y' is the face of the block attached to x that is the same face as y (i.e., {north, east, south, west}). The remaining face of the newly attached block, if any, is set to *open*. This strategy allows new blocks to set their states based on messages only from directly attached blocks.

In Figure 3-10C, the new block sets its north face to *done*, because of the attachment there; its west face to *corner* ($= f(\mathit{done})$), because there is a block to the north whose west face is *done*; its east face to *open* ($= f(\mathit{open})$), because there is a block to the north whose east face is *open*; and its south face to *open*, because no blocks are attached to its east or west faces.

Communication over distances further than a single block is only necessary when the first block is being attached in a new (*open*) row. Thereafter, permission for further attachment in that row can be given or denied without communication, based on local state only; and communication when further blocks are attached need not involve messages passed beyond immediate neighbors. As Figure 3-11 suggests, long *open* rows are rare and communication beyond immediate neighbors is seldom needed in practice. The total number of inter-block messages during construction experimentally scales linearly with the number of blocks in the structure.

Block power could be supplied centrally from the marker, or self-contained in each block. Power on the order of 40 mW total per block would be sufficient, using, e.g., Chipcon CC1000 chips for wireless communication with perimeter-following robots, and RS232 serial connections for inter-block communication (Table 3.1). Robot movement will be slow compared with processing and information exchange, and communication correspondingly infrequent. Thus power consumption can easily be reduced by having processors in a low-power standby mode most of the time, with the communication devices listening for incoming messages and waking their processor when necessary. One AA battery ($\sim$ 2000 mAh) would then power a block for about 50 hours.

Element	Current	Power
Processor	4 mA	13.2 mW
RS-232 port	1.5 mA	4.95 mW (×4)
CC1000 chip	7.4 mA	24.4 mW

Table 3.1: Sample power requirements for communicating blocks. All elements may operate at 3.3 volts.

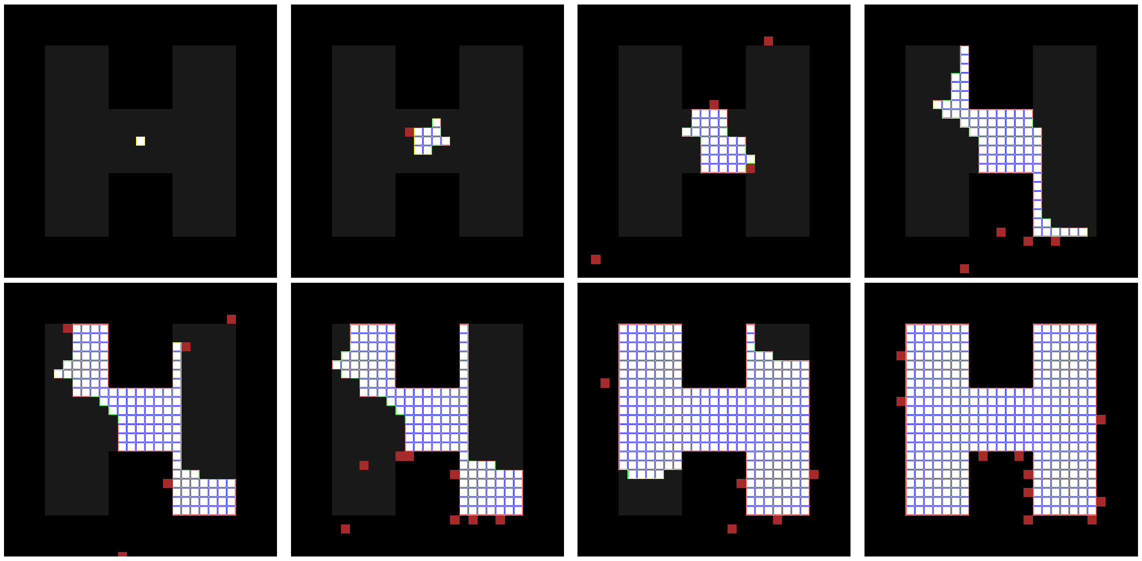


Figure 3-11: Simulated construction of a sample structure of communicating blocks, showing successive snapshots during the process of construction by ten robots. White: blocks; brown: robots carrying blocks; gray: empty cells where blocks should be attached; black: empty cells that should be left empty. Block side states are shown in the same colors as in Figures 3-9 and 3-10. The marker is in the center.

The memory and communication requirements with communicating blocks are these:

	Robots	**Blocks**
Static state	none	shape map
Dynamic state	none	side state (two bits per side); coordinates (two integers)
Communication	none	*with other blocks:* the shape map plus a few bits upon initial attachment, a few bits thereafter; *with robots:* one bit to indicate allowed or disallowed attachment

With this algorithm, blocks need to be able to indicate to robots where attachment is allowed.

		Inert	Distinct	Writable	Communicating
Blocks	State	None	Static label	Dynamic (C)	S, C, M
	Comm.	No	No	No	Yes
Robots	State	F, M	F, M, L	F, M	None
	Comm.	No	No	No	No

Table 3.2: Summary table of memory and communication capabilities needed for robots and blocks for each of the four variant approaches discussed. State abbreviations and memory required (font size in table reflects relative amount of memory):
M = shape map: memory required depends on size and complexity of desired structure; does not change during construction
C = coordinates: two integers
S = side state: a few bits per side
F = *seen-row-start* flag: one bit
L = label map: memory required depends on size of desired structure

3.2.6 Considerations for multiple robots

The algorithms described above for one robot will work nearly unchanged with multiple robots. Each robot acts in such a way that further consistent actions will lead to the successful completion of the structure. It makes no difference whether those further actions are taken by the same robot, or by other robots. However, there are a few considerations to be addressed when multiple robots are active.

General considerations

Interference: A swarm of robots can physically interfere with each other, getting in one another's way, and as a result slowing the course of construction because of the time they effectively have to spend avoiding each other rather than building. In the worst case, gridlock can occur, for instance if robots end up all around the structure perimeter such that none is able to move. Some scheme is necessary for resolving this problem when it occurs. Section 4.3 demonstrates the effects of interference, and discusses centralized and decentralized approaches to mitigating the problem.

Attachment delays: While a robot is in the process of attaching a block, it acts as an obstacle along the structure perimeter. Another robot that approaches that site during its perimeter traversal will have to respond somehow. The easiest approach is simply to wait for the first robot to finish, and then continue. This is the approach used here. However, this approach could lead to long backups in some circumstances, depending on the number of robots at work, the number of sites typically available for block attachment, the length of time it takes to attach a block as compared to the size of the structure and robot speed, etc. In some cases, it may be desirable to develop a scheme to let robots go around others, to keep the construction process moving.

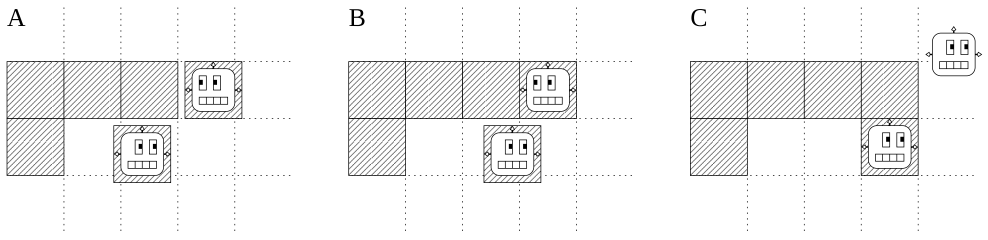


Figure 3-12: Potential complication when one robot follows another very closely. (A) The lower robot registers an end-of-row. (B) The upper robot attaches a block extending that row. (C) The lower robot registers the new end-of-row, and decides to attach a block at that site, inappropriately.

Noncommunicating blocks

In any of the three approaches using noncommunicating blocks, if all robots are consistent in following the scheme of Algorithms 1–3, then the actions of multiple robots will not conflict: Since robots all circle counterclockwise, and only start new rows at their counterclockwise end, it's not possible for separated blocks to wind up attached.

The only potential complication is if one robot is following another very closely; then, as illustrated in Figure 3-12, it may be possible for a violation of the separation rule to occur. While this scenario may be very unlikely to occur in practice with real robots, some scheme for preventing the situation is strictly necessary to guarantee correct operation. For instance, when a robot attaches a block, it may send out a short-range signal instructing any very nearby robots to reset their *seen-row-start* flags to false.

Distinct blocks

As described earlier, with many robots adding blocks in parallel, each can encounter blocks at the structure whose labels it has not previously seen. Thus upon reaching the structure, a robot follows the perimeter, keeping track of block labels along the way. It can add those labels to its label map once it encounters a block whose position it knows. As it goes on to search for a legal attachment site, it can continue to update its label map with any other unknown labels encountered along the way. Algorithm 2 includes these considerations.

With the marker located in the middle of the structure rather than along an edge, it is possible for a robot to return to the structure to find no blocks it recognizes along the entire perimeter. In such a case, this robot may not be able to contribute any further to construction. However, it is impossible for all robots to be lost in this way, since some robot(s) must have placed the blocks that form the perimeter; thus ultimate completion of the structure is not at risk. This form of robot loss can be avoided by requiring the marker to be along an edge of the desired structure, or, if communication is permitted, by having robots directly communicate map information to one another when they meet.

Writable blocks

Because every block label stores position information, there is no risk of robot loss through not recognizing any labels around the perimeter. Further, if blocks are rearranged or replaced during the course of construction (as may occur, e.g., if error correction becomes necessary (Chapter 8)), robots operating with statically-labeled blocks could encounter conflicts between their label maps and observations, whereas dynamically-labeled blocks will simply be rewritten.

Communicating blocks

Finite message propagation speed means that before a robot is given permission to attach a block in an *open* row, the structure must ensure that no other robot is being given permission at the same time elsewhere along the row. Thus when a robot requests permission to attach at an *open* site, communication along the structure row is necessary, to notify the rest of the row of the intent to attach, and to obtain confirmation that permission will not be granted elsewhere. Collisions between messages corresponding to simultaneous attachment requests must be resolved. Only then does the structure block give the robot permission to proceed, and send the message down the row to set faces to *closed*.

This consultation with the row is only necessary for the first block attached in an *open* row; thereafter the previously *open* sides in the row take on other states, and can allow or forbid robots to attach without reference to any other blocks.

An example of a full algorithm which handles these considerations is as follows (Algorithm 5; Figure 3-13). The set of block side states is augmented to include an additional state *waiting*, indicating that attachment in the adjoining row has been requested but not yet granted; $f(\textit{waiting}) = \textit{waiting}$. Messages, between attached blocks or between blocks and robots, are members of the set {query-from-robot, request, ok, no, cancel, attaching}. Messages have associated with them a block *side* to which they refer (e.g., {N, S, E, W}), and a *location* identifying the block to which a robot wants to attach. This side and location information associates together a set of messages related to establishing whether a given block can be attached.

To ensure that only one robot in a row receives permission to attach, a block sends a 'request' message in both directions (if applicable) down the row, and waits to receive 'ok' messages in return before granting permission. The *waiting* state temporarily forbids attachment, and will be replaced with *closed* if permission is granted (an 'attaching' message conveys this information) or *open* if not (a 'cancel' message conveys this case).

An alternative to this more complicated approach is to forgo all of this additional mechanism for preventing errors from ever arising, and instead to accept the possibility of occasional errors, which robots can detect and correct. Two separated blocks mistakenly attached in the same row can quickly determine that fact, and one can indicate to passing robots that it needs to be removed, while forbidding additional attachment to it. Error correction is discussed further in Chapter 8.

Algorithm 5 Message-passing for communicating blocks, to ensure only one robot at a time receives permission to attach in a given row. This algorithm gives the procedure a block should follow upon receiving a message {type, side, location}. Queries from robots have 'side' matching the side of the block where the robot is located.
The direction a message was traveling in can be determined, when required, from the location associated with the message and the block's own location.
A given block may also make reference to the states of the sides of blocks attached to it, information which may be obtained through additional block-to-block communication not explicitly described here.

```
switch type
  case query-from-robot:
    if (state(side) = waiting) or (state(side) = closed) then
      send robot away
    else if state(side) = corner then
      give robot permission to attach to side
    else if state(side) = open then
      state(side) ← waiting, waitloc(side) ← here
      send message {request, side, here} in both directions parallel to side
      {A lack of blocks in either direction is equivalent to receiving an 'ok' from that direction.}
  case request:
    if state(side) = open then
      state(side) ← waiting, waitloc(side) ← location
      if no further blocks in the direction of the message then
        send message {ok, side, location} back the other way
      else
        send message {request, side, location} in the original direction
    else
      send message {no, side, location} back the other way
  case cancel:
    send message {cancel, side, location} in the original direction
    if (waitloc(side) = location) and (state(side) = waiting) then
      state(side) ← open
  case ok:
    if (state(side) = waiting) then
      if location = here then
        if this is the second 'ok' to return to the original requesting block then
          send message {attaching, side, here} in both directions parallel to side
          give robot permission to attach to side
      else
        send message {ok, side, location} in the original direction
  case no:
    if state(side) = waiting then
      if (location = here) then
        send message {cancel, side, location} in both directions parallel to side
        send robot away
        state(side) ← open
      else
        send message {no, side, location} in the original direction
  case attaching:
    if (not ((state(side) = done) or (state(side) = corner))) and (waitloc(side) = location) then
      state(side) ← closed
    send message {attaching, side, location} in the original direction
```

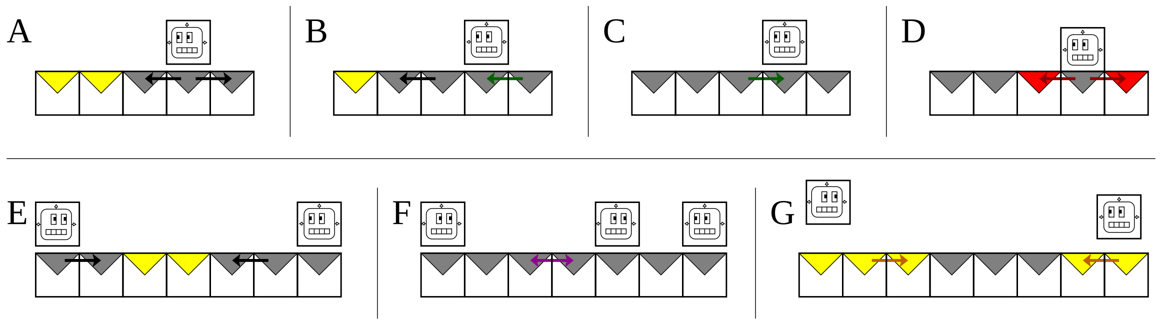
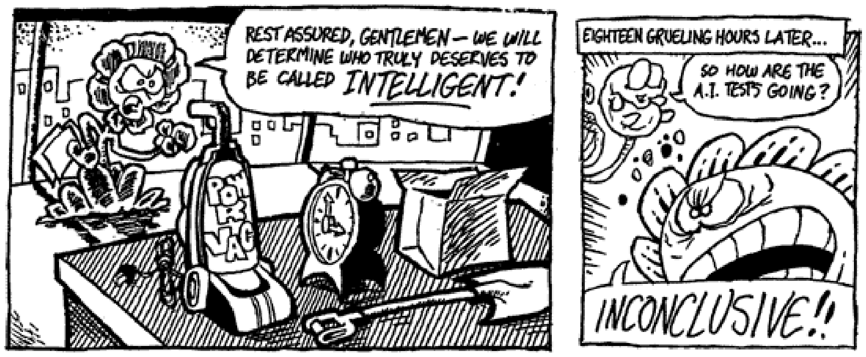


Figure 3-13: Example of extended communicating block algorithm for simultaneously active robots (Algorithm 5). Side states are shown only for the top side. *waiting* sides are shown in gray; messages are shown as arrows.
A: A robot approaches an initially *open* row with a 'query-from-robot'. The approached block sends 'request's in both directions; *open* blocks receiving the message switch to *waiting* and pass the message on.
B: The rightmost block has no further neighbors to pass the message to, and sends an 'ok' back.
C: Eventually a second 'ok' reaches the initially approached block.
D: That block gives the robot permission to attach, and sends 'attaching' messages in both directions. *waiting* blocks receiving the message switch to *closed* and pass the message on.
E: A different situation. Two robots approaching the same *open* row at about the same time will give rise to competing 'request's traveling down the same row.
F: With the (relatively) simple algorithm given, the conflict will result in 'no' messages being sent in both directions when the 'request's meet. A scheme for giving one request precedence could be used to avoid turning both robots away. Also shown: if another robot (middle) approaches a *waiting* row before other attachment queries have been resolved, it will be turned away.
G: Upon receiving the 'no' messages, the original blocks turn away the two robots and send 'cancel' messages down the row, returning all blocks to their original *open* state.

From *Bob the Angry Flower*
©1999 Stephen Notley
www.angryflower.com
Used with permission.


REST ASSURED, GENTLEMEN — WE WILL DETERMINE WHO TRULY DESERVES TO BE CALLED INTELLIGENT!
POW R VAC
EIGHTEEN GRUELING HOURS LATER...
SO HOW ARE THE A.I. TESTS GOING?
INCONCLUSIVE!!

Chapter 4

Experiments and analysis

In the last chapter I described several possible variant approaches to the construction problem, in which building blocks have different capabilities. This chapter compares them, in several respects.

In general, the more elaborate the capabilities put into the blocks, the more unnecessary robot action the system can avoid, the better advantage it can take of a greater number of robots, and the more robust it is to robot errors. Conversely, more elaborate capabilities mean increased system expense and potentially reduced block robustness.

Section 4.1 gives a brief comparison discussing qualitative similarities and differences between the variants. Section 4.2 investigates quantitative differences in construction speed and opportunity to exploit parallelism. Section 4.3 discusses approaches to reduce problems of interference between robots.

4.1 Qualitative comparison

The four variant approaches to construction described in the last chapter are similar in many ways.

- Agents use a fixed set of simple control rules regardless of the desired structure.
- There are no 'leader' robots with special roles that need to be elected or maintained.
- The task is executed by many robots running the same control rules asynchronously and in parallel.

- The robots act independently of one another; a robot does not maintain any state regarding other robots in the system.
- Robots do not need to maintain any explicit multi-hop communication structure; instead, the partial structure provides a coordination mechanism.

As a result of these properties, the algorithms automatically adapt to unexpected delays and to varying numbers of robots, which can be removed or added during the course of construction. However, they have differences in particular with respect to robustness, cost, and speed.

Robustness

With inert blocks, robots must keep track of their movement after passing the single landmark represented by the marker, over potentially long distances for large structures. The grid of blocks they move along is a crucial reference in this task. However, if a robot does miscount its movements somehow, it will be misaligned with the structure coordinate system and may attach a block at a site meant to be left empty. If a robot drifts away from the perimeter, then after regaining it, it will need to travel all the way back around to the marker to ensure knowing its location correctly. By contrast, with labeled or communicating blocks, location references are available throughout the structure—with writable labels or communicating blocks, available at every step—which can be used to correct such errors. Moreover, with communicating blocks, robots need not be able to count blocks as they pass them nor recognize geometric features of the structure; this greater simplicity can mean fewer failure points in the robots.

Communicating blocks have a different robustness problem. Because of the increased complexity, each block becomes a more likely failure point (though failure may be less likely than in robots because no additional mobility or actuation is involved). For large structures, failures will occur even with very high reliability components. It is possible to recover from block failures (section 8.2), but if the components are simpler, failure will occur less often. Noncommunicating blocks are more robust in this respect.

Cost

Among noncommunicating blocks, labeled ones will be more expensive in terms of materials and fabrication, and robots will require the additional capabilities necessary to interact with the labels. Depending on the implementation, e.g., using RFIDs, these costs need not be great: At present, the cost of RFID tags is on the order of $0.50 each, and of transceivers, $100; manufacturers argue that these costs could fall an order of magnitude or more in the foreseeable future.

Blocks capable of communication will in turn be more expensive than noncommunicating ones. The relative expense will depend in part on the application; if this approach is applied to structures assembled out of high-level prefabricated units, the relative additional cost of communication capabilities will be less significant than for structures assembled out of bricks.

Overall, work on pervasive computation [27], modular robots [50], and RFID technology for ubiquitous labeling [41] is reducing the costs of components that could be used for systems like those described here. Already significant progress has been made, and costs will decrease further as use of these technologies becomes increasingly widespread.

Speed

The respect in which the four variants can be most thoroughly compared (as I do quantitatively in the next section) is in how they affect construction speed. This will be a function of both (1) how much traveling robots need to do to collect enough nonlocal information to make a decision about whether to attach a block, both to establish position and to verify the separation rule; and (2) how many places there are to attach blocks at any given time. The latter affects both how far robots would have to travel to reach such a site even if they had all the nonlocal information they could use, and how much work there is that robots could be doing in parallel at once.

On average, construction with each successive algorithm (using inert, distinct, writable, or communicating blocks) should build any given structure as fast or faster than with the previous. In each successive case, robots are eligible to attach blocks sooner upon reaching the structure, and need not travel as far along the perimeter—with inert blocks, they must first reach the marker; with static labels, they must reach a known block; with writable labels, any block will do; with communicating blocks, robots are eligible to attach immediately upon reaching the structure, without needing to survey a full row first to ensure that no distant blocks have been attached. With any but inert blocks, the marker may be located in the middle of the desired structure, so that construction can proceed on all sides. With communicating blocks, new rows may be started anywhere, not just at the counterclockwise end, and can be extended in both directions. These factors will also tend to result in more sites where block attachment is permitted at one time, so that the parallelism of the group of robots can be better exploited.

In the next section, I investigate these issues through both analysis and simulation. In both cases, I make the assumption that robots and blocks occupy discrete positions on a grid corresponding to the occupancy matrix, as noted earlier. The consequences of relaxing this assumption, so that robots can have continuous-valued positions, are discussed in section 4.2.4.

4.2 Quantitative comparisons

4.2.1 Total distance traveled along perimeter

In general, the time taken to build a given structure will be a function of a number of factors:

Block algorithm	Best case	Worst case	Average case
Inert	$2n^3 - n - 1$	$5n^3 + 3n^2 - 4n - 5$	$\sim (2.99 \pm 0.04)n^{2.986\pm0.003}$
Distinct	$n^2 + n - 1$	$O(n^3)$	$\sim (1.4 \pm 0.2)n^{2.90\pm0.03}$
Writable	$n^2 + n - 1$	$O(n^3)$	$\sim (0.99 \pm 0.06)n^{2.954\pm0.014}$
Communicating	0	$O(n^3)$	$\sim (1.3 \pm 0.4)n^{2.56\pm0.08}$

Table 4.1: Summary of best, worst, and average cases for total number of steps $D'(n)$ that robots must take along the structure perimeter when building an $n \times n$ square.

- the size and shape of the structure;
- N, the number of robots;
- L, the time required for a robot to fetch a block and bring it to the structure;
- D, the distance a robot needs to travel along the perimeter after reaching the structure before it finds a valid attachment site;
- A, the time required for a robot to attach a block once it has found a valid attachment site.

To the extent that interference between robots can be avoided, N robots will complete the structure N times faster than one robot. However, as robot density in the workspace increases, there will be interference as robots have to maneuver to avoid each other, as the limit is reached as to the amount of work that can be done at one time, and so on. Interference is neglected for now, and considered in section 4.3.

L will typically be task-specific (depending on factors like the layout of the workspace and locations of building material), and A implementation-specific. D, by contrast, will depend primarily on the algorithm used, and so I focus on that quantity in comparing the three variants' performance. I consider a fixed number of robots building an $n \times n$ square, and investigate the total distance along the perimeter $D' = \Sigma D$ traveled by all robots during construction, measured in units of block length. This metric reflects the amount of time and effort robots spend in the vicinity of the structure, searching for a place to attach a block.

I first consider the theoretical limits on the best and worst cases for how D' can scale with the size of the structure, depending on where along the perimeter robots happen to reach the structure during the course of construction. The number of robots does not affect these limits, to the extent that interference can be neglected so that robots can be considered to reach the structure anywhere around its perimeter.

With inert blocks, the best case occurs if the robot always hits the structure adjacent to the labeled side of the marker. Because robots only become eligible to attach blocks after passing that landmark, the structure grows in a very stereotyped way, and an exact expression for the number of steps can be written. The expression is $O(n^3)$, because a robot typically has to travel the length of a side (n) to find a place to attach a block, for each of n^2 blocks. The worst case comes if the robot always hits the structure just past the landmark: then in addition to that same distance it needs to travel to find an allowed attachment site, it must first circle the entire structure to find the landmark. That adds an additional $O(n^3)$ steps to D'.

With labeled blocks (distinct or writable), the best case for $D'(n)$ is $O(n^2)$: New rows have to be started n times, and for each one, a robot has to travel distance $O(n)$ to survey the whole row to verify it's empty. However, additional blocks in existing rows can be attached with no travel required, if the robot always happens to get lucky and reach the structure at an inside corner. The worst case, again, is $O(n^3)$.[1]

With communicating blocks, the best possible case for $D'(n)$ is 0: It is possible for robots always to randomly end up on the perimeter at sites where blocks may be attached, so that no travel along the perimeter is ever necessary. The worst case is $O(n^3)$: if one full row of the structure is built first, and robots are especially unlucky about where they reach the structure thereafter, they may have to travel a distance $O(n)$ for each of the $O(n^2)$ blocks they attach. Worse than $O(n^3)$ is not possible for a square structure: A robot can never have to travel further than $O(n)$ to attach any block, and there are only n^2 blocks.

In addition to the best and worst cases for scaling of $D'(n)$, the average case is of interest, and can be measured experimentally. In simulation experiments, block-carrying robots are placed randomly on the circumference of a large circle surrounding the marker, and move inward until they reach the perimeter. After a robot attaches a block, it moves instantaneously back to that surrounding circle, and continues with another block. For structures of side lengths {10, 20, 30, 40, 50, 100, 200} and 10 robots, experimental results for D' appear in Figure 4-1 and in the table. Most data points are based on 100 independent runs; data points for noncommunicating blocks with $n \geq 100$, and all blocks with $n = 200$, are based on 10 independent runs.

With all three varieties of noncommunicating blocks, average scaling is close to the worst case. The multiplicative factor is considerably better for labeled than for inert blocks; while both scale similarly with structure size, the former is consistently two to three times faster. The performance of the two sorts of labeled blocks (distinct and writable) is similar. With communicating blocks, the average scaling behavior is significantly better than for the other variants, though still far from the theoretical best case. Overall, in the range considered, robots using inert blocks travel an order of magnitude further along the structure than those using communicating blocks, with the labeled approaches falling in between.

Table 4.1 summarizes these results.

[1]With distinct blocks, as discussed in §3.2.6, it is possible for a subset of robots to end up unable to contribute further to construction due to a lack of knowledge about any blocks along the perimeter. In the experiments described, a small fraction of runs (on the order of 10%) see robots getting 'lost' in this way. Counting their movement increases the total D' of the system severalfold. The resulting distribution of D' across independent runs has a long tail that throws off the statistics for scaling of D' with n. For that reason, the movement of those robots is omitted from the results shown in Table 4.1 and Figure 4-1. However, that occasional large negative contribution to the system performance is important to note.

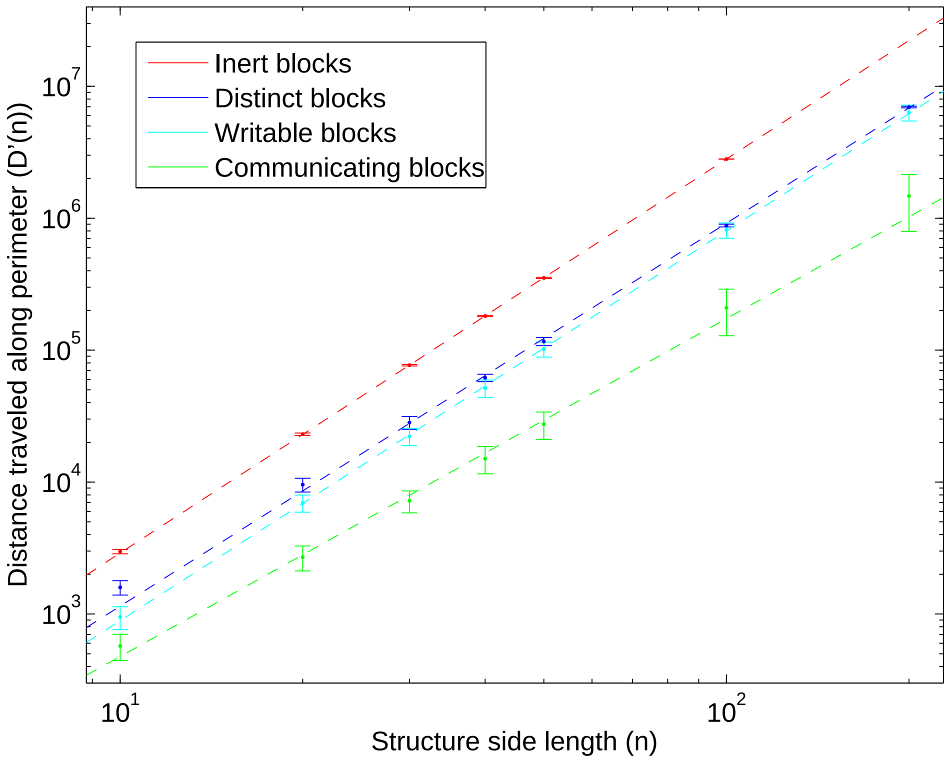


Figure 4-1: Total distance (in units of block length) traveled by all robots along structure perimeter during construction, as a function of side length of square structures.

4.2.2 Opportunity to exploit parallelism

At any stage of construction, there will be some number of simultaneously eligible sites where robots could attach blocks. That number reflects the shape the structure grows in; and it affects how much there is for robots to do at any given time, that is, the extent to which the parallelism of a swarm could be exploited.

Figure 4-2 shows the maximum number of available sites over the course of a run, for the experiments described in section 4.2.1 above with ten robots building square structures of varying side length. The very stereotyped way that structures grow with Algorithm 1 for inert blocks means that only one site is ever admissible for attachment at a time, no matter the size of the structure being built. Labeled blocks (with either read-only or read/write labels) allow slightly more parallelism: not being limited to a single landmark to establish position allows robots to work on all four sides of the structure at once. Still, not more than a few sites can be simultaneously available, regardless of the eventual size of the structure. With communicating blocks, the maximum number of sites simultaneously available for attachment is much greater still, so that robots can find one more readily and more robots can attach at once. Moreover, this number grows with the size of the structure, contributing to the shorter travel distances observed.

The reason can be seen by considering the following. If N is taken to be arbitrarily

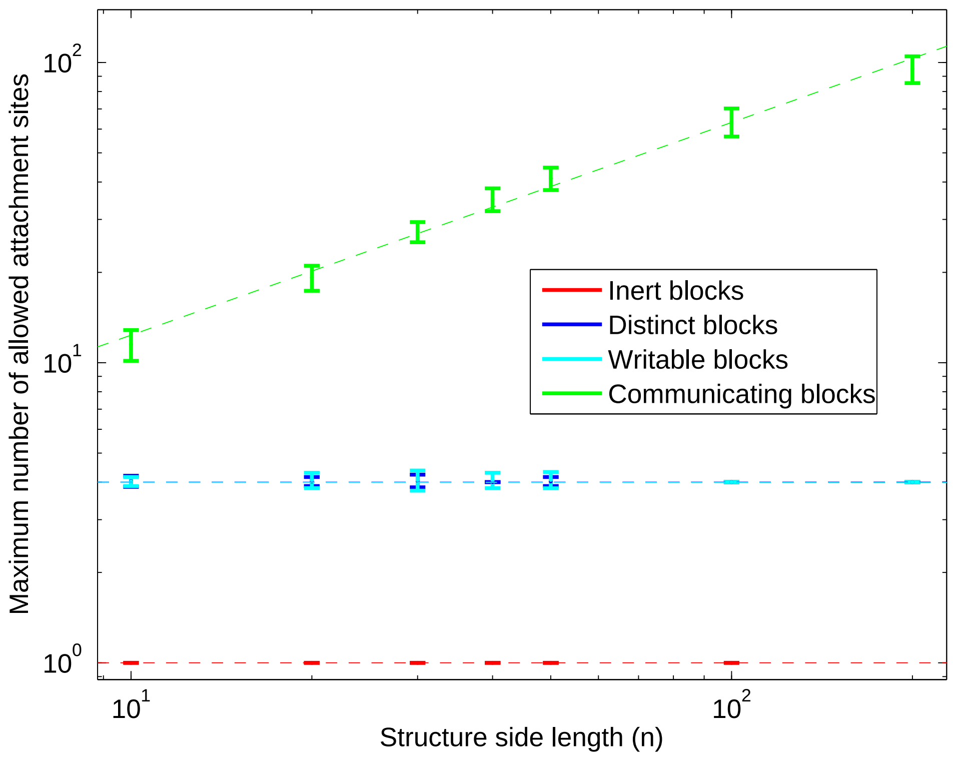


Figure 4-2: Maximum number of sites during construction where robots could simultaneously attach blocks.

large, and interference between robots is ignored, some robot will always be present to attach a block at any allowed site. Construction can then be seen as taking place in a series of steps of length A, at each of which blocks are attached at all allowed sites. Figure 4-3 shows that with writable blocks, there will be four allowed sites at each step, since robots must first pass through an end-of-row site before being allowed to attach a block at another end-of-row site. The time required to attach n blocks thus scales like nA. With communicating blocks, the number of allowed sites can increase linearly with the step number (along with the perimeter), if growth is not limited by the structure design. At step t, then, $O(t)$ blocks will be attached, so that the time to attach n blocks scales like $\sqrt{n}A$.

4.2.3 Theoretical lower bounds on construction time

Because not every site is available for attachment at all times, a limit exists to how quickly a structure could be built, even if there were an optimal centralized plan and an unlimited number of non-interfering robots. The minimum possible number of steps it could take to assemble any given structure, in this idealized case where construction progresses in a series of steps of length A, is equal to the longest shortest distance from the marker. That is, starting at any block in the desired final structure and taking one-cell hops to adjacent structure blocks, there is some minimum number

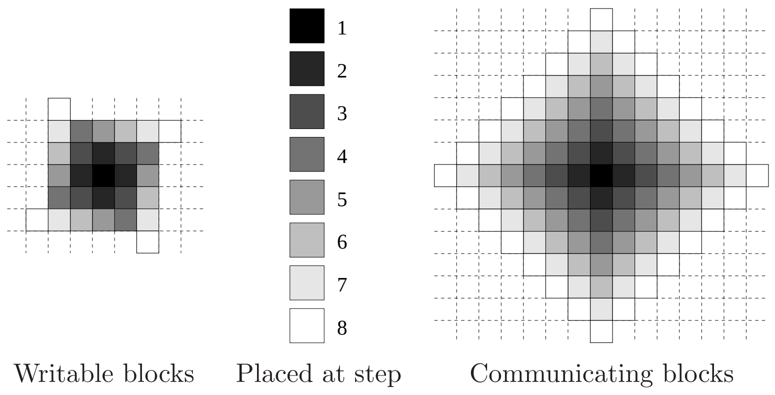


Figure 4-3: If the number of robots N is arbitrarily large, then at the end of each step of length A, we can assume that blocks will be attached at every site where it is legal to do so. In these diagrams, blocks attached at successive steps are shown successively lighter. Left, writable blocks; right, communicating blocks.

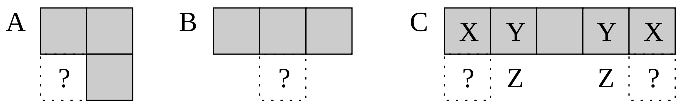


Figure 4-4: Three possible cases where blocks (marked ?) might be attached at step $t + 1$.

of hops d_{min} it takes to reach the marker; the fastest possible construction time is A times the maximum d_{min} among all blocks in the structure. This is because a block can only be attached if it has neighbors in the structure to attach it to; and an inductive approach shows that the earliest step on which any block could possibly be attached is the one equal to its shortest distance from the marker.

With communicating blocks and Algorithm 4, this minimum possible construction time for arbitrary structures can, in theory, in fact be achieved.

I now show that, with an unlimited number of non-interfering robots, the algorithm for communicating blocks results in each site that's meant to be occupied becoming occupied on the step which is equal to its shortest distance from the marker, d_{min}.

The proof is by induction. For sites adjacent to the marker, the statement is trivial.

Next consider a structure after t steps, where every perimeter block has $d_{min} \leq t$. $d_{min} < t$ is possible only if sites neighboring those blocks are intended to be left empty; otherwise blocks would already have been attached at those sites, by assumption. Any sites bordering the perimeter, which are meant to be occupied, will then have $d_{min} = t + 1$. I now show that every one of those sites will have a block attached at the next time step $(t + 1)$, without conflict.

There are three possible cases, shown in Figure 4-4.

(A) A block can be attached at an inside corner, extending two existing rows.

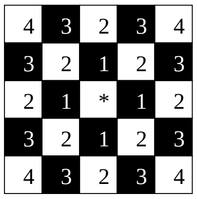


Figure 4-5: Sites are labeled with their distance from the marker (marked *). No two adjacent blocks can have the same distance.

Such an attachment cannot result in a violation of the separation rule: because a new block here starts no new rows, if there were to be a violation at the next time step, it would occur in the absence of this block as well. Thus attachment at this type of site will not result in conflict, and is always allowed.

(B) A block can start a new, previously unoccupied row. If only one such block is to be attached in that row at the next time step, then again attachment will not violate the separation rule.

(C) The only potentially problematic case, then, is if two blocks are supposed to be added in the same new row at the same time. In considering this case, I will use the following lemma:

Lemma: No two adjacent blocks can have the same distance from the marker d_{min}.

Proof: Because the marker is unique and occupies one site, the plane can be tiled like a checkerboard, where white squares are an even distance from the marker and black squares are an odd distance (Figure 4-5).

Therefore, the two blocks supposed to be added in the same new row (marked ? in Figure 4-4) must be separated by at least one site. If they are meant to be separated according to the shape map, then they can both be attached at once without conflict. A problem can then only arise if all sites between them are meant to be occupied.

The two blocks must border the structure on the same side, as in the figure; otherwise either the partial structure violates the separation rule, or the structure design has a loop in it, neither of which is allowed.

The old neighbors (marked X in the figure) of those two new blocks (marked ?) have $d_{min} = t$, by hypothesis. Since all sites between the new blocks are meant to be occupied, all sites between those old blocks must also be meant to be occupied, or else the structure design would have a loop. Further, all sites between the old blocks must already be occupied by step t, or the structure would have violated the separation rule.

The occupied site(s) adjacent to the old blocks (marked Y) must have $d_{min} = t-1$: If they had $d_{min} = t+1$, they would still be unoccupied by hypothesis; by the lemma above, they cannot have $d_{min} = t$; and if they had $d_{min} < t-1$, the old blocks (X) would have been occupied earlier, by hypothesis. Therefore the site(s) adjacent to both those and the new blocks (marked Z) have $d_{min} = t$, and thus must be occupied by hypothesis. But in that case, the new blocks to be added are not starting a new row after all. This contradiction completes the proof.

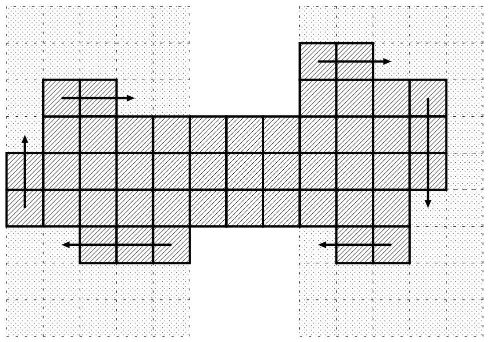


Figure 4-6: A desired structure (light shading) under construction using labeled blocks; those so far added have diagonal shading. One row per side of the structure can be simultaneously under construction at a time (arrows); each row will be completed before construction of the next layer begins. Separations caused by areas meant to be left empty, like those at top and bottom, allow rows on the same side to be treated independently.

With noncommunicating blocks, there are typically many fewer sites at any given time where blocks can be attached, as described in the previous section on parallelism. Under the assumption of discrete positions on a cellular grid, labeled blocks allow only one row per side to be simultaneously under construction. Separated rows on the same side can be under construction at the same time (Figure 4-6), but the next layer outward cannot be started until the previous row is completed. With the algorithm for inert blocks, blocks will be attached no more than one at a time in a predictable order. That will then require n attachment steps for an n-block structure, no matter how many robots there are.[2]

4.2.4 Continuous-valued positions

If the assumption of discrete locations along the structure grid is relaxed, then it may be possible for robots to reach the structure between grid sites such that they recognize an end-of-row but not an inside corner (Figure 4-7). If so, then that can allow robots building with labeled blocks to have two or more layers on the same side under construction at once. As a result, some of the results above for labeled blocks can change (results for communicating and inert blocks are not affected). The best possible case for $D'(n)$ is improved from $O(n^2)$ to $O(n)$. More than 4 sites can become available for block attachment at a time, allowing better use of swarm parallelism and faster construction—in fact, a structure could in theory be completed

[2]Chapter 3 outlined an alternative algorithm for inert blocks which does not rely on a distinct marker. With that algorithm, robots may be at any point along the structure perimeter when they establish their position. As a result, more than one site can be available for attachment at a time. However, with that algorithm, the structure must be built in such a way that the marker ends up with blocks attached to at most two of its sides. As a result, the opportunity for parallelism with that algorithm can never be better than with labeled blocks, for which the marker can be located anywhere in the desired structure.

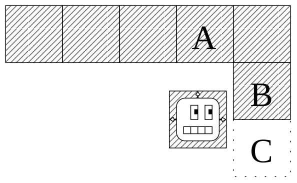


Figure 4-7: If a robot reaches a structure between grid sites in a situation like this, and its sensing is sufficiently limited that it can perceive the block at B but not at A, then it could start following the perimeter counterclockwise, register the end-of-row site as it turns the corner, and attach a block at C. Such an attachment is not possible under the assumption of discrete positions.

with labeled blocks as quickly as with communicating blocks, if robots consistently happen to come to exactly the right places at exactly the right times. However, the way the robot has to reach the structure for this case to occur is a low-probability event. Thus while the relaxation of the assumption about discrete positions can substantially improve best-case performance, it is unlikely to have a significant effect on average performance.

4.3 Interference between robots

While the parallelism of a swarm can result in considerable speedup, N robots will in general not be able to complete a structure N times faster than a single robot. One factor in that limitation is a saturation in the amount of work there is to be done at any given moment, as discussed in the previous sections. Robots added beyond that point will have nothing to contribute. This factor reflects diminishing returns in increasing the number of robots, in terms of improved construction speed.

Another factor is physical interference between robots, as they restrict one another's movement, have to maneuver to avoid each other, and so on. This factor reflects actual losses in construction speed; adding too many robots can hurt the overall performance of the system.

In section 4.3.1 I demonstrate the effects of interference in the model system. In section 4.3.2 I describe two possible strategies for reducing it, a centralized and a decentralized one.

The focus here is on interference near the structure: if it occurs anywhere in the workspace, it should occur around the structure, where robots are likely to be most densely packed (especially near the beginning of the construction process, when the structure perimeter is still short). Thus I continue to abstract away the details of what robots do away from the structure. In the experiments below, robots take L time steps to fetch a block and bring it to the structure perimeter, and A time steps to attach it. Traveling one block-length along the perimeter takes one time step. If blocked by another robot in front of it, a robot waits for the other to move on before continuing.

With noncommunicating blocks, robots can end up in a fully gridlocked situation: Because it is necessary to pass an end-of-row site before becoming eligible to attach a block, robots can end up packed around the perimeter, such that none can move nor yet may any attach a block. To keep this situation from preventing construction entirely, I treat robots that have spent 10 time steps at an end-of-row site as though they had attached a block: they are removed from their position along the perimeter and allowed to continue with the next step of fetching another block. This discontinuous removal could apply, for instance, to robots able to move freely in three dimensions (as in outer space or underwater), assembling a two-dimensional structure; such robots could leave the plane of the structure to return later without violating any physical laws.

4.3.1 Effects of interference

In this section I show some simple experiments illustrating the effects interference can have. These use $L = 1$ and $A = 1$, so that the time taken to build a structure will reflect primarily the time robots spend around the perimeter, looking for a place to put a block and potentially encountering other robots. I will discuss only writable and communicating blocks here.

Figure 4-8 shows the time required for N robots to build a $n \times n$ square structure, with $N = \{1, 2, 5, 10, 20, 50, 84, 100\}$ and $n = \{11, 21, 31\}$, for both writable and communicating blocks. Averages are over 100 independent runs.

Note in passing that construction with communicating blocks finishes much more quickly than construction with writable blocks, by more than a factor of 2 on average, for all numbers of robots tested.

For all three structure sizes, significant interference effects slowing the time to completion begin to appear with around $N \sim n$ robots. With communicating blocks, in the range for N tested, adding more robots gives diminishing returns on improved construction speed, but construction time does continue to decrease. With writable blocks, adding more robots eventually begins to slow the course of construction in an absolute sense.

4.3.2 Reducing interference

The results above caution against the use of too many robots. However, as the structure grows, an increasing number of robots can usefully contribute to construction. Even if the number of available sites for attachment fails to grow, as with noncommunicating blocks, a longer perimeter means that more robots could be searching for attachment sites at once without interfering with one another. This suggests that varying the number of robots during the course of construction could make it possible to take advantage of the parallelism of the swarm while avoiding interference.

The optimal number of robots N^*, in the sense of the largest number that can work on a structure simultaneously without causing significant slowdowns due to interference, can be estimated as follows. Consider a set of robots fetching blocks, bringing them to the structure perimeter, and circling until they find an allowed site at which to attach. In most cases, when a block is attached somewhere, the previous

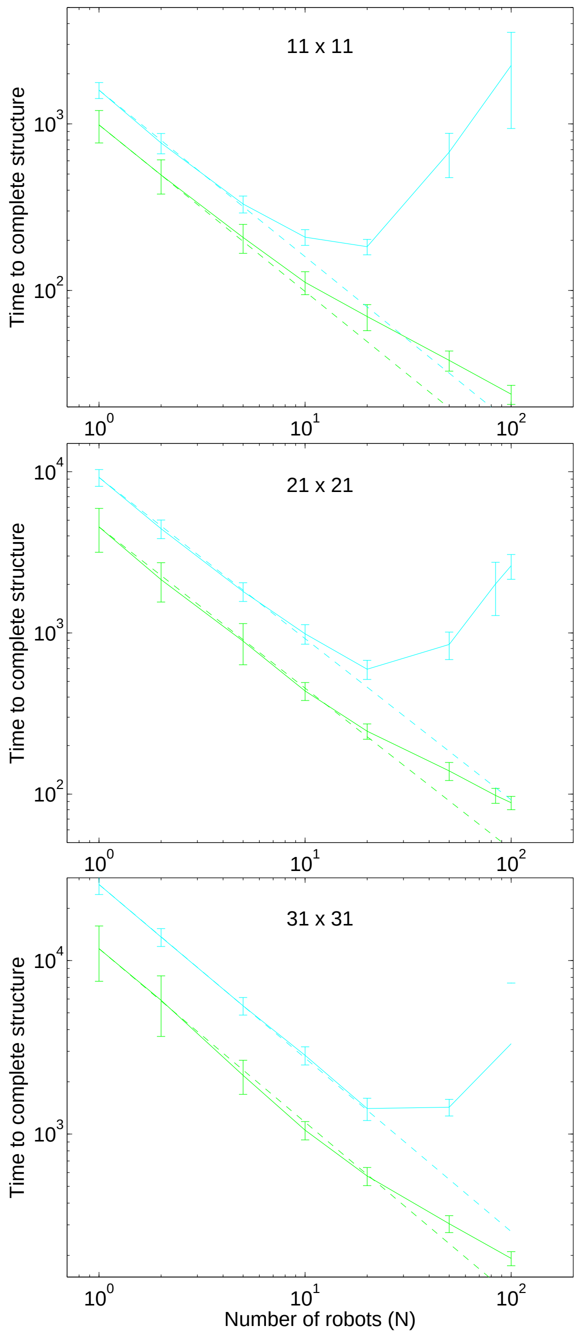


Figure 4-8: Time taken by N robots to build a $n \times n$ square structure. Blue, writable blocks; green, communicating blocks. Top, $n = 11$; middle, $n = 21$; bottom, $n = 31$. Dashed lines show how quickly completion would occur if there were no interference and N robots took $1/N$th the time taken by one robot.

site in the circuit becomes an inside corner and thus usually an allowed attachment point. Robots should thus ideally be spaced along the perimeter by a distance $A+1$, so that when one finishes attaching its block, another shows up at once to attach the next, without having to wait for the first to finish. Then $P/(A+1)$ robots will fill out a perimeter of length P evenly. Moreover, each time a robot finishes attaching a block and goes off to fetch the next, another robot can take its place along the perimeter. If $L = c(A+D)$, every robot along the perimeter can have c counterparts off fetching blocks, without interference. Therefore the prediction for the optimal number of robots is

$$N^*(t) \sim \frac{P(t)}{A+1}\left(1 + \frac{L}{A+D(t)}\right)$$

This value will increase as construction proceeds and P grows.[3]

The prediction could be used to direct the gradual addition of robots to a construction site, so that construction consistently proceeds as quickly as possible without substantial interference occurring.

Another way to address the issue, and a more natural and decentralized one, would be to use an adaptive tactic inspired by recruiting during foraging in social insects: have a pool of idle robots, which can be drawn from if more robots can usefully be added to the construction team, or increased if significant interference is occurring among the working robots. A distributed algorithm based on each robot's individual pattern of encounters has been shown to be effective in simulated foraging systems [9]. I demonstrate the suitability of a similar approach here, using writable blocks, which showed the most pronounced interference effects in the previous section.

Robots keep track of whether or not they have been able to move for each of the last t_a time steps. If that freedom of movement falls below some threshold θ_a, the robot leaves the construction area, and waits in some out-of-the-way location to avoid interference. A robot inactive in this sense has some fixed probability per unit time p_a of reactivating and rejoining the construction team. Traveling to and from the waiting area will incur some time cost T_a.

The experiments whose results are shown below use (somewhat arbitrarily) $t_a = 5, \theta_a = 0.3, p_a = 0.01$ per time step. In order to ask what is the best this approach can achieve, T_a is set to 0, so there is no switching cost between active and inactive. A pool of 100 robots assembles a square structure with side length 31 blocks. In the experiments of Figures 4-9–4-11, $L = A = 1$ as above; in those of Figure 4-12, $L = 31, A = 11$. Averages are over 100 independent runs.

Figure 4-9 shows the number of robots active as a function of time during the course of construction. That number initially decreases sharply, and then increases again slowly as the structure grows and more room around the perimeter is available for robots to occupy without interfering.

Figure 4-10 shows the fraction of sites along the structure perimeter which are occupied by robots. It can be seen that the algorithm acts to add or remove robots to maintain an equilibrium occupancy rate along the perimeter, as the perimeter

[3]While D, in the denominator of the second term, will also increase with time, it can grow no faster than P, so that $N^*(t)$ is monotonically increasing.

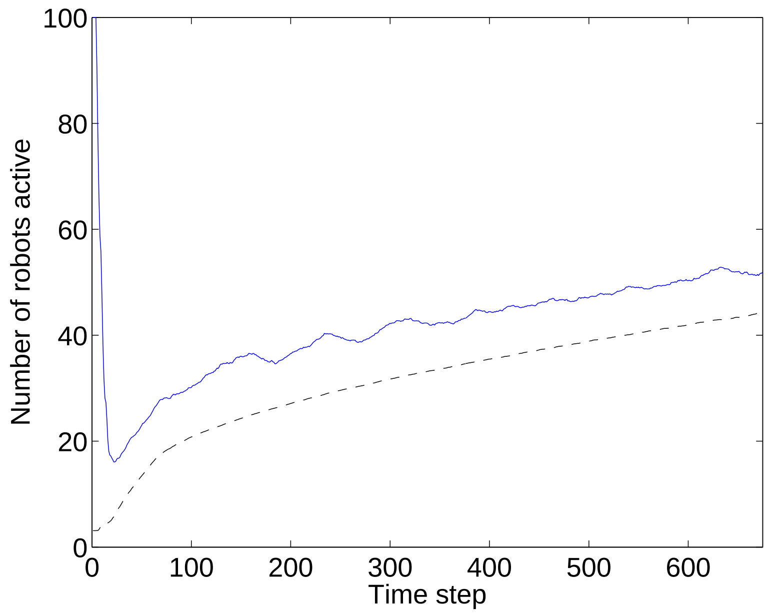


Figure 4-9: Number of robots active as a function of time with adaptive recruiting as described in the text. $L = 1, A = 1$. The dotted line shows $P/(A+1) \cdot (1+L/(A+D))$.

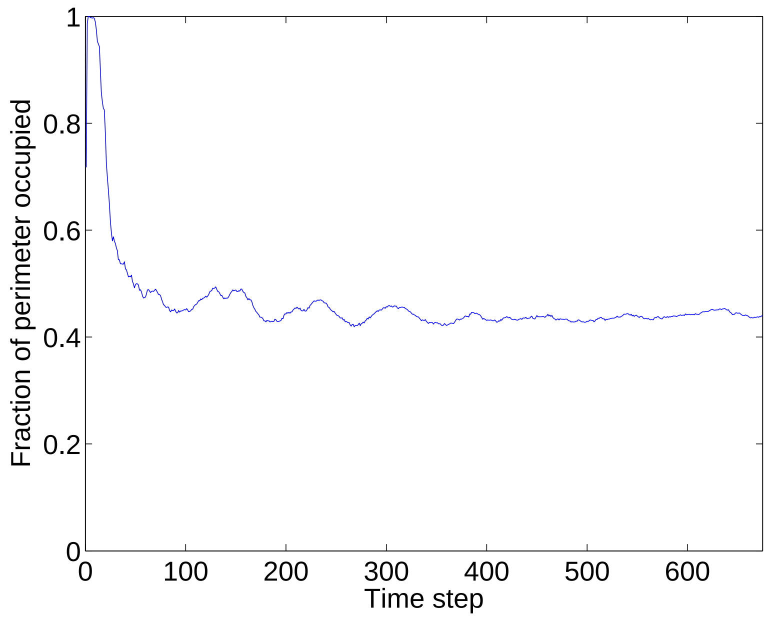


Figure 4-10: Fraction of structure perimeter occupied by robots as a function of time, with adaptive recruiting as described in the text. $L = 1, A = 1$.

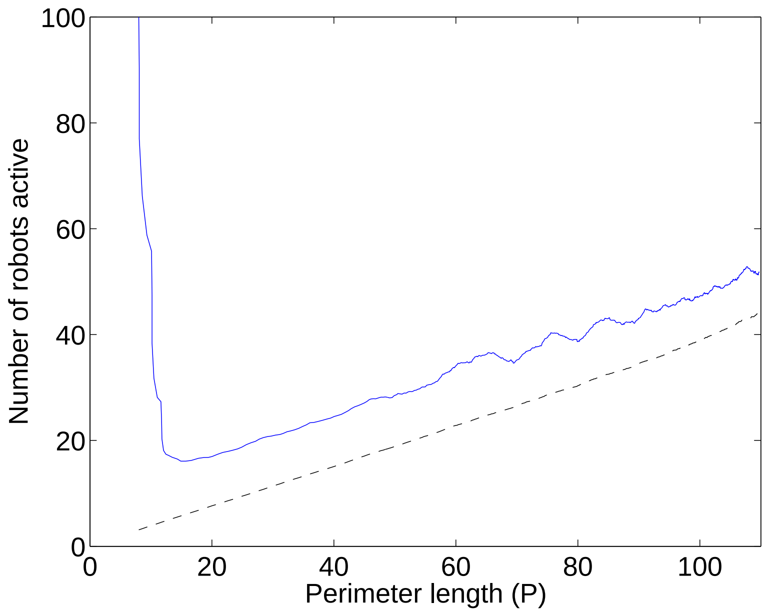


Figure 4-11: Number of active robots as a function of length of the structure perimeter, with adaptive recruiting as described in the text. The dotted line shows $P/(A+1)\cdot(1+L/(A+D))$. $L=1$, $A=1$.

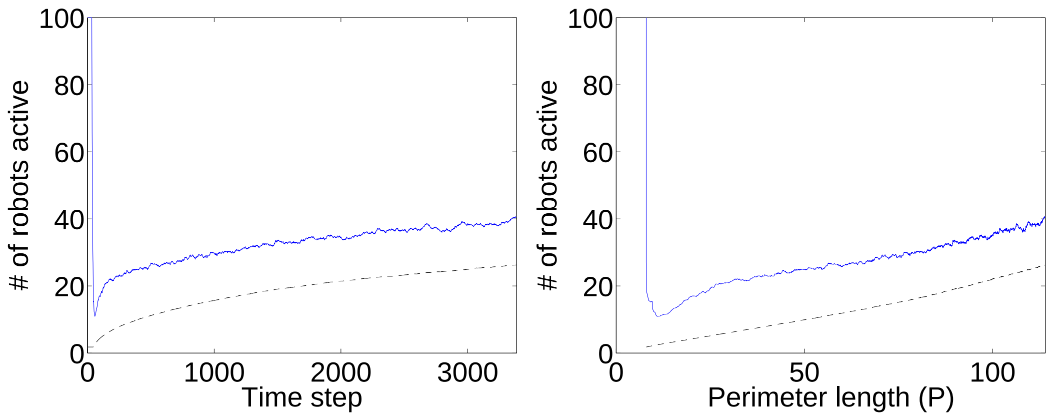


Figure 4-12: Number of active robots as a function of time (left) and of perimeter length (right), with adaptive recruiting as described in the text. The dotted line shows $P/(A+1)\cdot(1+L/(A+D))$. $L=31$, $A=11$.

grows. Figure 4-11 accordingly shows an approximately linear relationship between the length of the structure perimeter and the number of robots active.

In these experiments with $L = A = 1$, structures of writable blocks are completed faster with adaptive recruiting (1024 ± 175 time steps) than with any fixed number of robots.

Figure 4-12 shows the results of repeating these experiments with $L = 31$ and $A = 11$. For both sets of parameter values, the observed curves for $N(t)$ and $N(P)$ are consistent with the predictions for N^*, showing that this adaptive recruiting method can be effective in dynamically modifying the size of the construction team in a decentralized way.

From *Nukees*
©1998 Darren Bleuel
www.nukees.com
Used with permission.

I DUNNO. HE'S BEEN STARING AT THAT ALLEN WRENCH ALL DAY.
HELP! I'M JUST A THEORIST!

Chapter 5

Hardware

While the approaches I describe are high-level algorithmic ones, care has been taken to ground them on simple basic capabilities that can be made robust and self-correcting, crucial to a hardware realization. I have implemented a prototype system that demonstrates these key capabilities and can build arbitrary solid structures using any of the three algorithms for noncommunicating blocks (communicating blocks have not at present been implemented physically).

Section 5.1 describes the hardware. Section 5.2 gives details of the implementation of low-level capabilities. Section 5.3 evaluates its performance.

5.1 Overview

Figure 5-1 shows the hardware: a laptop controller drives an ER1 (Evolution Robotics) wheeled base and gripper, and obtains visual feedback from a CMUcam2 mounted to one side and in front, pointing downward. That robot hardware is sufficient for the use of inert blocks. For use with labeled blocks, a RightTag RFID read/write board and antenna is mounted on the left side of the robot, such that it will pass over the centers of neighboring blocks as the robot follows the perimeter.

Blocks are $8.5'' \times 8.5'' \times 1.5''$, made from sheet metal, with foam handles affixed to the top surface for the gripper to grasp. RFID tags can be mounted atop the handles to implement labeled blocks. For self-alignment, the blocks use neodymium magnets, mounted in alternating-polarity pairs on each side so that two blocks brought into proximity will be drawn together for any pair of faces. Magnets would be a poor choice for connectors in any real application—their range and strength is limited, and once connected, they are difficult to detach—but they are sufficient for this proof-of-concept demonstration.

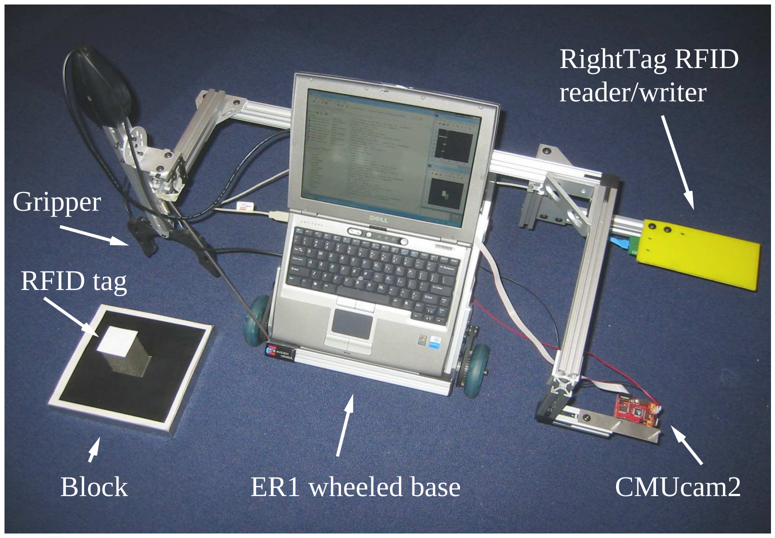


Figure 5-1: Hardware (robot and block).

The camera is configured to register white areas. Tests were conducted in various areas of MIT's Stata Center, in open spaces with blue or red carpeting. In these environments, it is sufficient to outline blocks with white borders; in a more complicated environment, a more complex approach or the use of different sensors could improve robustness.

The gripper has no vertical degree of freedom. It is mounted such that a gripped block is held above the ground with about 2″ of clearance. When the robot finds a site where a block should be attached, it maneuvers so that the held block is approximately above that site, and drops it; the magnets bring it into alignment.

The robot fetches new blocks from a single cache, implemented as a 2″ pedestal on which blocks are placed by hand to await pickup (Figure 5-2). A line on the floor nearby acts as a visual reference to guide the robot into position to pick up the block.

The robot demonstrates the key elements of the approaches with noncommunicating blocks (Figure 5-2):

- the ability to maneuver to a cache, pick up a block, and bring it to the structure;
- perimeter-following;
- recognition of grid points and sites where block attachment is valid;
- reading and writing block labels;
- attachment of blocks to form a desired final structure.

The additional key elements necessary for communicating blocks have been demonstrated in work on modular robots [50, 76].

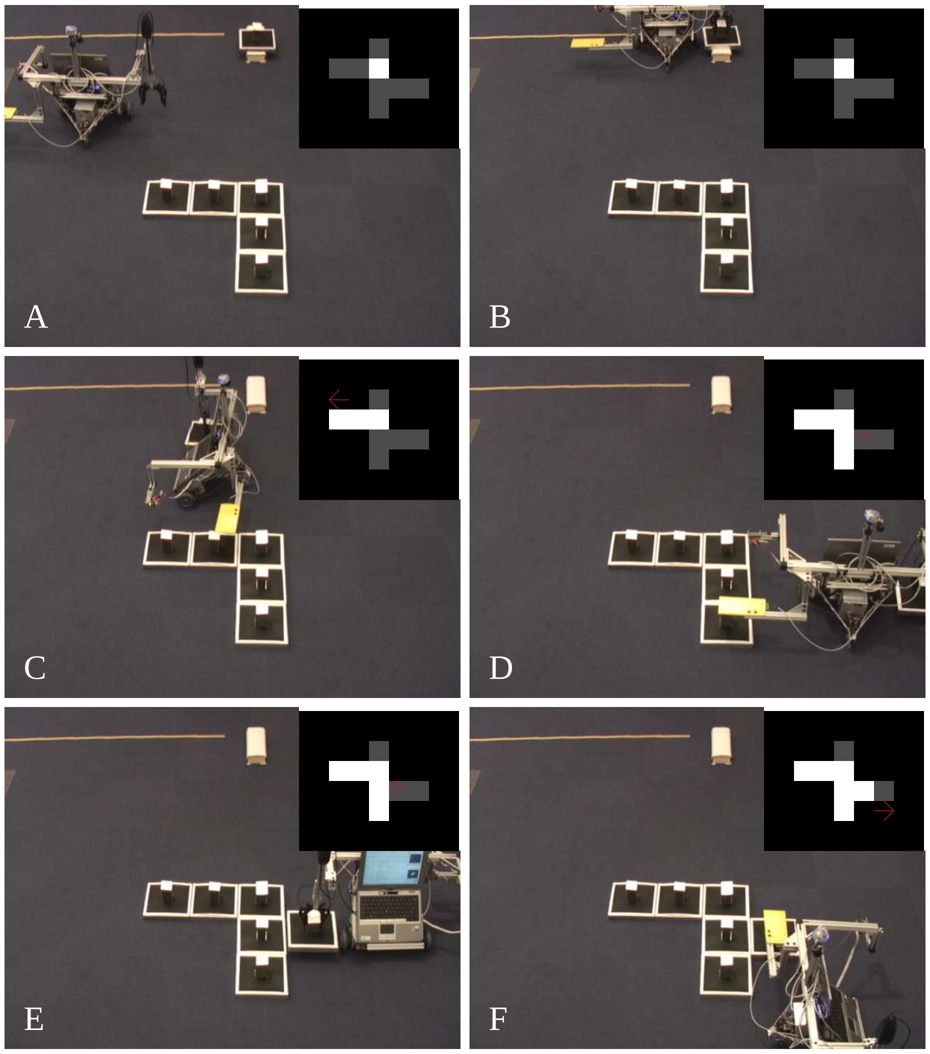


Figure 5-2: Process of adding one block to the structure, using writable blocks. The cache is at top, structure in progress at bottom, with the marker at its upper right. Inset: the robot's knowledge about the structure's progress and its own position: desired structure in gray, known existing blocks in white, robot location (if known) shown as arrow.
(A) The robot, traveling toward the cache from the vicinity of the structure, initially knows only that the marker must be present.
(B) Using the line on the floor as a reference, it maneuvers to and picks up a block from the cache.
(C) Once at the structure, it can use its RFID reader to determine its position, and its camera to follow the perimeter. Existing blocks are added to the robot's map as it observes them.
(D) Eventually the robot reaches an empty site where a block is desired, and where one may be attached according to Algorithm 3.
(E) It maneuvers to attach its block at that site, dropping it into place...
(F) ...and writes the block's new coordinates to its tag.

5.2 Implementing low-level behaviors

This section describes the routines used for perimeter-following (§5.2.1), block attachment (§5.2.2), block fetching (§5.2.3), and (with labeled blocks) interactions with block labels (§5.2.4).

The overall robot behavior in this prototype implementation is comparatively slow, particularly when the robot is moving through a featureless area searching for the structure or cache. This is in part a result of the very limited interface to the ER1 provided by Evolution Robotics. It would be desirable, for instance, to command the robot to move forward at some speed, while the camera continuously monitors the environment, and then to send a new motor command as appropriate when some feature comes into view. However, the ER1 API allows specifying only distances to travel (forward or backward, or rotations to the left or right), not translational or rotational speeds to be maintained indefinitely. Moreover, sending a new motor command has the effect of first bringing the robot to a halt, and then starting the new motor command.

For these reasons, I instead took the approach of using a series of discrete steps, in each of which the robot polls the camera, sends a motor command, and waits for that movement to be completed before continuing. Coupled with the small field of view the camera must have compared to block size (see below), the distance traveled in any given step is limited to about two inches, if no visual landmarks are available. When the robot is along the structure perimeter, block edges provide cues allowing it to take larger steps.

5.2.1 Perimeter-following

The CMUcam2 communicates with the laptop via a serial connection. This communication is too slow to allow transfer of full images (which take several seconds each). Instead, preprocessing is done onboard the camera and then only a few bytes are sent over the serial line.

The camera is configured as follows:

- *Poll mode:* The camera sends one packet at a time when requested by the computer, rather than sending information continuously.
- *Track color:* The image is thresholded such that any pixels with an RGB value within the range $\{150 \leq R \leq 255, 150 \leq G \leq 255, 0 \leq B \leq 255\}$ are considered 'active'.
- *Noise filter:* Pixels are considered on a row-by-row basis. Two consecutive active pixels are required before a third pixel is considered 'detected'. This filtering eliminates isolated pixels that do not represent real features.
- *Bounding box:* The camera sends the laptop only the coordinates of the bounding box around the set of detected pixels.

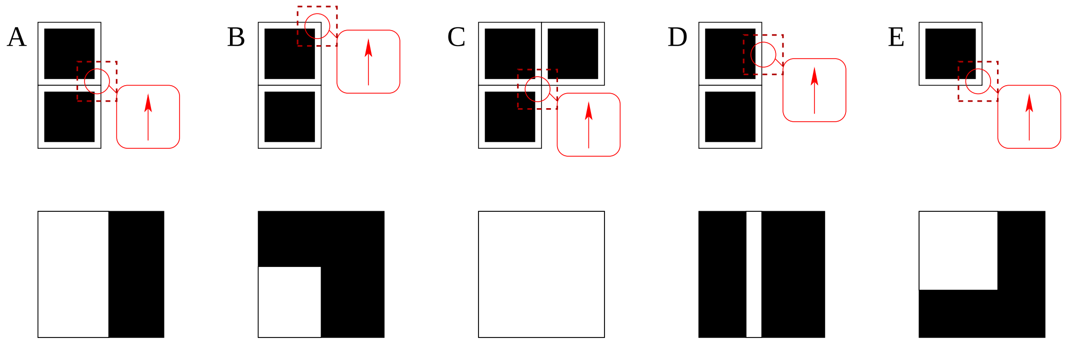


Figure 5-3: Using bounding boxes to distinguish different situations.
Top: Actual situation; boxes are black with white borders, robot is red with arrow indicating direction of travel, dotted outline indicates camera field of view.
Bottom: bounding box returned (white).
(A) At a junction between blocks, (B) at a left-turn corner, (C) at an inside corner, (D) mid-wall, (E) just finished turning a left-turn corner.

If the camera field of view is smaller than a single block, then this approach can distinguish between the various relevant possibilities for perimeter-following (Figure 5-3): (A) at a junction between blocks, (B) about to turn a corner to the left, (C) at an inside corner, (D) in the middle of a wall, or (E) just turned a corner to the left. The camera is mounted slightly ahead and to the left of the robot base so that it can recognize corners just before reaching them.

- In case (A), the robot can move ahead a distance equal to the block side length, which should put it into one of cases (A) through (C).
- In case (B), the robot can go around the corner to continue following the perimeter. The goal is, in effect, to pivot the robot around the camera. Because the robot is nonholonomic, this involves a predefined sequence of {move forward, turn left, move forward}, done blindly and with preset distances such that at the end, the robot is in situation (E) and can see the same corner of the same block from a different angle.
- In case (C), the robot can turn to the right by a similar predefined sequence of {move backward, turn right, move backward}, which is again equivalent to a rotation around the camera. With the current hardware, however, to have the effect of pivoting the robot around the camera, that last move-backward step would make the robot back up over part of the structure. Instead, then, it moves a shorter distance backward, and checks the camera to see if the wall now to its left continues far enough to be in view. If so, it can continue following that wall. If not, it infers the presence of a left-turn corner, and executes a modified left-turn sequence to put it into situation (E) at that corner (Figure 5-4).

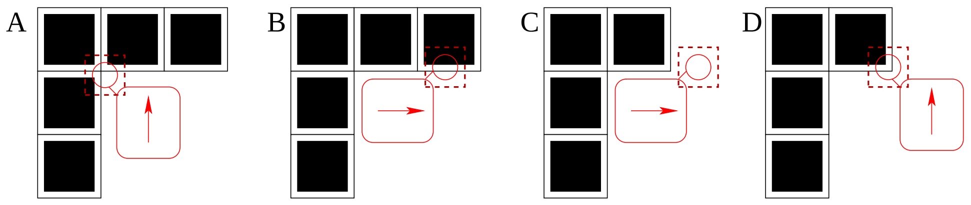


Figure 5-4: When the robot turns right at an inside corner, it may not be able to back up far enough at the end of the turn for the camera to see the structure. (A) Before the turn. (B) After the turn, if the wall now to the robot's left continues far enough, the camera can see it and wall-following can continue. (C) If the wall is not visible, the robot infers a left-turn corner and performs a modified left-turn sequence to put it at the position shown in (D).

- In case (D), the robot can take a series of small steps until it comes to the end of that block, which will put the robot into one of cases (A)–(C).
- Case (E) is functionally equivalent to case (A).

Further, the camera can be temporarily configured to return the bounding box of a set of detected pixels within a limited subwindow of the field of view. This capacity is useful in making finer adjustments to the robot's position along a wall, as described next.

Position adjustments

As discussed earlier, the robot can easily get off track in a continuous-valued world. Blocks can be used as landmarks to readjust position at any step. Readjustment is particularly important before executing multi-step operations that require the robot to move blindly, as when turning corners or attaching blocks.

The robot can adjust its angle and distance from a wall to its left by requesting a set of bounding boxes around detected pixels for thin horizontal slices of its field of view (Figure 5-5). Any slices for which a bounding box is too wide (indicating a horizontal block edge) can be discarded. Remaining slices can be used to fit a line indicating the robot's angle to the wall and its distance from it. On that basis, the robot can rotate to align itself more precisely with the wall. Again, because the robot is nonholonomic, adjusting distance from the wall must be done via a sequence of steps: rotate 90°, move forward or backward an appropriate distance, rotate 90°. When the robot faces a wall head-on (as when at an inside corner, or when first reaching the structure), it can make the equivalent adjustments using the horizontal wall by requesting a set of vertical slices.

At block junctions, a vertical slice further into the structure than the edge (Figure 5-5) will reveal how far forward or backward the robot is from perfect alignment with the junction. Similarly, just before or after left turns, such a vertical slice will tell the robot how far in its direction of travel it is from the corner, which will need to be taken into account in its next movement.

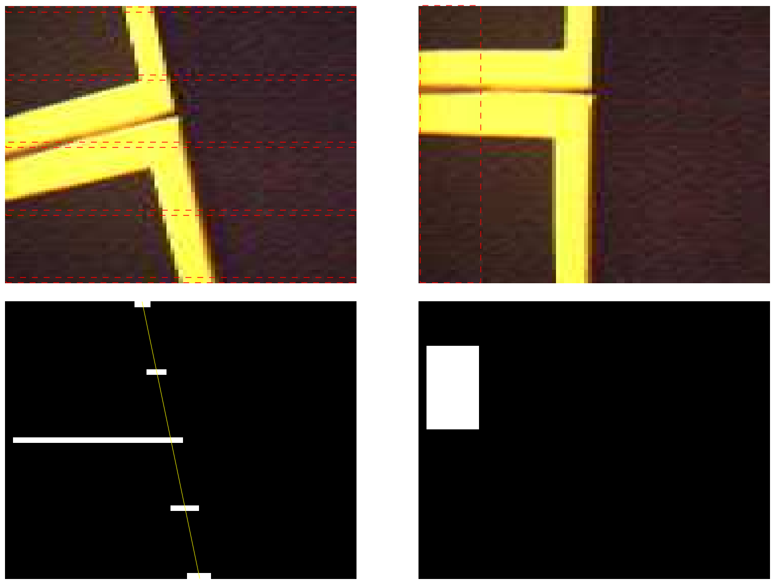


Figure 5-5: Left: Bounding boxes for horizontal slices of an image let the robot fit a line to estimate its angle from the wall. Wide bounding boxes (like the third from the top) are discarded. Right: A vertical slice within the body of the structure reveals the horizontal position of a block junction. Top, raw camera image with slices marked (dashed red lines); bottom, after processing.

5.2.2 Block attachment

There are three kinds of sites where a block might need to be attached: in the middle of a wall (Figure 5-3A), at the site before a left turn (Figure 5-3B), or at an inside corner (Figure 5-3C).

The attachment procedure for the first two cases is similar. The robot first adjusts its angle and distance from the wall, and then its distance from the junction or corner, as described above. Once in position as appropriate, it spins in place 180° and then opens its gripper to drop the block. The gripper is mounted so as to carry the block as close as possible to the wheeled base, to reduce the robot's footprint; the camera is mounted such that this 180° rotation after alignment will put the block in approximately the right location to be dropped into place. The magnets ensure precise block alignment.

The geometry prevents the robot from following the same procedure to attach a block at an inside corner. Instead, it first adjusts its position and angle with respect to both the wall to its left and the wall in front of it. Once in position, it moves {backward, left, forward} to put the block in place, and then opens the gripper.

5.2.3 Block fetching

To retrieve a new block from the cache, the robot starts in empty space (away from the structure), heading roughly in the direction of the cache and aimed anywhere to its left. It moves straight ahead in a series of short steps, checking the camera at each step. At some point it will reach the line on the floor. It can use that reference to orient itself towards the cache, adjusting its angle and direction as above. Next it follows the line along the floor until it reaches the end. That landmark triggers it to turn left and move forward a short distance.

The gripper has an infrared beam between the two ‘fingers’, and can be configured so that if that beam is broken, the gripper will close. The line, cache, and block are positioned such that when the robot takes that last step of moving forward, the gripper passes over the block so that the handle breaks the IR beam. The gripper then closes on the handle quickly enough to grasp it, although the robot’s forward movement continues. Once the gripper is fully closed, the robot backs up, rotates 180°, and starts moving forward.

The relative position of the cache and the marker is such that, if the robot moves straight ahead from its pose after obtaining a block, it will reach the marker (or, if present, a block that has been attached in front of the marker). This scheme for finding the structure is not general, but suffices for the prototype system.

The robot finds the structure the same way it finds the cache line, by taking a series of short steps while checking the camera. When it reaches the structure, it uses the perimeter as a reference to adjust its position and angle, then turns so the structure is to its left. Thereafter it can follow the perimeter as described above.

The remaining task is to return to the cache after attaching a block. Again the approach taken here is not general, but sufficient for this system. The robot uses its knowledge of the structure’s geometry and size to maneuver away from it and then to a pose where, with no structure ahead of it, it’s aimed to the left of the cache. The routine from the beginning of this section can then be used again to obtain the next block.

5.2.4 Interactions with block labels

The RFID transceiver is mounted such that when the camera is over the forward corner of one block, the transceiver is over the tag of the block behind it (Figure 5-6). This choice was made to help the robot in making right turns at inside corners; the transceiver provides an additional sensor which the robot can use to establish the presence of blocks behind those visible to the camera.

While perimeter-following, at each step where the robot is positioned with the camera at the forward corner of one block, it can use the RFID transceiver to read the coordinates of the block (if any) behind it. This lets it monitor its position continually so that, if there were an error in its discrete position estimate, it could correct it. Right and left turn sequences have an extra step in addition to the three described in §5.2.1 above, in which the robot pauses to read the position of a block as the transceiver passes its tag during the turn.

After attaching a block in the middle of a wall or at the site before a left turn, as described in §5.2.2 above, the robot next moves into position to write coordinates to the newly attached block. The position of the RFID transceiver lets it do this by a motion sequence {move forward, turn left, move backward}. At the end, the robot

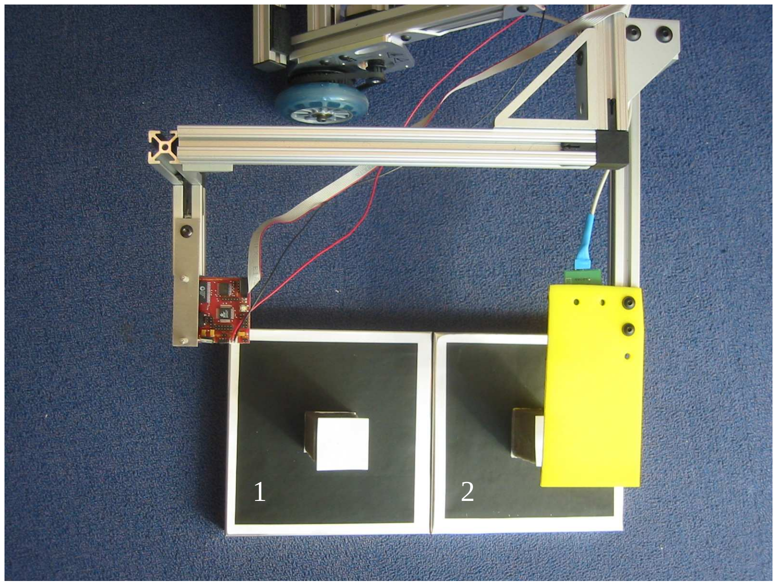


Figure 5-6: The camera is at the forward corner of block 1, relative to the robot, and the RFID transceiver is at the tag on the handle of block 2.

is facing away from the wall with the transceiver over the block tag. After attaching a block at an inside corner, a motion sequence {back up, turn right, back up, turn right} puts the transceiver into position without risking the robot running over the structure.

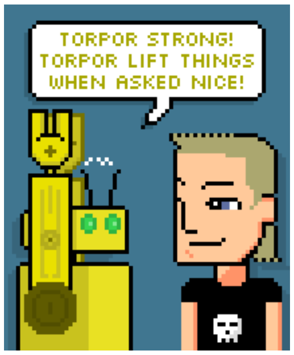
5.3 Evaluation

Several tests evaluated the reliability of the basic behaviors.

- The robot traveled repeatedly around the perimeter of a 2×2 square structure, measuring its progress by visual reference to the marked block edges, and checking this discrete position estimate for accuracy against the coordinates stored in read/write RFID tags. In addition, each time it passed a block, it wrote two random bytes to the tag and read them back. In approximately 30 minutes, it recognized 122 block boundaries to measure its progress by, turned 41 corners, located 82 block tags, read 328 bytes from and wrote 164 bytes to those tags, without any errors in any of these actions or in its discrete position estimate.
- The robot successfully found and retrieved blocks from the cache in 10 of 10 trials.
- In 10 trials for each of the three kinds of block attachments (mid-wall, just before a left-turn corner, and at an inside corner), the robot successfully attached blocks, and wrote their new coordinates to their RFID tags, respectively 9, 10, and 9 times.

As expected, the hardest robot tasks were those involving physical manipulation. This difficulty was compounded by the fact that with the current hardware configuration, the camera is the only source of feedback for position evaluation, and is more than 75 cm away from the gripper. When attaching a block, the robot adjusts its position using the camera at the attachment site, and then maneuvers blindly to move the gripper into place, relying on the not infallible precision of the ER1 drive system. Adding sensors at the gripper in the next hardware revision, to allow further position adjustment after the gripper is approximately in position, should greatly improve the reliability of block attachment.

From *Diesel Sweeties*
©2003 R. Stevens
www.dieselsweeties.com
Used with permission.



Chapter 6

Functional heterogeneity

In this chapter I consider the more general case where building blocks are identical in shape but heterogenous in function, and a given set of functional constraints governs how they can be put together.

With structures built from multiple types of building materials, it will in general be significant what materials are attached where. This factor introduces additional considerations into the construction algorithms.

If blocks can be arbitrarily custom-designed for particular structures, then any structure could be built using only local rules for assembly. A trivial, inefficient approach would be to designate specific blocks for specific locations in a blueprint, and engineer the blocks so that each one can be attached solely to the neighboring blocks in that layout and nowhere else.[1] However, I take the opposite approach, assuming that the block types are defined by the application, and the task is to work within those limits.

An example of such an application might be a system for building structures with more complex features such as plumbing and electrical wiring. Such features could be incorporated by prefabricating more elaborate modular blocks with embedded sections of pipe or wiring; attaching such a block to the structure would connect up those components, in order to form a network of these utilities [35]. These blocks would accordingly come in different varieties (containing straight segments, corner units, etc.), and a consideration in building would be attaching the various types at appropriate locations to produce a network as desired. Another application using even higher-level blocks is the building of underwater structures (for marine research,

[1]The separation rule will still restrict the order in which blocks may be attached.

oil drilling, etc.), where the building blocks are larger-scale pods that each fill one of several distinct and predefined roles—living quarters, lavatory, power generation, airlocks, emergency escape centers, laboratory space, cooking facilities, exercise and recreation areas, storage, etc. [1, 32].

The building design then might specify what block type should go at each location in the structure. Alternatively, the specifics of the layout may not be considered significant; instead there may be desired constraints on where different block types are attached relative to other types. For instance, in the latter application, one might want all living pods to be located in a contiguous block, no two airlock pods to be adjacent to one another, or no pod to be further than three pods away from an emergency escape center. The goal is then to build any structure satisfying those constraints.

Section 6.1 introduces the use of constraints of these sorts. Section 6.2 details several issues associated with the use of constraints that do not fully prespecify a structure, which are potentially more powerful but also more problematic.

6.1 Functional constraints

Overall, constraints on block placement can be divided into two classes: those independent of block type, and those based solely on block type. Geometric constraints, the former, are those discussed up until now; these ensure that the final structure is free of unwanted internal gaps and has a perimeter of the desired shape. Functional constraints, the latter, encompass whatever restrictions based on block types may be dictated by the particular application.

The algorithms previously presented considered only geometric constraints. Incorporating functionally heterogenous blocks involves an additional check before attachment is permitted. The two checks can be treated independently but both are necessary; the test becomes "are all geometric constraints satisfied, and are all functional constraints satisfied?"

Algorithms 6 and 7 give the appropriate generalizations of earlier algorithms for noncommunicating and communicating blocks, respectively. There are three main differences:

- An extra check must be performed (Algorithm 6, line 10; Algorithm 7A, line 4) to ensure that a potential attachment will satisfy the functional constraints.
- With noncommunicating blocks, robots must clear their *seen-row-start* flags in some situations to avoid unwanted gaps (Algorithm 6, lines 15–16).
- It is possible at any time for there to be no site where a block of a given type could legally be attached. As a result, robots must be able to leave the perimeter and get a different block type if no attachment site can be found (Algorithm 6, lines 12–13; Algorithm 7B, lines 8–9). What the robot does with the old block will depend on how blocks are obtained. If free blocks are scattered away from the structure, the robot could return the block to that region; if there are caches of blocks, each of a unique block type, the robot would need to return the

Algorithm 6 Pseudocode procedure for assembly of a solid structure of noncommunicating, functionally heterogenous blocks.

```
while structure not complete do
  fetch new block of type $t$
  go to structure perimeter
  establish position, by whatever means
  seen-row-start ← false
  while still holding block do
    if (site should have a block) and
        ((site just ahead has a block) or
        (seen-row-start and (at end-of-row))) and
        ($t$ at site consistent with functional constraints) then
      attach block here
    else if taking too long to find a place for block then
      discard block appropriately
    else
      if at inside corner then
        seen-row-start ← false
      if at end-of-row then
        seen-row-start ← true
      follow perimeter counterclockwise
```

block to the appropriate cache. In a weightless, three-dimensional environment, as in outer space or underwater, a cable extending outside the plane of the structure might have temporary attachment sites where blocks can be left in this situation. Then if and when such a block is called for later in the course of construction, it can be readily available so a robot need not travel as far to retrieve it.

Functional constraints may involve a full prespecification of the desired structure (section 6.1.1), or lower-level constraints on relative locations of different block types, the latter allowing structures to be built in a more adaptive way (section 6.1.2). Both sorts of constraints may be used in different regions of a single structure (section 6.1.3).

With noncommunicating blocks, robots are responsible for ensuring that functional constraints are satisfied. With structures not fully prespecified, whether a block may be attached at a given site depends not only on location but on the types of other blocks in the structure. In this case communicating blocks can make functional constraint satisfaction significantly easier, particularly for some classes of constraints (see section 6.2.2), because of limited robot perception. The use of communicating blocks also extends more naturally to three dimensions, where blocks can be walled in within a structure, so that robots would be unable to determine the types of blocks not accessible from the structure surface. For these reasons, when I discuss adaptive satisfaction of lower-level constraints involving relative block placement, I will consider only communicating blocks.

Algorithm 7 Pseudocode procedure for assembly of a solid structure of communicating, functionally heterogenous blocks. For convenience, this description assumes each block maintains a complete dynamic map of the structure, in addition to the shape map and the list of functional constraints.

A: Blocks

```
loop
  if a robot asks to attach a block of type $t$ to side $s$ then
    if (block at $s$ consistent with geometric constraints) and
        ($t$ at $s$ consistent with functional constraints) then
      coordinate with other blocks to lock out appropriate part of the structure
          from simultaneous attachment
      allow attachment
      update map with $t$ block at $s$
      pass message about $(t, s)$ to neighboring blocks
    else
      forbid attachment
  if neighboring block sends message about $t$ at $s$ then
    if message represents new information then
      update map with $t$ block at $s$
      pass message about $(t, s)$ to neighboring blocks
```

B: Robots

```
while structure not complete do
  fetch new block of type $t$
  go to structure perimeter
  while still holding block do
    ask adjacent structure blocks if $t$ can be attached here
    if all structure blocks answer yes then
      attach block here
    else if taking too long to find a place for block then
      discard block appropriately
    else
      follow perimeter counterclockwise
```

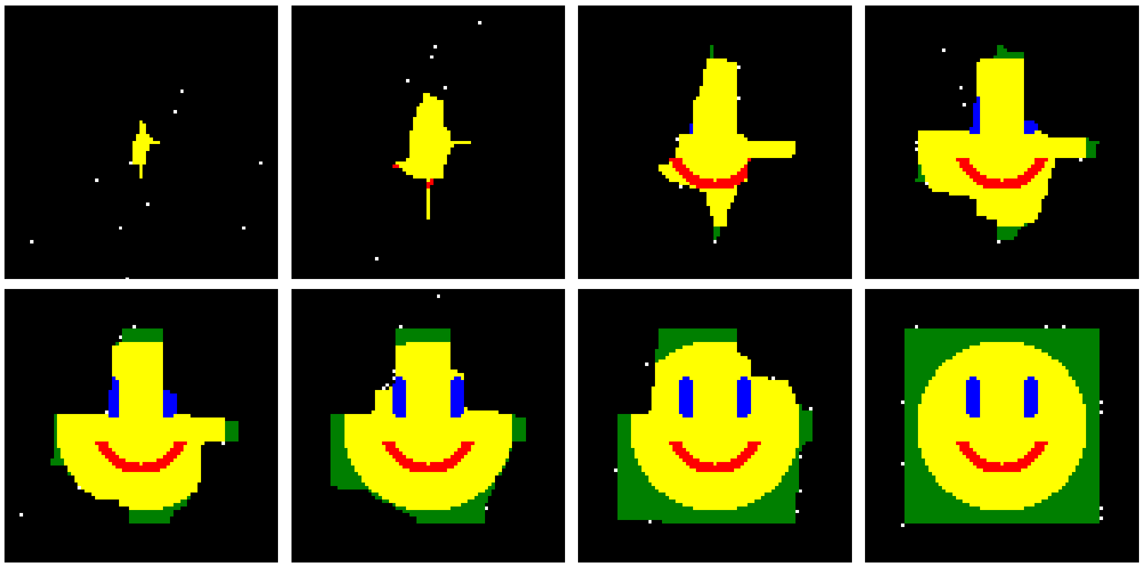


Figure 6-1: Snapshots of ten robots (white) building a prepatterned structure of four block types (yellow, red, blue, green), using communicating blocks.

6.1.1 Prepatterned structures

An important capacity of a construction system is the ability to create predefined patterns of block types. This can be achieved with functional constraints that act as a blueprint, specifying exactly what block type should be attached at every location in the desired structure. Like the shape map, this specification can be more compact than a full enumeration, just as an image specification can be more compact than a bitmap. Robots may only attach the blocks they carry if the latter are the single type allowed at the site in question. Figures 6-1 and 6-2 show rectangular and non-rectangular structures built using such functional constraints.

6.1.2 Adaptive patterns

In many cases the design of structures does not require specific patterns, but rather functional relationships between the locations of blocks of various types. Any structure that satisfies those relationships will be considered acceptable. The more relevant way to specify the constraint in this case is by reference not to absolute coordinates, but to relative placement of different block types. For instance, no two blocks of a given type may be permitted to be attached adjacent to one another; Figure 6-3 shows three possible structures built using such a constraint. The construction framework I have described extends naturally to building structures in this flexible way.

It is always a valid approach to come up with a blueprint ahead of time that satisfies all such constraints, give that to the system, and have it produce that particular structure. But satisfying constraints on the fly during the building process instead can let the structure be adaptive, and respond to changing circumstances or conditions unknown in advance.

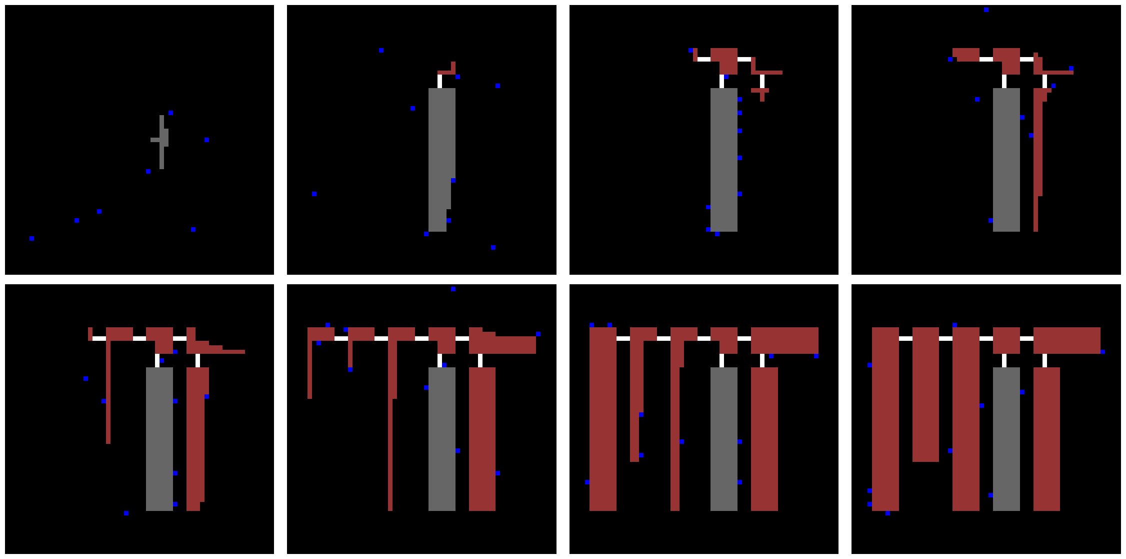


Figure 6-2: Snapshots of ten robots (blue) building a prepatterned structure with a complex shape, using communicating blocks.

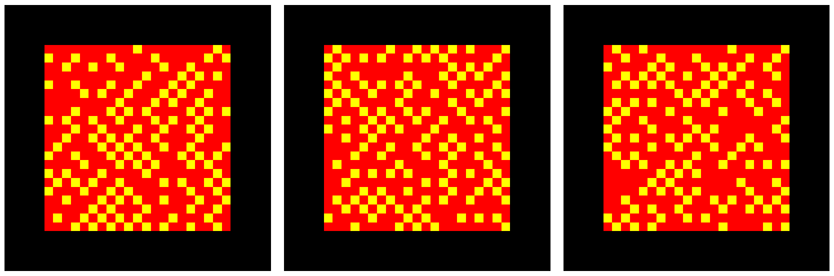


Figure 6-3: Examples of structures built with two block types, with the geometric constraint that the final structure be a 21×21 square, and the functional constraint that no two yellow blocks be adjacent.

Satisfying constraints on the fly can have additional benefits, such as increased speed of construction. As an example, consider the class of 21×21-block structures of Figure 6-3. If it is assumed that bringing blocks from caches to the structure perimeter, attaching a block, and moving one block length along the perimeter each take one time step ($L = A = 1$), then the time required to build a structure will reflect primarily the time robots spend following the perimeter looking for a place to attach their block. Averaging over 100 independent runs with 10 robots, on-the-fly construction completes in 650 ± 140 time steps, while building exactly the same 100 structures as prespecified patterns takes 1100 ± 300 steps, or 70% slower on average. Robots come to the structure already carrying some type of block; the speedup for on-the-fly construction occurs because sites can accept any type of block consistent with the constraints, rather than having to wait for a particular block type to appear, and robots can accordingly find places to attach their blocks more readily.

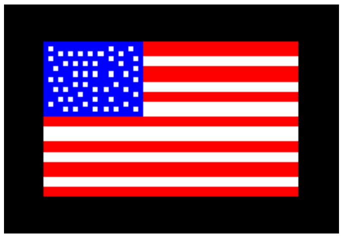


Figure 6-4: Example of a structure assembled according to the following constraints:
Region 1, border of upper-left area: only blue blocks allowed.
Region 2, interior of upper-left area: blue and white blocks both allowed; no white block may be in the eight-neighborhood of another white block.
Region 3, elsewhere: blocks must be either red or white depending on their y-coordinate (prepatterned).

6.1.3 Regional constraints and multiple levels

Constraints may also be specified on multiple levels of organization. Different *regions* of the structure can be defined, in each of which some different set of constraints is to be imposed. Figure 6-4 gives an example where such regions are designated according to specific coordinates, combining prepatterned and adaptive constraints into a single structure.

Regions can be designated in this rigid way, or more fluidly: it may be permissible for a region to occur anywhere in the structure, with variable size, aspect ratio, orientation, multiplicity, etc. For instance, instead of requiring that a particular region of red blocks be located in the rectangle between coordinates (3,5) and (6,10), the structure description might specify only that a region of 4 by 6 red blocks be located somewhere; that a region of 4 by n blocks, $4 \leq n \leq 10$, be located somewhere; or that m such regions, $1 \leq m \leq 12$, be located throughout the structure.

In this way, regions can be treated analogously to blocks, in that they can be positioned according to a fully prespecified blueprint, or given lower-level constraints and flexibility in satisfying them adaptively. In the same vein, just as blocks can have constraints on their relative locations restricting their placement, so can regions occur in the structure in locations restricted based on the locations of other regions. Dimensions, etc. of regions may also be specified with reference to each other (e.g., region A must be larger than region B).

Moreover, regions can be nested in this framework. An application might, for instance, call for a complex, which is composed of wings. One wing might be residential, made up of apartments; another might be made up of lab spaces. An apartment can be broken down into bedroom, bathroom, kitchen, etc.; and so on until the lowest level, which is decomposed into blocks as before. Each region may have its own constraints governing the placement of its sub-regions.

Figure 6-5 gives an explicit, abstract example. An 8×8 structure is constrained to have four 4×4 regions, one of type A, one B, two C. In a region of type A, only red blocks are allowed; in B, only blue blocks are allowed. In C, there must be four

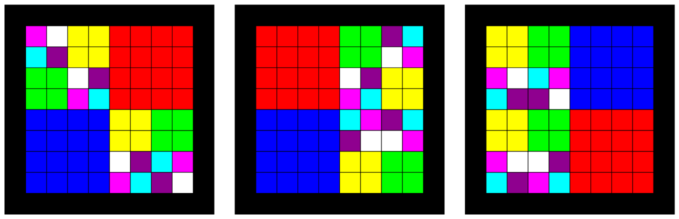


Figure 6-5: Three examples of structures that can all equally well be built based on the constraints given in the text.

2 × 2 subregions, one of type D, one E, two F. In a region of type D, only yellow blocks are allowed; in E, only green blocks are allowed. Finally, a region of type F must contain an arrangement of {purple, cyan, magenta, white} blocks, in that order clockwise; rotation is permitted, reflection is not. A great variety of structures can be built consistent with this set of constraints, all equally valid; the figure shows three.

6.2 Adaptive constraint satisfaction

There are a number of considerations associated with the use of adaptively satisfied constraints on relative block placement. To begin with, there's the question of whether a solution to a given set of constraints can be found (section 6.2.1). Low-level functional constraints can be classified into different types, with different requirements on block memory (section 6.2.2). Constraints may be satisfied in an immediate way, or in a way looking ahead to future stages of the structure's completion (section 6.2.3). And keeping multiple robots from taking mutually contradictory actions becomes slightly more involved than when a full blueprint is used (section 6.2.4). This section describes these issues in terms of low-level constraints between blocks, but the same considerations apply to low-level constraints between regions.

6.2.1 Satisfiability

A task involving fitting a given set of blocks into a given shape subject to given constraints may have no solution; and if one exists, it may be hard to find. In general, the problem of finding a solution to such a scenario, or showing that none exists, is NP-complete [63]. However, many cases of interest will have the property that a solution can be found quickly by a straightforward trial, and no exhaustive search is necessary. The system here provides a natural and effective way of assembling structures for constraint sets that are easy to satisfy in this sense. Constraint sets difficult to satisfy would best be handled by finding a solution prior to construction, and building that solution as a blueprint.

6.2.2 Types of adaptive constraints

In general, there are several classes of low-level functional constraints to consider, which may be characterized as follows:

- *Absolute*, without reference to neighboring blocks: e.g., type A may be attached; B is forbidden.
- *Proximity*, where the types of neighboring blocks are important but their locations are not: e.g., every A must have at least one B somewhere within a neighborhood of radius 3; no C can be adjacent to a D.
- *Relational*, involving both types and locations of neighboring blocks: e.g., every A must have a B bordering its west edge and a C bordering its north edge. Rotation and reflection may each be allowed or forbidden in such a constraint.

Proximity and relational constraints imply an associated distance d within which blocks are relevant to satisfaction of the constraint. One approach with communicating blocks is for blocks to maintain a local map of their area of the structure, whose size is at least of radius d. Blocks refer to this map to determine whether robots should be allowed to attach blocks at a neighboring site; when a new block is attached, a message to that effect is sent to blocks within d, for them to update their maps accordingly. The larger the value of d, the more memory is required of blocks, but the farther-reaching constraints may be.

Another type of constraint can result if only finite numbers of each block type are supplied. For instance, there might be only two blocks of type A, which must be attached adjacent to one another. The first A attached to the structure could be put anywhere, but the second would have to be added next to the first. In order to give an A-carrying robot the proper instruction, blocks would need to establish whether another A had already been attached anywhere, a global property of the structure. In such a case, blocks might use one bit of memory to indicate whether an A had yet been attached somewhere. Such an approach would be memory-efficient but not generalize well—what if there were four blocks of type A, which had to be attached in adjacent pairs?

The most straightforward way to ensure being able to handle all possible constraints, though not the most memory-efficient, would be to have blocks capable of maintaining a dynamic global map of the structure. Such a map could be of continued use after the structure is completed—in the underwater-structure example given above, for instance, every pod would be able to give its occupants a map of the full layout, directions to the nearest escape pod in case of emergency, and so on.

6.2.3 Short- and long-term constraint satisfaction

There are two ways to satisfy the condition that no functional constraint be violated by attaching a block of type t at a site s. One is to consider only blocks that are part of the structure at that time. The condition is simply: if attaching t at s would result in a structure that violates any constraint, then forbid the attachment. The examples of Figures 6-1–6-4 were assembled in this way.

In many cases, however, it will be desirable to look further ahead in time while ensuring constraints are met. For some constraint sets, situations may occur where attaching a block gives an acceptable structure, but one to which no further blocks can legally be attached. Looking ahead can prevent becoming trapped in such dead ends. A structure like that of Figure 6-4 gives one example: guaranteeing exactly 50 'stars' would require looking ahead during assembly of Region 2, to ensure sufficient remaining room for the white blocks not yet attached. It may also be desirable to allow constraints to be violated temporarily, so long as some future way of satisfying them remains possible. Consider, e.g., a structure of two block types, each of which is constrained always to have at least one neighbor of its own type. Without the possibility of violating the constraint temporarily, the structure would necessarily turn out to consist entirely of one block type: no block of the other type could ever be attached, since it would have no neighbor of its own type until a second such block were attached to it.

Such looking ahead may be limited to a fixed number of steps f into the future. Like constraints associated with larger d, larger f will be more powerful but more difficult for the blocks. Putting no limit on f, so that blocks always look ahead to the completion of some finished structure, will be slow but will prevent the structure from getting trapped in dead ends of partial completion.

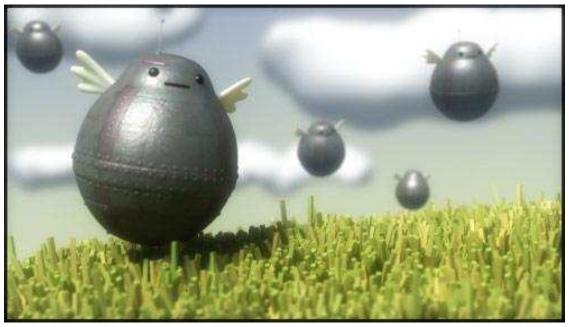
6.2.4 Considerations for multiple robots

Section 3.2.6 discussed the need with communicating blocks to ensure that no two robots be given near-simultaneous permission to attach blocks at conflicting sites. The approach described there to satisfy geometric constraints involved locking out a row from further attachment before giving a robot permission to attach.

For functional constraints with finite d, the corresponding approach is to lock out all blocks within radius d before giving a robot permission to attach. When d is large (and, in particular, since for some constraints d may extend over the entire structure), so doing can become unwieldy: considerable message-passing over the entire structure will take place for every block attached, and robots may be turned away from attaching more often than necessary.

However, message-passing within the structure will be much faster than physical movement of robots. As described in section 3.2.5, geometric constraints only require consulting with other blocks when the first block in a row is being attached (and typically, only a few blocks will need to be consulted); thereafter, allowing or forbidding other blocks to be attached in the same row can be reliably done without communication within the structure. And d is small for many functional constraints of interest.

All this means that, even without this strategy of locking out before attaching, errors should be infrequent. An alternative approach, then, is to correct errors when they arise rather than being scrupulous about preventing them in the first place. Blocks can allow or forbid robots to attach based solely on their own knowledge; if that causes a conflict that is only detected later, the blocks can direct robots to correct the error by removing the offending blocks, as described in section 8.2.1.



Chapter 7

Three dimensions

A construction or fabrication system that can only produce objects in two dimensions will, in most cases, be of limited utility. The real goal is for the automated construction team to be able to produce fully three-dimensional artifacts.

While implementation issues can become significantly more complicated, the algorithmic approach can be extended fairly directly to three dimensions in the case of communicating blocks. Here I describe that extension, with three variants on how blocks can find their way along the surface of the structure to attachment sites.

In this chapter, I will consider the problem in terms of two components: where blocks need to go, and how they can get there.

1. For any target structure, there are constraints on where elements can be added at any valid intermediate stage of construction, to avoid reaching dead-end states where construction cannot proceed.

2. Elements must reach, or be transported to, sites allowed by those constraints.

Section 7.1 considers the first of these components. Section 7.2 considers the second, and discusses three variant approaches—based respectively on a systematic search, random walks, and the use of gradient information—in which different information needs to be communicated, and in which different amounts of robot and block state are necessary. Section 7.3 experimentally compares the tradeoffs of these variants with respect to simplicity of approach, amount of communication, and construction speed.

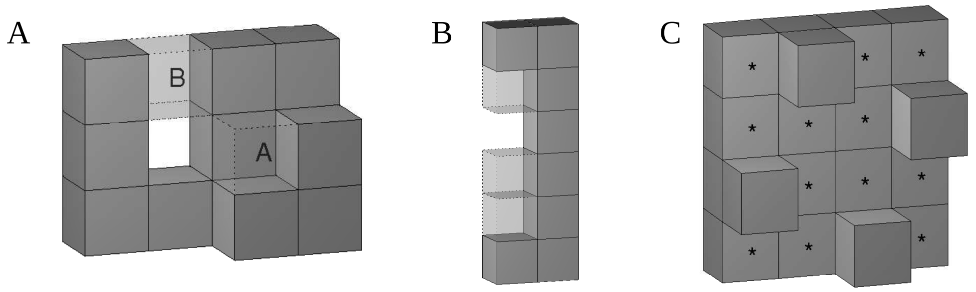


Figure 7-1: (A) Physical restriction on block placement. A block could be attached, e.g., at site A. Site B, though it has fewer neighbors, cannot accomodate a block because neighboring blocks occupy sites at opposite faces.
(B) As in two dimensions, if two separated blocks are attached in the same row, then an unfillable gap will ultimately result as blocks are added between them (shaded).
(C) This partial structure can occur without violating the separation rule. However, attaching further blocks at any of the twelve sites marked * would violate that restriction.

7.1 Allowed attachment

I assume the same basic constraint on block placement as before: a potential attachment site will be too physically restricted to accomodate a block if there are neighboring blocks at any pair of opposite faces (Figure 7-1A). Again, the corollary of that restriction is that no two non-adjacent blocks may be attached in the same row if all spaces between them are intended to be occupied. Otherwise, the further addition of blocks will eventually lead to an unfillable gap (Figure 7-1B).

In three dimensions, unfillable gaps can result in another way, via situations like that shown in Figure 7-1C. Configurations can arise in which the separation rule has been satisfied up to that point, but additional blocks cannot be added to any sites meant to be occupied without breaking it. In that example, building a $4 \times 4 \times 2$ target structure taking only the separation rule into account could result in the partial structure shown. But that partial structure is a dead-end state: no further blocks can be attached without violating the separation rule.

A sufficient way to prevent such situations is as follows. Consider any 2-D slice through the structure design, perpendicular to one of the Cartesian axes. Desired blocks in that plane form one or more contiguous groups. For any such group, require that as blocks are added, they must be added contiguously: i.e., once the first block

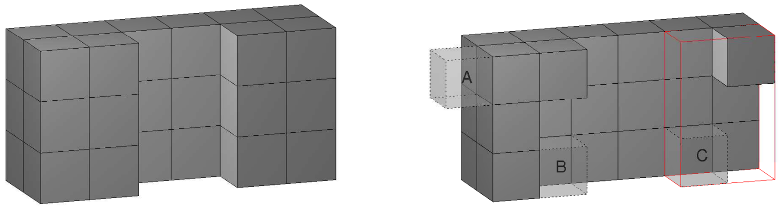


Figure 7-2: Left: A desired structure. Right: a possible intermediate stage in its construction. Site A violates the first check: the structure design specifies that no block is to be attached there. Site B passes checks 1 and 3 but fails 2: there is a block above it and an intermediate site that needs to be occupied. Site C passes checks 1 and 2 (since the design specifies that the center region is to be left empty) but fails 3: it would not be attached contiguously to the block previously attached in the locally contiguous group in one plane (red border).

in such a group has been added to the structure, successive blocks in that group must be added adjacent to previous ones[1] (Figure 7-2C).

Thus with functionally homogenous blocks, the following three checks are sufficient when a site is being considered for attachment (Figure 7-2):

1. Does the structure design specify this site is to be occupied?

2. For each of the three axes passing through this site, is the separation rule satisfied?

3. For each of the three planes passing through this site, is a block here being attached contiguously to the locally contiguous group?

If the answer to all three of the above questions is 'yes', then block attachment may proceed. Otherwise, attachment is forbidden.

With functionally heterogenous blocks, a fourth check can be added to ensure that a prospective attachment satisfies all functional constraints as well.

For robots building with noncommunicating blocks, ensuring that all of these checks are satisfied can be a significantly more complicated task than in the two-dimensional case. In particular, for functionally heterogenous blocks and corresponding constraints, robots will be unable to determine the types of noncommunicating blocks that have become entirely buried within the structure. Communicating blocks, by contrast, will still be able to determine those block types as necessary; and as before, they can share information to check that geometric constraints are satisfied. For these reasons, I consider only communicating blocks in this chapter.

[1]This rule can be viewed as the analogue of the separation rule, in the next higher dimension. Conversely, the separation rule can be viewed as a statement that contiguous one-dimensional groups of blocks must be assembled contiguously.

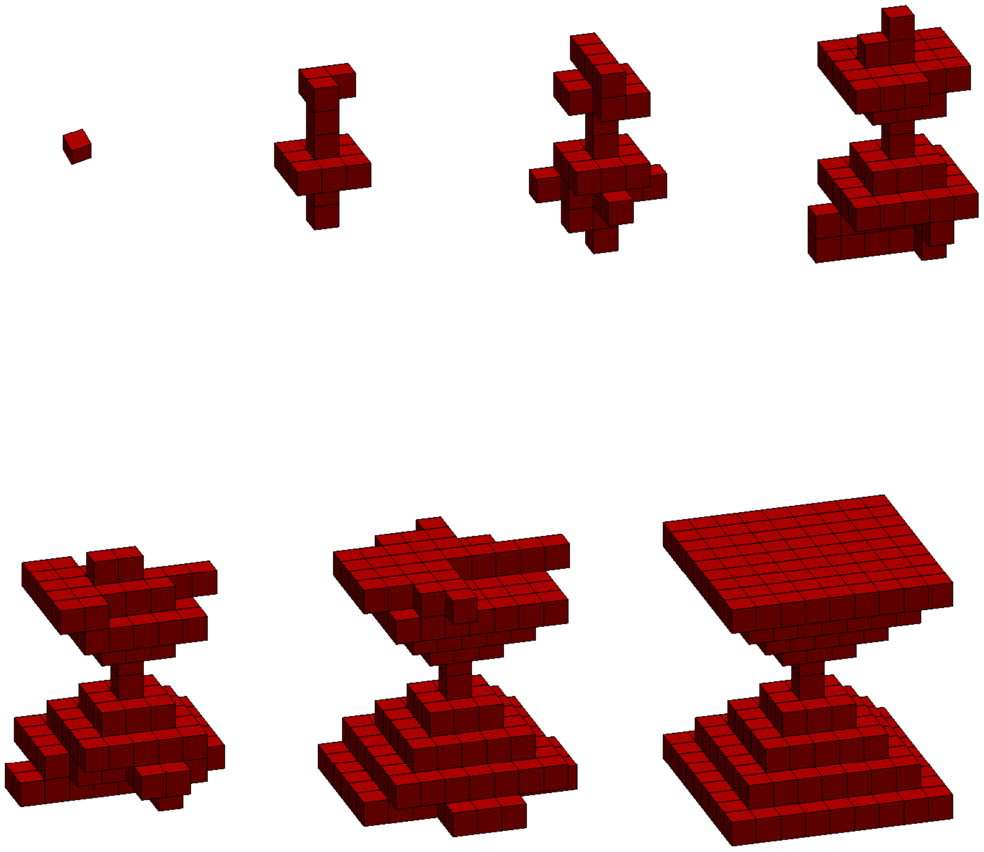


Figure 7-3: Simulated construction of a sample structure in three dimensions, showing successive snapshots during the process of construction by ten robots, using gradient-based movement (§7.2.3). For clarity, only blocks that have been attached to the structure are shown.

The algorithm for blocks is analogous to the two-dimensional case. Each side of each structure block now maintains state associated with blocks in the adjoining plane and in the two orthogonal rows passing through the adjoining site. Before any blocks have been attached in a given plane or row (label it "P_1"), the corresponding side of blocks in the adjoining plane/rows ("P_2") will be in a state indicating that attachment in P_1 is allowed. When the first block ("B") in P_1 is attached, the associated blocks in plane/row P_2 take on a state locking out further attachment in plane/row P_1, except for the allowed sites adjacent to block B.[2] Passing messages over distances is needed only when the first block is being added to a new row, or to a new contiguous group in a plane. For the addition of further blocks in that row or plane, communication beyond immediate neighbors is not required.

7.2 Block transportation

While the approach is straightforward to extend from the two-dimensional case algorithmically, it would be significantly more difficult physically. One obvious complication in terrestrial settings is the presence of gravity. In the following, gravity is neglected. Thus the solutions presented here could be applied to weightless situations (e.g., outer space or underwater), but more work would be required to extend them to terrestrial contexts.

There are several different methods by which blocks might reach the structure and the potential attachment sites.

7.2.1 Systematic search

Recall that in the 2-D case, block-carrying robots come to the structure and then circle it counterclockwise until reaching a site where their block can be attached. A direct extension of that approach works for a limited class of solid 3-D structures.

The straightforward modification is to treat the structure as a stack of layers. Robots[3] bring blocks to the structure and circle it at the level at which they arrive (constant z-coordinate), keeping track of visited locations. If they return to a previously visited site without having found any available sites at that level, they move up or down one level and circle again.[4] To implement this approach, robots should have a sensor indicating which direction is up. Additionally, either robots need suffi-

[2]Once again, an alternative to locking out planes/rows is to accept the possibility of occasional errors and to expect to have to correct them (section 8.2).

[3]In some applications, blocks may be self-mobile, or driven by, e.g., ambient fluid forces. In such cases, with no separate class of mobile agents to transport blocks, references to robots in the discussion may be replaced by references to the blocks.

[4]Depending on the geometry of the structure, moving up or down a level may first require some movement at the original level to reach a point where vertical movement is possible, or some movement might be necessary at the new level to reach the structure perimeter in that layer. Further, robots using this approach must have some way of distinguishing 'inward' from 'outward', when moving along the top or bottom surface of the layer next to their own while not adjacent to any blocks in their own layer. Otherwise they may have no basis for choosing a direction of movement.

cient memory to keep track of visited locations, to be able to determine when they've returned to a previously visited site; or blocks need to determine when there are no available sites on a given level, and direct robots to higher or lower levels.

A primary disadvantage of this approach is that it imposes the following requirements on permissible structures:

1. Every level of the structure design must have a block at $x = y = 0$, which further must be the first block attached at that level.
2. Each level must constitute a contiguous solid structure.

The first requirement ensures that a block will be able to find some place at each level where attachment is desired. The second reduces the problem at each level to a previously solved case.

7.2.2 Random movement

On small enough scales, or in weightless environments, blocks may drift at random or via Brownian motion [26, 30, 74]. Blocks will then eventually reach potential attachment sites at random of their own accord, and can, e.g., be drawn into position when close enough through the use of some attractive force [74].

Advantages of this approach are its parallelism (all blocks can be in motion and eligible for potential attachment at once), its potential suitability to very small scales, its applicability to structures that are not limited by the requirements in the previous section, and the lack of a need for a separate class of mobile robots at all. Disadvantages include difficulty of feasibility at larger scales and in terrestrial and other environments, and the fact that stochasticity can result in very long assembly times: some sort of more directed search is likely to be faster.

Another approach based on random movement, more applicable to the case of building blocks manipulated by mobile robots, is one in which robots bring blocks to the structure (for example, by following a beacon) and thereafter move at random along the structure's surface. This approach will be slower in general than the systematic search, but is a very simple way to avoid the latter's constraints on admissible structures.

7.2.3 Gradient-based movement

A third approach, which both avoids the systematic search's restrictions on admissible structures and can significantly shorten the search for an attachment site, is for the structure to direct a searcher's movement more explicitly. Any blocks in the structure with exposed faces at which further blocks could be attached send out a directed numerical gradient along the surface of the structure, reflecting this availability [55, 56].

The gradient value represents additional state maintained by and communicated between blocks in the structure, and must be communicated also to robots moving along the structure surface. For the latter, following this gradient means moving in the direction of the closest site where a block could be attached. The fact that the gradient is restricted to the surface prevents becoming stuck in local minima. The drawback to this approach is that it involves much more extensive message passing.

With functionally heterogenous blocks, a separate gradient must be associated with each block type. A site that can accomodate more than one type of block sends out information associated with each corresponding gradient.

7.3 Experiments

This section compares these three different assembly methods by their performance on several example structures. Relevant quantities are the time necessary for structure completion, and the amount of communication required. The latter factor includes both messages passed within the structure (that is, between blocks which are physically connected), and messages passed from the structure to robots moving along its surface.

As in two dimensions, the physical presence of robots will be an implementation-dependent restriction on possible structures: any deliberate infoldings of the surface need to leave enough room for robots to move over the surface freely. The size and geometry of robots will also affect how close they can get to one another. In these experiments I do not explicitly consider the physical presence of robots; a robot carrying a block takes up the same space (one grid cell) as a block alone. The effect is equivalent to self-mobile blocks. Blocks are allowed to move in any direction along the structure surface, and to approach each other as closely as to occupy two adjacent sites.

Experiments proceed as follows. A structure begins with a seed block in an empty workspace. Ten robots carrying blocks are placed at random a constant distance away. Each moves inwards until it reaches the growing structure, then moves along its surface according to one of the three approaches discussed above. At each step adjacent to the surface, a robot receives a message from neighboring blocks about whether it can legally attach at that site; with the gradient approach, direction information is communicated as well. Additional communication may take place within the structure, as blocks are added and allowed attachment sites change, and (when applicable) as gradient information is updated. The simulations implement the algorithms in a centralized way, but calculate at each step the approximate number of messages that would be sent in a decentralized implementation. Each time a robot finishes attaching a block, it moves instantaneously back to a random position at the original distance from the seed, holding a new block. Messages travel infinitely fast compared to the speed of robot movement. Movement updates are asynchronous.

Figure 7-4 shows experimental results, averaged over ten runs each for various structures, for the total distance robots move along the surface of the structure during assembly, the number of messages sent to robots from the structure, and the number of messages passed within the structure.

1. The random walk approach requires more robot travel than do the other approaches. This difference increases for structures with greater surface area.

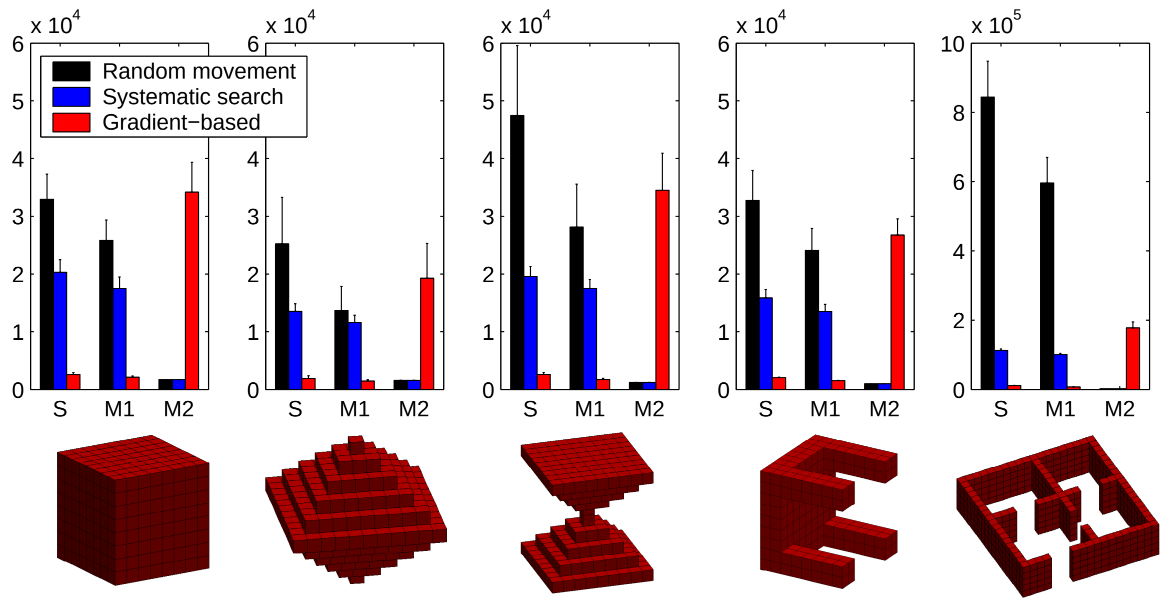


Figure 7-4: Performance comparison for the three movement approaches, for various structures. S: Steps taken along surface. M1: Messages passed to robots from structure. M2: Messages passed within structure.

2. The systematic search eliminates a great deal of repeated movement and guarantees that robots will reach every allowed site in bounded time.

3. With the gradient-based approach, robots travel distances an order of magnitude smaller still, as they are directed straight to the closest available site. However, the cost is at least an order of magnitude more communication within the structure, to establish and maintain the gradient. Additionally, the (fewer) messages sent to robots from the structure will need to contain more information, specifying movement direction in addition to whether attachment is allowed. These costs may be bearable in practice for the sake of the speedup in construction, since communication between blocks which are physically connected should be rapid, unambiguous, and reliable, and hence of low cost compared to movement or less direct communication.

My giant killer robots should have completed their attack on the Tsahir mansion. Now to reap the spoils of my—
WHAT'S THIS?

From *Narbonic*
©2000 Shaenon Garrity
www.narbonic.com
Used with permission.

Chapter 8

Other extensions

This chapter discusses other extensions to the approach described in previous chapters. Section 8.1 discusses how temporary structures could be disassembled by the same robot construction team. Section 8.2 discusses recovery from failures (of the blocks, robots, or marker). Section 8.3 discusses building structures with deliberate holes. Section 8.4 discusses the isomorphism of this approach using square or cubic blocks to a "Tinkertoy"-style approach using rods and hubs to build open lattices. This chapter describes these issues in two dimensions, but three-dimensional extensions are straightforward.

8.1 Disassembly

Many applications will involve structures intended to be temporary. If block attachments are reversible, a swarm construction system like those described here can dismantle a structure following its use and reuse its parts for another structure. Disassembly can be accomplished by having robots follow the structure perimeter, removing any blocks they find that satisfy the following criteria:

1. The block has at most two neighbors, bordering adjacent sides; i.e., if that site were unoccupied, it could physically accomodate a block.

2. If it does have two neighbors along adjacent sides, then the other site which those two neighbors border is occupied by a block (Figure 8-1).

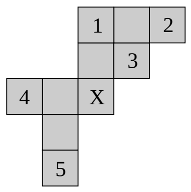


Figure 8-1: A structure in the process of disassembly. Numbered blocks can be removed. The block marked X meets criterion 1 in the text but not 2; if removed, it would split the structure into two parts.

Also, if the application involves a marker which acts as a tether and/or a beacon to let robots find the structure, then the marker must be the last block removed.

The first criterion assumes that physical constraints on block motion are the same for disassembly as they are for assembly. The second criterion ensures that throughout disassembly, the structure remains one contiguous piece. Violating this rule can result in part or all of the remaining structure becoming isolated. In a system where robots follow a beacon to reach the structure, a disconnected component will be lost to perimeter-following robots. If the structure is physically tethered to some anchor point via the marker (as would likely be the case, e.g., in underwater or space-based applications), then violating those criteria would mean that blocks could drift off and be lost in a more literal sense.

Care must also be taken with multiple robots that two blocks not be chosen for removal at the same time such that each block considered alone satisfies the criteria, but would violate criterion 2 if the other block were removed. Blocks 1 and 3 in Figure 8-1 illustrate such a pair.

Either blocks or robots may be responsible for determining whether blocks satisfy these criteria, according to whether communicating or noncommunicating blocks are involved.

8.2 Recovery from failures

There are a number of possible failure modes to consider. These include cases where a block breaks or otherwise needs to be replaced; where a beacon breaks, if applicable; or where robots go wrong. Section 8.2.1 describes the case for block replacement; section 8.2.2 outlines considerations for other failure modes.

8.2.1 Block error correction

Suppose that a block mistakenly ends up attached at a location where it violates some constraint, or needs to be replaced due to malfunction or some other reason. It may be some time before this error arises or is detected. I will thus consider the problem of replacing any arbitrary block in the structure. I outline the high-level approach and omit elaboration of an explicit algorithm to be carried out by individual robots or blocks.

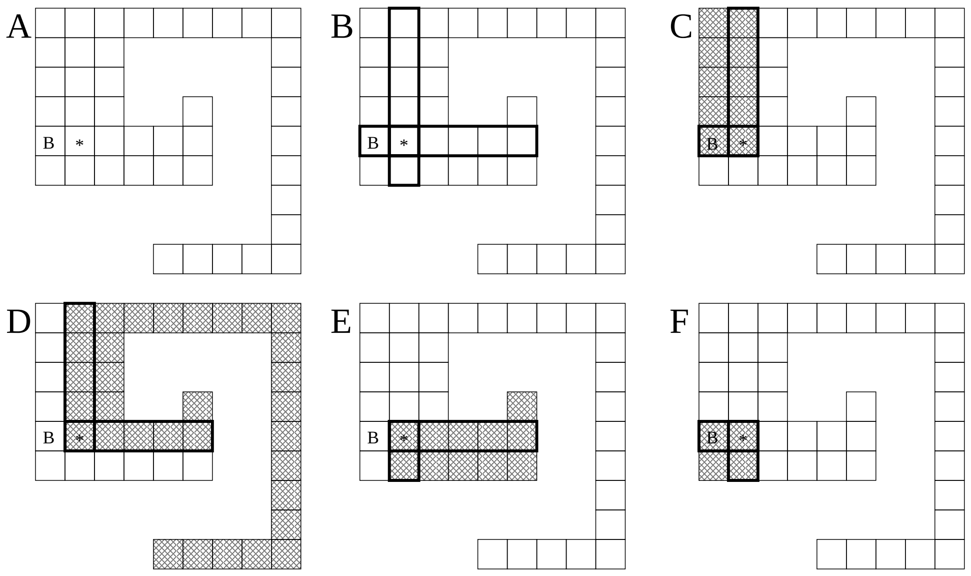


Figure 8-2: Error correction. The letter B indicates a beacon; the block to be replaced is marked *. See text for discussion.

Because of the assumption that a block can only be removed from the structure if it is already free on two sides, it is necessary to remove all blocks in an entire 'quadrant' of the structure in order to free up the target block. Figure 8-2 demonstrates the demarcation of quadrants. (A) shows a completed structure, using a marker which acts as a beacon, marked with the letter B; a block to be replaced is marked *. Extending horizontal and vertical arms from the target block, as in (B), defines the four (overlapping) quadrants, crosshatched in (C)–(F). A quadrant is constructed essentially by flood-filling the area including and to one side of a pair of arms.

Note that a quadrant may contain blocks whose absolute coordinates go beyond those of the associated arms (as in (D), where a quadrant nominally above the right-hand arm includes blocks that extend below it). Also, any blocks that would otherwise be left isolated from the rest of the structure if a quadrant were removed must be included in that quadrant. This issue arises if an arm lies along an edge of the structure for all or part of its length. In this example, the right-hand arm lies along such an edge, at the top side of the second and third blocks from the target block; any blocks above and further along that arm must therefore be included in the quadrant nominally below it, as shown in (E).

For efficiency, the quadrant containing the fewest blocks should be chosen for removal. As with disassembly, a block representing a beacon or anchor must remain intact, and so any quadrants containing that beacon are omitted from consideration in this choice. Here the quadrant in (F) contains the fewest blocks, but cannot be removed because it includes a beacon (as does the quadrant in (C)). The quadrant chosen for removal will be that in (E).

Once the target block has been removed, that section of the structure can be restored by construction as usual. Construction may also progress as usual in other parts of the structure throughout the correction process.

Previous chapters suggested, as an alternative to temporarily locking out parts of the structure from attachment to prevent errors, instead accepting the possibility of inappropriate attachments transiently occurring, and correcting them when they do. This sort of error would likely be detected quickly, and require the removal of only one or a few blocks along the perimeter, rather than the potentially large numbers necessary to replace a failed block in the middle of the structure.

Noncommunicating vs. communicating blocks

As with other issues, the tasks described above can be accomplished much more easily with communicating blocks than with noncommunicating ones. Several aspects involve the use or dissemination of nonlocal structure information: the determination of which quadrant to remove, the restriction against attaching additional blocks in that quadrant during correction, the requirement for blocks in that quadrant to be removed, and even the detection of the erroneous block.

With noncommunicating blocks and no inter-robot communication, some of these tasks may be very difficult to accomplish. One possibility could be to make use of a probabilistic strategy, where additional blocks may (inappropriately) continue to be attached in the quadrant chosen for removal, but at a lower rate than that at which blocks are (appropriately) removed. If communication between robots is allowed, another possibility could be to make use of a team approach, for instance setting up robots to act as sentries marking a region for removal.

Conversely, communicating blocks make these tasks straightforward. A block may determine that it needs to be replaced, or if it malfunctions more severely, its neighbors may make that determination. Any block could obtain the structure information necessary to carry out the computation determining which quadrant is to be removed, and send messages out to the affected blocks accordingly. Blocks in the chosen quadrant could forbid robots from attaching any further blocks to them, and ask to be removed.

8.2.2 Other failure modes

In keeping with the swarm approach, the algorithm does not depend on particular robots completing particular assigned tasks, nor on blocks being attached in a particular order. Thus breakdowns of individual robots almost anywhere in the workspace need not impede the system's ability to complete the structure. The exception is at the edge of the growing structure, where a broken robot could serve as an obstacle to others following the perimeter, and prevent growth of the structure in that area. Such perimeter breakdowns would need to be detected and towed away by other robots.

In an approach in which robots use marker and cache beacons to find materials and return to the structure, those beacons represent failure points, in that if they fail, the system may be unable to complete its assigned task. One possible safeguard against their loss would be to build into robots or blocks the potential to replace them [68].

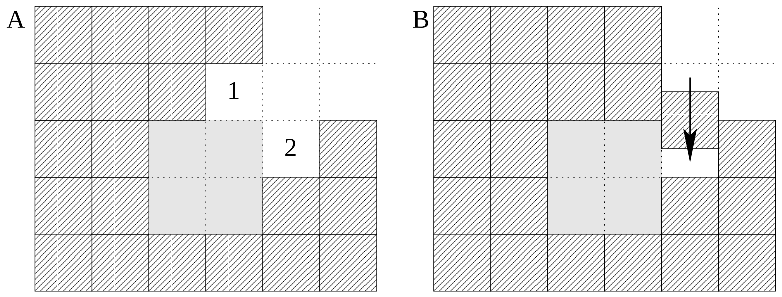


Figure 8-3: (A) Naïve application of Algorithm 4 to a structure intended to contain internal gaps frequently leads to the situation shown, which can easily be reached without the separation rule ever being violated. Blocks are shown with diagonal shading; cells intended to be left empty have light shading.
(B) Attaching a block at either of sites 1 or 2 will give a configuration in which it may be difficult for a block to reach the other site.

8.3 Deliberate holes

In all of the preceding, I have been treating internal gaps as an evil to be avoided at all costs. But what if we want to build a structure which intentionally contains gaps—a wall with windows in it, for instance?

Unfortunately, the algorithm for communicating blocks does not extend easily to include deliberate gaps. Applying it without modification leads to situations like that shown in Figure 8-3A. If blocks are placed in both a cell above site 1 and a cell to the right of site 2 before either 1 or 2 is filled in, which is possible using Algorithm 4, then the problem shown will result.

That configuration does not involve explicitly attaching a block between two others on opposite sides, and so might be construed as allowed. However, in order to maneuver it into place, the block will have to pass through a position where there are blocks on both sides at once (Figure 8-3B). Thus it is preferable to disallow it for the same reason for disallowing the more explicitly problematic case.

8.3.1 Alternate algorithm

An approach significantly different from that so far described can handle deliberate holes, for a limited class of structures. Moreover, it lends itself more readily to decentralization in a sense, in that no information is needed about whether distant blocks have been attached. Once position is established, the check necessary to determine whether attachment at a given site is allowed depends solely on the presence or absence of immediate neighbors.

That check can be described informally as follows (Algorithm 8, Figure 8-4). If a site is meant to have a block attached according to the shape map, check the two neighboring sites which are one step closer to the marker, one in each coordinate. (If the site in question is in the same row or column as the marker, there will be only one such neighboring site.) For each, if the shape map specifies a block there but the

Algorithm 8 Pseudocode procedure for assembly of a structure which may contain deliberate holes. The text and Figure 8-5 describe the class of structures this approach applies to. The marker is defined to have coordinates $(0, 0)$.

```
while structure not complete do
  get block from cache
  go to structure
  establish position, by whatever means
  while still holding block do
    $(x, y) \leftarrow$ coordinates of prospective attachment site
    $A \leftarrow ((x = 0)$ or
         (shape map specifies no block at $(x - \mathrm{sgn}(x), y)$) or
         ((shape map specifies a block at $(x - \mathrm{sgn}(x), y)$) and
          (block at $(x - \mathrm{sgn}(x), y)$)))
    $B \leftarrow ((y = 0)$ or
         (shape map specifies no block at $(x, y - \mathrm{sgn}(y))$) or
         ((shape map specifies a block at $(x, y - \mathrm{sgn}(y))$) and
          (block at $(x, y - \mathrm{sgn}(y))$)))
    if (shape map specifies block at $(x, y)$) and $A$ and $B$ then
      attach block here
    else
      follow perimeter counterclockwise
```

site is unoccupied, then attachment at the site in question is forbidden. Otherwise, attachment is allowed.

Let $\mathcal{A}$ be the following class of structures: Every block in the structure must be reachable by some sequence of steps, starting at the marker and moving to adjacent blocks, that involves at most two directions (e.g., 'right' and 'up'). Structures that would require taking steps in more than two directions to reach any given block will not be correctly built by this algorithm.

Algorithm 8 will reliably produce structures of class $\mathcal{A}$ which have only rectangular holes. It may be able to produce any structure of that class, but not reliably: with non-rectangular holes and no further restrictions on ordering of block attachments, some blocks near holes could fail to be attached (Figure 8-5). The algorithm will reliably fail on structures outside that class.

Class $\mathcal{A}$ is the same class of structures that Jones and Matarić's Transition Rule Set Compiler [30] applies to; and this algorithm (derived independently) appears similar to theirs. Their approach eliminates some state redundancy in blocks, and so is closer to the style of approach where a compiler generates minimal but incomprehensible rulesets, though it does not claim to achieve minimalism.

I have been investigating extensions to the approach of Algorithm 8. For instance, an approach which decomposes a structure into multiple regions, each of which uses a separate 'seed' as position reference in the equivalent of the above algorithm, can extend the class of admissible structures. However, a single algorithm that handles

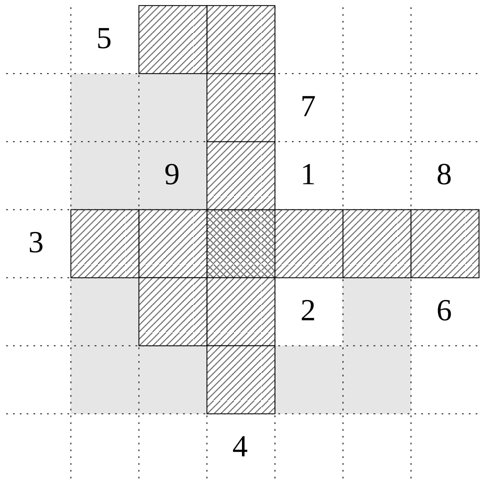


Figure 8-4: Examples of sites where attachment is allowed or disallowed according to Algorithm 8. Blocks are shaded with diagonal lines; the marker is crosshatched; sites meant to be left empty have light shading.
Attachment is allowed at sites 1 and 2, because the two neighboring sites in the direction of the marker are occupied; at 3 and 4, because they are in the row/column of the marker and the one neighboring site in the direction of the marker is occupied; and at 5 and 6, because although only one of the neighboring sites in the direction of the marker is occupied, the other is meant to be left empty.
Attachment is forbidden at sites 7 and 8, because one neighboring site in the direction of the marker is unoccupied and meant to have a block; and at 9, because although both neighboring sites are occupied, the shape map specifies that the site itself should be left empty.

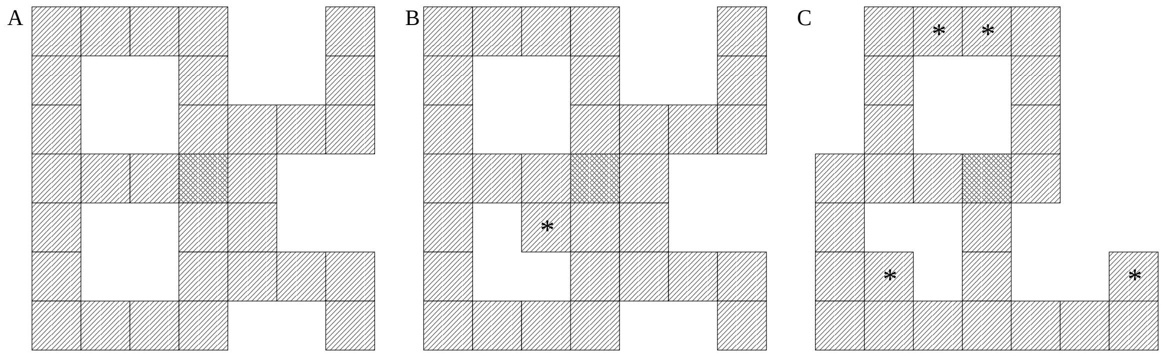


Figure 8-5: (A) Example of a structure Algorithm 8 will reliably produce; (B) example of a structure it may be able to produce, depending on ordering of block attachment; (C) example of a structure it will fail on. The marker is in the center, crosshatched.
In (B), the blocks bordering the lower-left hole could be attached before the block marked *, making it impossible for the latter site to be reached.
In (C), the blocks marked * are impossible to attach with this algorithm.

truly arbitrary structure designs, with respect to both perimeter and internal holes, remains elusive. Finding one is an important area for future work.

8.3.2 Non-algorithmic solutions

In future work I hope to find a manageable algorithmic solution to the problem, or prove that it is in general impossible to avoid. Other possible resolutions could include the following:

- Stipulate that blocks must be constructed such that they can be added to both sites 1 and 2 in the situation of Figure 8-3. That both makes things harder for the mechanical designer, and undermines the rationale for maintaining the separation rule in the first place.

- Leave one of the two cells unfilled and let construction continue. The resulting structure will have a gap in the designated place but not of exactly the designated shape. For that reason, this approach is unsatisfying as a solution to the problem of constructing designated structures.

- Take advantage of the third dimension by adding blocks in such configurations from out of the plane, avoiding that difficult transition. This approach will not extend to higher dimensions.

- Build strictly according to the algorithms already described, filling in cells intended to be part of gaps with a special class of blocks which are designed so as to be able to be removed at the end of construction. This approach is analogous to apoptosis (programmed cell death) in morphogenesis.

8.4 Building open lattices

Many structures are built not as solid units but as open frameworks of struts and connectors. These frameworks might represent the final structure, as in the examples of Figure 8-6, or they might be intended as an internal skeleton which will support additional material at a later stage of construction.

The approach so far described is isomorphic to such a case, if the block shape is modified appropriately (Figure 8-7A, B). Cross-shaped blocks could be used to build an open lattice, with no modifications to the algorithms.

In practice, cross-shaped blocks are likely to be unwieldy to manipulate, to be more difficult to attach together than solid blocks, and to have weaker connections. A better alternative is to build using rods and node connectors as in Figure 8-7C and D. The same algorithmic approach could still be used; only the implementation details of the robots, and how they manipulated the building elements upon attachment, would change.

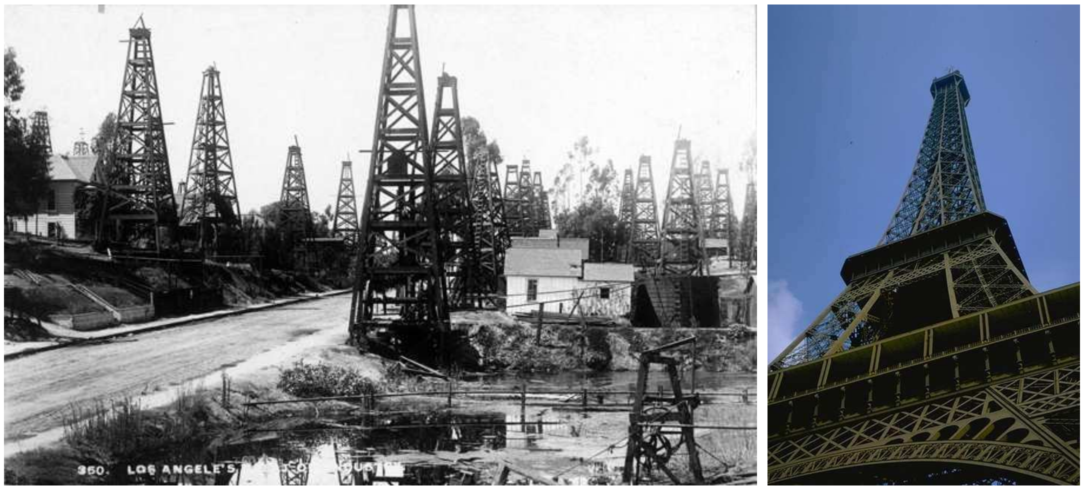


Figure 8-6: Many structures are built as open frameworks rather than as solid units.

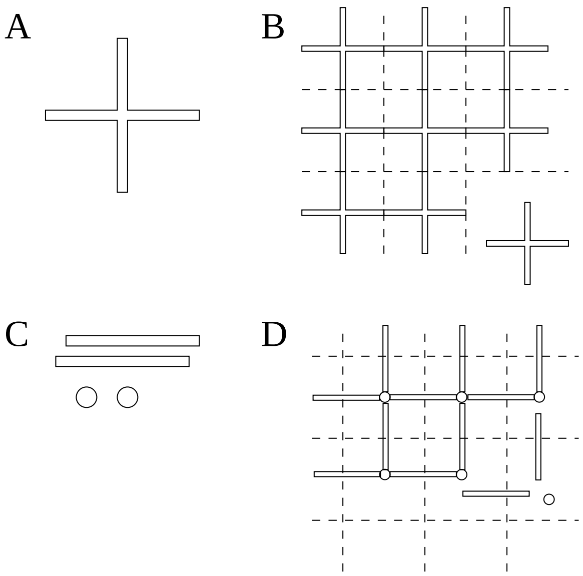


Figure 8-7: The same framework described above can be used to build an open lattice rather than a solid figure.
(A) A cross-shaped block. (B) Treating such blocks exactly like the square blocks hitherto described leads to the assembly of a lattice.
(C) A method likely to enable easier and stronger connections, and more portable building materials, is for the structure to be built out of rods and node connectors as sketched here. (D) Adding a block, in the original formulation, then corresponds to putting a connector in the center of the associated cell and attaching it with rods to any neighboring connectors.

From *Piled Higher and Deeper*
©2005 Jorge Cham
www.phdcomics.com
Used with permission.

IT'S... IT'S SNOWING.
IN JUNE.
I FEEL A GREAT DISTURBANCE IN THE FORCE!
MEANWHILE, IN HELL:
brrr.
JORGE CHAM © 2005
TO BE HONEST, MIKE, I NEVER THOUGHT THIS DAY WOULD COME.
ME NEITHER.

Chapter 9

Conclusions

In this thesis, I have presented decentralized algorithms by which a collection of simple robots could assemble structures out of square blocks according to a high-level user-specified design. I demonstrated how extending the idea of stigmergy, by putting increased capabilities into the building materials, could improve construction speed and the system's ability to make use of the swarm's parallelism. This chapter concludes with a summary and outline of open questions for future work.

9.1 Summary

Earlier chapters explored the following issues:

- Robots can reliably build a specific structure, even when all blocks are identical, by using the partially completed structure as a reference and block edges as landmarks.

- Labeling blocks, as with passive RFID tags, can allow faster construction. Writable blocks speed construction still further.

- Blocks which can communicate enable the dissemination of nonlocal structure information still more readily. As a result, robots can be made simpler; the parallelism of the swarm can be better exploited; and structures can in theory be completed as quickly as with any centralized algorithm.

- Interference between robots can be effectively reduced in a decentralized way by having individuals temporarily remove themselves if they encounter too much interference. In this way the size of the construction team changes dynamically according to the size of the task.

- Structures can be built using heterogenous blocks, in either a fully prepatterned way, or adaptively according to a set of low-level constraints.
- The algorithmic approach extends to three dimensions, particularly with communicating blocks.
- Recovery from various kinds of failures is feasible.
- A limited class of structures with deliberate holes can be built.
- Building with open lattices rather than square blocks can be cast as the same problem algorithmically.
- A hardware prototype demonstrates building solid 2-D structures with noncommunicating blocks.

9.2 Future work

A number of areas for future work have been pointed out along the way. These include the following:

- Algorithms for building fully arbitrary structures, with internal holes of any shape as well as arbitrary perimeters.
- Algorithms for three-dimensional structures which take into account factors such as gravity.
- Improvements to hardware. For instance, additional sensors near the gripper should increase the success of block attachment. Further, extensions to allow awareness of other robots (for, e.g., collision avoidance) would permit physical swarms consisting of more than one member.

Other important open questions include the following:

- While functional heterogeneity has been discussed, the problem of shape heterogeneity—assembling a structure out of blocks whose shape may vary—remains for future work. It is worth noting that the related polyominoes problem, tiling a space of a given shape with blocks from a given set, is NP-complete [38]. Thus, as with the situation with functionally heterogenous blocks, building a structure in an adaptive way rather than a fully prespecified way will likely be appropriate only in cases where the set of requirements turns out to be easy to satisfy.

- This dissertation considers the problem of reliably building structures of specified shape. I have stressed this problem as distinct from the way social insects build, which is to produce any structure that "works", but not necessarily any particular one. That specificity is a desirable feature in most building projects. However, in some situations, it will be desirable to produce structures whose overall shape is flexible. It should be possible to develop algorithms that take a set of high-level shape requirements and guarantee that the system will come up with some shape that satisfies them. Solving that problem will also allow the structure to adapt to an immutable environment: for instance, to build around a boulder or tree if that obstacle cannot be moved.

- In the formulation presented here, one robot can complete the entire task by itself if necessary; and N robots can potentially achieve an N-fold speedup purely through parallelism. Robots act independently and, except for considerations like collision avoidance, essentially ignore what the others are doing. However, there may be advantages that can be gained from explicit cooperation [31]. Tasks can be considered which one robot working alone could not perform, but multiple robots can accomplish together [19]. A coordination of effort could potentially allow a greater than N-fold speedup.

- Other problems may benefit from the use of extended stigmergy. It may well prove to be a useful framework for coordinating other multiagent systems which can affect their environment, with the potential for advantages like those shown here over the basic stigmergy that's been employed in foraging and other swarm applications.

Bibliography

[1] E. Eugene Allmendinger. A seafloor-based system for surveying and maintaining subsea oil-gas production complexes. Technical report, Engineering Design and Analysis Laboratory, University of New Hampshire, Durham, August 1975. EDAL Report No. 114.

[2] Daniel Arbuckle and Aristides Requicha. Active self-assembly. In *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, pages 896–901, New Orleans, Louisiana, 2004.

[3] Ronald Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[4] Erkin Bahceci, Onur Soysal, and Erol Sahin. A review: Pattern formation and adaptation in multi-robot systems. Technical Report CMU-RI-TR-03-43, Carnegie Mellon University, Pittsburgh, PA, USA, 2003.

[5] Karen Baker. Navigation and positioning systems: Finding your way through today's offerings. *UnderWater Magazine*, Winter 1997.

[6] Neil Barbour, John Elwell, and Roy Setterlund. Inertial instruments—where to now? In M. V. Jessick and D. L. Knobbs, editors, *Proceedings of AIAA Guidance, Navigation and Control Conference*, pages 566–574, August 1992.

[7] Robert Barish, Paul WK Rothemund, and Erik Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Letters*, 5(12):2586–2592, 2005.

[8] J. Bishop, S. Burden, Eric Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Self-organizing programmable parts. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005.

[9] Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press Inc., 1999.

[10] Eric Bonabeau, Guy Théraulaz, Jean-Louis Deneuborg, Nigel Franks, Oliver Rafelsberger, Jean-Louis Joly, and Stephane 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.

[11] Adrian Bowyer. Automated construction using co-operating biomimetic robots. Technical report, University of Bath Department of Mechanical Engineering, Bath, UK, 2000.

[12] R. A. Buswell, R. C. Soar, M. C. Pendlebury, A. G. F. Gibband F. T. Edum-Fotwe, and A. Thorpe. Investigation of the potential for applying freeform processes to construction. In *Proceedings of the 3rd International Conference on Innovation in Architecture, Engineering and Construction*, pages 141–150, Rotterdam, The Netherlands, January 2005.

[13] Johanna Darlington. A method for sampling the populations of large termite nests. *Annals of Applied Biology*, 104:427–436, 1984.

[14] Jacob Everist, Kasra Mogharei, Harshit Suri, Nadeesha Ranasinghe, Berok Khoshnevis, Peter Will, and Wei-Min Shen. A system for in-space assembly. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2356–2361, Sendai, Japan, 2004.

[15] John Feddema, Chris Lewis, and David Schoenwald. Decentralized control of cooperative robotic vehicles: Theory and application. *IEEE Transactions on Robotics and Automation*, 18(5):852–864, October 2002.

[16] John Feddema, Rush Robinett, and Raymond Byrne. An optimization approach to distributed controls of multiple robot vehicles. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, USA, 2003.

[17] Nigel Franks, Andy Wilby, Bernard Walter Silverman, and C. Tofts. Self-organizing nest construction in ants: sophisticated building by blind bulldozing. *Animal Behaviour*, 44:357–375, 1992.

[18] Jakob Fredslund and Maja Matarić. A general, local algorithm for robot formations. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, October 2002.

[19] Brian Gerkey and Maya Matarić. Pusher-watcher: an approach to fault-tolerant tightly-coupled robot coordination. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, pages 464–469, Washington, D.C., USA, 2002.

[20] Ivan Getting. The global positioning system. *IEEE Spectrum*, 30(12):36–47, December 1993.

[21] Seth Copen Goldstein, Jason Campbell, and Todd Mowry. Programmable matter. *Computer*, 38(6):99–101, June 2005.

[22] Seth Copen Goldstein and Todd Mowry. Claytronics: A scalable basis for future robots. In *Robosphere 2004*, Moffett Field, CA, November 2004.

[23] Pierre-Paul Grassé. La reconstruction du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[24] Saul Griffith. *Growing Machines*. PhD dissertation, Massachusetts Institute of Technology, MIT Media Laboratory, September 2004.

[25] Saul Griffith, Dan Goldwater, and Joseph Jacobson. Self-replication from random parts. *Nature*, 437:636, September 2005.

[26] Ying Guo, Geoff Poulton, Phil Valencia, and Geoff James. Designing self-assembly for 2-dimensional building blocks. In *ESOA'03 Workshop*, Melbourne, Australia, July 2003.

[27] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *ASPLOS-IX*, Cambridge, MA, 2000.

[28] B. Hoffmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice*. Springer-Verlag, New York, NY, 1997. fourth edition.

[29] Bert Hölldobler and Edward O. Wilson. *The Ants*. Belknap Press of Harvard University Press, Cambridge, Massachusetts, 1990.

[30] Chris Jones and Maja Matarić. From local to global behavior in intelligent self-assembly. In *Proceedings of 2003 IEEE International Conference on Robotics and Automation*, pages 721–726, Taipei, Taiwan, 2003.

[31] Chris Jones and Maja Matarić. Automatic synthesis of communication-based coordinated multi-robot systems. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 381–387, Sendai, Japan, 2004.

[32] Michael E. Jones. *Deepwater Oil Production and Manned Underwater Structures*. Graham & Trotman, London, 1981.

[33] Michael E. Jones. *Logistic Support of a Manned Underwater Production Complex*. Graham & Trotman, London, 1983.

[34] István Karsai and Zsolt Pénzes. Comb building in social wasps: self-organization and stigmergic script. *Journal of Theoretical Biology*, 161:505–525, 1993.

[35] Behrokh Khoshnevis. Automated construction by contour crafting—related robotics and information technologies. *Journal of Automation in Construction*, 13(1):5–19, January 2004. Special issue: The best of ISARC 2002.

[36] Behrokh Khoshnevis and George Bekey. Automated construction using contour crafting—applications on earth and beyond. *Journal of Rapid Prototyping*, 9(2):1–8, 2003.

[37] Eric Klavins, Robert Ghrist, and David Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 2006. To appear.

[38] Don Knuth. Dancing links. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millenial Perspectives in Computer Science*, pages 187–214. Palgrave, Houndmills, England, 2000.

[39] Keith Kotay and Daniela Rus. Algorithms for self-reconfiguring molecule motion planning. In *Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.

[40] Keith Kotay and Daniela Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[41] Manhattan Associates. RFID: The UPC of the 21st century. White paper, 2003.

[42] Zachary Mason. Programming with stigmergy: using swarms for construction. In *Proceedings of Artificial Life VIII*, pages 371–374, Sydney, Australia, 2002.

[43] James McLurkin. Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots. Master's project, Massachusetts Institute of Technology, May 2004.

[44] Chris Melhuish, Owen Holland, and Steve Hoddell. Collective sorting and segregation in robots with minimal sensing. In *5th Conference on Simulation of Adaptive Behaviour*, Zurich, Switzerland, 1998.

[45] Chris Melhuish, Jason Welsby, and Charles Edwards. Using templates for defensive wall building with autonomous mobile ant-like robots. In *Proceedings of Towards Intelligent Autonomous Mobile Robots 99*, Manchester, UK, 1999.

[46] Petter Ogren, Edward Fiorelli, and Naomi Leonard. Formations with a mission: Stable coordination of vehicle group maneuvers. In *Proceedings of 15th International Symposium on Mathematical Theory of Networks and Systems*, August 2002.

[47] Chris Parker, Hong Zhang, and Ronald Kube. Blind bulldozing: multiple robot nest construction. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, USA, 2003.

[48] Nissanka Bodhi Priyantha. *The Cricket Indoor Location System.* PhD dissertation, Massachusetts Institute of Technology, June 2005.

[49] Paul WK Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):e424, 2004.

[50] Daniela Rus and Gregory Chirikjian, eds. *Autonomous Robots*, 10(1):5–124, January 2001. Special issue on self-reconfigurable robots.

[51] Dylan Shell. A (partially) annotated bibliography of papers that make use of the word “stigmergy”. http://robotics.usc.edu/∼dshell/stigmergy.php, October 2003.

[52] Rupert Soar, Alistair Gibb, Tony Thorpe, and Francis Edum Fotwe. Application research for freeform construction processes. IMRC Proposal.

[53] Rupert Soar, Dennis Loveday, Weratunga Malalasekera, J. Scott Turner, and Henk Versteeg. 3D-mapping of Macrotermes michaelseni mounds and simulation of their homeostatic function: Lessons for human construction? EPSRC Proposal.

[54] William Spears and Diana Gordon. Using artificial physics to control agents. In *IEEE International Conference on Information, Intelligence, and Systems*, Bethesda, MD, USA, 1999.

[55] Kasper Støy and Radhika Nagpal. Self-reconfiguration using directed growth. In *Proceedings of Distributed Autonomous Robotic Systems 2004*, Toulouse, France, 2004.

[56] Kasper Støy and Radhika Nagpal. Self-repair through scale independent self-reconfiguration. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[57] Kazuo Sugihara and Ichiro Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[58] Yuzuru Terada and Satoshi Murata. Automatic assembly system for a large-scale modular structure: hardware design of module and assembler robot. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2349–2355, Sendai, Japan, 2004.

[59] Guy Théraulaz and Eric Bonabeau. Coordination in distributed building. *Science*, 269:686–688, 1995.

[60] Guy Théraulaz and Eric Bonabeau. Modelling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology*, 177:381–400, 1995.

[61] J. Scott Turner. A superorganism’s fuzzy boundaries. *Natural History*, 111(6):62–67, July-August 2002.

[62] J. Scott Turner. Extended phenotypes and extended organisms. *Biology and Philosophy*, 19:327–352, 2004.

[63] Peter van Emde Boas. The convenience of tilings. In Andrea Sorbi, editor, *Complexity, Logic and Recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Mathematics*, pages 331–363. Marcel Dekker Inc., 1997.

[64] Serguei Vassilvitskii, Mark Yim, and John Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, pages 117–122, Washington, DC, USA, 2002.

[65] Marsette Vona and Daniela Rus. Crystalline robots: self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[66] Abraham Warszawski and Ronie Navon. Implementation of robotics in buildings: current status and future prospects. *Journal of Construction Engineering and Management*, 124(1):31–41, 1998.

[67] Jens Wawerla, Gaurav Sukhatme, and Maja Matarić. Collective construction with multiple robots. In *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[68] Justin Werfel. Building blocks for multi-agent construction. In *Proceedings of Distributed Autonomous Robotic Systems 2004*, Toulouse, France, 2004.

[69] Justin Werfel, Yaneer Bar-Yam, and Radhika Nagpal. Building patterned structures with robot swarms. In *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[70] Justin Werfel, Yaneer Bar-Yam, and Radhika Nagpal. Construction by robot swarms using extended stigmergy. AI Memo AIM-2005-011, MIT CSAIL, Cambridge, MA, 2005.

[71] Justin Werfel, Yaneer Bar-Yam, Daniela Rus, and Radhika Nagpal. Distributed construction by mobile robots with enhanced building blocks. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, Orlando, USA, 2006.

[72] Justin Werfel and Radhika Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21(2):20–28, March-April 2006.

[73] Paul White, K. Kopanski, and Hod Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, pages 2888–2893, New Orleans, Louisiana, 2004.

[74] Paul White, Victor Zykov, Josh Bongard, and Hod Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics: Science and Systems I*, Cambridge, MA, 2005.

[75] Randall Wilson and Jean-Claude Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1995.

[76] Mark Yim, Ying Zhang, and David Duff. Modular robots. *IEEE Spectrum*, 39(2):30–34, 2002.

[77] Junku Yuh, Tamaki Ura, and George Bekey, editors. *Underwater Robots*. Kluwer Academic, Boston, 1996. Reprinted from a special issue of Autonomous Robots, vol. 3, nos. 2 & 3, June 1996.

[78] Robert Zlot, Anthony Stentz, M. Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation*, May 2002.