

# Adaptive Behavior

<http://adb.sagepub.com>

---

## Construction in a Simulated Environment Using Temporal Goal Sequencing and Reinforcement Learning

Anand Panangadan and Michael G. Dyer

*Adaptive Behavior* 2009; 17; 81

DOI: 10.1177/1059712308101787

The online version of this article can be found at:  
<http://adb.sagepub.com/cgi/content/abstract/17/1/81>

---

Published by:



<http://www.sagepublications.com>

On behalf of:



[International Society of Adaptive Behavior](#)

**Additional services and information for *Adaptive Behavior* can be found at:**

**Email Alerts:** <http://adb.sagepub.com/cgi/alerts>

**Subscriptions:** <http://adb.sagepub.com/subscriptions>

**Reprints:** <http://www.sagepub.com/journalsReprints.nav>

**Permissions:** <http://www.sagepub.co.uk/journalsPermissions.nav>

**Citations** <http://adb.sagepub.com/cgi/content/refs/17/1/81>

# Construction in a Simulated Environment Using Temporal Goal Sequencing and Reinforcement Learning

Anand Panangadan<sup>1</sup>, Michael G. Dyer<sup>2</sup>

<sup>1</sup>*Saban Research Institute, Childrens Hospital of Los Angeles*

<sup>2</sup>*Computer Science Department, University of California Los Angeles*

A behavior-based architecture (*ConAg*) with a connectionist action selection mechanism is introduced that enables a society of autonomous agents to construct arbitrary structures in their simulated two-dimensional world. Construction in this environment involves the agents picking up colored discs and dropping them at incomplete parts of the structure being built.

The *ConAg* architecture provides both reactive behaviors which are used to maintain the viability of the agent and navigational planning behaviors that are used for construction. The action selection mechanism enables learning the sequence of behaviors required for construction by reinforcement learning. The navigational planning behaviors use a grid-based representation of the world. The shape of the structure to be built is also encoded on an internal spatial map. Path planning is implemented by spreading activations on sets of grid-based maps so that the agents perform the construction task efficiently. Construction of arbitrary structures is supported by temporal sequencing of goals. We present simulation results that demonstrate the performance of the architecture and algorithms.

**Keywords** construction · reinforcement learning · spatial map · spreading activation

## 1 Introduction

As robot technology improves in the future, robots will be expected to exist autonomously and provide services that are useful to man. Construction and repair of physical structures will be one of the most important of such tasks. Artificial systems that can construct large-scale structures in the real world do not exist yet. However, ants (and other social insects) build complex structures, though each ant only has limited perception and there is no centralized control structure. Moreover, the behaviors exhibited by these creatures have useful qualities such as being robust to unexpected changes to

the structures being built and to failures of individual insects (Bonabeau, Theraulaz, Deneubourg, Aron, & Camazine, 1997; Bonabeau et al., 1998).

Researchers have used some of the features of these natural systems for designing construction mechanisms in simulated 2- and 3-dimensional environments (Bonabeau, Guerin, Snyers, Kuntz, & Theraulaz, 2000; Theraulaz & Bonabeau, 1995a). These systems are predominantly *reactive*, that is, the individual agents behave by following fixed rules that depend only on a limited view of the structure being built. For instance, agents can only see the contents of their neighboring cells in a grid world. The shapes of the

*Correspondence to:* Anand Panangadan, Saban Research Institute, Childrens Hospital of Los Angeles, 4650 Sunset Blvd, MS 139, Los Angeles, CA 90027, USA. *E-mail:* anandvp@usc.edu  
*Tel.:* +1 323-361-2413 *Fax:* +1 323-361-3512

Copyright © 2009 International Society for Adaptive Behavior (2009), Vol 17(1): 81–104.

DOI: 10.1177/1059712308101787

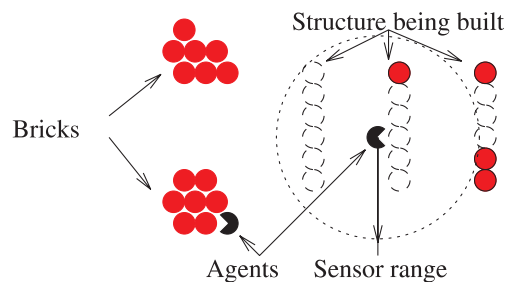
Figures 1, 4–6, 11–13, 17–18 appear in color online: <http://adb.sagepub.com>

structures in these simulated environments correspond to those built by insects such as pillars and walls (Bonabeau et al., 1998). Researchers have also used this approach for construction in the physical world using robots with limited sensory capabilities (Jones & Mataric, 2003; Stewart & Russell, 2006; Werfel, 2004; Werfel, Bar-Yam, Rus, & Nagpal, 2006). The chief issue in these works is determining a set of reactive rules which on repeated application will lead to the agents/robots arranging objects in the environment to a desired shape.

However, restricting agents to being reactive causes some significant drawbacks in the resulting construction mechanism. When construction relies only on immediate sensory feedback it is not clear what individual behaviors will collectively lead to a particular structure. This makes it difficult to design a group of agents for arbitrary structures. Moreover, structures built using only local rules are sensitive to the initial layout of the environment and the number of agents themselves (Deneubourg, Theraulaz, & Beckers, 1992; Theraulaz & Bonabeau, 1995a). Also, learning in such systems is not effective because behaviors learned while building one structure may not be applicable to others. Greedy heuristics that are used in stigmergetic systems lead to good solutions only for certain problems such as collectively finding shortest paths in an ant colony (Dorigo, Maniezzo, & Coloni, 1996). In the case of construction, a greedy approach leads to inefficient construction for certain shapes. For instance, if the desired shape consists of a structure enclosed within another, then building the outer structure first will block access to the inner structure.

Our hypothesis is that providing agents with an explicit spatial representation leads to a construction mechanism that alleviates these drawbacks and hence has significant advantages over purely behavior-based approaches. In this article we present the *ConAg* (for Construction Agent) series of architectures. We will show that implementation of this architecture enables a group of autonomous agents to construct arbitrarily shaped structures in an artificial 2-dimensional (2-D) world. We will also describe how this architecture facilitates learning of construction behaviors and path planning for efficiently completing the construction task.

The *ConAg* architecture is behavior based with connectionist action selection and is coupled with an internal spatial representation. The environment con-



**Figure 1** Two agents build three “walls” from the two clusters of discs to the left of the walls. The sensor range is not long enough for the agent at the right to sense the two sources of bricks. The agent at the left is close enough to a brick to pick it up.

tains only colored discs. Agents can move in this environment and sense discs around them. They can grab and drop a disc, and carry the disc as they move. Construction in this environment thus involves arranging these discs to form a specific 2-D pattern (also referred to as *structures* or *configurations*). An example is shown in Figure 1. Although highly abstracted from the real world, the sensors and effectors on the simulated “robots” (called *agents*) resemble those found in current physical robots. The internal spatial representation is grid based and also stores a blueprint of the structure to be built. This blueprint is a bird’s-eye view of the structure to be built and it is easy for a human to describe the structure in the form required by the agent (compared with the case where a procedural description of building the structure has to be provided). Moreover, any arbitrary 2-D shape can be specified. The advantage of separating the structure representation from agent behaviors is that behaviors learned while building one kind of structure can be reused to build other structures by just changing the blueprint provided to the agent. The connectionist action selection mechanism enables the agent to learn the sequence of construction steps by imitating a “teacher” agent and also to learn correlations between behaviors. We implemented path planning on the spatial maps using the biologically inspired method of spreading activations. In this approach, goal locations receive highest activation which is then spread to neighboring nodes (except those representing obstacle locations). The shortest path to a goal location is determined by following the local gradient at a node. This method is also utilized in other computing areas inspired by nat-

ural systems such as ant colony optimization (Deneubourg et al., 1991; Dorigo et al., 1996). Spreading activation is attractive from an implementation standpoint as the algorithm can be executed in parallel. We have also developed an algorithm that utilizes the spatial maps and the blueprint of the structure to be built to plan an efficient order of moving objects, that is, objects should not impede paths to objects that are yet to be moved. The algorithm performs this computation by using spreading activation in a novel way (multiple activations are spread simultaneously and their interactions determine shortest paths).

We first describe existing approaches to robot construction in Section 2. We then describe our simulation environment in Section 3. Section 4 describes the architecture of the agent and illustrates the use of this architecture to perform a construction task. Section 5 describes a reinforcement-learning procedure that can learn sequences of navigational planning behaviors. Section 6 describes a planning algorithm that uses the spatial representation to determine the order in which discs are to be moved in order to complete the construction task efficiently. Each section gives simulation results illustrating the performance of the presented techniques and a discussion of these results. The experiments are conducted in simulation since it is difficult to study learning in physical systems because of the time it takes to complete a robot trial.

## 2 Related Work

Researchers have invented mathematical models to describe how purely local interactions between certain social insects (e.g. wasps and ants) and their immediate environment result in the construction of large physical structures. These mathematical models abstract out only the features most relevant to construction: the movement abilities of insects, deposition and decay of pheromones, and effects of wind (Bonabeau et al., 1998; Ladley & Bullock, 2005). The ability of these social insects to build and maintain physical structures in their spatial environment has inspired researchers to design artificial systems with similar capabilities in simulation or with physical robots. Many of these artificial systems provide individual construction agents with a predominantly reactive architecture with no explicit spatial maps. Theraulaz and Bonabeau (1995a,

1995b) demonstrated a 3-D grid world where simulated agents built structures that resembled those built by wasps. The agents used only local stimulus–response rules in their work. Bonabeau et al. (2000) provide fitness functions that can be used to analyze the types of structures that result from such local rules. They state that there was an upper limit to the complexity of the structures that resulted from this approach. In general, one of the main challenges of this approach is that of determining local behaviors that lead to the arrangement of objects into a specific shape. Jones and Mataric (2003) describe an algorithm that generates local rules from the pattern of the structure to be built. In their approach, the shape of the structure is utilized only during the offline rule generation phase. Thus their system cannot accommodate any subsequent changes in the desired shape. Moreover, not all shapes can give rise to a correct set of local rules for construction. Howsman, O’Neil, and Craft (2004) describe construction in a similar simulated grid world in 3-D. In that work, the local rules were provided by the researchers. Werfel et al. (2006) describe a construction task in a simulated grid world containing entities similar to the construction agents and bricks described in our work. Their work presented algorithms that generated local rules that would lead to the arrangement of tiles into desired 2-D patterns. Unlike our work, the pattern shape was used in the rule-building stage only and the rules were dependent on relabeling of the tiles. A distinguishing characteristic of construction using the local rules approach from that of our internal spatial representation-based approach is that of efficiency of completing the construction task. Agents relying on local rules typically perform an *explore* action until a location where one of the rules becomes applicable is found. Thus, the structure may be completed in a less optimal manner than if explicit path planning was used (as illustrated by the example in Figure 13). If all of the agent’s behaviors rely only on local sensor information, then agents have to *tag* the environment in order to keep track of the construction of a large structure. Forms of tagging include painting discs (Crabbe & Dyer, 1999) and assigning state variables to building objects (Jones & Mataric, 2003). Modifying the environment is required if parts of the structure to be built cannot be distinguished from local sensor information (perceptual aliasing). The use of a large internal spatial representation in the ConAg architecture removes the need for environment-marking behaviors in our con-

struction agents. Theoretical studies of the complexity of construction in 2-D environments using only local rules are available in Adleman, Cheng, Goel, and Huang (2001) and Rothmund and Winfree (2000).

Researchers have demonstrated physical systems in which a robot can manipulate specially designed objects. Wawerla, Sukhatme, and Matarić (2002) demonstrated a real-world system where a robot with a behavior-based controller could arrange colored cylinders in a line. Stewart and Russell (2006) showed how a distributed collection of robots could build even more complex shapes by enabling one of the mobile robots (the organizer) to provide a time-varying environmental cue (a gradient of light) to the other robots (builders). Werfel (2004) demonstrated how a radio beacon could provide the time-varying environmental cue that robots use to coordinate their behaviors and lead to the construction of complex shapes in a simulated 2-D environment. Terada and Murata (2004) proposed the use of regular hexahedrons with genderless and rotation invariant connectors in every face as a building block. These blocks could then be manipulated by an assembler robot. Kotay, Rus, Vona, and McGray (1998) proposed a design for a 3-D building block (named a *molecule*) that is even capable of rotations around a fixed axis. The molecule is capable of moving among other similar units to form larger structures.

The construction task is related to that of self-reconfiguration in modular robots (Castano, Behar, & Will, 2002; Kotay & Rus, 1999). In reconfiguration, the shape refers to that of the agent/robot itself and possible configuration changes are limited by the physical constraints of the robot body. Cellular robots (CEBOTS) are composed of *cells*, each of which has its own sensors and performs actions such as moving and bending (Kawauchi, Inaba, & Fukuda, 1993). The cells can communicate with one another and physically couple with other cells to perform different manipulation tasks. CONRO robots are reconfigurable robots that are composed of small, inter-connecting, homogeneous modules (Castano, Shen, & Will, 2000).

The simulation environment used in this work is based on that introduced by Crabbe and Dyer (1999). That work demonstrated a society of autonomous agents that work together to build simple structures such as walls and briar patches in a 2-D world containing colored discs. The architecture of the agents in that work is completely neural and the agents do not maintain an internal model of the world. The lack of a

spatial representation causes small changes in the environment to lead to significant changes in the shape of the structure being built. Brooks, Maes, Matarić, and Moore (1990) presented a society of simulated agents that jointly perform tasks such as digging trenches and collecting rocks which might be useful in planetary base construction. The approach used is inspired by insect colonies as the agents are completely distributed and do not communicate with each other.

Architectures have been proposed to enable a robot or a small group of robots to manipulate their environment in a limited form. The Control Architecture for Multi-robot Planetary Outposts (CAMPOUT) is a distributed architecture used to coordinate the actions of two mobile robots that together carry a beam over rough terrain (Pirjanian et al., 2000; Schenker et al., 2000). CAMPOUT contains both a deliberative component for task-level planning (higher level actions) and a reactive component for executing lower level behaviors that require close interactions between sensors and motors. Matarić, Nilsson, and Simsarian (1995) presented two autonomous robots that, with the help of communication, push a box together toward a goal region. Khatib (1999) described an architecture that allows groups of mobile arm robots to interact with each other or with a human to do construction or building tasks. The main focus of these works is the closely coupled coordination required between a small number of agents. The effect of the agents' actions on the environment (such as where exactly a box has to be pushed to) is not studied.

Architectures have also been proposed that facilitate learning of behavior sequences. The MAXSON-VI architecture can learn sequences of goals through imitation in an environment similar to that used in our work (Crabbe & Dyer, 2000b). As in ConAg, a teacher agent is already endowed with the ability to perform the correct sequence. The sequence learning in MAXSON-VI is *one-shot*, that is, one observable sequence of a teacher is sufficient for the student to learn the appropriate response compared with the multiple training instances required in ConAg. The PerAc (perception-action) architecture (Gaussier & Zrehen, 1995) is a neural architecture with an innate reflex system as in the ConAg architecture. This architecture learns sensory/motor associations by reinforcing links between sensory input patterns and motor actions. The PerAc architecture has been used for both landmark-based

navigation (Gaussier, Joulain, Zrehen, Banquet, & Revel, 1997) and for navigation by building a topological map (Revel, Gaussier, Lepretre, & Banquet, 1998). The Agent Network Architecture is a non-hierarchical action selection mechanism—behaviors can excite or inhibit other behaviors (Maes, 1990, 1991). These links between behaviors can be learned by identifying dependencies between behaviors. A similar approach has been used in the ConAg architecture to enable learning of spatial and temporal correlations (Panangadan & Dyer, 2002). The agent network architecture does not include a spatial representation. In ConAg, the spatial maps do not directly specify the direction in which the agent has to move—this is determined by the action selection network. This separation between the spatial and goal representation enables a ConAg agent to satisfy different goals at different times. Tyrrell (1993a, 1993b) has compared action selection mechanisms in a simulated environment and has found that hierarchical action selection outperforms flat mechanisms.

The ability of a distributed group of ants, each with limited sensing, to discover efficient paths to food sources has led to ant colony optimization methods to provide solutions for combinatorial optimization problems such as the traveling salesman problem (Dorigo et al., 1996) and the assignment problem (Gambardella, Taillard, & Dorigo, 1999), and for other problems such as sorting (Deneubourg et al., 1991; Holland & Melhuish, 1999) and network routing (Caro & Dorigo, 1998). The main features of such groups that make these tasks possible are positive feedback (good solutions such as short paths are reinforced), distributed computation (prevents the group from settling too quickly into a poor solution), and greedy heuristics (which lead to acceptable solutions rapidly; Dorigo et al., 1996). The collective laying of pheromone by ants while searching for food can be considered as a form of spreading activation: shorter paths are reinforced with larger amounts of pheromone and the gradient of pheromone gives a path leading to home. In our work, we make use of spreading activation not only for path planning to a goal but also for goal selection. The concept of spreading activation for path planning has also been used for routing in sensor networks (Intanagonwawat, Govindan, & Estrin, 2000). The egocentric spatial maps (ESMs) used in the ConAg architecture are similar to concentric spatial maps (CSMs; Chao & Dyer, 1999; Lagoudakis, 1998). CSMs are egocentric maps

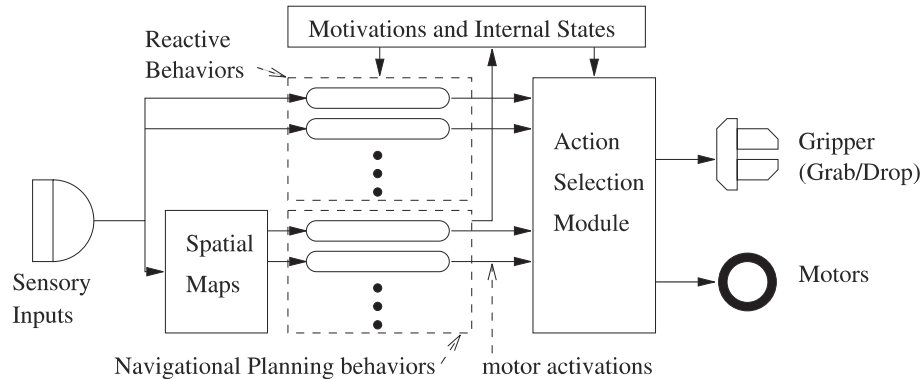
in which the neurons representing the space around the agent are arranged in concentric circles.

### 3 Environment and Agents

We now describe the simulation environment which we used to demonstrate the ConAg architecture. The agents and the colored discs (shaded in printed copy) exist in a simulated 2-D continuous world, similar to that described by Crabbe and Dyer (2000a). The discs represent objects that are relevant to an agent—food (green), water (blue), and bricks (red). Agents are equipped with distance sensors that enable them to sense discs and agents. Distance sensors for each color are distributed all around the agent (360° field of vision). The sensors have a limited sensing range that covers only a small portion of the environment around them. Each agent also has a compass and this is used to align all sensor readings in one global direction. Random errors are added to the sensors. The mean error is proportional to the distance of the object being sensed (closer objects are sensed more accurately). Another source of sensor error is the use of only a discrete number of distance sensors—a sensor cannot distinguish the presence of more than one disc within its field of vision and also cannot determine the exact angle of a disc within this sector.

The “motors” on the agent enable it to move forward and turn, through motor commands that consist of the speed and the angle of a turn. The agents have inertia and thus the motors cannot react instantaneously to motor commands. The distance moved by the agent in each time step is obtained from the wheel encoders and this odometry information is used to maintain the spatial maps.

An agent has a *gripper* with which it can pick up a brick located near its current position and carry the brick as it moves. Since this work does not focus on the low-level details of gripper positioning, it is assumed that a gripper can grasp any brick that is within a pre-specified range from the agent as a single action. An agent can carry only one brick at a time. Bricks carried by other agents are not visible. The agent can drop the brick at its current location. One may imagine that after grabbing a brick, the agent slides the brick under its belly (making the brick invisible to other agents) and therefore when the agent drops the brick, it occupies the same position as that of the agent.



**Figure 2** Block diagram of the ConAg architecture.

We simulate the energy requirements of a physical entity by requiring that an agent has to “eat” and “drink” periodically. These actions are accomplished by touching food and water discs respectively. The need for eating and drinking are triggered by two *motivations* (hunger and thirst). In addition, there is an *avoid* motivation that is always active to trigger collision-avoidance behaviors.

## 4 ConAg Architecture

The ConAg architecture is a behavior-based architecture (Arkin, 1998). In such an architecture, every module, called a *behavior*, directly accesses the sensory information and produces a motor activation. These motor activations are then sent to an action selection module that generates a single motor activation that is sent to the motors. Behavior-based architectures have been widely used in physical robot implementations because of their fast reaction times (each behavior performs a simple computation on the sensor input) and robustness (failure of a single behavior will not bring down the robot completely; Brooks, 1986; Mataric, 1992).

The ConAg architecture uses a connectionist action selection that implements a winner-take-all selection. The behaviors are divided into two groups:

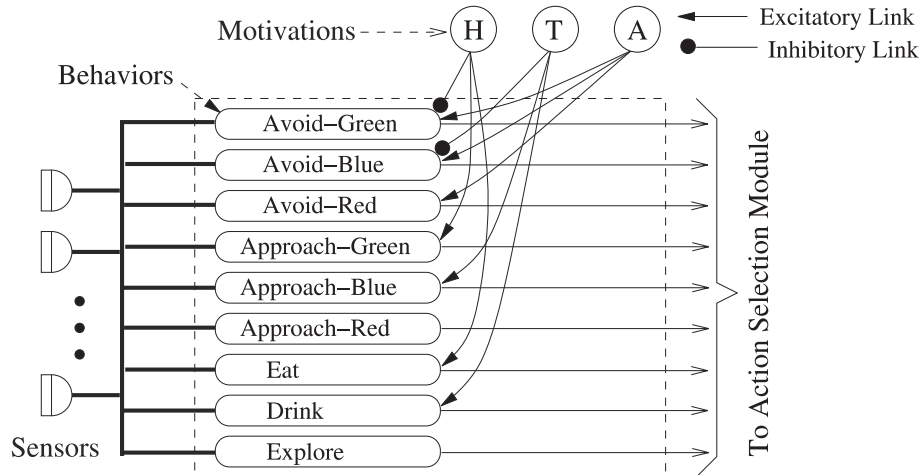
1. **Sensory/Motor Behaviors:** The sensory/motor module generates motor actions to maintain the physical requirements of the agent (obstacle avoidance, eating, drinking). These behaviors are reactive, that is, the output of each behavior depends only on the current sensory input.

2. **Navigational Planning Behaviors:** The navigational planning module builds and maintains a spatial representation of the environment, learned from the history of sensory inputs. These maps are then used for both construction and self-preservation goals, by planning paths to locations of bricks and food and water.

Since different behaviors have to be performed at different stages of the construction task, the current state of construction is encoded into *Internal State* nodes and the activation on these state nodes regulates the action selection. The block diagram of the architecture is shown in Figure 2 and the modules are described in detail below.

### 4.1 Sensory/Motor Behaviors

The reactive behaviors available to the agent are shown in Figure 3. These include *Approach* behaviors that generate motor commands that take an agent to the nearest disk of a particular color and *Avoid* behaviors that move the agent away. There are also *Eat* and *Drink* behaviors, and an *Explore* behavior. The Explore behavior outputs a random motor action and does not depend on sensor activations. Second-order links between motivations and behaviors are used to excite or inhibit behaviors. The Avoid motivation excites all the Avoid behaviors for obstacle avoidance. The Hunger motivation excites the Approach-green behavior and inhibits the Avoid-green behavior. This results in the agent moving toward green discs when it is hungry while the inhibitory link suppresses the green disc avoidance behavior. There are no links to the Approach-red behavior from



**Figure 3** Sensory/Motor behaviors. Activations flow from the sensors, through the behaviors that are regulated by the *Hunger* (H), *Thirst* (T) and *Avoid* (A) motivations, to the action selection module.

the motivations as this behavior is needed only during construction and hence is regulated by the internal state nodes (described later). The Explore behavior is always excited and thus this behavior is the default behavior if none of the other behaviors are active.

## 4.2 Navigational Planning Behaviors

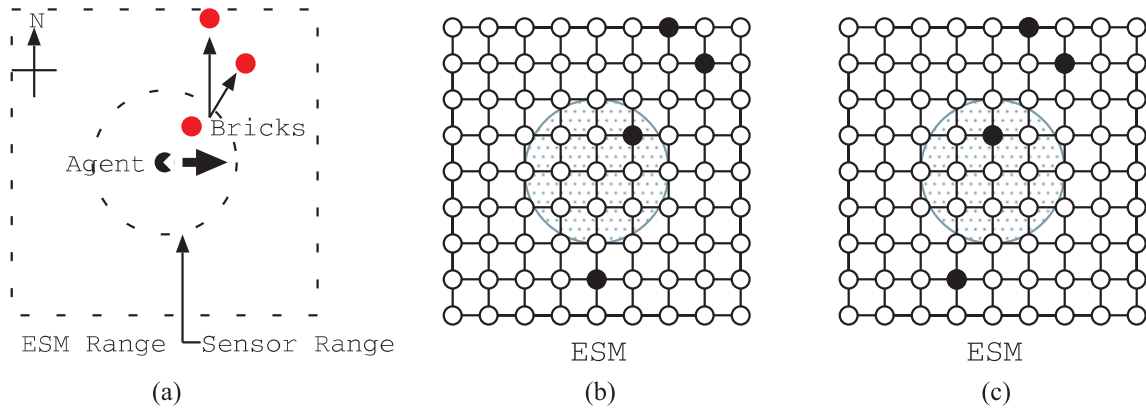
To plan paths (taking into consideration regions of the world that are out of sensor range) an internal representation of the world is necessary. A representation of space is also needed to compare the current layout of the world with the pattern of the structure to be built. Every agent uses egocentric spatial maps (ESM; Chao & Dyer, 1999) to represent the spatial relationship between the agent and each kind of disc in the environment. Navigational maps that receive activation from the ESMs are then used to compute paths.

**4.2.1 Egocentric Spatial Maps (ESM)** An ESM divides the area around the agent into a grid of small uniform squares. The area covered by each grid cell is approximately equal to the size of one disc. The central cell (neuron) in the grid is the cell on which the agent is currently present. The ESM contains neurons arranged in a grid to correspond to these squares such that the central neuron in the ESM corresponds to the square on which the agent is located. A neuron is connected to its eight neighboring neurons and this repre-

sents the adjacency relationships between the areas represented by each ESM cell. The activation of a neuron indicates if a disc is present in the corresponding square. Thus at every point of time, each agent maintains a bird's-eye view of the world around it. The neurons close to the center of the ESM correspond to the space sensed by the sensors. At every time step, new sensory input is integrated into the center portion of the map (Figure 4b). Thus, changes in the world, such as the addition or disappearance of discs, are reflected in the ESM. Though this representation uses a pre-fixed number of nodes and interconnections, assigning the center of the ESM to the current location of the agent ensures that the space closest to the agent is always mapped.

As the agent moves in every time step, the neuron activations are passed to neighboring neurons to maintain the egocentric nature of the map. The activations on the ESM are passed in a direction opposite but proportional to the distance moved by the agent in each time step (Figure 4c). The amount moved by the agent is obtained from the dead-reckoning inputs. If the activation of a neuron is moved out of the map's range, then the agent "forgets" about the corresponding disc.

Each agent has a separate ESM for each kind of disc in the environment: the *food ESM*, *water ESM*, and the *brick ESM*. The structure to be built is also represented in a ESM called the *configuration ESM*. Like the other ESMs, the activations on this map are also shifted as the agent moves so that egocentricity of



**Figure 4** Updating an ESM. (a) Layout of bricks and agent moving to the right. (b) The shaded portion of the ESM represents the space sensed by the sensors. For clarity, only four (out of the eight) adjacent nodes are shown connected to each node. (c) Shifting activations to the left when agent moves to the right.

the map is preserved, but the initial activations on the configuration ESM (that encode the structure to be built) are set a priori and are not updated by the sensors.

**4.2.2 Navigation Maps (NM)** Paths are computed by spreading activation on navigation maps which also consist of a grid of neurons. Activation spreads from nodes representing goal locations while nodes representing obstacles inhibit this activation (food and water discs and cells where bricks must be placed are goal locations). Let  $n$  denote an arbitrary neuron in an NM and  $nb(n)$  the set of eight neighboring neurons of  $n$ . Let  $a_n(t)$  be the activation of  $n$  at time  $t$ .

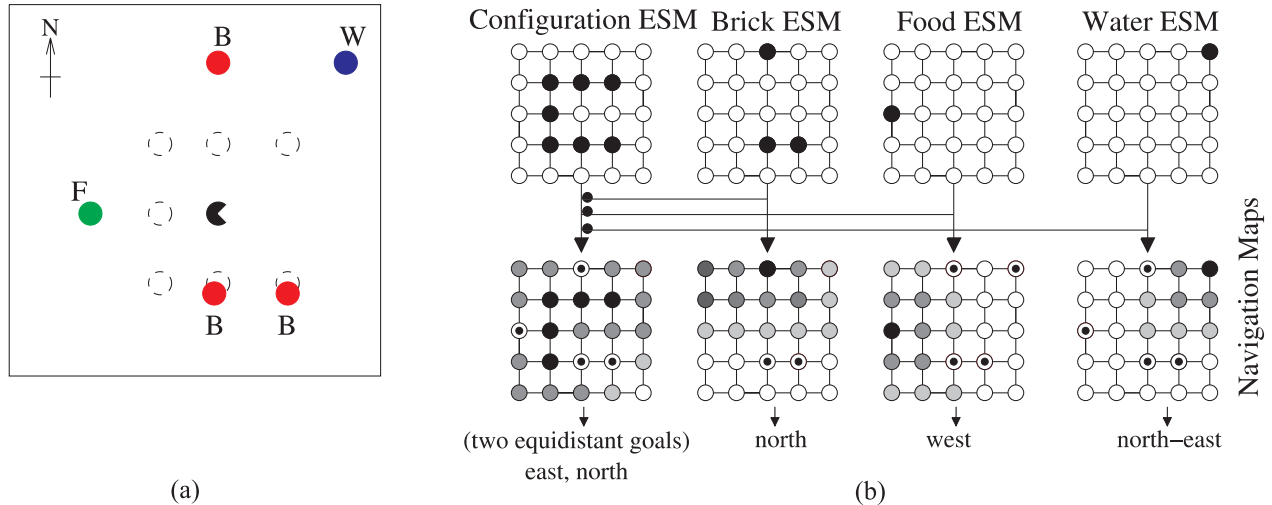
$$a_n(0) = \begin{cases} 1, & \text{if } n \text{ represents a goal location} \\ -1, & \text{if } n \text{ represents an obstacle} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$a_n(t+1) = \max_{m \in nb(n)} (a_m(t) - d(n, m)), a_n(t) \geq 0, \quad (2)$$

where  $d(m, n)$  is proportional to the Euclidean distance between the locations represented by nodes  $m$  and  $n$ . Equation 2 is iterated until the activations stop changing. Spreading activation is a parallel implementation of Dijkstra's shortest path algorithm (Cormen, Leiserson, & Rivest, 1990). The gradient created by the activation is the planned path to the nearest goal location. To reach this goal from its current location, the agent should move in the direction of the maximum gradient of activation at the center node of

the NM. The motor commands based on the maximum gradient will only be in one of the eight compass directions (corresponding to each of the eight neighboring nodes of the central node). However, the path of an agent will be smooth because of the inertia of the agent.

**4.2.3 Integrating Multiple ESMs and NMs** Every ESM is associated one-to-one with an NM, that is, each node in the ESM has an excitatory link to its corresponding node in the NM. The excitatory links from active ESM nodes activate the NM goal nodes that will compute a path to the nearest disc of the color represented in the ESM. Inhibitory links from the active nodes in other ESMs activate obstacles in this path. The *configuration navigation map* computes a path to locations where bricks should be dropped. Hence, every configuration NM node is activated by the corresponding node from the configuration ESM and inhibited by the activations from the nodes on the food and water ESMs (since food and water discs are obstacles). The configuration NM is also inhibited by the brick ESM because if a brick is already present at a desired location, then the agent should not place another disc there. Spreading activation is carried out on the configuration NM and the maximum gradient at the center node gives the motor activation that has to be taken to move toward the nearest drop site. The idea is extended to generate the activations on the *brick navigation map*, which computes a path to a brick that is



**Figure 5** Interconnections between ESMs and NMs (a) The agent is surrounded by a food (F), a water (W) and three brick (B) discs (where two of them are already in their goal positions). The dashed circles indicate the horseshoe structure to be built. (b) The four ESMs and their corresponding NMs. The darkness of shaded circles indicates the magnitude of activation while circles with filled centers indicate negative activation (obstacles). For clarity, only one connection from each ESM to its corresponding NM and only the connections to the configuration NM are shown. The direction to the nearest goal is listed under the corresponding NMs.

available for construction. It is excited by the nodes on the brick ESM and inhibited by the nodes on the food and water ESMs. Inhibitory links from the configuration ESM distinguish available bricks from ones that already compose parts of the structure being built.

Figure 5 shows a subset of the interconnections between the ESMs and the NMs. The node activations on the configuration ESM are in the shape of the structure to be built. (In this figure, the configuration to be built has the shape of the letter C.) The brick ESM has three activations, two of which align with two of the activations on the configuration ESM. These represent bricks that already form part of the structure being built. The mismatches between the ESMs represent the location of a brick that is available for construction. These activate the corresponding nodes on the configuration and brick NMs. Activations spread from these nodes to the center and give the direction the agent should move to reach the unbuilt parts of the structure and to reach the brick available for construction.

### 4.3 Internal State Nodes

While motivations monitor the vital internal “energy” levels of the agent that are essential for its survival,

*Internal State Nodes* are used to keep track of the current stage of the construction task:

- *Holding-Brick*: Is active when the agent is carrying a brick.
- *At-Drop-Site*: Indicates if the agent is at a location where it can drop a brick.
- *At-Brick*: Is active when the agent is at a location from where it can pick up a brick.

The activations of the At-Drop-Site and At-Brick internal state nodes can be computed directly from the NMs. For instance, if the central node of the configuration NM is active, then this implies that the agent is currently at a location where the structure is missing a brick. Hence, the activation of the At-Drop-Site internal state node is set from the central node of the configuration NM. Similarly, the activation of the At-Brick state node is set from the central node of the brick NM. The activation of the Holding-Brick state node is directly available from the current position of the grippers (grasping or open). The use of egocentric spatial maps greatly reduces the number of neurons needed to alter any agent behavior because such behavior-controlling neurons are only required at the center of each map.

#### 4.4 Action Selection

The action selection module implements a fixed priority selector—behaviors that are crucial to the survival of the agent, eating and drinking, have the highest priority, followed by avoiding obstacles, and finally approaching food and water. The prioritizing of these behaviors is implemented by lateral-inhibition links from the high-priority behaviors to all of the lower ones. The lateral-inhibition connections are assumed to be innate and are not modified.

If none of the self-preservation goals are active, then the agent attends to the construction task. The agent performs the configuration navigation behavior if the Holding-Brick state node is active, else the brick navigation behavior is performed. If Holding-Brick and At-Drop-Site internal state nodes are both active, then the agent drops the brick. If Holding-Brick is inactive but At-Drop-Site is active, then the agent attempts to grab the disc near it. Excitatory and inhibitory connections from the internal state nodes to the outputs of the navigational planning behaviors are used to select the motor action. Weights on these links can be learned (described in Section 5).

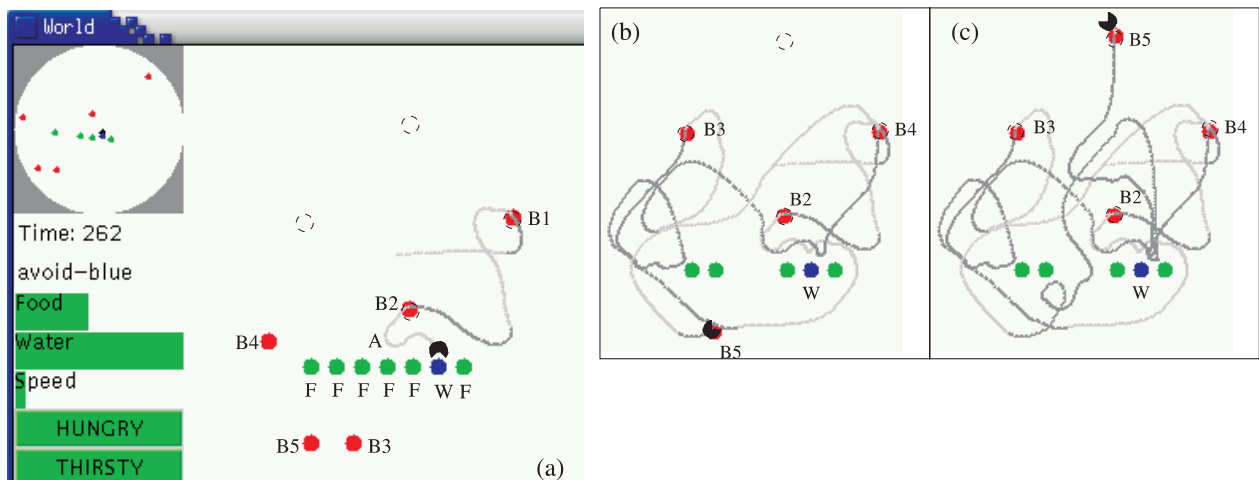
Integrating low-level behaviors with higher level ones using this priority mechanism is an instance of selecting *consummatory* behaviors over *appetitive* behaviors (consummatory behaviors are those that directly affect a motivation while appetitive behaviors

are those that do not directly affect any motivation; McFarland, 1981). This preference for consummatory over appetitive behaviors is one of the requirements for an action selection mechanism since otherwise an agent would repeatedly perform a behavior such as moving toward food instead of eating even when food was available to the agent (Tyrrell, 1993).

#### 4.5 An example

In Figure 6, screen shots from a sample construction run are shown. The initial arrangement of discs consists of a wall made of food and water discs that acts as an obstruction and five bricks (labeled B1 through B5) scattered around the environment (Figure 6a). The dashed circles (arranged in a diamond pattern) indicate locations where bricks are to be placed. After moving bricks B1 and B2 and en route to place the third, the agent became thirsty and thus suspended construction (at point A) to reach the water disc.

In Figure 6b, the agent's ability to repair is demonstrated. We removed brick B1 when it was out of the agent's sensor range. However, when the agent became thirsty again, it navigated to the blue disc to drink. From that location, the missing brick location is within sensor range and the brick ESM is updated. After drinking, it correctly drops the brick it was carrying at the location that was initially occupied by brick B1.



**Figure 6** Example construction task: Positions of discs and agent at time (a) 262, (b) 1194, and (c) 1644. Carrying a brick. Construction sites are marked by dashed circles. The "wall" is made of food (F) and water (W) discs. Portions of the simulation program are shown on the left side, displaying the current sensor activations, motivations of the agent, and time-step.

In Figure 6c, the agent's ability to take advantage of unexpected changes in the environment is demonstrated. When the agent moved to pick up brick B5, we removed two green discs, thus creating a gap in the wall. When this gap came into sensor range, the ESMs were updated and the path to the northern-most drop site was recomputed to navigate the agent through the gap.

## 5 Construction Sequence Learning

One of the main advantages of a connectionist action selection module is that the sequence of steps that have to be repeated for construction can be learned, that is, the interconnections between the internal state nodes and the navigational planning behaviors do not have to be set a priori. The construction task requires an orchestration of both sensory/motor and navigational planning actions, summarized in the action sequence given in Figure 7. Note that this sequence of actions for construction is independent of the shape of the structure to be built. Thus this sequence can be pre-programmed into the agent and then learning becomes unnecessary. We demonstrate the learning capability of the ConAg architecture in order to show that the architecture is suitable for use in tasks that require a different sequence of actions (for instance, if bricks have to be colored to provide cues to other agents as in: Crabbe & Dyer, 2000a; Jones & Matarić, 2003; and Werfel, 2004).

We introduce three new state nodes to identify stages of the construction task:

3. *Near-Brick*: Becomes active when a brick is sensed close to the agent. Near-Brick is set from the activations of the red disc sensors.

4. *Brick-Available*: Is active if the spatial maps indicate a brick that is not part of the structure being built. The activation of this state node is set from the sum of activations on the nodes of the configuration NM: if the structure is complete, then there will be no active node on the configuration NM.
5. *Drop-Site-Available*: Is active if the spatial maps indicate that there are parts of the structure that still require a brick to be placed somewhere. The activation of this state node is set from the sum of activations on the nodes of the brick NM: If there are no bricks that are not part of the structure, then none of the nodes on the brick NM will be active.

Based on sensory inputs and internal state nodes, this sequence can be performed by the action selection module through the connections and weights shown in Figure 8. Each node in the action selection module may have both excitatory (arrows) and inhibitory (dots) connections from the internal state nodes. Weights are positive for excitatory and negative for inhibitory, and all activations are summed and thresholded:

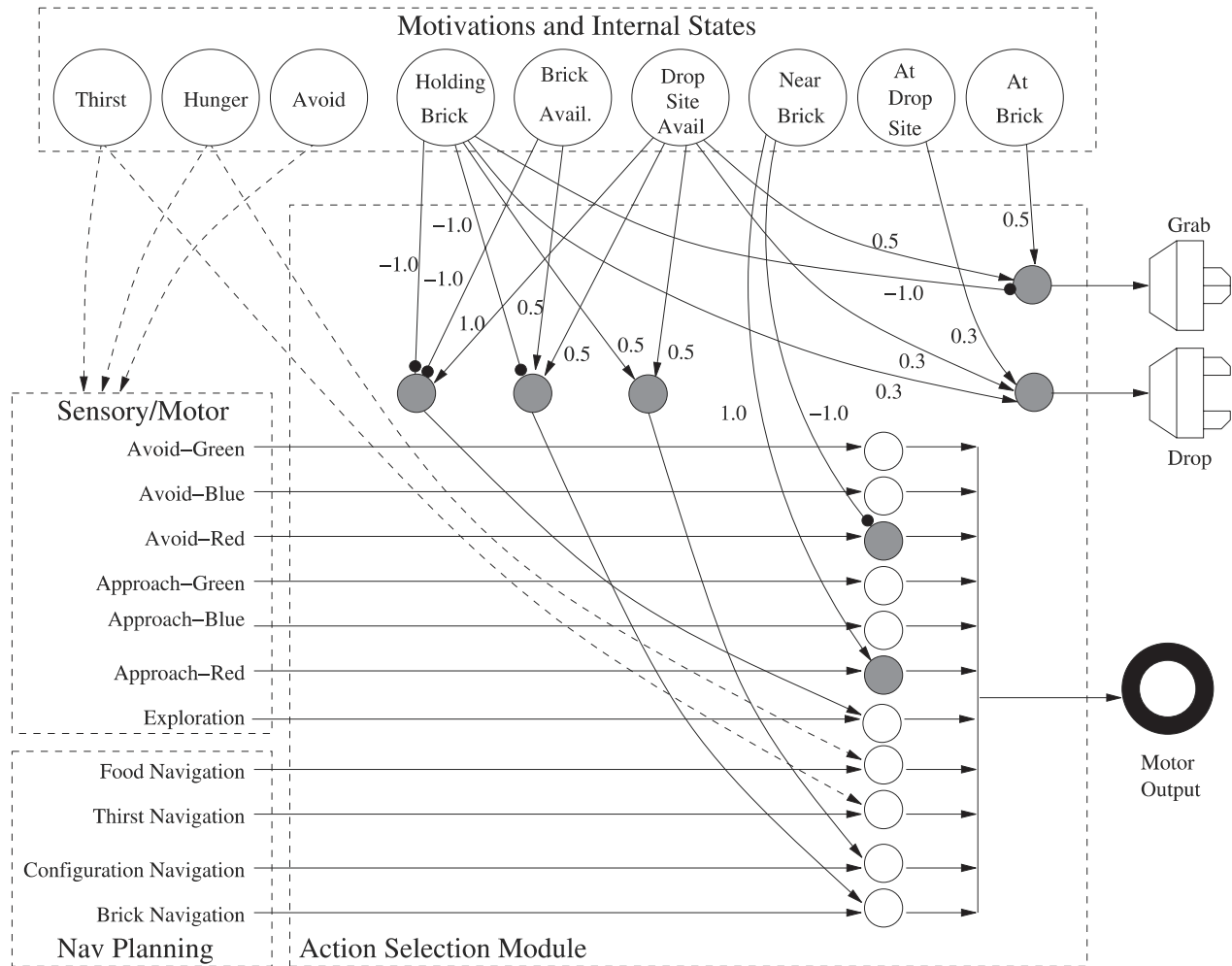
$$a_j = \psi \left( \sum_i \left( w_{ij}^{excite} \times a_i + w_{ij}^{inhibit} \times a_i \right), T \right),$$

where  $a_j$  is the activation at node  $j$ ,  $\psi$  is the step function with a pre-defined threshold,  $T$ .

The excitatory links activate a node when a set of conditions is met, such as enabling *Grab* when (a) the agent is at a disc (At-Brick internal state node is active) and (b) the structure is missing discs (Drop-Site-Available internal state node is active). Therefore, 0.5 is

- |     |   |
|-----|---|
| 1:  | while construction is incomplete                                    |
| 2:  | To locate an available disc:  |
| 2a: | if close to a brick, select <i>Approach-Red</i>                     |
| 2b: | else if bricks are available, select <i>Brick navigation</i>        |
| 2c: | else select <i>Explore</i> behavior                                 |
| 3:  | If brick is within reach, <i>Grab</i>                               |
| 4:  | To navigate to the location where the structure is missing a brick: |
| 4a: | select <i>Configuration navigation</i>                              |
| 5:  | If at a drop-site, <i>Drop</i> the brick                            |

**Figure 7** Sequence of actions required to perform construction.



**Figure 8** The action selection module with engineered connections and weights for construction. The shaded circles indicate the output nodes.

assigned to both weights to ensure that Grab is active only when both input nodes are firing to surpass the threshold (0.7). However, if the agent already has a disc (Holding-Brick is active), then the Grab node should be inhibited so the agent does not try to grab another disc. This is achieved by the inhibitory link, more than sufficient to prevent the activation from surpassing the threshold. Since self-preservation goals have higher priority, the reactive response of avoiding the disc must be turned off when the agent needs to grab a red disc. This is achieved by the inhibitory link from the Near-Disc internal state node to the Avoid-Red behavior.

### 5.1 Reinforcement Learning of the Construction Sequence

An agent adjusts the weights on the links in its action selection module through a process of reinforcement learning. The agent learns to mimic the actions of a teacher agent that is already capable of performing the construction sequence. The teacher agent is not physically situated in the environment, but it has access to the same sensory input as the learner agent and the teacher's actions are determined by the engineered action selection network shown in Figure 8. The teacher then monitors the student's external actions and generates two positive or negative reinforcement signals, one for the arm action and the other for the motor action of the

student. These signals are generated by comparing the student's external actions with the actions the teacher would have taken. This is analogous to the case where the teacher agent is closely following the learner and is able to compare the learner's outputs with its own ideal output at every step. The student only has access to the teacher's actions, but not to its internal state.

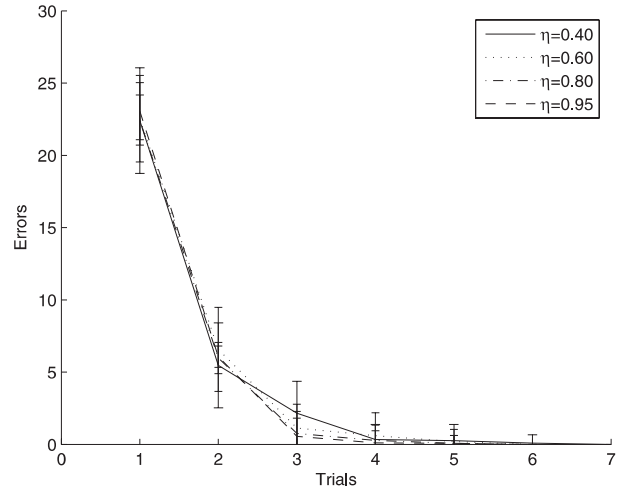
The student begins with a fully connected network. The network's inputs represent the internal state nodes (Holding-Brick, At-Brick, Near-Brick, Brick-Available, Drop-Site-Available, and At-Drop-Site) and the outputs represent the possible actions (Grab, Drop, configuration navigation, brick navigation, Explore, Avoid-Red, and Approach-Red). Since the links between any two nodes can be either excitatory or inhibitory, two weights have to be learned simultaneously. Through immediate rewards, the sequence of actions to carry out construction is learned by adjusting the weights to minimize error. The delta learning rule is used for the single layer network:

$$w_{ij}^{excite}(t+1) = w_{ij}^{excite}(t) + \eta \times R_j(t) \times a_i$$

$$w_{ij}^{inhibit}(t+1) = w_{ij}^{inhibit}(t) - \eta \times R_j(t) \times a_i,$$

where  $w_{ij}(t)$  is the weight between nodes  $i$  and  $j$  at time  $t$ ,  $a_i$  is the activation of node  $i$ , and  $R_j(t)$  is the reinforcement signal at node  $j$ .  $\eta$  is the learning rate. The reinforcement signal generated by the teacher can take two values:  $-1$  and  $+1$ . With a positive reinforcement, all excitatory weights that activated the behavior are increased and the inhibitory weights decreased. If a node is incorrectly activated in the student, a  $-1$  reinforcement from the teacher causes opposite adjustments and reduces future activations under the same inputs.

This learning approach was evaluated by having the actions of one student agent reinforced in the environment shown in Figure 6. The student was repeatedly placed in the same environment until it no longer made any mistakes, at which point it was placed into a novel environment to validate that the learning is generalized and adaptive to other construction scenarios. The student agent was given no other a priori knowledge except for the structure to be constructed, that is, the activations on the configuration ESM. The action selection network has all of its weights randomly initialized to be between 0 and 0.5 (excitatory) or 0 and  $-0.5$  (inhibitory). Each trial consisted of 1,000 time



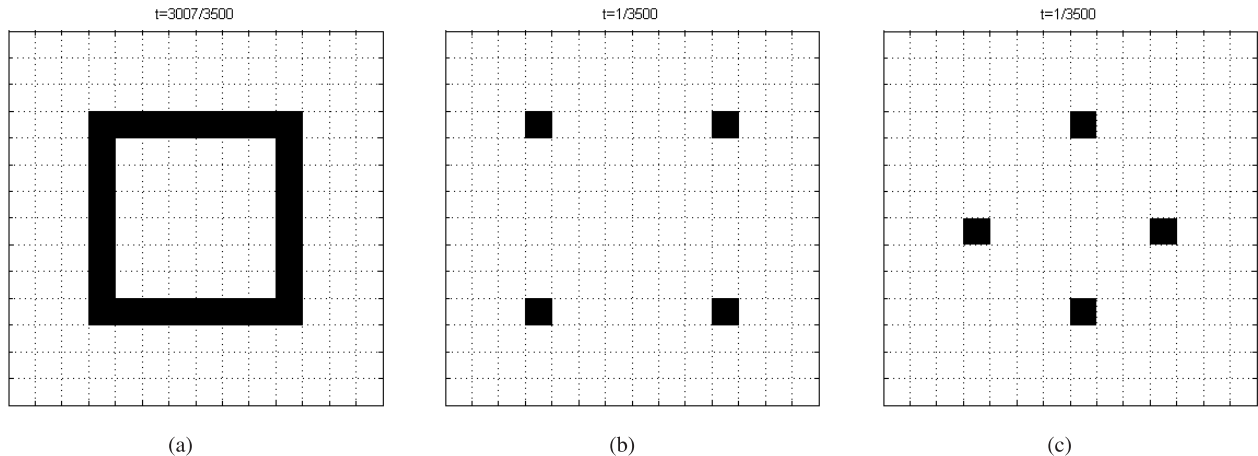
**Figure 9** Number of errors made by the student agent during construction sequence learning at different learning rates.

steps at the end of which the agent was moved to its initial position in the environment. The number of errors is calculated as the number of time steps when the output actions of the learner did not match those of the teacher. Figure 9 shows the reduction in errors over trials for different learning rates. The error bars mark one standard deviation from 50 runs. The number of errors per trial was not significantly sensitive to the learning rate.

More complex reinforcement algorithms such as Q-learning (Watkins, 1989) that can handle delayed reinforcement are also applicable in this environment. In our experiments, we found that Q-learning with a suitable set of parameters showed comparable error rates. The use of nodes to represent state in the ConAg architecture obviates the need for a delayed reinforcement learning mechanism to learn the construction sequence.

## 5.2 Comparison with Rule-Based Systems

We compared the performance of the ConAg architecture with that of a representative reactive system for construction. We implemented the construction system described by Jones and Matarić (2003). Their system was demonstrated (in simulation) on a 2-D square lattice. Construction agents move through this lattice one cell at a time. A structure gets built in this environment by agents moving into the appropriate cells



**Figure 10** The reactive construction rules vary with the initial configuration of bricks. (a) Structure to be built is square shaped. (b) The “seed” bricks are placed at the corners. (c) The “seed” bricks are placed at the sides.

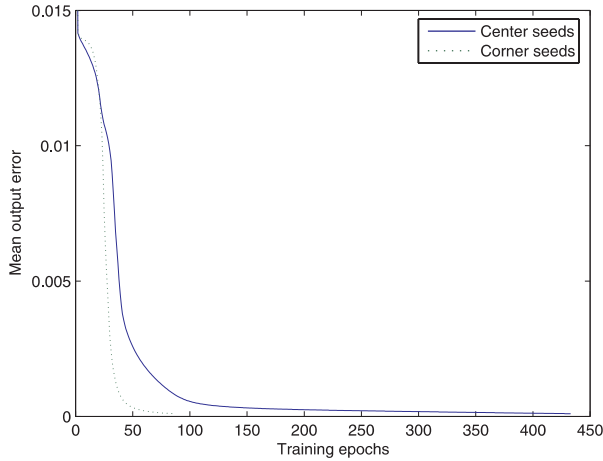
(thus the agents become bricks). The agents have a completely reactive action selection mechanism. A construction agent can only sense the occupancy of its neighboring cells. Agents determine their action by comparing the occupancy pattern to a lookup table of pre-computed patterns and associated actions (such as moving to one of the unoccupied neighboring cells). The lookup table of rules was called the *transition rule set*. In addition, an agent maintains a state variable that is visible to neighboring agents. To construct a structure, the environment is instantiated with bricks in specific locations and each of the agents is provided with a particular lookup table of patterns and actions. Repeated execution of the actions specified in the lookup table leads to the desired structure being built. The state variable is required in order to encode the construction rules for complex structures. Jones and Mataric (2003) also describe an automated way of generating the lookup table given the shape of the structure to be built.

The features of the construction system of Jones and Mataric (2003) are similar to those found in other reactive rule-based systems (Bonabeau et al., 2000; Crabbe & Dyer, 2000a; Werfel, 2004). The main distinction between such systems and the ConAg approach is that the reactive systems keep track of the progress of the construction task by marking the environment (such as by assigning a state variable to agents: Jones & Mataric, 2003 and Werfel, 2004; moving different types of bricks: Bonabeau et al. 2000; or coloring bricks: Crabbe & Dyer, 2000a). The ConAg architec-

ture, on the other hand, keeps track of the progress of construction by updating the *internal* spatial representation within each agent. We compared the progress of the construction task using the ConAg approach and the system proposed by Jones and Mataric (2003).

We considered the case where the shape of the structure to be built is a square (Figure 10a). Construction using the ConAg approach requires that the configuration NM be initialized with this shape. In the rule-based system of Jones and Mataric (2003), the transition rule set that leads to the building of a square depends on the initial configuration of bricks. In particular, the number of states and rules in the transition rule set varies with this initial configuration. If the initial bricks are at the corners of the square, then four distinct states and eight rules are sufficient (Figure 10b). However, if the initial bricks are at the sides of the square, the number of distinct states and rules increase with the size of the square (Figure 10c).

We incorporated learning of the transition rule set to the rule-based construction system by imitation learning from a teacher agent as in our ConAg system. We used a feed-forward neural network as the learning mechanism. The training input data to the neural network consists of the states of neighboring cells as observed by the teacher agent during construction and the training output data are the corresponding actions. Figure 11 plots the decrease in errors during learning of the transition rule set for the two starting configurations (bricks at corners and bricks along the side) during a typical learning trial. The rate of learning the



**Figure 11** Learning the transition rule set for building a square from two starting configurations: seed bricks at corners and seed bricks at sides.

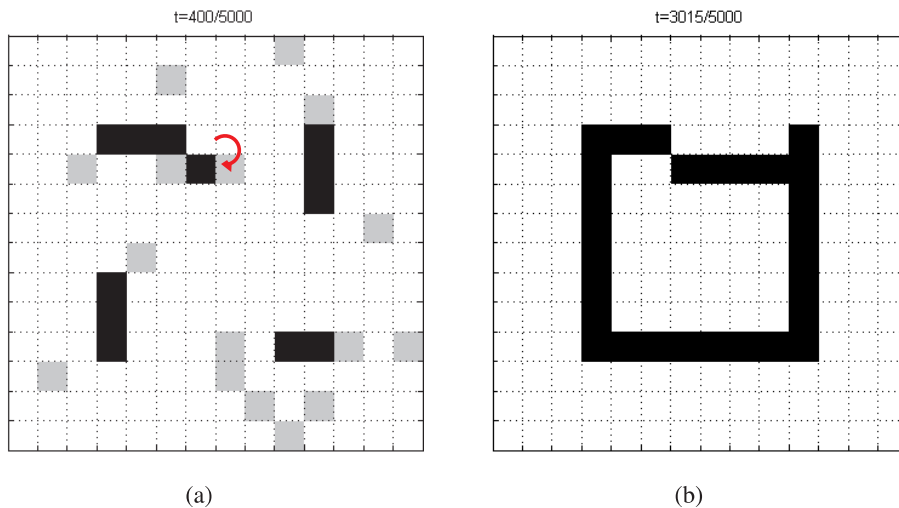
transition rule set is dependent on the initial configuration due to the difference in complexity of the dent on the initial configuration due to the difference in complexity of the rule set for the two cases. The time of completion of learning also differs between the ConAg and rule-based systems. Learning the construction task in ConAg involves only learning the sequence of moving a brick to a drop site. As this sequence is independent of the shape of the structure to be built, learning

can be completed before the teacher agent finishes constructing the structure. However, in the rule-based system, the learner has to observe all the actions of the teacher agent up to the completion of the structure since the transition rule set depends on the shape of the structure being built.

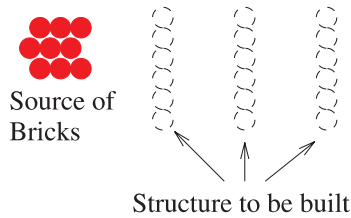
We studied the performance of the two approaches with regards to repair of structures. We showed in Section 4 that the ConAg architecture could continue to build the desired structure even if a brick was moved during construction. The rule-based system, on the other hand, is sensitive to the location of all bricks in the environment. Even if one of the bricks places by an agent is moved to a neighboring cell, a different structure gets built (Figure 12).

## 6 Path Planning via Temporal Sequencing of Goals

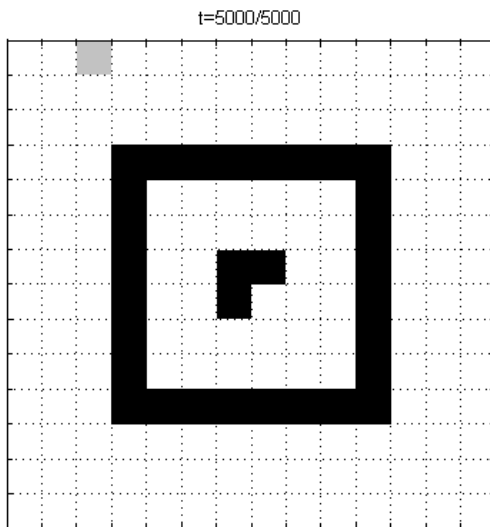
The ConAg architecture, as described thus far, always moves a brick to the closest drop site. However, the total distance traveled by an agent depends on the order in which bricks are placed. For instance, in Figure 13, the structure to be built consists of three parallel walls. Ideally, the agent should build the right-most wall first, but the ConAg architecture would build the left-most wall first and this would block direct paths from the source of bricks at the left to the other two walls. In



**Figure 12** Construction using reactive rules is affected by unexpected changes in the environment. (a) We moved one of the bricks in the structure as it was being built. (b) The resulting structure is not the desired square. The gray cells indicate agents; the black cells indicate bricks.



**Figure 13** Order of placing bricks determines distance traveled.



**Figure 14** Building a square enclosing a smaller square using purely reactive rules. As the outer square is completed before the inner square, the construction agent (gray cell) is blocked from adding bricks to the inner square.

the extreme case, certain structures (for instance, a pile of bricks completely enclosed within a larger ring) would be impossible to build if the order of placing bricks is not planned. This is illustrated in Figure 14 where we attempt to build two squares one within the other using purely reactive rules in a grid world. If the outer square is built before the inner one, then the construction agents cannot access the inner square.

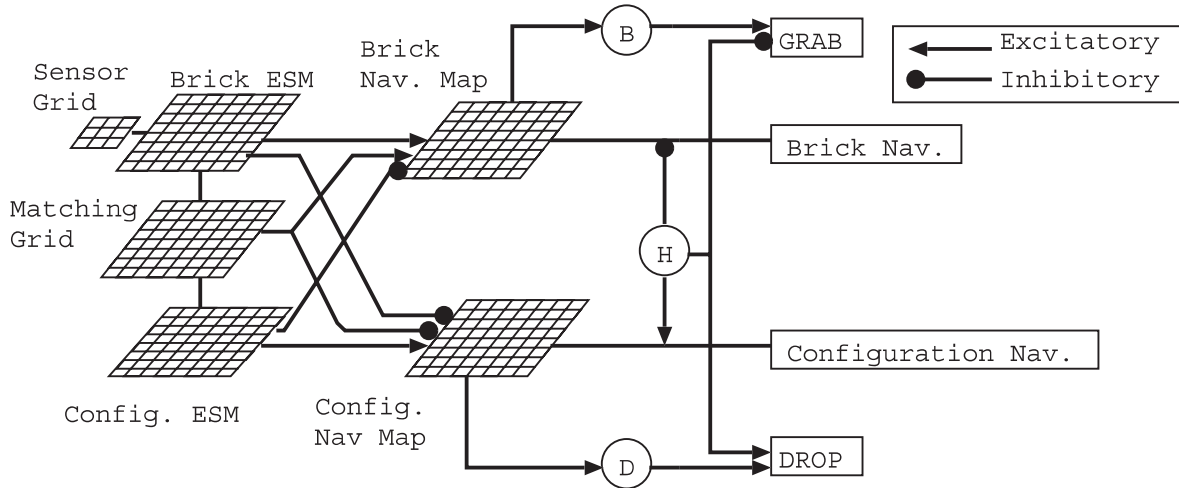
To prevent dropped bricks from blocking direct paths to other drop sites, the agents have to plan the temporal sequence of goal locations such that dropped discs do not obstruct paths to the remaining drop sites. The algorithm described in this section determines such a sequence in two steps. First, each agent

matches discs to drop sites represented in its ESMs using a greedy heuristic (*calculateMatching* algorithm). Second, it determines which of the drop sites do not block paths between a disc and its matching drop site (*drop site selection*) and proceeds to fill these first. We earlier introduced spreading activation for path planning using navigation maps (Section 4.2.2). Here we extend the concept by spreading multiple activations simultaneously, representing paths to different types of goal locations. The interactions of these activations will be used to match pairs of bricks and drop sites which are close together. This implementation using NMs enables the algorithms to be integrated into the existing ConAg architecture. The idea of spreading activation from both source and destination has also been used in other applications that require path planning such as data dispersion in sensor networks (Intanagonwiwat et al., 2000).

## 6.1 Matching Algorithm

The algorithm *calculateMatching* is implemented on a grid of nodes called the *Matching grid*. The algorithm consists of a set of local rules that are repeatedly applied by each of the nodes on the Matching grid. The rules are local in the sense that the actions applicable at a cell depend only on its state and that of its neighboring cells. The algorithm matches each unfilled drop site to its closest available brick. At algorithm completion, all nodes on the Matching grid that are on the shortest path between a matched brick and drop site pair are marked with a *match* value of 1. The values on the Matching grid will be used in the drop site selection step to determine which drop sites are to be filled first.

There is a one-to-one correspondence between the cells on the Matching grid and those on the brick and configuration ESMs. The interconnections between the Matching grid and the ESMs are shown in Figure 15. The source nodes for the spreading activation (the locations of unmatched bricks and drop sites) are excited by the corresponding nodes on the brick and configuration ESMs. Matching grid nodes representing obstacles (bricks that already form a part of the structure) are inhibited by their corresponding nodes on the brick ESM. The nodes of the Matching grid can initiate and spread activations among neighboring nodes. The nodes make use of the gradient of these activations to implement the *calculateMatching* algorithm.



**Figure 15** ConAg-TS architecture: the lines between the Matching grid and the ESMs represent links between every corresponding pair of neurons. Three internal state nodes are also shown: At-Brick (B) is set from the brick navigation map, At-Drop-Site (D) is set from the configuration navigation map, and Holding-Brick (H).

The `calculateMatching` algorithm is listed in Figure 16. It is an iterative algorithm, where every iteration determines the brick and drop site pair that are closest to each other. This pair is identified by spreading activations from all brick and drop site locations. One set of activations (denoted by  $a$ ) is spread from the nodes representing locations of available bricks (step 1). The activation is spread as described for navigation maps (Equation 2). The number of nodes in the grid is denoted by  $M \times M$  and the local rules are repeated  $M$  times to ensure that activation spreads throughout the grid.  $nb(n)$  denotes the set of all neighboring nodes of node  $n$ . All discs that are not bricks free to be picked up act as obstacles to the spreading activation. Similarly, a second set of activations (denoted by  $b$ ) is independently spread from all the nodes representing unfilled drop sites (step 2). These activation values at a cell are inversely proportional to the shortest distance to a brick or drop site. The sum of these two activations at a node is inversely proportional to the distance of the path between the closest brick and drop site that passes through the corresponding location. Hence, if the sum of these two activations at a node is a local maximum, this indicates that the corresponding location is on the shortest path between some brick and drop site. In addition, all nodes on this shortest path will have the same value. To identify the path between the brick and drop site pair that are closest to each other from all other brick and drop site pairs, the maximum activation value

is distributed to all nodes (step 3). This step effectively suppresses all activation that is not on the path between the desired brick and drop site. A path from the drop site is identified by following the gradient toward the closest brick (step 4).  $succ(n)$  is defined as the neighboring node of  $n$  that is along the gradient of activation  $a$ .

$$succ(n) = \underset{n' \in nb(n)}{\operatorname{argmax}}(a[n']). \quad (3)$$

The matching between the drop site and brick is marked by setting the corresponding nodes on the Matching grid to 1 (step 5). Note that this path is initiated from the neighboring node of the drop site (not the drop site itself). This fact will be used in the goal selection step. These steps are repeated until all drop sites are matched.

**6.1.1 Running Time of the Algorithm** As all the rules are local, the steps in the `calculateMatching` algorithm can be executed in parallel. Each step of the algorithm involves a node communicating only with its neighbors and performing a fixed number of computations. Each spreading activation requires every node to execute Equation 2  $M$  times. At the end of each loop, one drop site is matched. Therefore, the main matching loop has to be executed at most  $D$  times where  $D$  is the number of unfilled drop sites. Thus, the running time when `calculateMatching` is executed in parallel is

```

Initialization
  match[n] := 0  $\forall n \in N$ 
  S[n] := 1 if n represents a brick, 0 otherwise
  T[n] := 1 if n represents a drop-site, 0 otherwise
  X[n] := 1 if n represents an obstacle, 0 otherwise

while there are non-zero entries in T
1: Spread activation a
  a[n] := 1 if S[n] = 1; a[n] := -1 if X[n] = 1; a[n] := 0 otherwise
  repeat M times
    a[n] := maxm ∈ nb(n)(a[m] - d(n, m)),  $\forall n \in N, a[n] > 0$ 

2: Spread activation b
  b[n] := 1 if T[n] = 1; b[n] := -1 if X[n] = 1; b[n] := 0 otherwise
  repeat M times
    b[n] := maxm ∈ nb(n)(b[m] - d(n, m)),  $\forall n \in N, b[n] > 0$ 

3: Identify cells with maximal a+b activation
  c[n] := a[n] + b[n]  $\forall n \in N$ 
  hasMaxValue[n] := true  $\forall n \in N$ 
  repeat M times
    if (a[n] > 0) and (c[n] < a[m] + b[m]), n ∈ N, m ∈ nb(n)
      c[n] := a[m] + b[m]
      hasMaxValue[n] := false

4: Identify drop-site to be matched cells on shortest path between drop-site and brick
  onShortestPath[n] := false  $\forall n \in N$ 
  if (T[n] = 1) and (hasMaxValue[n] = true), n ∈ N
    onShortestPath[succ(n)] := true
    T[n] := 0
  repeat M times
    if (onShortestPath[n] = true), n ∈ N
      onShortestPath[succ(n)] := true

5: Set match value of all cells on shortest path to 1
  if (onShortestPath[n] = true), n ∈ N
    match[n] := 1
end while

```

**Figure 16** Algorithm calculateMatching.

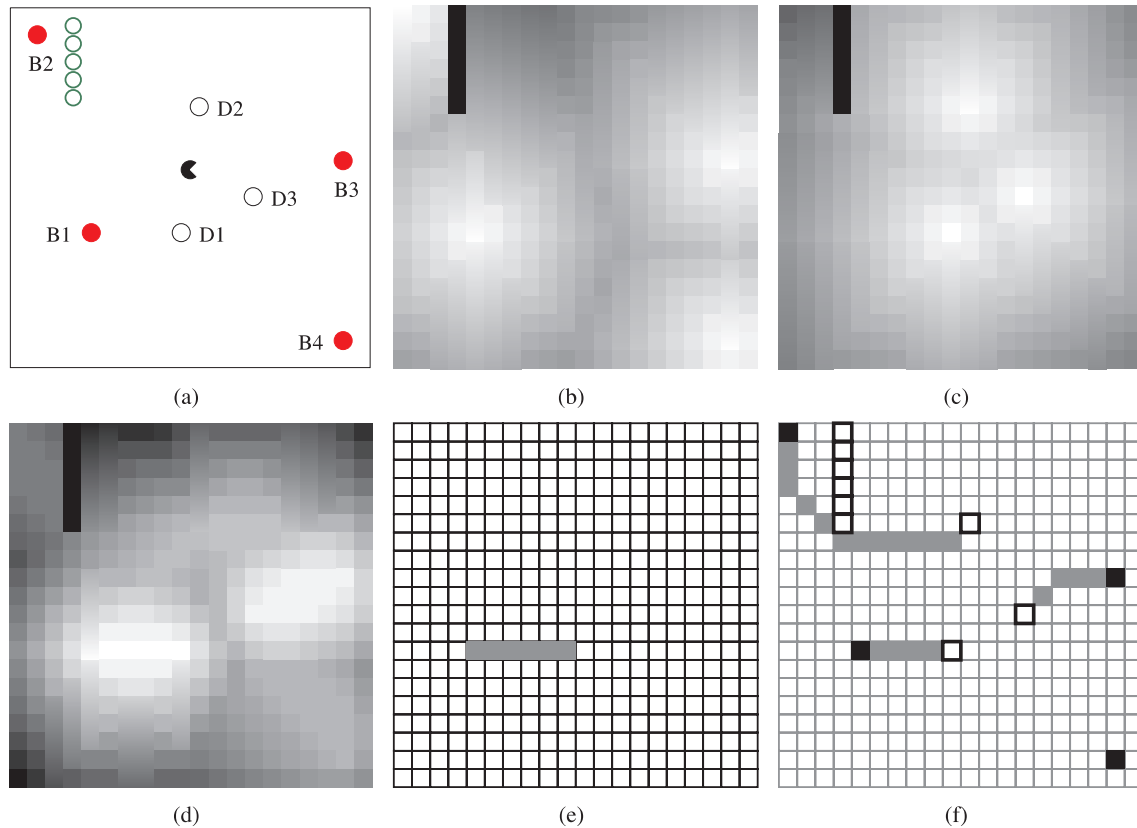
proportional to  $M \times D$ . Note that the area covered by the navigational maps is proportional to  $M^2$ .

An example illustrating the calculateMatching algorithm is shown in Figure 17. Figure 17a shows an environment containing four bricks and three drop sites. The spread of activation  $a$  from the nodes representing bricks is shown in Figure 17b and activation  $b$  initiated from the nodes representing drop-sites is shown in Figure 17c. The sum of these two activations is shown in Figure 17d. Figure 17e shows the shortest

path identified from the sum of activations. The final values in the Matching grid are shown in Figure 17f indicating the matched brick and drop site pairs.

## 6.2 Drop Site Selection (DSS)

In the example described above, the paths between the three pairs of matched bricks and drop sites did not block each other and hence the three bricks may be moved to their corresponding drop sites in any order.

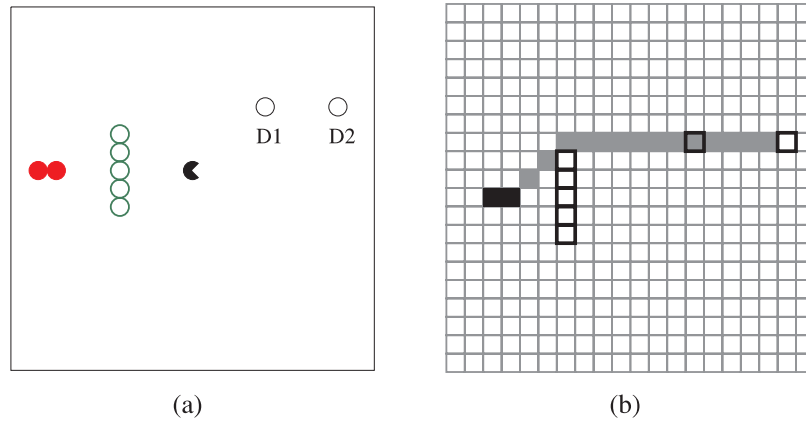


**Figure 17** Example illustrating algorithm `calculateMatching`. (a) Environment containing four bricks (B1, B2, B3, and B4) and three drop sites (D1, D2, and D3). There is also a wall of obstacles. (b) Activation  $a$  spreading from nodes representing bricks. (c) Activation  $b$  spreading from nodes representing drop sites. (d) Sum of  $a$  and  $b$  activations. (e) Brick and drop site pair that are closest to each other. (f) Final matches between bricks and drop sites.

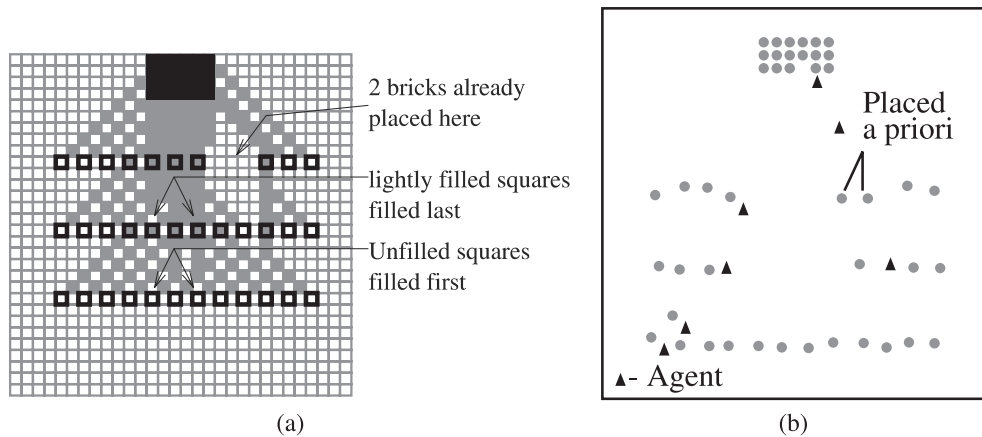
The DSS algorithm determines the order in which drop sites are to be filled such that a dropped brick will not block paths between other bricks and drop sites. The match values are set starting from a neighboring node of a drop-site node (step 4 of the `calculateMatching` algorithm), not the drop-site node itself. Thus, if a drop-site is not present on the path between another drop-site and its matched brick (as is the case in Figure 17f), then its `match` value is 0. Therefore, bricks must be dropped at drop sites whose corresponding `match` value is 0. For instance, consider the two bricks and two drop sites in Figure 18a and the nodes in the corresponding Matching grid that have a `match` value of 1 (Figure 18b). The path from the bricks to the farther drop site D2 passes through the nearer drop site, D1. Therefore, only the `match` value of the node corresponding to D1 is 1. Thus, drop site D2 should be filled *before* D1.

The behaviors that lead to the appropriate selection of bricks and drop sites are as follows. If the agent is not holding a brick, it should move toward the nearest brick that has been matched during the `calculateMatching` algorithm. If the agent is holding a brick, it should move toward the nearest drop site that would not block subsequent access to other unfilled drop sites. Since the `calculateMatching` algorithm is initiated by the activations on the navigation maps, any change in the layout of the environment will cause the activations on the Matching grid to change. In this way, path planning adapts to the changes in the environment caused by the movement of bricks by other construction agents. For instance, after a brick is dropped it becomes an obstacle for all future path planning.

The goal selection behaviors are implemented by having inhibitory connections from the nodes in the Matching grid to the corresponding nodes in the con-



**Figure 18** Example illustrating goal selection. (a) Environment containing two bricks, two drop sites (D1 and D2), and a wall of obstacles. (b) The corresponding Matching grid. Filled gray squares indicate cells with *match* value of 1.

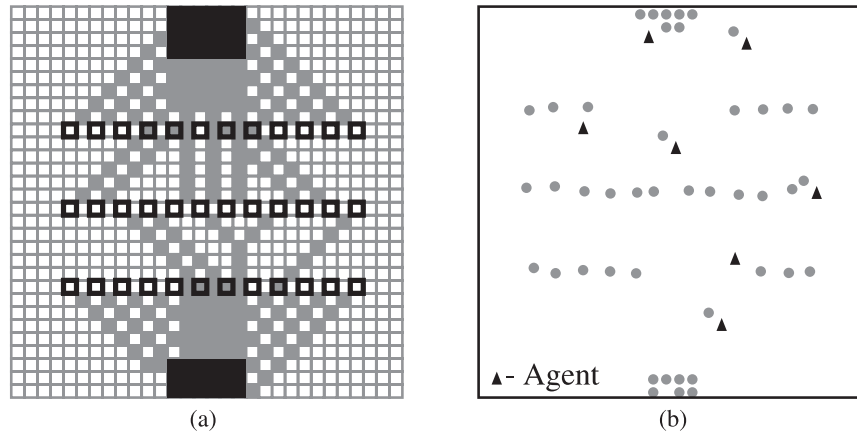


**Figure 19** Environment with one brick source and three walls to be built. (a) *match* values on the nodes of the Matching grid of an agent at the very beginning of construction with two bricks already in place. Black filled squares indicate locations of bricks, squares with thick edges indicate locations of drop sites, and lightly filled squares indicate neurons  $n$  where  $match[n] = 1$  so they will be filled last. (b) The agents have built the bottom-most wall, and parts of the other walls.

figuration NM and excitatory connections to the corresponding nodes in the Brick NM. The inhibitory connections prevent those drop sites that block paths between some matched brick and drop site (i.e., drop sites whose corresponding node on the Matching grid has *match* value 1) from being considered as goal locations. Similarly, the excitatory connections to the brick NM ensure that only those bricks that are matched to some drop site (i.e., bricks whose corresponding node on the Matching grid has *match* value 1) will be considered as goal locations. Since path planning is carried out by spreading activation on the NMs, the agent moves toward the closest selected

brick or drop site (that does not block a farther away site).

**6.2.1 Example 1 of Goal Sequencing** Figure 19a shows the *match* values after running the calculate-Matching algorithm on the Matching grid of an agent at the very start of construction. The task is to build three parallel walls from a single source of bricks. Two of the bricks were placed a priori. The activation flowed around those nodes representing the two bricks placed a priori since these are obstacles to paths between the source of bricks and drop sites.



**Figure 20** Environment with two brick sources and three walls to be built. (a) Activations on the Matching grid of an agent at the very start of construction. Black filled squares indicate locations of bricks, squares with thick edges indicate locations of drop sites, and lightly filled squares indicate neurons  $n$  where  $\text{match}[n] = 1$ . (b) The agents have built the middle wall, and parts of the other walls.

Figure 19b shows the state of the environment after the agents have been performing construction for a while. The goal sequencing behaviors filled the farthest wall first (and those parts of the other walls which do not block a direct path to the farthest wall). An algorithm that always chose the closest goal would have filled these rows of drop sites in the reverse order, requiring the agents to move around the walls placed first. Figure 19b shows that the agents have already built the bottom-most wall, and parts of the other two walls.

**6.2.2 Example 2 of Goal Sequencing** Figure 20a shows the  $\text{match}$  values on the Matching grid of an agent at the start of construction. Some of the drop sites on the middle wall were matched with bricks from the top source, while the other drop sites were matched with bricks from the bottom source since both the brick sources are equidistant from the middle row of drop sites. The paths between the drop sites in the middle row and their matched bricks pass through the central portions of the top and bottom row of drop sites. Thus, the nodes corresponding to the central portions of the top and bottom row of drop sites have their  $\text{match}$  values set to 1 while the  $\text{match}$  value of the nodes corresponding to the middle wall is 0. Therefore, in this environment, the sequencing algorithm fills the middle wall first and then fills the top and bottom walls (Figure 20b). An algorithm which

always selected the closest goals would have filled the two outer walls first requiring the agents to move around these outer walls to reach the middle wall.

### 6.3 Conclusions and Future Work

We presented the ConAg architecture that enables a group of autonomous agents to construct arbitrary structures in their simulated 2-D environment. The ConAg architecture couples a behavior-based action selection mechanism with an explicit spatial representation of the environment around the agent and the shape of the structure to be built. We showed that this arrangement enabled an agent to learn a sequence of construction tasks that was independent of the shape of the structure to be built. The ConAg architecture is an alternative to purely reactive construction mechanisms (Bonabeau et al., 1998, 2000; Howsman et al., 2004; Jones & Matarić, 2003; Theraulaz & Bonabeau, 1995a). The main problem to be solved in a purely reactive construction mechanism is to determine a specific set of behaviors for every different shape of the structure to be built. The ConAg approach of separating the spatial representation of the structure to be built from the representation of the behavior sequence alleviates this problem: different shapes structures are built by initializing the internal spatial map of each agent with a bird's eye view of the structure.

The ConAg architecture has other significant advantages over purely reactive approaches. A ConAg

agent determines its action at every step by comparing the shape of the structure to be built (maintained in a spatial map) with the layout of the spatial environment around it. If parts of the structure are moved after it is built, agents can detect the discrepancy between the desired and actual shapes of the structure, and this triggers the construction behaviors. Thus, a ConAg agent can also repair the structure as part of the construction task. (This also implies that construction can proceed from any initial configuration of the environment.) Purely reactive agents on the other hand utilize features of the environment to keep track of the progress of the construction task. An unexpected change in the environment can cause unintended behaviors to be triggered. Thus, construction ability does not imply the ability to repair in reactive agents. The separation of spatial representation from the behavior sequence also facilitates learning. The time to learn the construction task is independent of the complexity of the structure to be built. The construction sequence that is learned can then be coupled with different spatial maps to build differently shaped structures.

We also showed how path planning for efficient construction can be performed using the spatial representation. This enables agents to move construction material in such a way that parts of the structure being built do not become obstructions. The ability to plan paths is particularly important while building enclosed structures (e.g., shaped like concentric circles). If the outer structure is built before the inner structure, then agents will not be able to complete construction of the inner structure. Explicit path planning is not possible in purely reactive systems.

The ConAg architecture and the path planning algorithms are designed to be used by each agent in a completely distributed multi-agent system. In the current work, we have studied the ConAg architecture in simulation using only a few agents (less than five). However, since the ConAg approach makes extensive use of spatial maps, the performance of the construction task depends on the fidelity of the spatial maps. In the real world, the spatial map is affected by occlusion, sensor and motor errors, and perceptual aliasing. We intend to study how the presence of a large number of agents in the environment affects the maintenance of the spatial maps and the performance of the construction task.

We are also interested in applying the ConAg architecture to a heterogeneous society of construction

agents. For instance, consider a group of agents consisting of a few “foremen” equipped with the ConAg architecture (i.e., with explicit spatial representation) and many purely reactive “worker” agents. The foremen agents use the spatial planning capabilities of their ConAg architecture to mark the environment with cues for the worker agents. The worker agents can utilize these cues to build simple parts of the structure (such as walls).

## Acknowledgments

The authors wish to thank Gerald Chao for his collaboration in developing the sequence learning algorithms for the ConAg architecture. This work was supported in part by an Intel University Research Program grant to the second author.

## References

- Adleman, L. M., Cheng, Q., Goel, A., & Huang, M.-D. A. (2001). Running time and program size for self-assembled squares. In *ACM Symposium on Theory of Computing* (pp. 740–748).
- Arkin, R. (1998). *Behavior-based robotics*. Cambridge, Massachusetts: MIT Press.
- Bonabeau, E., Guérin, S., Snyers, D., Kuntz, P., & Theraulaz, G. (2000). Three-dimensional architectures grown by simple “stigmergic” agents. *Biosystems*, *56*, 13–32.
- Bonabeau, E., Theraulaz, G., Deneubourg, J. L., Aron, S., & Camazine, S. (1997). Self-organization in social insects. *Trends in Ecology and Evolution*, *12*, 188–193.
- Bonabeau, E., Theraulaz, G., Deneubourg, J.-L., Franks, N., Rafelsberger, O., Joly, J.-L. et al. (1998). 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.
- Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*, 14–23.
- Brooks, R. A., Maes, P., Matarić, M. J., & Moore, G. (1990). Lunar base construction robots. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS)* (pp. 389–392).
- Caro, G. D., & Dorigo, M. (1998). AntNet: Distributed stigmergic control for communication networks. *Journal of Artificial Intelligence Research*, *9*, 317–265.
- Castano, A., Behar, A., & Will, P. (2002). The Conro modules for reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, *7*, 403–409.

- Castano, A., Shen, W.-M., & Will, P. (2000). CONRO: Towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8, 309–324.
- Chao, G., & Dyer, M. G. (1999). Concentric spatial maps for neural network based navigation. In *Proceedings of the 9th International Conference on Artificial Neural Networks*.
- Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Cambridge, Massachusetts: MIT Press.
- Crabbe, F., & Dyer, M. G. (1999). Second-order networks for wall-building agents. In *Proceedings of the International Joint Conference on Neural Networks*.
- Crabbe, F., & Dyer, M. G. (2000a). Observation and imitation: Goal sequence learning in neurally controlled construction animats: VI-MAXSON. In *From Animals to Animats 6: Proceedings of the 6th International Conference on Simulation of Adaptive Behavior* (pp. 373–382). Cambridge, Massachusetts: MIT Press.
- Crabbe, F., & Dyer, M. G. (2000b). Goal directed adaptive behavior in second-order neural networks: Learning and evolving in the MAXSON architecture. *Adaptive Behavior*, 8 (pp. 149–172).
- Deneubourg, J. L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., & Chrétien, L. (1991). The dynamics of collective sorting: Robot-like ants and ant-like robots. In *From Animals to Animats: Proceedings of the 1st International Conference on Simulation of Adaptive Behavior* (pp. 356–365). Cambridge, Massachusetts: MIT Press.
- Deneubourg, J.-L., Theraulaz, G., & Beckers, R. (1992). Swarm-made architectures. In *Proceedings of the 1st European Conference on Artificial Life* (pp. 123–133). Cambridge, Massachusetts: Bradford Book/MIT Press.
- Dorigo, M., Maniezzo, V., & Colomi, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26, 29–41.
- Gambardella, L., Taillard, E., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50, 167–176.
- Gaussier, P., Joulain, C., Zrehen, S., Banquet, J. P., & Revel, A. (1997). Visual navigation in an open environment without map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 545–550).
- Gaussier, P., & Zrehen, S. (1995). PerAc: A neural architecture to control artificial animals. *Robotics and Autonomous Systems*, 16, 291–230.
- Holland, O., & Melhuish, C. (1999). Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5, 173–202.
- Howsman, T. G., O'Neil, D., & Craft, M. A. (2004). A stigmergic cooperative multi-robot control architecture. In J. Pollack, M. Bedau, P. Husbands, T. Ikegami, & R. A. Watson (Eds.), *Artificial life IX: Proceedings of the 9th International Conference on the Simulation and Synthesis of Living Systems* (pp. 88–93). Cambridge, Massachusetts: MIT Press.
- Intanagonwivat, C., Govindan, R., & Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*.
- Jones, C., & Mataric, M. J. (2003). From local to global behavior in intelligent self-assembly. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (pp. 721–726).
- Kawauchi, Y., Inaba, M., & Fukuda, T. (1993). A principle of distributed decision making of cellular robotic system (CEBOT). In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 833–838).
- Khatib, O. (1999). Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26, 175–183.
- Kotay, K., & Rus, D. (1999). Locomotion versatility through self-reconfiguration. *Robotics and Autonomous Systems*, 26, 217–232.
- Kotay, K., Rus, D., Vona, M., & McGray, C. (1998). The self-reconfiguring robotic molecule. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (pp. 424–431).
- Ladley, D., & Bullock, S. (2005). The role of logistic constraints on termite construction of chambers and tunnels. *Journal of Theoretical Biology*, 234, 551–564.
- Lagoudakis, M. G. (1998). *Mobile robot local navigation with a polar neural map*. Unpublished master's thesis, University of Southwestern Louisiana.
- Maes, P. (1990). Situated agents can have goals. *Robotics and Autonomous Systems*, 6, 49–70.
- Maes, P. (1991). A bottom-up mechanism for action selection in an artificial creature. In *From Animals to Animats: Proceedings of the 1st International Conference on Simulation of Adaptive Behavior* (pp. 238–246). Cambridge, Massachusetts: MIT Press.
- Mataric, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8, 333–344.
- Mataric, M. J., Nilsson, M., & Simsarian, K. T. (1995). Cooperative multi-robot box-pushing. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- McFarland, D. (1981). *The Oxford companion to animal behaviour*. New York: Oxford University Press.
- Panangadan, A., & Dyer, M. G. (2002). Learning spatial and temporal correlation for navigation in a 2-dimensional continuous world. In *Proceedings of the 11th International Conference on Machine Learning*. San Francisco: Morgan Kaufmann.

- Pirjanian, P., Huntsberger, T., Trebi-Ollennu, A., Aghazarian, H., Das, H., Joshi, S. et al. (2000). CAMPOUT: A control architecture for multi-robot planetary outposts. In *Proceedings of the SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems III* (Vol. 4196, pp. 221–230).
- Revel, A., Gaussier, P., Lepretre, S., & Banquet, J. P. (1998). Planification versus sensory-motor conditioning: what are the issues? In *From Animals to Animats 5: Proceedings of the 5th International Conference on Simulation of Adaptive Behavior* (pp. 129–138). Cambridge, Massachusetts: MIT Press.
- Rothmund, P. W. K., & Winfree, E. (2000). The program-size complexity of self-assembled squares. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (pp. 459–468).
- Schenker, P., Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Das, H., Joshi, S. et al. (2000). Robot work crews for planetary outposts: Close cooperation and coordination of multiple robots. In *Proceedings of the SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems III* (Vol. 4196, pp. 210–220).
- Stewart, R. L., & Russell, R. A. (2006). A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior*, 14, 21–51.
- Terada, Y., & Murata, S. (2004). Automatic assembly system for a large-scale modular structure – hardware design of module and assembler robot. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vol. 3, pp. 2349–2355).
- Theraulaz, G., & Bonabeau, E. (1995a). Coordination in distributed building. *Science*, 269, 686–688.
- Theraulaz, G., & Bonabeau, E. (1995b). Modeling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology*, 177, 381–400.
- Tyrrell, T. (1993a). *Computational mechanisms for action selection*. Unpublished doctoral dissertation, University of Edinburgh.
- Tyrrell, T. (1993b). The use of hierarchies for action selection. *Adaptive Behavior*, 1, 387–420.
- Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Unpublished doctoral dissertation, Cambridge University, Cambridge, England.
- Wawerla, J., Sukhatme, G. S., & Mataric, M. J. (2002). Collective construction with multiple robots. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2696–2701).
- Werfel, J. (2004). Building blocks for multi-robot construction. In R. Alroni (Ed.), *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*.
- Werfel, J., Bar-Yam, Y., Rus, D., & Nagpal, R. (2006). Distributed construction by mobile robots with enhanced building blocks. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2787–2794).

## About the Authors



**Anand V. Panangadan** is a research specialist at the Saban Research Institute of the Childrens Hospital Los Angeles and a post-doctoral affiliate of the Jet Propulsion Laboratory. His current research is in developing machine learning based information processing algorithms for wireless sensor networks. These algorithms are to be deployed in human health monitoring systems (body area networks) and environmental sensor networks. He received his Ph.D. degree in computer science from the University of California, Los Angeles in 2002 and his B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Bombay in 1996.



**Michael G. Dyer** received his Ph.D. in computer science at Yale University in 1982. He joined the UCLA Computer Science Department a year later as assistant professor and became full professor in 1992. He is the author of over 100 publications in artificial intelligence, connectionist/neural networks, natural language processing, cognitive modeling and animat-based modeling. He is author of the book *In-Depth Understanding*, MIT Press (1983) and serves on the editorial board of the journals: *Applied Intelligence*, *Connection Science*, *Knowledge-Based Systems* and *Cognitive Systems Research*. His research is in extraction and acquisition of language semantics via symbolic, neural and evolutionary methods. *Address*: Computer Science Department, 4532F Boelter Hall, University of California at Los Angeles, Los Angeles, CA 90095-1596. *E-mail*: [dyer@cs.ucla.edu](mailto:dyer@cs.ucla.edu)