Building Patterned Structures with Robot Swarms

Justin Werfel[1,2] jkwerfel@mit.edu
Yaneer Bar-Yam[2,3] yaneer@necsi.org
Radhika Nagpal[3] rad@eecs.harvard.edu

[1]MIT CSAIL, Cambridge, MA 02139
[2]New England Complex Systems Institute, Cambridge, MA 02138
[3]Harvard University, Cambridge, MA 02138

Abstract

We describe a system in which simple, identical, autonomous robots assemble two-dimensional structures using prefabricated modules as building blocks. Modules are capable of some information processing, enabling them to share long-range structural information and communicate it to robots. This communication allows arbitrary solid structures to be rapidly built using a few fixed, local robot behaviors. Modules are identical in shape but may be functionally distinct, with constraints governing the location of different classes. We present algorithms for assembly of solid structures of any shape, both when the layout of module classes is fully specified in advance, and when functional constraints are satisfied during the building process, allowing for adaptive structures. This approach demonstrates a decentralized, autonomous, flexible, simple, and adaptive approach to construction.

1 Introduction

In this paper we discuss the design of a system for automated construction, in which elements are separated into mobile and structural components (robots and modular blocks, respectively). Our approach is to use robots that are simple in capabilities and behavior, and that communicate indirectly through information stored in the environment; that information is embodied by the blocks, which themselves can communicate with one another when attached. Here we focus on construction when the blocks are identical in shape but heterogenous in function, and a given set of functional constraints governs how they can be put together. We first show how specific structures can be built, and then discuss the assembly of structures which have no prespecified plan but satisfy a given set of constraints.

Our approach relies on very simple procedures and therefore should be practical for the complexities and difficulties of real robots in physical environments. The limited requirements for implementation are that robots be able to follow a perimeter, communicate with immediately adjacent blocks, and push together blocks with self-aligning connectors. 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.

The ability to automate construction would be useful particularly in settings where human presence is dangerous or problematic; for instance, robots could be initially sent to underwater or extraterrestrial environments, to create habitats to await later human travelers. Swarm approaches, involving larger numbers of simpler robots rather than one or a few with greater sophistication, have advantages for such a goal, in particular with respect to decentralization and robustness. Such systems can typically absorb the loss of many components without a significant impact on task completion; similarly, they tolerate components acting in no specified order, a useful quality since it is difficult to preplan robot behavior in detail in uncertain environments.

If blocks can be arbitrarily custom-designed for particular structures, then any structure can be built using only local rules for assembly; a trivial, inefficient way to do it is 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.[1] However, we take the opposite approach, assuming that the block types are defined by the application, and our task is to work within those limits.

An example of such an application is the building of underwater structures (for marine research, oil drilling, etc.), where the building blocks are high-level prefabricated pods that each fill one of several distinct and predefined roles—living quarters, power generation, emergency escape centers, laboratory space, etc. There may be desired constraints on which classes are attached relative to which other classes—for instance, using the previous example, we might want all living pods to be located in a contiguous block, or no two airlock pods to be adjacent to one another.

In §2 we review related work, and in §3 describe the problem we address and our assumptions about the system. §4 gives algorithms the system can use to assemble structures. §5 covers geometric constraints on block placement. §6 and §7 describe structures which are fully prespecified or adaptive, respectively, and elaborate on the satisfaction and specification of functional constraints. §8 discusses issues associated with the use of multiple robots.

[1]Geometric constraints will still restrict the order in which blocks may be attached, as discussed in §5.

2 Related work

Several previous papers address topics related to construction, though not with the goal of producing user-specified building designs. For example, [Wawerla *et al.*, 2002; Jones and Matarić, 2004] study aspects of the use of communication among robots, to increase their effectiveness on tasks involving rearrangement of functionally heterogenous blocks. [Bowyer, 2000] discusses issues of mechanical design for robots meant to build arches, towers, and walls, using extruded foam. [Melhuish *et al.*, 1999] deals with minimization of capabilities for behavior-based robots arranging pucks into walls, using environmental cues.

[Werfel, 2004] describes a framework for arranging blocks into arbitrary lines and curves, specified geometrically; [Mason, 2002] describes one in which structures can be specified through the use of static, global 'pheromone' signals. Both make assumptions about the precision with which their mobile agents operate (the former having to do with odometry, the latter with ability to evaluate the strength of distant signals) that may cause the shapes they generate to suffer from lack of robustness to noise.

Our approach is particularly motivated by research in two areas: insect and insect-inspired construction [Théraulaz and Bonabeau, 1995], and self-reconfigurable modular robotics [Rus and Chirikjian, 2001]. Social insects which build do so with the principle of stigmergy, communicating indirectly by storing information in the environment. While this approach can be used to produce structures with given qualitative characteristics, it does not easily allow the consistent production of *specific* structures (potentially arbitrary and complicated ones); nor has a general principle been described for taking a particular desired structure and extracting a set of low-level behaviors that building agents can follow to produce it. Work in modular robotics has produced hardware systems with many capabilities we will take advantage of here, including connections that self-adjust so that inexact alignment is corrected, and communication links between physically attached modules. Such communication is reliable, unambiguous, rapid, and scalable, compared to the communication by external signaling that robots typically use. Modular robot algorithms typically require that all modules remain connected at all times, and that individual modules be capable of mobility, neither of which are appropriate for automated construction applications. Very recently, modular robotics researchers have begun to describe hardware systems for automated assembly of structures [Terada and Murata, 2004; Everist *et al.*, 2004]. These systems are based on inert, identical modules, and have not yet explored issues of specifying distributed robot behavior and creating complex user-specified structures.

Elsewhere we describe a similar system to that presented here [Werfel *et al.*, 2005], concerning identical blocks only. There we show that giving blocks limited communication abilities is an effective way to enable the system to construct arbitrary solid structures, while allowing robots to remain simple and with fixed, local behaviors. The block communication provides the nonlocal knowledge about the structure necessary to assemble arbitrarily complicated shapes.

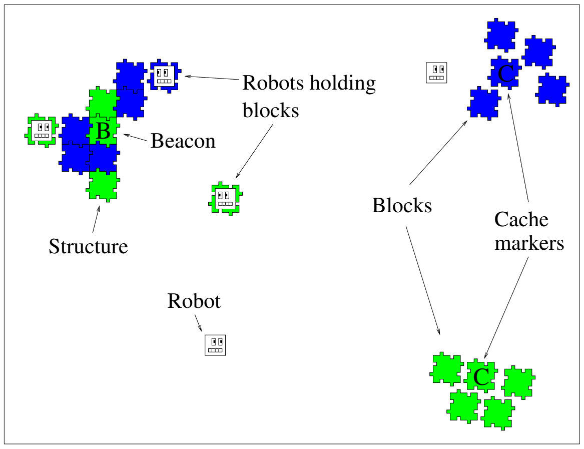

Figure 1: A sketch of the kind of system described here. As the structure is assembled, it forms a lattice with an implicit coordinate system, of which the beacon can be taken to be the origin.

This approach provides other benefits, including increased efficiency of construction, alleviation of interference between robots, and better exploitation of the parallelism of the swarm, without requiring additional capabilities from the robots. As computation becomes less expensive, it has been proposed to make it more pervasive [Hill *et al.*, 2000]. Particularly if the building blocks in question are high-level units, as in the underwater application suggested above, it should be feasible to incorporate these computation capabilities into the blocks without making the system prohibitively more expensive.

3 General problem and assumptions

We consider the problem where mobile robots and caches of building blocks are deployed at random into an obstacle-free workspace, along with a beacon indicating the location for the start of construction. The goal is for robots to collect blocks from the caches and arrange them into a structure, satisfying some set of criteria, starting at the beacon (Fig. 1). The beacon and caches send out distinct long-range signals which robots can use to navigate to them. Each cache contains a single block type.

We will work with square blocks, to be assembled in the horizontal plane. An important constraint is that a block can be added to the growing structure if and only if the potential attachment site has at least two adjacent sides open; otherwise there is insufficient room to maneuver to add the new block (Fig. 2). Though a system has been demonstrated in which cubical blocks can be slid into spaces like that at (D) in that figure [Terada and Murata, 2004], the task of mechanical design will be simplified and the precision with which robots must operate reduced if we maintain this constraint. In particular, if we prevent gaps like that at (D), then situations where a robot needs to maneuver a block down a longer 'tunnel', like that around (C), will also be prevented. A block attached to the structure can immediately obtain from its neighbors its

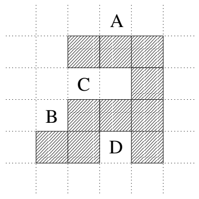

Figure 2: Examples of valid and invalid prospective docking sites for a sample structure (shaded squares represent grid sites already occupied by blocks). A new block can be attached at A or B; C and D are too spatially constrained to allow a block to be maneuvered into position.

position in a common coordinate system, along with information about the present and desired final structure. Blocks along the perimeter of the structure and robots traversing the perimeter can also communicate, with lower bandwidth; this communication is very short-range, avoiding problems associated with signaling over distances, interference when many robots are active, and ambiguity in which agents are signaling.

Robots can move in any direction in at least the two dimensions of the plane, alone or while holding a block. They can detect the presence of other robots, for collision avoidance, either by active short-range signaling or by passive perception. Once in the immediate vicinity of the growing structure, they can follow its perimeter, and communicate with adjacent blocks.

4 Algorithms for blocks and robots

Blocks attached to each other to form the structure have access to information about the entire structure, via communication with other blocks. Our approach will be for blocks to be the agents responsible for ensuring that the structure remains consistent with all the constraints. Robots can then simply circle the perimeter of the structure, and rely on it to tell them when a site is acceptable for attachment of their payload.

We can divide the constraints on block placement into two classes: those independent of block type, and those based solely on block type. Geometric constraints, the former, ensure that the 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. Both classes are discussed further below (§5–7).

Algorithm 1 summarizes high-level procedures for robots and blocks to follow to build structures using this approach.

Algorithm 1 High-level pseudocode procedure for assembly of a close-packed structure by a single robot.

```
A: Robots
  while structure not complete do
    choose a block type $t$
    get $t$ from cache
    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
        return block to cache
      else
        move one site counterclockwise along perimeter
B: Blocks
  loop
    if neighboring robot asks to attach a block of type $t$ at site $s$ then
      if (block at $s$ consistent with geometric constraints) and ($t$ at $s$ consistent with functional constraints) then
        answer yes
        update map with $t$ block at $s$
        pass message about $(t, s)$ to neighboring blocks
      else
        answer no
    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
```

4.1 Robustness

Physical robots in real-world environments are subject to a wide variety of internal and environmental perturbations. A hardware implementation would need to be robust to such factors. Our approach is intended 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: directed movement, block pickup, perimeter following, block joining, limited communication with immediate neighbors.

Noise affecting robot movement, e.g., due to motor unreliability or currents in an underwater setting, can be compensated for. For instance, if robots lose track of the perimeter during their tour, they need only revert to a previous step of their algorithm, find the perimeter again and proceed from there. The lack of robot internal state means no important information is lost in such an event. Self-aligning connectors drawn from modular robotics effect the fine details of block attachment. Communication between robots and blocks can be physically limited, e.g., through appropriate geometric design of an optical interface, to ensure that robots only communicate with immediately neighboring blocks and that crosstalk with more distant blocks is prevented. Interblock communication is via a physical interface.

More serious robot failures can also be absorbed. 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.

Blocks, less complicated mechanically, may be less subject to failure. Malfunctions that do occur could be detected by neighboring blocks, which could alert robots to the need for replacement. Similarly, in the event that a block is mistakenly attached to the structure so as to violate a constraint, the active nature of the blocks makes it easier to detect and report that error. §9.2 outlines a scheme for disassembly and reconstruction of the necessary part of the structure.

5 Geometric constraints

Positions can be specified by reference to the common coordinate system which blocks share via their communication.

5.1 Rectangular structures

The simplest possible perimeter is a rectangle; the geometric constraint enforcing that shape is simply a restriction against robots attaching blocks at sites with coordinates outside the desired rectangle.

The other geometric constraint to be followed is the generally applicable one that the structure should be solid, without internal holes. To avoid unwanted gaps, it is necessary to avoid situations where two blocks end up in the same row, unless they are adjacent. A block can check, based on its communication with the other blocks in its row [Werfel *et al.*, 2005], whether a potential attachment site satisfies this criterion, and forbid robots from attaching further blocks at that site if it does not. This rule is sufficient to generate solid structures.

5.2 Complex shapes

With only a small modification, the system extends to solid structures with perimeters which are not necessarily rectangular, nor even convex, but can be of any arbitrary shape.[2] As usual, the desired shape is specified with respect to the shared coordinate system and downloaded by each new block attached to the structure. To achieve the extension to arbitrary shapes without letting unwanted gaps creep in, it is sufficient to relax the geometric rule against attaching two non-adjacent blocks in the same row, to allow two such blocks so long as they are separated by space intended never to be occupied by blocks. The desired shape can be represented compactly, e.g., as a collection of rectangles whose superposition gives the desired structure [Støy and Nagpal, 2004]. Fig. 3 gives an example of an H-shaped structure being built, with geometric constraints only.

6 Functional constraints: Patterned structures

An important aspect of such a construction system, hitherto unaddressed, 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

[2]We will require that any spaces between blocks in the desired final structure be wide enough for two block-carrying robots to pass each other, so that perimeter-following can continue unimpeded.

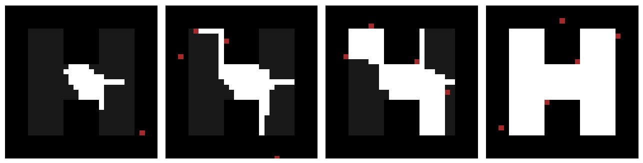
Figure 3: Snapshots of five robots (brown) building a structure of a single block type (white), using only geometric constraints. Sites that the structure plan specifies should eventually be occupied by blocks are shown in dark gray. Robots not appearing in a particular frame are off fetching blocks.

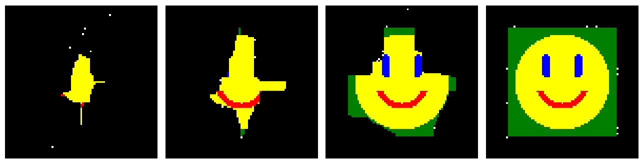
Figure 4: Snapshots of ten robots (white) building a structure of four block types (yellow, red, blue, green).

should be attached at every location in the desired structure. This specification can be more compact than a full enumeration, just as an image specification can be more compact than a bitmap. Structure blocks forbid robots to attach the blocks they carry unless the latter are the single type allowed at the site in question. Figs. 4 and 5 shows rectangular and nonrectangular structures built using such functional constraints.

7 Functional constraints: Adaptive structures

In many cases the design of structures does not require specific patterns, but rather requires functional relationships between the locations of blocks of various types. For instance, with the underwater habitat example we gave above, it may be that the exact locations of escape pods are unimportant, but safety considerations require that no pod be further than three steps away from one. Any structure that satisfies that constraint will be considered acceptable. The more relevant way to specify the constraint in this case is by reference not to absolute coordinates, but to relationships between the block types. Our approach extends naturally to building structures in this flexible way. Figs. 6 and 7 show structures built using relative constraints of this sort.

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.

Satisfying constraints on the fly can have additional benefits, such as increased speed of construction. As an example, consider the class of 21 $\times$ 21-block structures of Fig. 6. If we assume that robots take no time to bring blocks from caches to the structure perimeter, and moving one block-length and attaching a block each take one time step, then the time required to build a structure will reflect primarily the

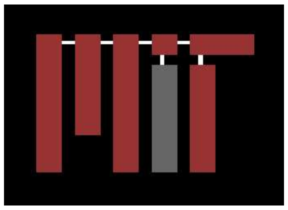
Figure 5: Example of a patterned structure with a complex shape.

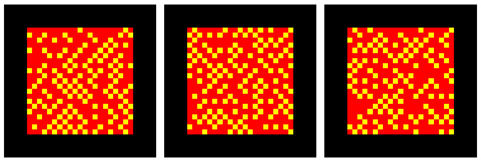
Figure 6: Examples of square structures built with two block types and the constraint that no two yellow blocks be adjacent.

time robots spend following the perimeter looking for a place to put 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.

7.1 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 [van Emde Boas, 1997]. 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.

7.2 Types of functional constraints

In general, there are several classes of functional constraints we can consider:

- *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.

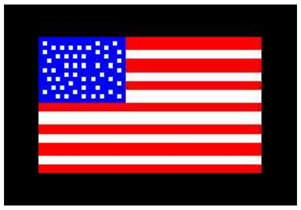
Figure 7: Example of a structure assembled according to the following list of constraints:
Region 1, border of upper-left area: only blue blocks allowed.
Region 2, interior of upper-left area: blue and white blocks both allowed; no white block may be in the eight-neighborhood of another white block.
Region 3, elsewhere: blocks must be either red or white depending on their y-coordinate.

- *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 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 functional constraint can result from finite numbers of each block type. 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 added anywhere, but the second would have to be 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. Having blocks maintain a global map of the structure would be the most straightforward way to ensure being able to handle all possible constraints, though not the most memory-efficient.

7.3 Multiple levels

Constraints may also be specified on multiple levels of organization. We can define different *regions* of the structure, in each of which some different set of constraints as listed above is to be imposed. Figs. 4 and 7 give examples where such regions are designated according to specific coordinates. The former designates regions compactly by reference to geometric shapes and lines (e.g., a region of red blocks is defined as those cells within a given distance from an arc with given center, radius, and endpoints, etc.). The latter combines patterned 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

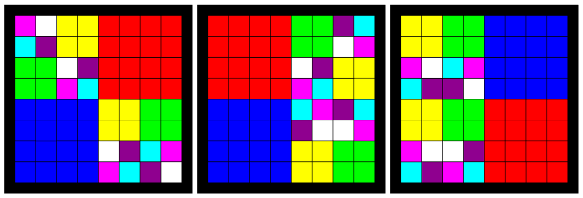

Figure 8: Three examples of structures that can all equally well be built based on the constraints given in the text.

in the structure, with variable size, aspect ratio, orientation, multiplicity, etc. Just as blocks can have proximity and relational constraints 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 1 must be larger than region 2).

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 8 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 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.

7.4 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 Figs. 4–7 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 Fig. 7 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.

This looking ahead may be limited to a fixed number of steps f into the future. Like functional 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.

8 Multiple robots

Algorithm 1 gives a procedure which will work for a single robot. However, when more than one robot is working on construction at once, additional precautions need to be taken. Finite speed of message propagation between blocks means that two robots can request to attach at mutually exclusive sites at about the same time, and both can be given permission by blocks working with outdated information.

The violation of geometric constraints (two non-adjacent blocks being attached in the same row) can be prevented, e.g., by sending messages to lock out a previously empty row when a robot requests to attach a block in it, and not giving the robot permission until all blocks in the adjacent row have indicated they have not and will not let other robots 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. 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 [Werfel *et al.*, 2005]. 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 which is only detected later, the blocks can direct robots to correct the error by removing the offending blocks, as described in §9.2 below.

9 Additional construction procedures

We briefly consider how two related procedures, disassembly and error correction, could be achieved with such a system.

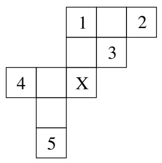

Figure 9: A structure in the process of disassembly. Numbered blocks can be removed. The block marked X meets criterion 1 in the text but not 2; if removed, it would split the structure into two parts. The beacon may be at the location of any of the unmarked blocks.

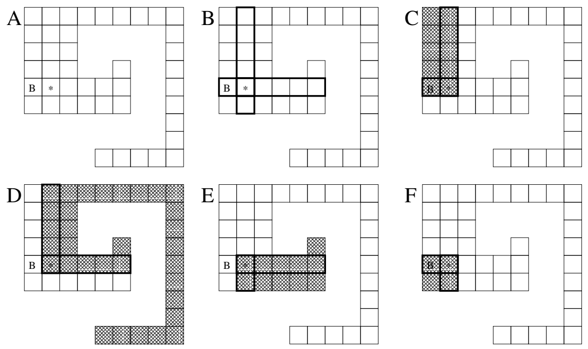

Figure 10: Error correction in a compound structure. The beacon is marked with the letter B; the block to be replaced is marked *. See text for discussion.

9.1 Disassembly

Many applications will involve structures intended to be temporary; the modular nature of this system is especially suited to such cases, allowing a structure to be dismantled following use and its parts reused 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. It has at most two neighbors, bordering adjacent sides; i.e., if that site were unoccupied, it could physically accomodate a block, as in Fig. 2.
2. If it does have two neighbors along adjacent sides, then the other site which those two neighbors border is occupied by a block (see Fig. 9).
3. It is not attached at the beacon location, unless no other blocks remain; the block at the beacon must be the last one removed.

Again, blocks can determine whether they satisfy these criteria, and indicate that to robots. The second and third criteria ensure that throughout disassembly, the structure remains one contiguous piece, which a robot homing in on the beacon signal will be able to find. Violating either of those rules can result in part or all of the remaining structure becoming isolated, and thereby lost to robots no longer able to reach it by the simple procedure of following the beacon signal to the structure and following the perimeter thereafter. If the structure is physically tethered to some anchor point via the first block attached (as would likely be the case, e.g., in 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 three criteria, but would violate criterion 2 if the other block were removed. Blocks 1 and 3 in Fig. 9 illustrate such a pair.

9.2 Error correction

Suppose that a block mistakenly ends up attached at a location where it should not be, or needs to be replaced due to malfunction or some other reason. It may be some time before this error arises or is detected; we will thus consider the problem of replacing any arbitrary block in the structure. We outline the high-level approach and omit elaboration of an algorithm to be carried out by individual blocks.

Because of our 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. Fig. 10 demonstrates the demarcation of quadrants. (A) shows a completed compound structure; the beacon is marked with the letter B, and the 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 in a compound structure, 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, the block attached at the beacon 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 the beacon (as does the quadrant in (C)). The quadrant chosen for removal will be that in (E).

Blocks in the quadrant chosen forbid robots from attaching any further blocks to them, and ask to be removed. Upon receiving such a request, some robots may switch their role temporarily from 'assembly' to 'disassembly'. 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.

10 Conclusions

We have demonstrated that this approach, of building structures with communicating blocks and multiple simple robots, can achieve a variety of construction objectives. Structures can be built with perimeters of any shape, and with prespecified patterns of block types or adaptive satisfaction of constraints. Other approaches using swarms of simple robots have demonstrated the ability to create structures which satisfy locally specified constraints [Théraulaz and Bonabeau, 1995; Melhuish *et al.*, 1999]. However, this system with communicating blocks is unique in its ability to handle fully specified and locally adaptive constraints alike.

We are currently working on implementing a version of this system using the AMOUR robot of the Rus Robotics Lab at MIT [Vasilescu *et al.*, 2005].

Acknowledgments

We thank Daniela Rus, Jason Redi, and Jake Beal for useful discussions. This work was supported by NSF grant EIA-0130391.

References

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

[Everist *et al.*, 2004] 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.

[Hill *et al.*, 2000] 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.

[Jones and Matarić, 2004] 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.

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

[Melhuish *et al.*, 1999] 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.

[Rus and Chirikjian, 2001] Daniela Rus and Gregory Chirikjian, eds. *Autonomous Robots*, 10(1):5–124, 2001.

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

[Terada and Murata, 2004] 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.

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

[van Emde Boas, 1997] 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.

[Vasilescu *et al.*, 2005] Iuliu Vasilescu, Paulina Varshavskaya, Keith Kotay, and Daniela Rus. Autonomous modular optical underwater robot (AMOUR): design, prototype and feasibility study. In *Proceedings of 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005.

[Wawerla *et al.*, 2002] 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.

[Werfel *et al.*, 2005] 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.

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