University of California
Los Angeles

Construction using Autonomous Agents in a Simulated Environment

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Anand Panangadan

2002

Abstract

A behavior-based architecture 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. All objects in the environment are colored discs. Agents can move in their environment and sense discs located around them through distance sensors. An agent can also pick up discs close to them and carry this disc as it moves to be dropped at another location. Construction in this environment involves the agents picking up discs, and then moving to incomplete parts of the structure being built and dropping these discs. The order in which parts of the structure are built affects the completion time of the construction task since discs can obstruct the paths of agents.

An agent has both reactive behaviors which are used primarily to maintain the viability of the agent and navigational planning behaviors that are used for construction. The navigational planning behaviors use an egocentric grid-based representation of the world. Path planning is implemented by spreading activations on sets of grid-based maps. The shape of the structure to be built is also encoded on an internal spatial map.

The connectionist action selection mechanism can learn to exploit spatial and temporal regularities in the environment using its reactive behaviors and also learn to perform sequences of navigational planning behaviors by imitation learning. Each agent monitors its progress to detect deadlocks arising from interactions with other agents and uses unsupervised learning to change its behavior so that the deadlock is broken. Algorithms are implemented on the spatial maps to decide the order in which discs are to be picked up and carried to incomplete parts of the structure in order to reduce the time taken to complete construction. Communication between agents is used to reduce the effect of random sensory and odometry errors on the accuracy of the spatial maps.

We present simulation results that show how the various algorithms perform as parameters and environmental factors are modified.

Acknowledgment

I would like to express my gratitude to my advisor, Prof. Michael G. Dyer, for his constant guidance, support, and encouragement throughout my graduate studies. The many discussions that I had with him played an important role in the shaping of this dissertation. I also thank Prof. Milos D. Ercegovac, Prof. Adnan Darwiche, and Prof. Charles Taylor for serving on my committee. This dissertation was supported in part by an Intel Faculty Development grant.

The work on sequence learning in chapter 5 was jointly done with Gerald Chao and I am grateful for the interest he has shown in this dissertation. This project started where Frederick Crabbe left off and I am thankful for his help in the early stages. I also thank Akash Nanavati for the enthusiastic discussions on various topics in computer science theory.

A lot of people have made my stay at UCLA thoroughly enjoyable. My thanks go to Verra Morgan for always having time for me inspite of her busy schedule, to all my friends, especially Murali Mani who has helped me during many last minutes, and to Lourdes Abellera for her love and encouragement.

My deepest gratitude goes to my parents for their immense love and care.

Contents

Chapter 1

Introduction

As robot technology improves in the future, groups of robots will be expected to exist autonomously and provide services that are useful to man. Construction and related repair of physical structures will be one of the most important of such tasks. For example, a group of robots could build a shelter for humans on a planet otherwise uninhabitable for man. Since the possible environments in which robots can exist, and also the actual physical characteristics of the robots (the sensors, effectors, physical size, etc.) can be very different from each other, this project studies the problem of construction in an artificial two-dimensional world. Although highly abstracted from the real world (as it is 2-dimensional), the simulated environment used here has features that are representative of the real world. The "robots" (called *agents*) are simulated in this environment and their sensors and effectors are similar to those found in many current physical robots. The goal of this project is to construct a society of autonomous agents that live in this artificial two dimensional environment and are capable of building arbitrary structures in the same environment.

The environment contains only colored discs of fixed radius. Agents can move around in this environment and sense discs around them. They can grab a disc located close to them and carry it as they move. This disc can also be dropped at the agent's current location. Construction in this environment involves arranging these discs to form a specific two-dimensional pattern (We will also refer to such patterns to be built as *structures* or *configurations*). An example is shown in figure 1.1.

This dissertation proposes a series of behavior-based architectures with connectionist action selection mechanism that enables each agent to perform the construction task. The connectionist action selection enables the agent to learn to exploit both special features of the environment and also how to learn the construction sequence itself. For instance, an agent can learn to associate discs of one color with those of another color if these differently colored discs always appear together in the environment (*spatial correlations* in the environment). The agent can also learn to associate following a trail of colored discs to reach another colored disc (*temporal correlations* in the environment).

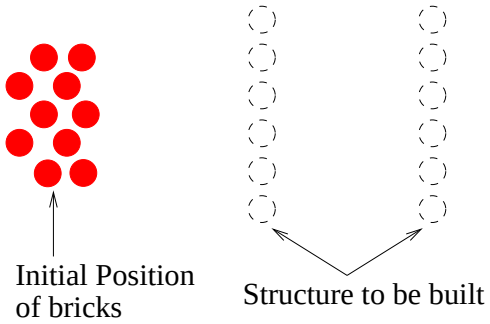


Figure 1.1: Example construction pattern and distribution of discs. Here, agents are to build two walls by using the "pile" (cluster) of discs to the left of the walls.

The architecture also makes use of an egocentric internal spatial representation of the world around the agent. Thus the agent is able to navigate to parts of the environment that are out of its present sensory range. This spatial representation also stores a blueprint of the structure to be built. The advantage of this approach (over hard wiring the agent for one particular kind of structure) 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. Moreover, the spatial maps are grid-based and thus this blueprint is stored as a "bird's eye-view" of the structure to be built. Thus, it is easy for a human to describe the structure in the form required by the agent (compared to the case where a procedural description of building the structure has to provided).

The dissertation also describes algorithms that operate directly on the internal spatial representation to plan the sequence in which parts of the structure are to be built. This is important for efficiency since certain orders of building the structure can block direct paths to remaining parts of the structure. The algorithms all use spreading activations on the grid-based representation and therefore all operations are local computations on the individual grid cells. Thus these algorithms can be parallelized for greater speed.

1.1 Overview of the Chapters

Chapter 2 describes the simulation environment. This includes the sensory and motor capabilities of each agent and also the actions that are required to perform construction. Chapter 3 describes the architecture of the agent designed for a single agent scenario. An example illustrating the use of this architecture to perform construction is shown. Chapters 4 and 5 describe how the connectionist action selection mechanism can be used for learning. Chapter 4 details the changes required to take advantage of regularities in the environment using only reactive behaviors. Chapter 5 describes an imitation learning procedure that can learn sequences of navigational planning behaviors.

Chapter 6 starts the transition into a multi-agent environment. In this chapter, the single agent architecture is used to learn a deadlock escape mechanism that arises out of interactions between agents. The next two chapters describe modifications to the architecture so that the construction task can be performed efficiently by a group of agents. Chapter 7 introduces disc placement strategies that use randomization while chapter 8 describes a more complex planning algorithm to determine the order in which discs are to be placed as part of the structure being built.

The algorithms described in these chapters assume that the spatial maps provide an accurate representation of the locations of the discs around the agent and the shape of the structure being built. Chapter 9 describes the use of communication to reduce the error in spatial maps that are introduced by random errors in the agent's sensors and motors.

Each chapter gives simulation results illustrating the performance of the presented techniques and a discussion of these results. Each chapter ends with references to related work and comparison of these works with the system presented here.

Chapter 2

Environment, Sensors, and Motors

The agents and the discs exist in a simulated 2-dimensional continuous world, similar to that introduced in [Crabbe, 2000; Crabbe and Dyer, 2001]. The discs represent objects that are relevant to an agent - "food", "water", and "bricks". The different kinds of discs are distinguished by their color: green discs are food, blue discs are water and red discs are bricks. All the discs have the same radius (1 unit). Each simulated agent is also simulated as a black disc with the same radius as that of the "passive" discs.

Agents are equipped with distance sensors that enable it to sense other discs. There are separate distance sensors for each color (in multi-agent environments, agents also have sensors that can sense black discs to identify other agents). The sensors are distributed all around the agent (360° field of vision) and each sensor is sensitive to its particular sector. The sensors have a limited sensing range, that covers only a small portion of the environment around them. The maximum range of a sensor can be different for different colors. The activation of each sensor is inversely proportional to the distance of the nearest disc in its field of vision (discs that are farther from the nearest disc are occluded). For instance, let G be the set of green sensors and G_i be the activation of green sensor $i \in G$. If R is the maximum sensor range for green sensors and d the distance from the agent to the center of the nearest green disc present in the sector to which sensor i is aligned, then

$$G_i = 1 - \frac{d}{R} \tag{2.1}$$

Thus, G_i varies from 0 (no green disc present in the sector within sensor range) to 1. Sensor activations are illustrated in figure 2.1. Each agent also has a compass and this is used to align all sensor readings in one global direction.

Errors are an inescapable part of real-world sensors and motors. Random errors are added into the activation levels of the sensors which are used by reactive behaviors. The random error is proportional to the distance of the object being sensed. Thus, as the object being sensed gets closer to the agent, it is sensed more accurately. Another source of sensor error is the use of only a discrete number of distance sensors. A distance sensor cannot distinguish the presence of more than one disc within its field of vision and also cannot determine the angle of a disc within this sector. Real-world sensors also exhibit systematic errors (i.e, not random) [Borenstein *et al.*, 1996]. However, such errors are not modeled in this dissertation.

Repeatedly incorporating sensor data containing errors into the internal spatial map results in inaccuracies accumulating in the map. Construction requires dropping bricks at pre-defined locations and hence an accurate map is necessary to provide navigation goal locations. Moreover, the construction environment is inherently dynamic and not all sensor-world mismatches are caused by sensor errors. Therefore sensor and odometry data that is incorporated into the spatial maps is assumed to be free of error. Chapter 9 describes Kalman filtering methods and communication between agents that reduce the impact of random errors on the internal spatial maps.

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 maximum speed of an agent is 1 unit per time-step. The agents also have inertia and this implies that the motors cannot react instantaneously to the motor commands that are generated by the architecture (the agent cannot speed up, slow down, or turn by arbitrary amounts in a single time-step). The distance moved by the agent in each time-step is obtained from the "wheel encoders".

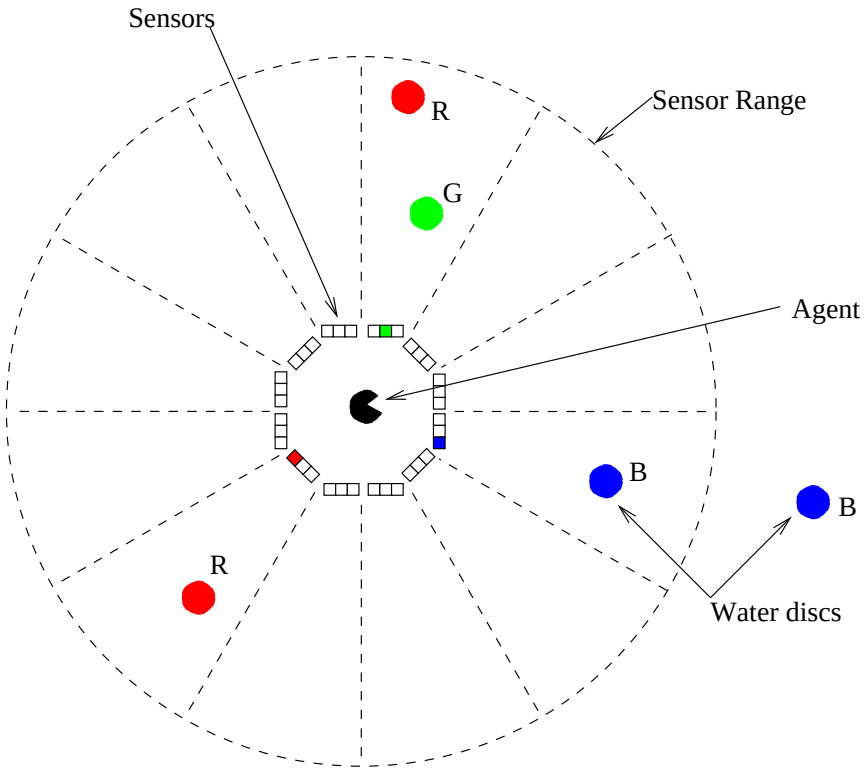


Figure 2.1: Sensor activations around an agent (throughout, a "pac-man" style icon will be used to indicate an agent): Discs can have one of three colors: red (R), blue (B), or green (G). Thus there are three distance sensors oriented toward each of the 12 sectors. The boxes around the agent indicate the activations of each of these sensors (a filled box indicates a non-zero activation). The sensors are sensitive only to the nearest disc in their sector (thus though there is a red and a green disc in one of the sectors, only the green sensor is activated).

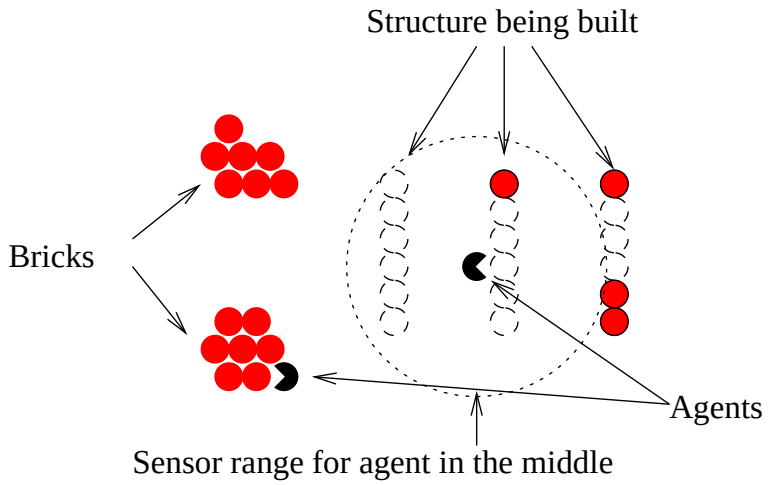


Figure 2.2: Environment with two agents: The sensor range is not long enough for the agent at the right to sense the two sources of bricks. The agent at the left is close enough to a brick to pick it up.

This reading is called odometry information and is used to maintain the spatial maps (chapter 3). If an agent attempts to move into a location that is occupied by another agent or disc, it will not succeed (i.e, the agent will remain in its earlier position). In this case the motors are incapable of following the motor commands and the odometry information indicates that no movement was made.

An agent has a "gripper" with which it can can pick up a brick (food and water discs can not be picked up) 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 (i.e, it is not necessary for the agent to turn and face a brick before attempting to grab it). The agent and the brick that it is carrying occupy the same position. An agent can carry only one brick at a time. Carrying a brick does not affect the sensory capabilities of the agent, but bricks being carried by other agents are no longer visible. The agent can also 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. After a brick is grabbed, the grippers continue to grip it until the drop motor activation is generated.

Construction in this world involves moving toward a brick, "grabbing" the brick when the agent is sufficiently close, moving to one of the designated configuration locations and then "dropping" the brick at that location. An example is shown in figure 2.2.

The "health" of the agent is represented by internal food and water levels. These levels decrease by a constant amount in every time-step (does not depend on the agent carrying a disc). The rate at which the internal food and water levels decrease (per time-step) is denoted by f_0 and w_0 respectively. *Motivations* are internal sensors that monitor these levels and consist of *hunger* and *thirst*. Two thresholds (T_0^h and T_1^h) control the activation of these two motivations. When the internal food level falls below threshold T_0^h, the hunger motivation is activated (the agent becomes "hungry"). To increase the internal food level, the agent must "eat" by touching a food (green) disc. This is a generalization of the tasks that a physical entity will have to perform to ensure continuous operation. For instance, mobile robots have to be charged periodically and they often have to move to a fixed charging station to do so.

The rate at which the internal food and water level increases (per time-step) while eating is denoted by f_1 and w_1 respectively. As in the case of grasping, the agent does not have to face the food (water) disc to eat (drink). Touching a food disc in a time-step increases the internal food level by a constant amount in that time step. When the food level increases above T_1^h, the hunger motivation becomes inactive (the agent is "satiated"). This process is illustrated in figure 2.3 which shows the values of the internal food level and the hunger motivation. Similarly, the internal water level is monitored by the thirst motivation and is

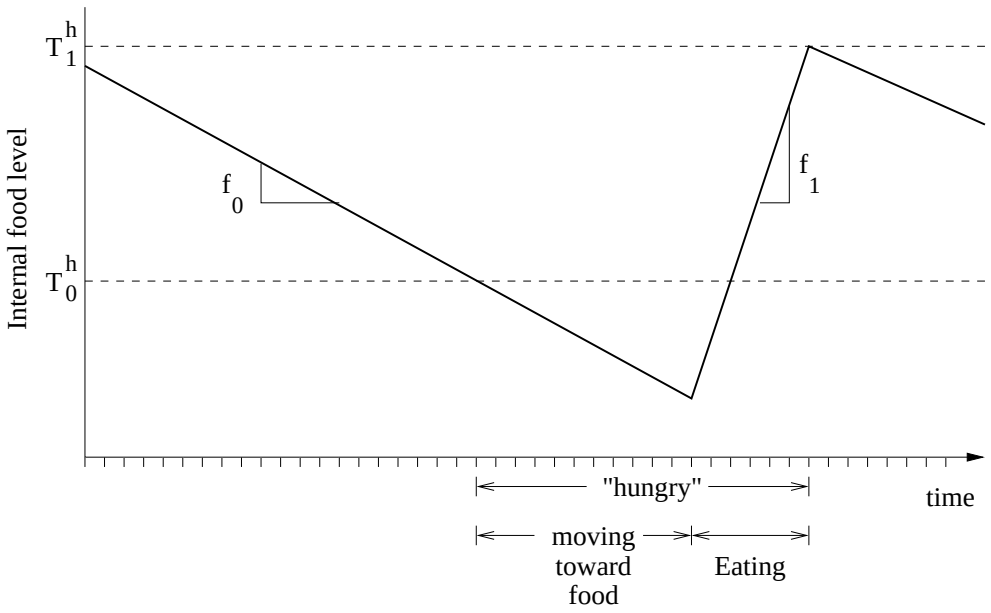


Figure 2.3: Change in internal food level and hunger motivation: Internal food level against time with rate f_0. When the food level falls below T_0^h the agent becomes "hungry" (hunger motivation is activated) and the agent starts moving toward a food (green) disc. When the food level goes above T_1^h the hunger motivation becomes inactive. Eating (touching a food disc) increases the food level by an amount specified by the rate f_1.

controlled by thresholds T_0^t and T_1^t. The agent "drinks" by touching a water (blue) disc. Eating or drinking does not change the food or water disc (either in size or location). In addition, there is an *avoid* motivation that is always active so that the agent does not collide with objects.

Biological systems have motivational units that indicate when the agent is hungry or thirsty. These include nerve receptors in the stomach to detect hunger and in the mouth for thirst [Toates, 1986]. It has also been shown in simulation that organisms can evolve to match the activations on their motivational units to those behaviors that affect the motivations [Cecconi and Parisi, 1992]. Such organisms also exhibit more adaptive behavior and have a higher fitness (as defined for the simulated environment of [Cecconi and Parisi, 1992]).

2.1 Simulation vs Physical Implementation

The work has been demonstrated in simulation because the computational time of the algorithms make them impossible to be executed in real time as would be required if they were to be implemented on physical robots. The construction task also requires good sensor and effector capabilities since the agents have to get close to the discs, pick them up and carry them to their destination. Another important reason to require good sensors and motors is that an accurate internal spatial map has to be maintained not only for navigation as in most physical systems, but also for constant comparison with the blueprint of the structure to be built. A large proportion of the current work using physical robots do not manipulate their environment (that is only navigation is performed; for example, [Mataric, 1992; Kuipers and Byun, 1987]). Physical systems that do manipulate their environment are often implemented as purely reactive machines (for example, [Østergaard *et al.*, 2001]). Coarse sensing capabilities are sufficient for reactive robots since they do not maintain an spatial representation. Moreover, when there are multiple construction agents present, the environment can significantly change due to the actions of agents that are beyond sensor range. Chapter 9 provides techniques to maintain the accuracy of the spatial maps in the presence of random sensory and motor errors.

It is much more difficulty to study learning in physical systems compared to simulated systems because of the time it takes to complete a robot trial. This is one of the main research themes behind the Simulation League in the RoboCup robotic soccer championships [Stone, 2001]. Rapid progress is being made in the sensory capabilities of robots. For example, laser range finders which are much more accurate than traditional sonar range finders are becoming more prevalent on robots. The issues arising from sensing are thus likely to change in the future.

2.2 Related Work in Construction

Crabbe and Dyer presents a society of autonomous agents that work together to build simple structures like walls and briar patches in a two dimensional world containing colored discs similar to the one used in this dissertation [1999a]. The architecture of the agents is completely neural and the sequence of steps for construction is performed using second-order networks (the architecture is described in more detail in chapter 3). The agents do not maintain an internal model of the world and hence the steps in construction are marked by "painting" the discs. For example, to build a wall, agents perform the following steps:

1. Move towards a red disc.
2. Carry red disc to a brown disc (the "pile-marker") and color the red disc orange (orange discs are ready to be used as part of a wall).
3. Carry an orange disc to a purple disc. The start of the wall is marked by a purple disc placed initially by the designer.
4. Color orange disc purple.

These steps are shown in figure 2.4. The wall is built between the (initially placed) purple and brown discs. Steps 1 and 2 are performed by scavenging agents while steps 3 and 4 are performed by wall-building agents (the type of an agent is fixed by the designer). Similar steps were shown for constructing corridors, intersections, and briar patches.

The nest-building abilities of colonies of ants and wasps have inspired construction systems in simulated 2- and 3-dimensional environments [Therauluz and Bonabeau, 1995; Deneubourg *et al.*, 1992]. The individual agents behave by following fixed rules that depend 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).

This is an instance of *stigmergy* which is the indirect coordination between individuals through their actions on the environment [Beckers *et al.*, 1994]. The emphasis in these works is on building natural looking structures. For instance, a simulated colony of ants could build pillars and walls using only the concentration of pheromones to guide their behavior [Bonabeau *et al.*, 1998]. (This is an example of *quantitative stigmergy* where agents react only to the magnitude of some stimulus [Theraulaz and Bonabeau, 1999]. *Qualitative stigmergy* is exhibited when agents recognize and react to different kinds of stimuli. For instance, by building new hexagonal cells at the junctions of already placed cells, wasps can construct large nests [Karsai, 1999]). The main issue is to specify a set of conditions that these local rules must satisfy for the structures to be built reliably (that is independent of the order in which different agents take their steps). This approach is different from that in [Crabbe and Dyer, 1999a] in that the behavior of a construction agent is dictated solely by its local view of the structure being built. The behaviors of the scavengers and wall-builders of Crabbe and Dyer do not change with the changing shape of the structure being built. However, these two approaches are similar in that the rules do not make use of a general purpose internal spatial representation and thus it is not obvious how arbitrary structures can be constructed. The structures built using these systems also depend on the initial positions of the elements of the environment. Small changes in the environment can lead to significant changes in the shape of the structure being built. For instance, if an agent approaches a purple disc from a slightly different direction, the angle of the wall will begin to diverge from its intended direction [Crabbe and Dyer, 1999a].

The work presented in this dissertation removes some of the inherent limitations of these approaches. Firstly, an internal spatial representation is used to remove the need for changing the environment for keeping

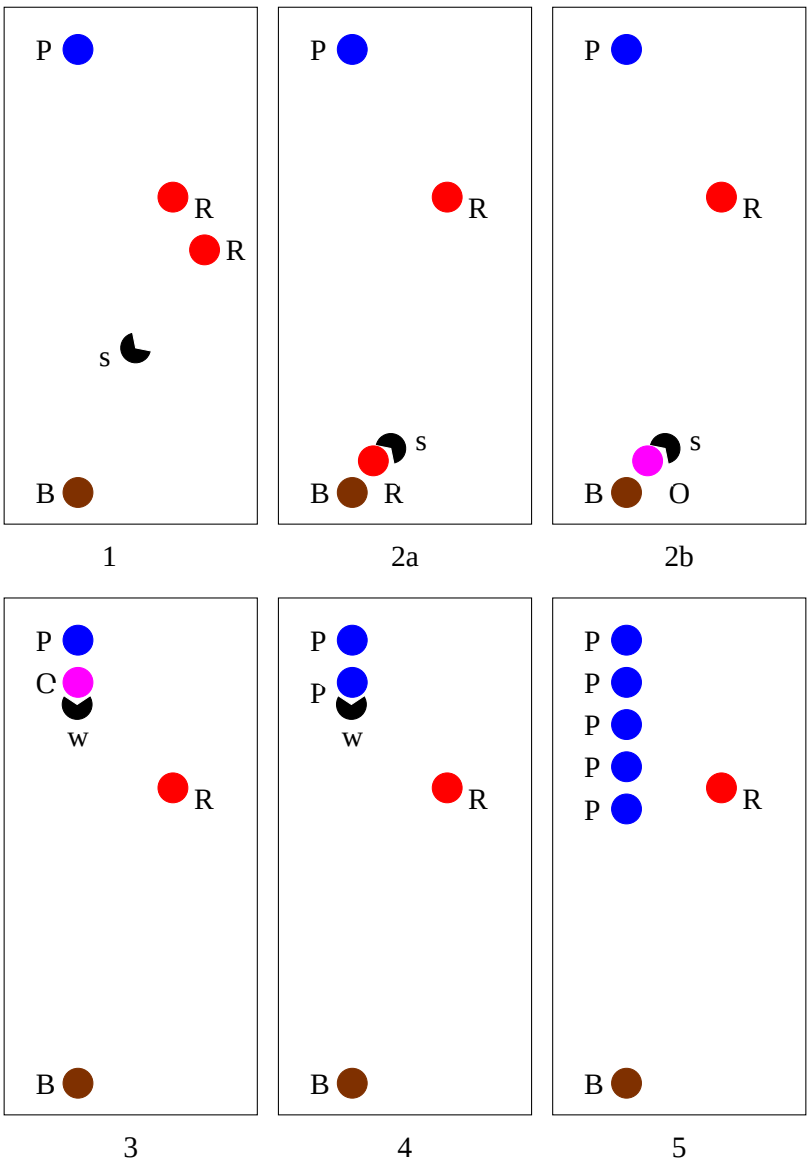


Figure 2.4: Steps to build a wall - arranging discs in a line [Crabbe and Dyer, 1999a]. The wall is to be built between the purple (P) and Brown discs (B). 1. Scavenger agent (s) moves toward a red (R) disc. 2a. Agent s carries a red disc to a brown disc and drops it. 2b. Agent s colors red disc, orange (O). 3. Wall-builder agent (b) carries an orange disc to a purple disc and drops it. 4. Agent b colors dropped disc purple. 5. Repeated application of steps 1-4 results in the wall being extended.

track of the construction sequence (coloring discs). Moreover, this also allows more reasonable sensor ranges (the sensor ranges were either essentially infinite [Crabbe and Dyer, 1999a] or limited to the immediate surroundings [Therauluz and Bonabeau, 1995; Deneubourg *et al.*, 1992]). Secondly, the agents are provided with a blueprint of the structure to be built. This removes the need for specifying the different steps required to build each different shape – one sequence of steps with goals obtained from the spatial representation can build all structures. Also, the shape of the structure being built is continuously matched with the blueprint so that any discrepancy can be removed throughout the construction sequence. Thirdly, the agents can determine the order in which parts of the structure have to be built for more efficient construction. This order is determined by performing computations on the internal spatial representation of both the structure to be built and also the layout of the environment (discussed in detail in chapter 7 and chapter 8). Finally, the architecture is designed to facilitate learning of the construction sequence from a teacher (chapter 5).

Foraging is the task of first searching the environment for certain objects that when found have to be moved to a pre-specified "home" area. It is similar to construction in that agents have to move objects from one location in the environment to another. Though foraging can be performed by one agent, it can be accomplished more efficiently by multiple agents. For this reason it is used for demonstrating and comparing various multi-robot architectures [Cao *et al.*, 1997]. Both foraging and construction systems face some similar issues such as reducing interference and efficiently dividing the task between the many agents. However, the major difference between foraging and construction is that while in foraging objects have to be deposited at a general home location, construction agents have to move objects and place them in a specified pattern. Thus construction agents face the additional issues of representing the structure to be built and computing different navigational goals at every step of the task (foraging tasks have only one "home" location). Moreover, construction agents require better sensory capabilities than foraging tasks since the objects have to be placed in precise locations. (Foraging systems are further discussed in chapter 6).

A more limited form of environment manipulation takes place in tasks such as coordinated object moving. Mataric *et al.* present two autonomous robots that with the help of communication, push a box together towards a goal region [1995]. Khatib describes an architecture that allows groups of mobile arm robots to interact with each other or with a human to do construction or building tasks [1999]. The robots interact by grabbing the same piece of construction material (eg., dry wall) and using the forces exerted by each robot/human to guide the task. 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]. 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. The main focus of all 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. In this dissertation, the fine-grained actions that would be required to manipulate a physical object are not studied (for instance, an atomic "grab" operation is assumed to be available to the agents).

Physical systems that can construct large-scale structures in the real world do not exist yet. One of the important practical applications of autonomous construction robots will be in building habitable structures for humans on other planets. The CAMPOUT architecture was designed with this long-term goal [Schenker *et al.*, 2000]. Brooks *et al.* present a society of simulated agents that jointly perform tasks like digging trenches and collecting rocks which might be useful in planetary base construction [1990]. The approach used is inspired by insect colonies as the agents are completely distributed and do not communicate with each other.

An extreme case of robots that manipulate the environment are robots that can change their own structure to perform different type of tasks. Cellular Robots (CEBOTs) are composed of *cells*, each of which has its own sensors and perform actions like moving and bending [Kawauchi *et al.*, 1993]. The cells can communicate with one another and physically couple with other cells to perform different manipulation tasks. The architecture is distributed, but some cells are designated as *master* cells and these coordinate sub-tasks. CONRO robots are reconfigurable robots that are composed of small, inter-connecting, homogeneous modules [Castano *et al.*, 2000]. The modules can be connected in different ways to create different shaped robots (such as a "snake" or a ring) and can also form different sized robots (many small robots or one large robot). The different configurations (shapes) lead to different gaits for movement. Simple physical robots were made to "evolve"

by specifying their generic structure and manufacturing them using rapid prototyping technology [Lipson and Pollack, 2000]. The robots consisted only of bars connected by ball joints (no sensors). Offspring could have a different number and configuration of blocks/joints from their parents. Their fitness was measured by how well they could move. Self-organization has also been studied theoretically using the Tile Assembly Model [Rothemund and Winfree, 2000]. This model considers a collection of square tiles in a 2-dimensional plane with different kinds of "glue" on each side. The kind of "glue" determines which tiles can stick together. The tiles come into contact based on a probabilistic model. The agents described in this dissertation have no "self-knowledge" and lack the ability to change their internal architecture or structure.

Chapter 3

Architecture

3.1 Introduction

Traditional robots like Shakey [Nilsson, 1984] had a hierarchical control structure with the planner at the top. Though Shakey had a good planner, its moves were prone to failure. It also took a long time to plan even a single step. This was because it is difficult and computationally expensive to maintain a reliable model of the world from the information provided by the sensors. As the time between sensing and acting increased, the accuracy of the internal world model decreased. Moreover, real world sensors are inherently noisy and this noise will be manifested in the internal world model.

In a robot with a behavior-based architecture, every module directly accesses the sensory information and produces a motor activation. Each of these modules, called *behaviors*, performs only a simple computation on the sensor input and this ensures fast reaction times. The robot is also robust since all behaviors produce a motor action and the failure of a behavior will not bring down the robot completely. Figure 3.1 shows the generic layout of a behavior-based architecture as described by Arkin [1998]. f is a function which takes the motor outputs generated by the behaviors (labeled $B_1, B_2, \ldots, B_n$) and generates the motor activation that is sent to the motors. f could either select just one of the motor outputs generated by the behaviors ("winner-take-all" selection) or some combination of all the motor outputs (for example, all the behaviors could vote on each of the possible motor actions and the motor action that receives the most votes would be performed).

The architecture described in this dissertation, called the ConAg architecture, is behavior-based with connectionist action selection. A grid-based spatial representation is used for planning paths to locations that are out of sensor range and also to represent the pattern of the structure that is to be built. The

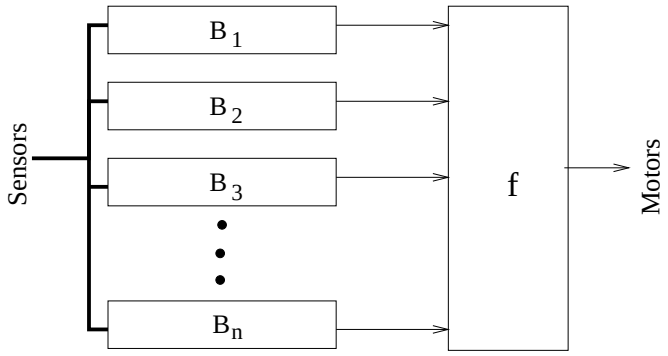


Figure 3.1: Block diagram of a behavior-based architecture. The behaviors are labeled $B_1, B_2, \ldots, B_n$. f is a function generates the motor output based on the motor actions generated by the individual behaviors [Arkin, 1998].

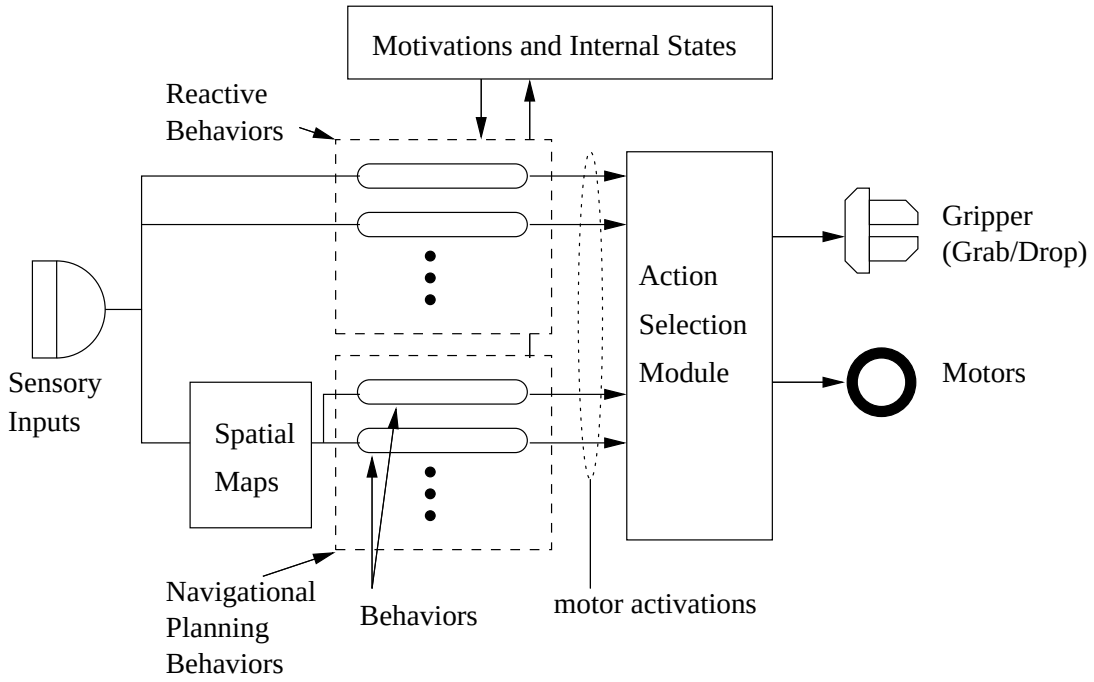


Figure 3.2: Block diagram of the ConAg architecture. The sensory inputs are processed by the Sensory/Motor and the Navigational Planning behaviors, producing all possible motor actions. They are then concurrently fed into the Action Selection module, which then chooses one action and sends it to the motors. The current stage of the construction task is encoded into the internal state nodes while the motivations are indicators of the health of the agent.

behaviors are divided into two groups:

1. Sensory/Motor Behaviors: The Sensory/Motor module maintains the overall health of the agent by generating motor actions in response to objects within the sensor range. These behaviors are therefore purely reactive (the output of each behavior depends only on the current sensory input).
2. Navigational Planning Behaviors: The Navigational Planning module builds and maintains the internal 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 desired locations such as to food discs when the agent is hungry.

The motor activation outputs of these behaviors are then sent to the action selection module which selects one of them to be sent to the motors. The Action Selection module attaches a higher priority to those behaviors responsible for maintaining the agent's health over those required for construction. Since the behaviors that have to be performed will be different during different stages of the construction task, the current state of construction is encoded into *Internal State* nodes. The block diagram of the architecture is shown in figure 3.2 and the modules are described in detail below.

3.2 Sensory/Motor Behaviors

The reactive behaviors available to the agent are shown in figure 3.3. These include *Avoid-x* and *Approach-x* behaviors where x is a color: Green, Blue, or Red. The "avoid" behaviors take the agent in a direction away from visible discs of that color (for obstacle avoidance), while the "approach" behaviors take the agent toward the nearest visible disc of that color. There are also *Eat* and *Drink* behaviors, and an *Explore* behavior that outputs a random motor action.

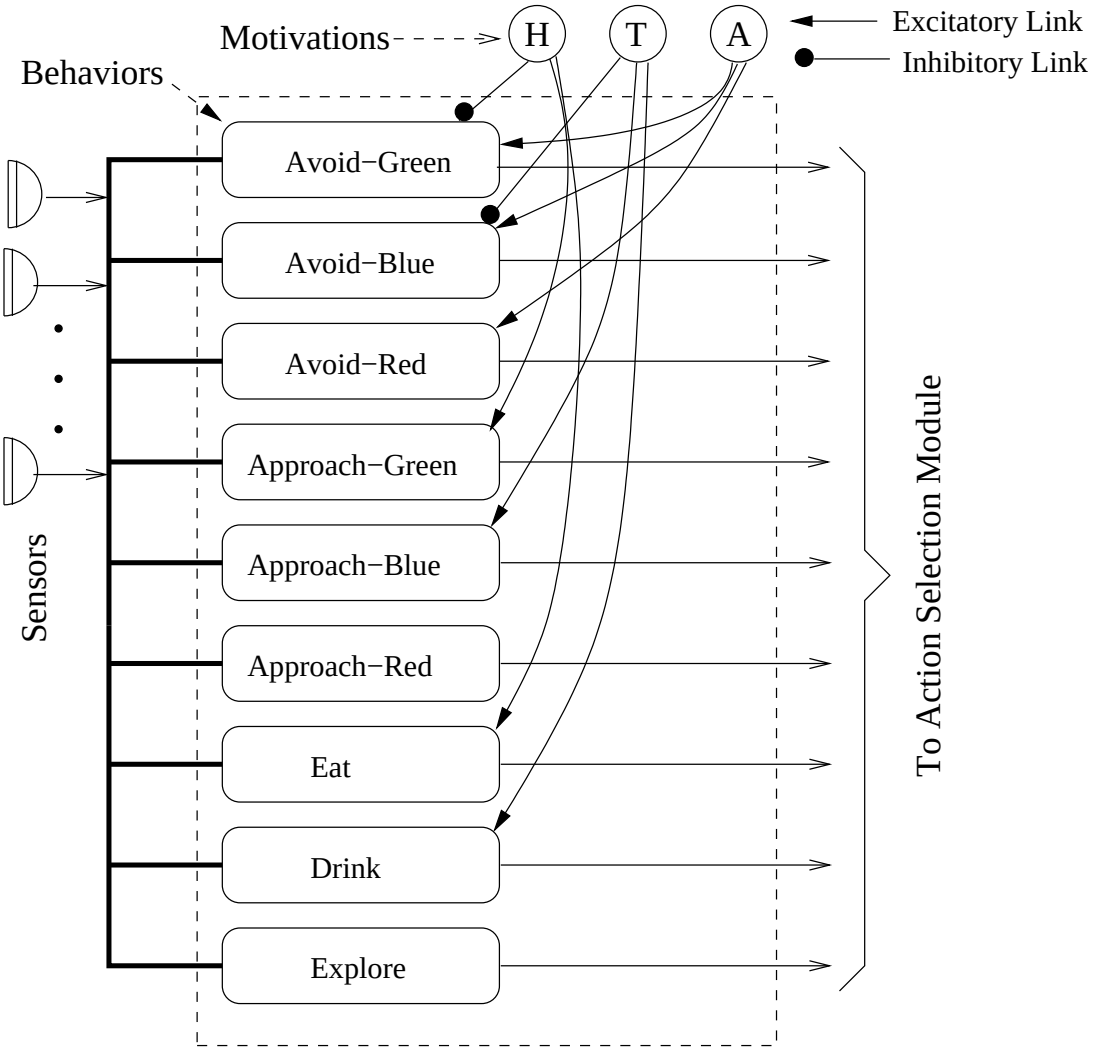


Figure 3.3: Sensory/Motor behaviors. Activations flow from left to right: from the sensors, through the behaviors that are regulated by the *Hunger* (H), *Thirst* (T) and *Avoid* (A) motivations, to the action selection module.

Each behavior takes the sensor data concurrently and outputs a motor activation obtained from simple computations on these sensor activations. For instance, the Approach-Green behavior takes the activations from the green sensors and outputs the direction which has the maximum sensor activation. Since sensor activation is inversely proportional to the distance of the sensed disc, this motor activation leads the agent to the nearest green disc. The avoid behaviors output motor activations that direct the agent away from discs that are situated close to the current location of the agent. A more detailed description of the approach and avoid behaviors is given in chapter 4. The Eat (Drink) behavior generates motor activations that enable the agent to eat (drink) from a green (blue) disc situated near it. The Eat (Drink) behavior is activated only when the food (water) disc is sensed very close to the agent. The motor activations generated by these behaviors set the motor speed to zero so that the agent stops moving and remains close to the food or water disc. 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. Thus, when the agent passes close to a disc, these avoid behaviors generate motor activations that take the agent away from the disc. 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 (so that the agent can get close enough to the green disc to eat). Similarly, the Thirst motivation excites the Approach-Blue behavior and inhibits the Avoid-Blue behavior. Note that there are no links to the Approach-Red behavior from the motivations. This behavior is needed only during construction and hence is regulated by one of the Internal State nodes (the connections from the internal state nodes are learned and this is described in chapter 5). The Hunger and Thirst motivations also excite the Eat and Drink behaviors respectively. Since these behaviors cause the agent to stop close to a food (water) disc, the agent eats (drinks) as long as these behaviors are selected. Once the internal food (water) level has increased beyond T_1^h (T_1^t), the hunger (thirst) motivation becomes inactive, another behavior is selected and the agent moves away from the food (water) disc and stops eating (drinking). The explore behavior is always excited and thus this behavior is the "default" behavior if all of the other behaviors are inactive.

3.3 Navigational Planning Behaviors

The Sensory/Motor module responds only to current sensory inputs and is thus completely reactive. 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 (ESMs) [Chao and Dyer, 1999] to represent the spatial relationship between the agent and each kind of disc in the environment.

3.3.1 Egocentric Spatial Maps

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 represents 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 "birds-eye" view of the world around it. The neurons close to the center of the ESM also 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 3.4). Thus changes in the world, such as the addition or disappearance of discs, are reflected in the ESM.

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. The amount moved by the agent is obtained from the dead-reckoning inputs. In general, the distance moved by an agent will not be an integral multiple of the ESM grid size. To account for smaller movements, each ESM cell also stores the (x, y) offsets of the disc represented in that cell. In every time-step, the dead-reckoning input is added to

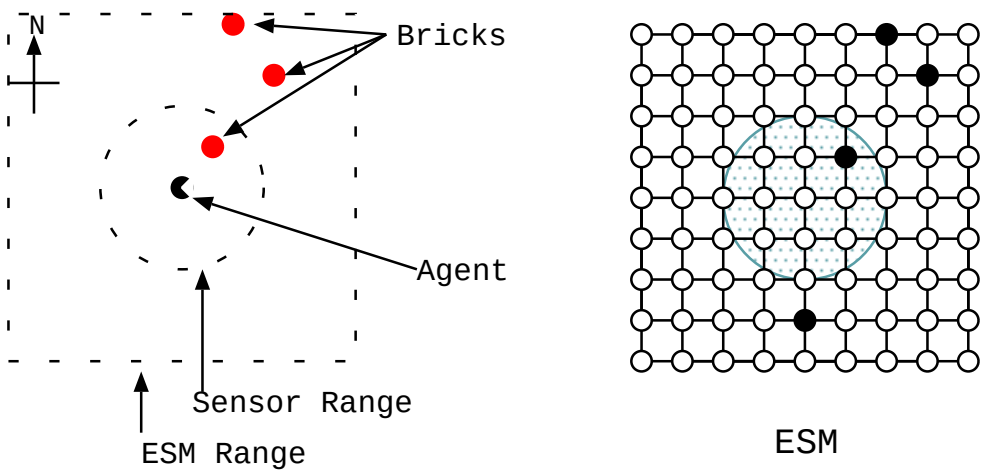


Figure 3.4: Updating an ESM from sensors. The central portion (shaded) of the ESM represents the same space sensed by the distance sensors. These sensor activations are integrated into the ESM to keep the map current. Activations of neurons that encode space outside the sensor range might not correspond to actual discs in the environment due to actions of other agents (the ESM shown contains an activated neuron "south" of the center that has no corresponding disc in the environment). Only connections from each neuron to its nearest four neighbors are shown for clarity.

the offset stored in each cell. If this offset falls outside the area represented by the cell, the cell activation is passed to the neighboring cell along with the offset (since the maximum speed of an agent is 1 unit per time-step, the offset will never fall outside the area represented by the neighboring neurons). The activation of the original cell is reset to zero to indicate that a disc is no longer present in the area represented by that cell. Though a cell offset can store continuous values, the ESM still effectively discretizes the space around the agent because a cell can only store one offset value. In particular, if two or more discs are located within the space represented by a cell, only the offsets for one will be stored in the cell. Thus, the grid size of an ESM determines how finely space is represented. The shifting of neuron activations is done procedurally by purely local computations (the new neuron activation and offset value depend only on the neighboring ESM neurons). Figure 3.5 shows an agent surrounded with two red discs (only one of which lies completely in an ESM cell) and the corresponding activations on its Brick ESM. As the agent moves by a distance half the cell size, the activation of one of the ESM neurons is passed to a neighboring cell while the activation on the other neuron remains the same.

If the activation of a neuron is moved out of the map's range, then the agent "forgets" about the corresponding disc. This process is illustrated in figure 3.6.

3.3.2 Integrating Multiple ESMs

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 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.

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 (equivalently, nodes representing obstacles and their interconnections are removed from the grid). Let n denote an arbitrary neuron in a Navigation map and $nb(n)$ the set of neighboring neurons of n. Let $a_n(t)$ be the activation of n at time t.

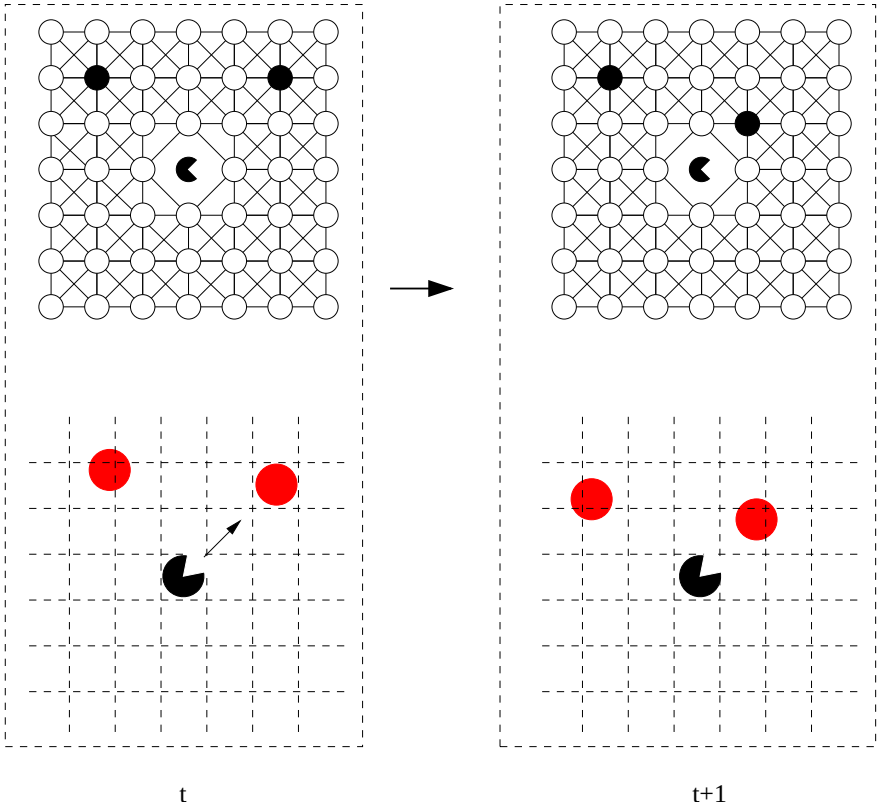


Figure 3.5: Updating an ESM with agent's movement: The arrangement of bricks at time t and corresponding ESM (darkened circles represent firing neurons). The dashed lines indicate the discretization of space by the ESM neurons. As the agent moves diagonally by about half the grid size (time $t + 1$), the ESM activations are shifted in the opposite direction. Only the activation of one neuron is shifted to a neighboring neuron; the offset of the other neuron remains inside the same grid cell at both times t and $t + 1$.

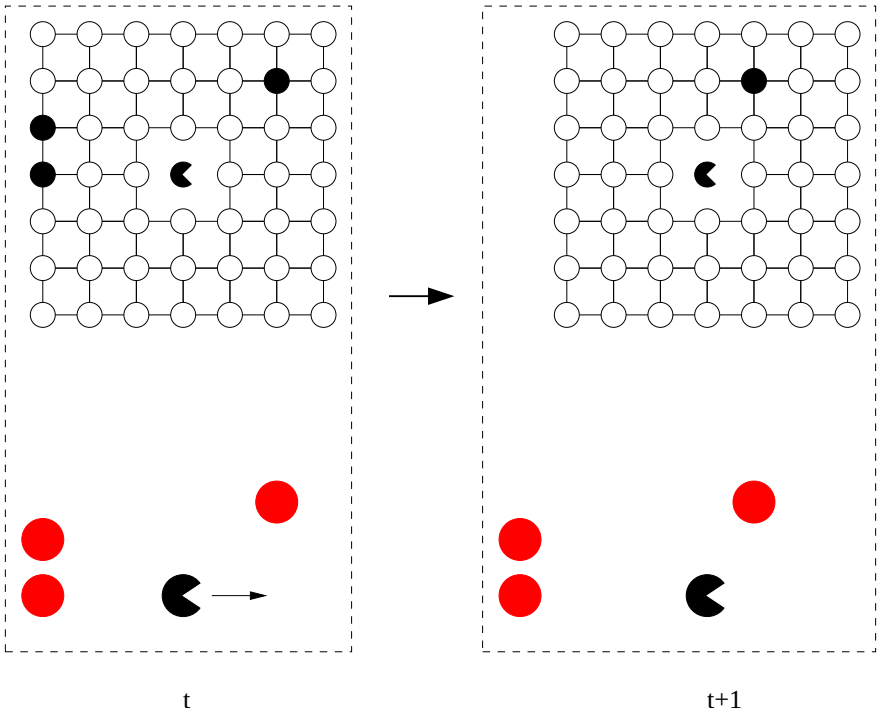


Figure 3.6: Forgetting activations as discs move out of range: The arrangement of bricks at time t and corresponding ESM (darkened circles represent firing neurons). As the agent moves to the right (time $t+1$), the ESM activations are shifted to the left to reflect the change. The two discs to the left are out of ESM range at time $t+1$ and their corresponding activations are "forgotten". Only connections from each neuron to its nearest four neighbors are shown for clarity.

$$a_n(0) = \begin{cases} 1, & \text{if } n \text{ represents a goal location} \\ -1, & \text{if } n \text{ represents an obstacle} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$a_n(t+1) = \max_{m \in nb(n)} (a_m(t) - d(n,m)), a_n(t) \geq 0 \tag{3.2}$$

where $d(m,n)$ is proportional to the Euclidean distance between the locations represented by nodes m and n. $d(m,n)$ encodes the distance relationship between neighboring nodes and can take only two values:

$$d(n,m) = \begin{cases} 1/M, & \text{if } n,m \text{ are horizontal/vertical neighbors} \\ \sqrt{2}/M, & \text{if } n,m \text{ are diagonal neighbors} \end{cases} \tag{3.3}$$

where M is a large number such that activations of non-obstacle nodes do not take negative values. Since the maximum distance between two nodes cannot be larger than the number of nodes in the grid, M can be chosen such that it is larger than the total number of nodes.

Equation 3.2 is iterated until the activations stop changing:

$$a_n(t+1) = a_n(t) \forall n \tag{3.4}$$

Spreading activation is a parallel implementation of Dijkstra's shortest path algorithm [Cormen *et al.*, 1990]. The gradient created by the activation is the planned path to the nearest goal location. Let n and m be neighboring nodes. Then, the gradient at n in the direction from n to m, $gradient(n,m)$, is given by

$$gradient(n,m) = \frac{a_m - a_n}{d(m,n)} \tag{3.5}$$

To reach this goal from its current location, the agent should move along the maximum gradient of activation at the center of the Navigation map. This is achieved by moving toward the area represented by that node c^* from central node c, where

$$gradient(c^*, i) = \max_{i \in nb(c)} (gradient(c,i)) \tag{3.6}$$

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 (the motors cannot immediately turn the agent in the direction given by the maximum gradient of activation).

The spreading activation algorithm actually calculates the shortest path to the nearest goal location from any node, not just the central node. At iteration i, the activation is set on all nodes from where a goal location can be reached by passing through exactly i cells. Thus, once the activation of a node is set in an iteration, that node's activation will not change in later iterations. Therefore the iterations can be stopped once the activation has reached the central node. The shortest path from a goal node to the central node can contain between 0 and N nodes, where N is the total number of nodes. Thus, equation 3.2 has to be iterated at most N times. However, in non-maze like environments, the number of nodes in the shortest path is of the order of $\sqrt{N}$ nodes and therefore the spreading activation can be terminated earlier.

An example illustrating spreading activation from two goal locations is shown in figure 3.7. The maximum gradient at the center is available after two iterations. Further iterations will only set the activations of those nodes that were not set in the first two iterations but will not change the maximum gradient at the center.

If there is more than one node that satisfies equation 3.6, then any one of those nodes may be chosen arbitrarily. This is shown in figure 3.8. In this case, one of the goal locations is behind a "wall" of obstacles. The activation flows around this wall and reaches the center from either end of the wall. Thus, there are two shortest paths, and the agent can move either to the area represented by the upper-right or lower-right nodes.

Every ESM is associated one-to-one with a Navigation Map i.e, each node in the ESM has an excitatory link to its corresponding node in the Navigation map. The excitatory links from active nodes represent the goal nodes for computing a path to the nearest disc of the color represented in the ESM. Inhibitory links from the active nodes in other ESMs represent obstacles in this path. For instance, the Food ESM is associated

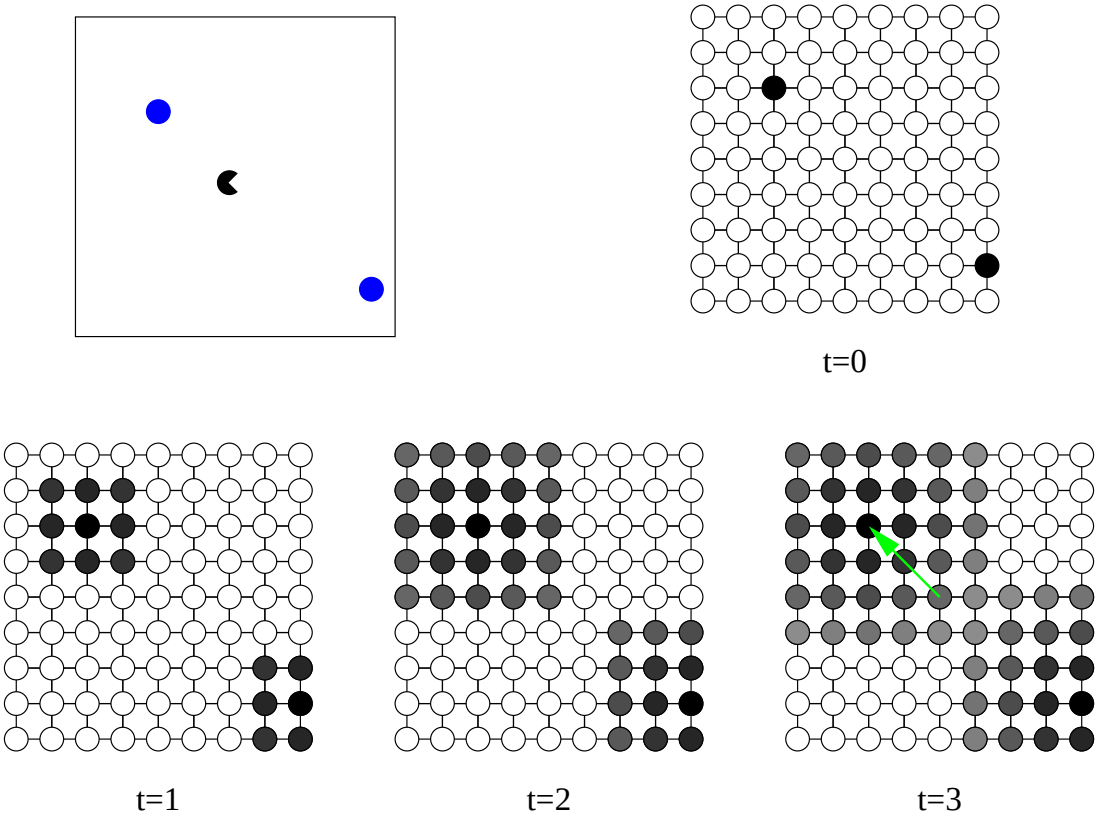


Figure 3.7: Spreading activation from two goal locations: The agent is surrounded by two blue discs (goal locations). The darkness of a circle indicates the magnitude of activation. The activations of the neurons at consecutive iterations of spreading activation are shown. At $t \geq 2$, the maximum gradient at the central node is toward the upper-left node. The planned path (sequence of maximum gradient nodes) is shown superimposed on the grid at $t = 3$.

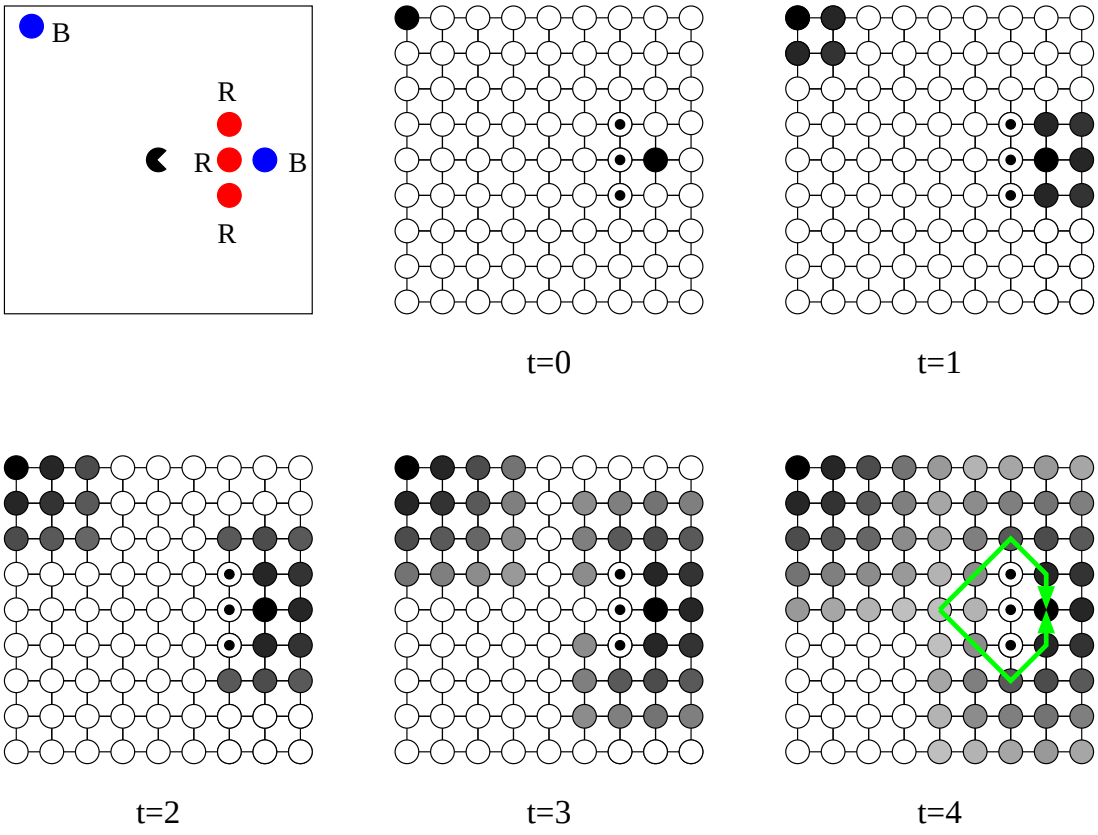


Figure 3.8: Spreading activation from two goal locations: Spreading activation in the presence of obstacles: The agent is surrounded by two blue (B) discs (goal locations) and a "wall" of three bricks (R; obstacles). The darkness of a circle indicates the magnitude of activation while circles with filled centers indicate nodes with negative activation. The activation flows on either side of the wall and there are two shortest paths from the center to the goal location at the right. The planned path (sequence of maximum gradient nodes) is shown superimposed on the grid at $t = 4$.

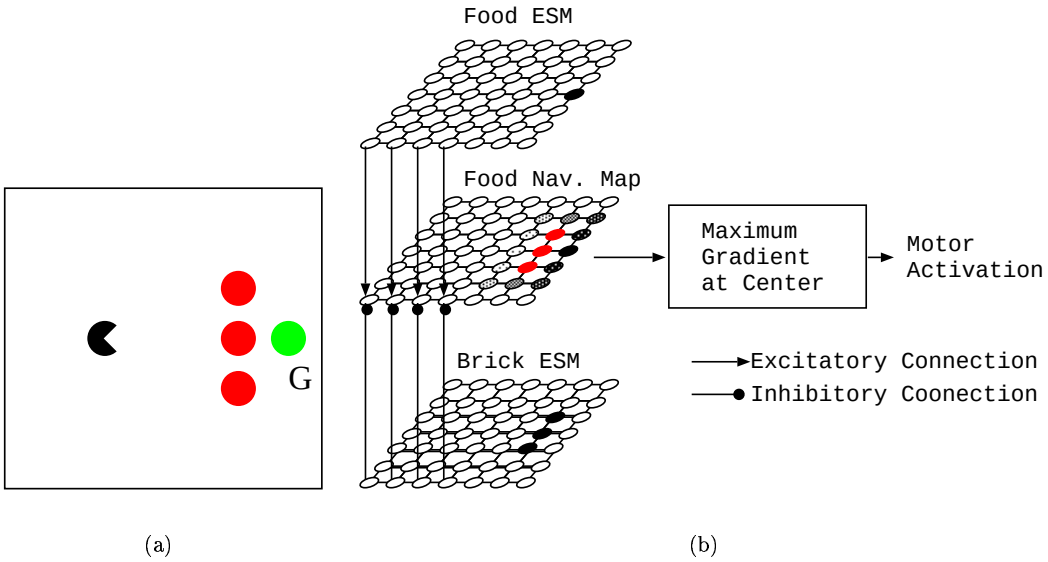


Figure 3.9: Food Navigation map: (a) A wall of three bricks separating agent from a food disc (G). (b) Food ESM, Brick ESM and Food Navigation Map. Only the excitatory (from Food ESM to the Food Navigation map) and inhibitory (from Brick ESM to the navigation map) connections for four of the nodes are shown for clarity. The Water ESM has also been omitted for clarity. The nodes on the Food Navigation map compute a spreading activation (a red node indicates negative activation while the darkness of shaded circles indicate magnitude of activation) and the maximum gradient at the center is the motor output of the Food navigation behavior.

with the *Food Navigation Map* that computes a path to the nearest green disc. The active neurons from the Food ESM represent the goal locations and activate their corresponding neurons in the Food Navigation Map. All other types of discs are obstacles in this path (for instance, water discs become obstacles when the agent is planning a path toward food) and therefore inhibitory links are projected from the Water and Brick ESMs to the corresponding nodes on the Food Navigation Map. This is illustrated in figure 3.9 which shows the Food and Brick ESMs and the Food Navigation Map. There is a food disc behind a "wall" of three bricks. To plan a path to the food disc, the activated node in the Food ESM excite its corresponding node in the Food Navigation Map. At the same time, the three activated nodes in the Brick ESM inhibit their corresponding nodes in the Food Navigation Map. Spreading activation is carried out on the Navigation Map and the maximum gradient at the center of the Food Navigation map gives the motor activation that has to be taken to move towards the food disc. Similarly, the *Water Navigation Map* receives excitatory links from the corresponding nodes in the Water ESM and inhibitory links from the Food and Brick ESMs.

The *Configuration Navigation Map* is used to compute a path to locations where bricks should be dropped. Hence, every 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 Navigation Map 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. The same idea is extended to generate the activations on the *Brick Navigation Map*, which is used to compute a path to a brick that is available for construction. It is excited by the nodes on the Brick ESM and inhibited by the nodes on the Food and Water ESMs. Moreover, there are inhibitory links from the Configuration ESM that distinguish available bricks from ones that compose parts of the structure that are already built.

Figure 3.10 shows the interconnections between the ESMs and the Navigation Maps. The Configuration ESM has seven activations in the shape of a "C" (representing the structure to be built). The Brick ESM has three activations that align with three of the activations on the Configuration ESM. These represent

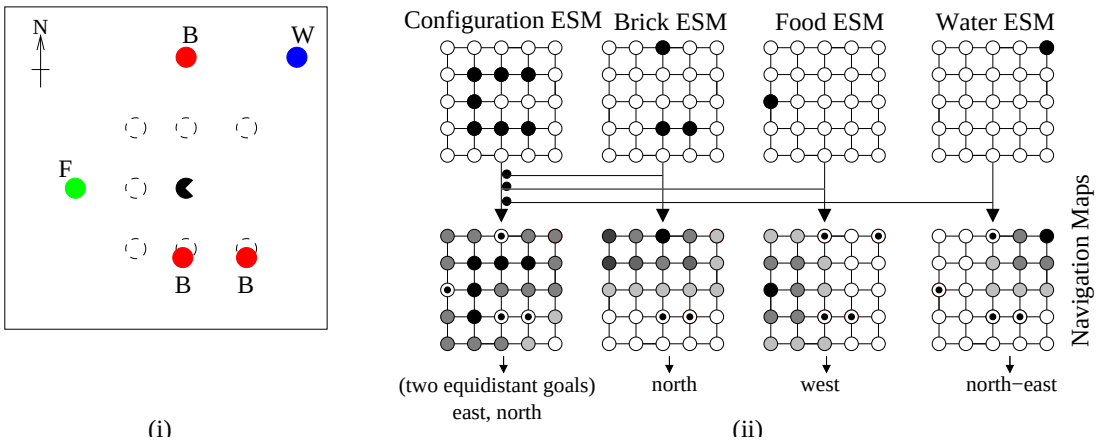


Figure 3.10: Interconnections between ESMs and Navigation maps (i) The agent is surrounded by a food (F), a water (W) and three brick (B) discs (where two of them are already in their goal positions). The dashed circles indicate the structure to be built. (ii) The four ESMs and their corresponding Navigational Maps. Darkness of shaded circles indicate the magnitude of activation while circles with filled centers indicate negative activation (obstacles). For clarity, only one connection from each ESM to its corresponding Navigation Maps is shown and only the connections from Brick, Water, and Food ESM to Configuration Navigation Map is shown. The direction to the nearest goal is listed under the maps. Note that since the two bricks south of the agent are already part of the structure, the Brick Navigation map plans a path to the brick in the north. There are five goal locations for Configuration navigation; however, the goal nodes that are horizontally and vertically above the central node are chosen as they have a greater gradient compared to the diagonal goal nodes. If the agent is hungry then it will go west; if thirsty it will go north-west.

bricks that already form part of the structure. The mismatches between the ESMs represent the location of a brick that is available for construction and the missing parts of the structure. These "misalignments" activate the corresponding nodes on the Configuration Navigation and Brick Navigation maps. Activations spread from these nodes to the center and give the direction the agent should move to reach the unbuilt part of the structure (two equidistant locations: north and east) and to reach the brick available for construction (north). The activations from the Food and Water Navigation maps give the directions to reach the closest food (west) and water disc (north-west). The agent then has to properly choose the right action given the status of its internal state.

3.4 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 on a cell where it can drop off a brick.
- *At-Brick*: Is active when the agent is next to a cell 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 navigation maps. For instance, if the central node of the Configuration Navigation map 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 Navigation map. Similarly, if the central node of the Brick Navigation map is active, then the agent is currently at a location where there is a brick that is not part of the structure being built. This brick is therefore available to be

grabbed and carried to a drop-site. Hence, the activation of the At-Brick state node is set from the central node of the Brick Navigation map. The activation of the Holding-Brick state node is directly available from the current position of the grippers (grasping or open). These three internal state nodes do not recognize the completion of the construction task. In chapter 5, more internal state nodes are introduced that capture this aspect of the construction task.

3.5 Action Selection

The Action Selection module selects one of the motor actions from the Sensory/Motor and Navigational Planning behaviors to be sent to the motors. The Action Selection module implements a fixed priority selector - all the behaviors are prioritized and the behavior with the highest priority is chosen. 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. Therefore, if a higher priority behavior is active, all other lower priority behaviors are silenced ("winner-takes-all" kind of selection). 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. Construction for an agent is a repeated sequence of moving to the location of a brick that is not part of the structure, picking it up, moving to a location that requires a brick and dropping it there. The internal state nodes correspond to these steps and the action selection module selects the behavior currently required for the construction task based on the activations of the internal state nodes. 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. In chapter 5 it is shown how weights can be assigned to these connections and the construction sequence can be learned by imitation learning. Figure 3.11 shows how the internal excitatory and inhibitory connections select one of the reactive and navigational planning behaviors for execution.

3.6 An example

In figures 3.12-3.17, screen shots of a sample construction run are shown. All the discs and the construction sites are confined within an area of 50×50 units. The size of each ESM is 50×50 nodes (it is possible for the agent to "forget" locations of discs if the agent goes to the edge of the world since the maps are egocentric). The sensor range is 20 units (same for food, water, and bricks). The thresholds controlling the hunger and thirst motivations ($T_0^h = T_0^t = 0.2$, $T_1^h = T_1^t = 1.0$), and the rate at which energy/water is consumed in each step is set such that the agent becomes hungry every 800 steps and thirsty approximately every 500 steps ($f_0 = -0.001$, $w_0 = -0.0015$). Eating and drinking each take 10 steps to complete ($f_1 = w_1 = 0.1$).

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 3.12(a)). Initially, there is no activation on the nodes of the Food, Water, and Brick ESMs. The agent's Configuration ESM has node activations set to represent the structure to build, in this case a diamond consisting of four widely spaced bricks (figure 3.12(b)). The four construction sites are shown as dashed circles in the figures. 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 3.12(a), the agent has located brick B1 (path in lighter grey) and moved it to the closest corner of the structure (path in darker grey). After properly relocating another disc and en route to place the third, the agent became thirsty and thus suspended construction (at point A) to reach the water disc (figure 3.13(a)). Resuming the high-level task after drinking, the agent goes around the wall of green discs to pick up brick B3 and begins moving to the third drop-site (figure 3.14(a)). On the way, it gets hungry

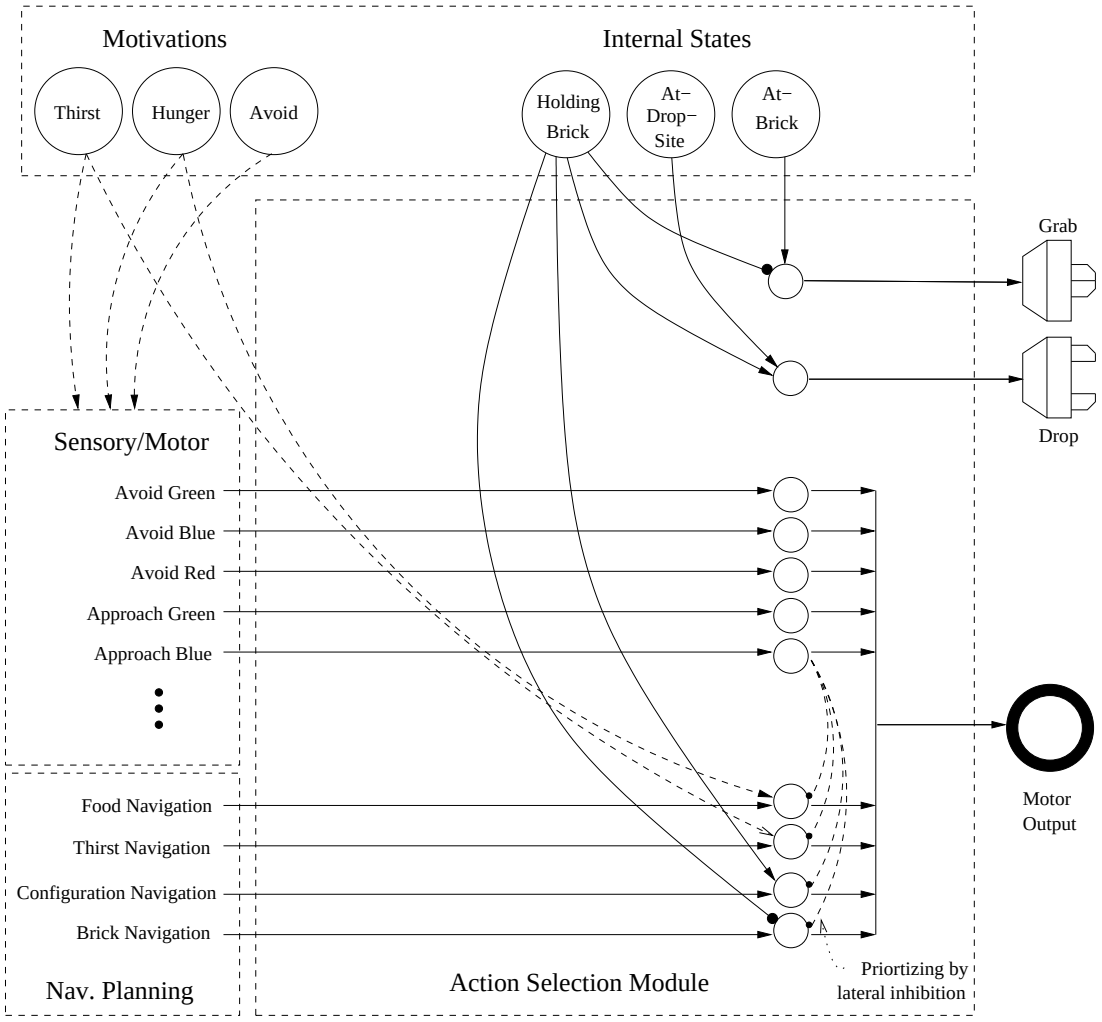
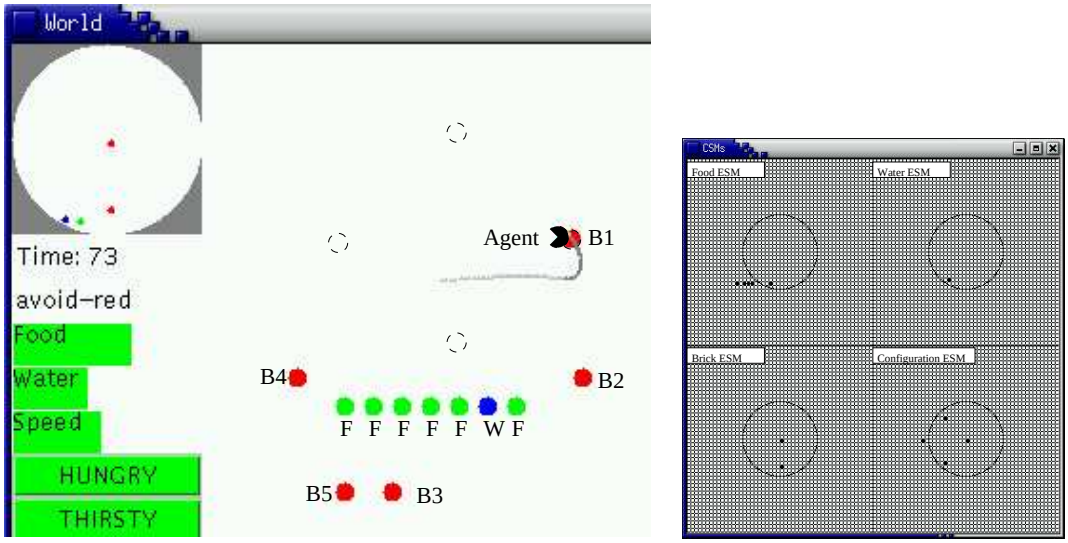


Figure 3.11: Action selection network showing links between internal state nodes and behaviors. Only one set of the lateral inhibition links for prioritizing is shown.



(a) (b)

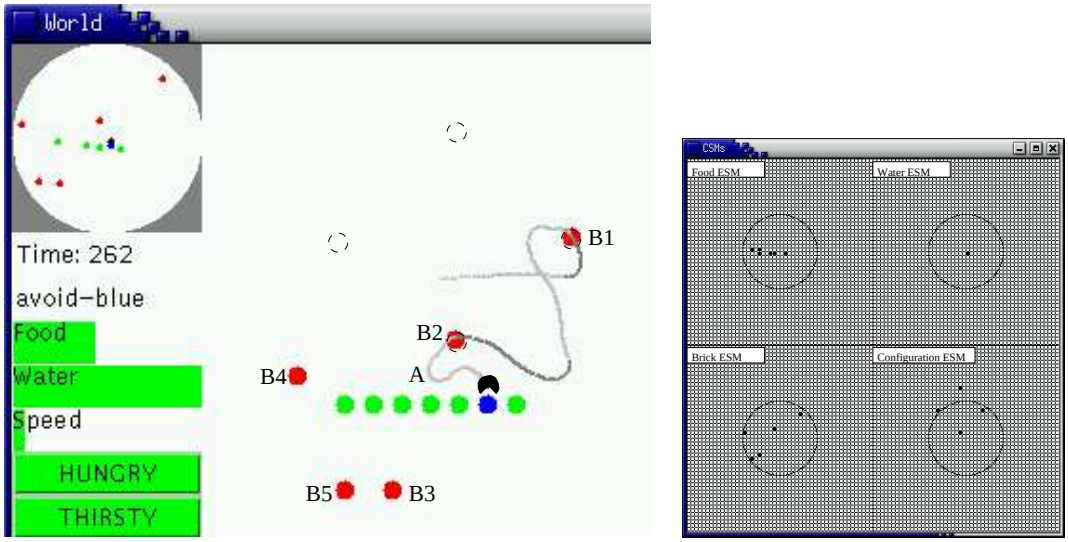
Figure 3.12: Example construction task: (a) Configuration of discs at time 73. The light grey line shows the agent's path when not carrying a brick. The darker line indicates when the agent is carrying a brick. At first the agent moves East toward brick B1 because it has the strongest gradient. Then it picks up the brick and drops it at the nearest goal location. Construction sites are marked by dashed circles. The "wall" is made of food (F) and water (W) discs. (b) Activations of ESMs at time 73. Some of the food is still within sensing range (within circle) but food out of range is remembered within the Food ESM as it is shifted while the agent moves. Brick B1 is in the center of its ESM since that is where the agent currently is located. Brick B2 is sensed to the south while bricks B3, B4, and B5 do not appear in the Brick ESM because there were out of sensory range throughout the path taken by the agent so far. The activations on the Configuration ESM show the "diamond" to be built.

and moves to the nearest green disc while continuing to carry brick B3. This is because as described in chapter 2, the grippers continue to grip a brick until a drop activation is generated. The Approach-Green behavior does not generate this drop activation (neither does the Approach-Blue behavior). At this point, only one green disc is visible (the one it is eating); all other green discs are occluded behind this one.

In figure 3.15(a), the agent's ability to repair is demonstrated. When the agent was carrying brick B3 to its drop-site, the author removed brick B1 which was out of the agent's sensor range. However, when the agent became thirsty (location C in figure 3.16(a)), 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 drops the brick it was carrying at the location that was initially occupied by brick B1.

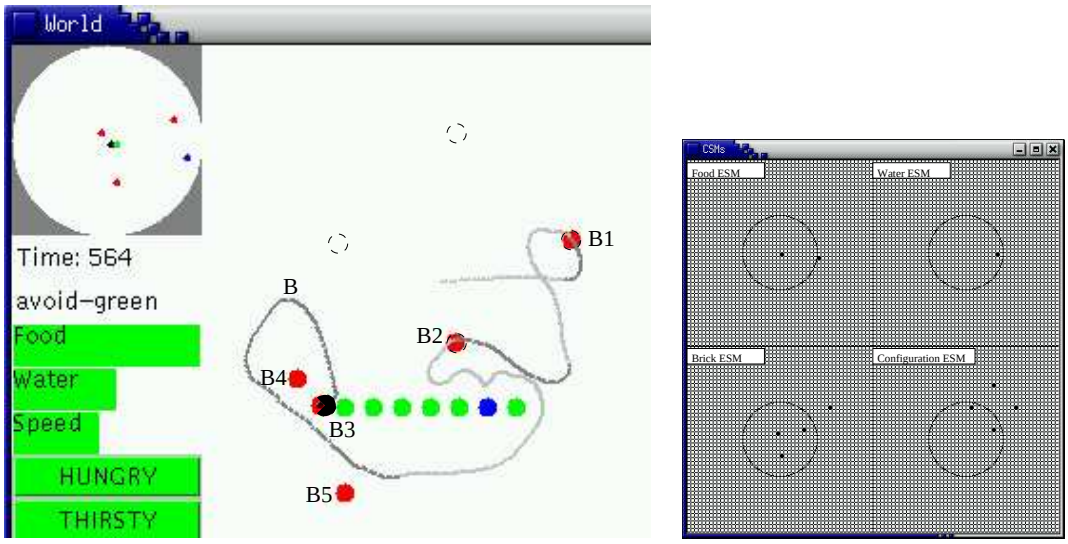
In figure 3.17(a), the agent's ability to take advantage of unexpected changes in the environment is demonstrated. When the agent moved to pick up brick B5, the author removed two green discs, thus creating a gap in the wall. When this gap came into sensor range, the ESMs were updated and the path to the northern most drop site was recomputed to navigate the agent through the gap. On its way to the last drop-site, the agent became both thirsty and hungry. It therefore moved to blue disc and then to the green disc next to it (while continuing to carry brick B5). The corresponding ESMs show that the agent's belief about disc locations are not completely accurate. This is because the sensors are unreliable at long range.

The four navigation maps that were used to plan a path for the time-steps shown in figures 3.12-3.17 are displayed in figure 3.18. The spreading activation flows around the obstacles which are marked as yellow squares (negative activation). When inhibiting an obstacle node, all its neighboring nodes are inhibited to account for the size of the agent (whose width is equal to the side of a grid cell). This is required to prevent



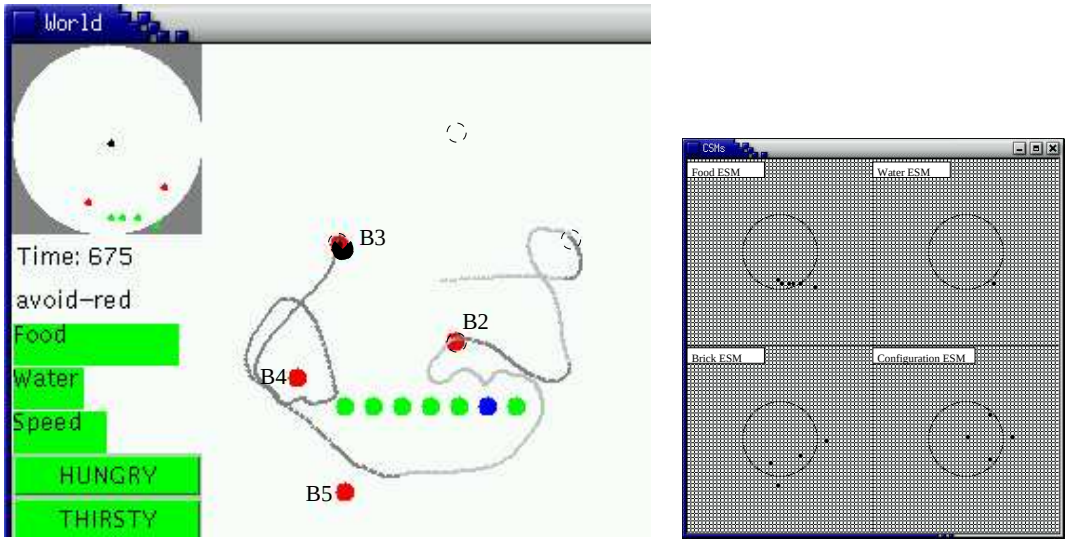
(a) (b)

Figure 3.13: Example construction task (cont.): (a) Configuration of discs at time 262. After picking up and dropping brick B2 at a construction site and on its way to brick B4, the agent became thirsty (at point A) and moves to the water (blue) disc to drink. (b) Activations of ESMs at time 262. The central node of the Water ESM is active because the agent is close to the blue disc and drinking. The agent now has knowledge of the positions of all the bricks in the Brick ESM as all bricks and food are currently within sensory range. Two goal sites are out of sensory range but are remembered on the Configuration ESM.



(a) (b)

Figure 3.14: Example construction task (cont.): (a) Configuration of discs at time 564. The agent goes around the wall of green discs (since it is an obstacle) to pick up brick B3 and begins moving to the third drop-site. At point B it gets hungry and moves to the closest food disc while continuing to carry the brick. (b) Activations of ESMs at time 564. Only one of the green discs can be sensed from the agent's current location - the others are occluded behind the nearest green disc.



(a) (b)

Figure 3.15: Example construction task (cont.): (a) Configuration of discs at time 675. The agent drops brick B3 at a drop-site. Meanwhile, the author has removed brick B1. (b) Activations of ESMs at time 675. Since brick B1 that was removed is out of sensor range, the Brick ESM continues to have an activated node for B1. The Food ESM contains activations for all the food discs again.

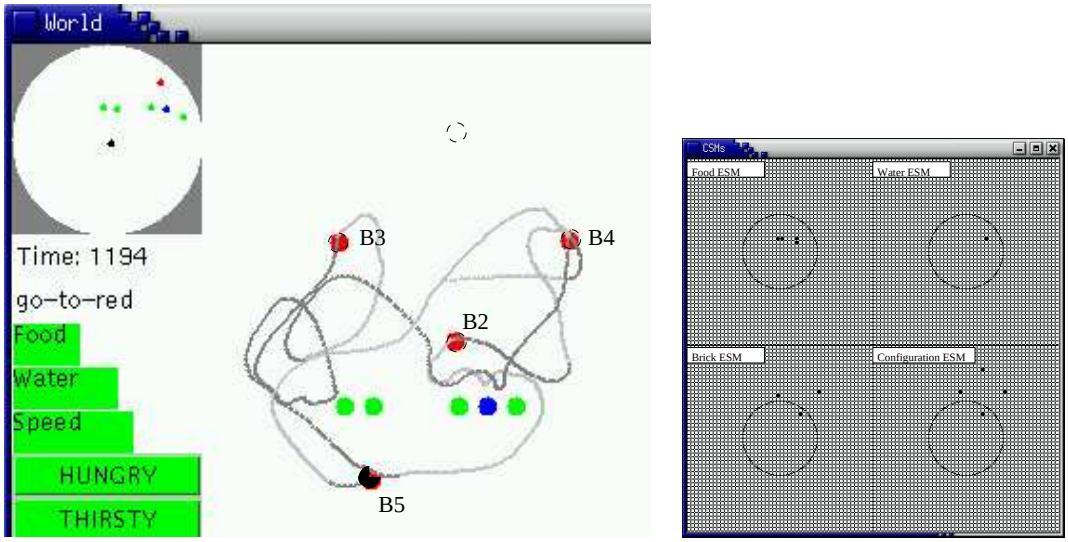
an agent from colliding with an obstacle disc because the accuracy of the ESMs is limited to its grid size. Collision avoidance by "growing" obstacles is often used as a simple (but sub-optimal) method to model the size of the robot (for example, in [Zelinsky and Yuta, 1993], the obstacles are "grown" by half the width of the cylindrical robot for efficient obstacle avoidance). The spreading activation was stopped as soon as activation reached the center node (the current location of the agent).

3.7 Discussion

3.7.1 Action Selection

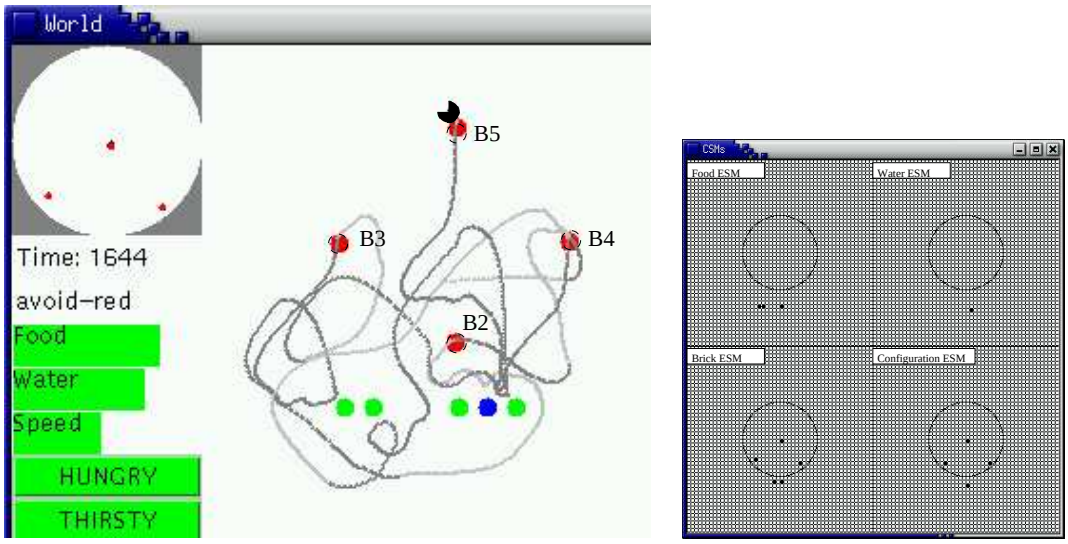
A behavior-based architecture with connectionist action selection has several features that make it suitable for an environment where agents have to survive autonomously and adapt to changing situations. The behavior-based approach ensures that though the agent has a large number of behaviors, the number of levels of processing between sensors and motors remains the same. Thus, this approach scales well and the time taken for the agent to react to the sensors remains small.

Implementing the action selection mechanism as a connectionist system offers the advantages associated with artificial neural networks such as smooth integration, ease of incremental learning, and biological plausibility [Rumelhart *et al.*, 1986]. The Action Selection module chooses the actions specified by the reactive behaviors if they are active over that of the Navigational planning behaviors. This is a simple yet effective way of integrating low-level behaviors with higher ones - the reactive behaviors operate directly on sensor information, and hence the actions specified by them are more likely to succeed. However, high-level tasks can also override these reactive behaviors such as when the agent is close to an available brick (Near-Brick internal state node is active), Approach-Red behavior is chosen while Avoid-Red is inhibited. Moreover, by selecting the behavior with the highest activation, actions that are most likely to succeed are chosen over others (the activation of a behavior is proportional to how strongly the sensors are activated, which in turn



(a) (b)

Figure 3.16: Example construction task (cont.): (a) Configuration of discs at time 1194. The agent picks up brick B4 and gets thirsty at location C. It then moves to the water disc. From that point, the initial location of brick B1 was within sensory range and the agent realizes that brick B1 is missing. Therefore, after drinking it drops brick B4 at that location and moves to pick up brick B5 after navigating around the wall. Meanwhile, the author removed two green discs, creating a short-cut. (b) Activations of ESMs at time 1194. The Brick ESM again shows activations of three bricks that are in their final positions.



(a) (b)

Figure 3.17: Example construction task (cont.): (a) Configuration of discs at time 1644. The construction task has been completed. The agent made use of the short-cut created by the author to move to the northern most drop-site and dropped brick B5. On the way, the agent became both hungry and thirsty. It moved to the blue disc and then to the green disc next to it (while carrying brick B5). (b) Activations of ESMs at time 1644. The activations of the Brick and Configuration ESMs match at the end of construction.

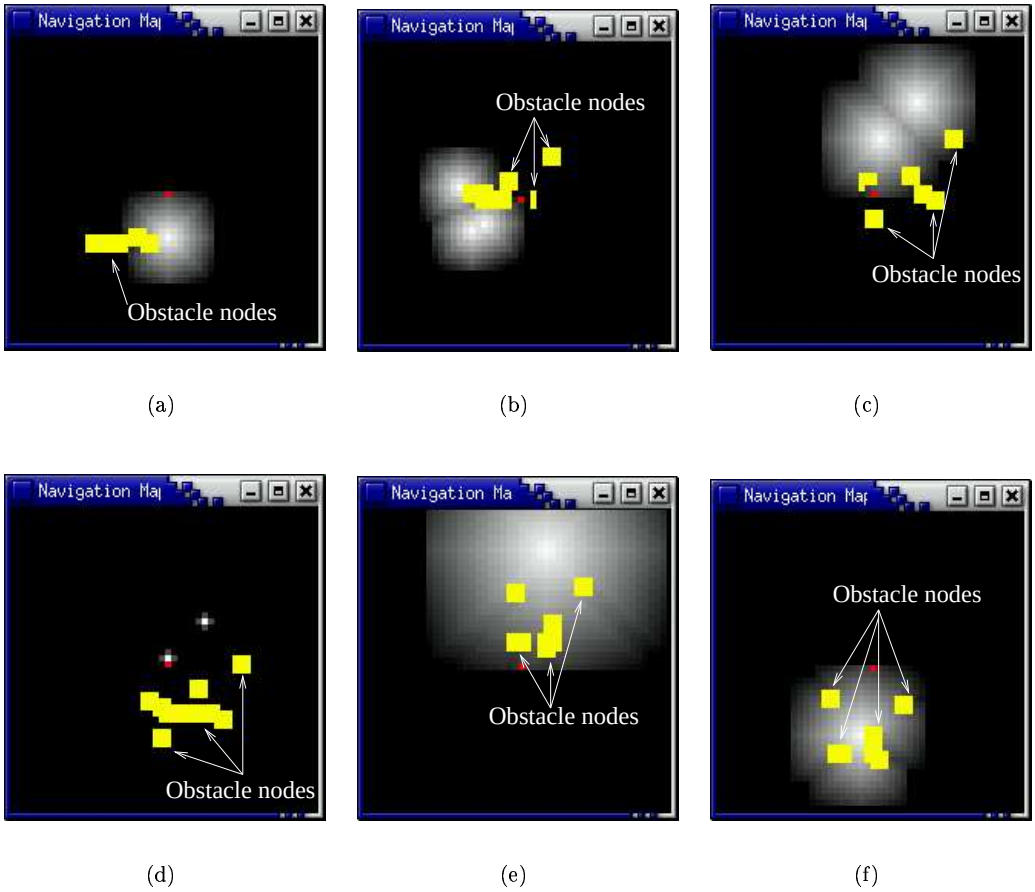


Figure 3.18: Activations on navigation maps that provided the final motor activation: (a) Brick Navigation map at time 73. (b) Configuration Navigation map at time 262. (c) Configuration Navigation map at time 564. (d) Configuration Navigation map at time 675. (e) Configuration Navigation map at time 1194. (f) Brick Navigation map at time 1644. Square blocks indicate obstacle nodes and the brightness of a cell is proportional to the magnitude of its activation.

depends on how close the discs are to the agent), i.e, *consummatory* behaviors are chosen over *appetitive* behaviors. Consummatory behaviors are those that directly affect a motivation [McFarland, 1981]. For example, eating would immediately reduce the hunger motivation. Appetitive behaviors are those that do not directly affect any motivation. For example, performing the Approach-Green behavior does not cause a decrease in an agent's hunger motivation. The preference for consummatory behaviors over appetitive ones is one of the requirements for an action-selection mechanism [Tyrrell, 1993a].

The action selection mechanism in ConAg is "winner-take-all" (WTA) because only one behavior, triggered by one motivation, is active at any given time. Werner proposed a neural architecture for an agent that has to look for food, water and avoid predators [1994]. The architecture is based on the work by Braitenberg where motors are controlled by direct connections from sensors [1984]. A part of the architecture is shown in figure 3.19 (no spatial representation is maintained). Nodes representing hunger and thirst motivations control direct links from sensor nodes to motor nodes through second-order connections. Since, all the motor nodes can be active simultaneously, the overall action of the agent is determined by the state of all its motivations.

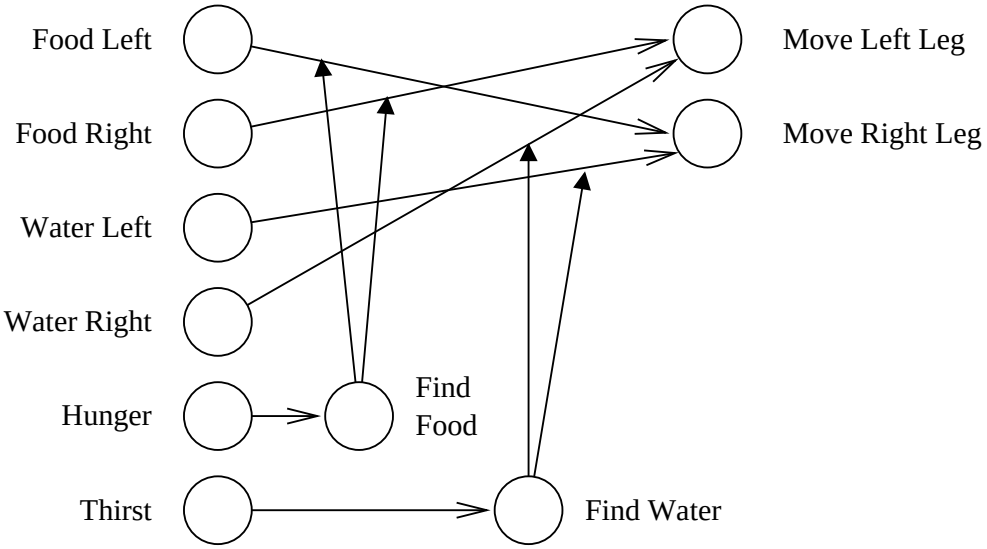


Figure 3.19: Action selection that combines behavior outputs [Werner, 1994]. Motivations (hunger and thirst) control links from sensor nodes (Food Left etc.) to motor nodes (Motor Left Leg, Motor Right Leg) through second-order connections. If the hunger node is active and food is sensed to the left (Food Left node is excited), the Move Right Leg node is activated due to the second-order connection from Find Food node causing the agent to turn toward the food source.

Werner's architecture is compared to the WTA action selection of the ConAg architecture along the following metrics:

1. *Persistence*: An action selection mechanism is said to be *persistent* if it continues to perform a consummatory action even after it is not the most important behavior to an agent [Tyrrell, 1993a]. Consider the situation depicted in figure 3.20. An agent that is both hungry and thirsty senses food and water. It moves toward the food as it is closer and begins eating. This reduces its hunger level to a point that moving toward water becomes more desirable than continuing to eat. However, on the way to water, its hunger level could rise to a point where eating again becomes more desirable. Thus, the agent will repeatedly attempt to move toward water only to move back to food without drinking. The ConAg architecture is persistent because consummatory behaviors are always preferred over other behaviors due to the fixed hierarchical ordering of behaviors (i.e, the agent would continue to eat until the internal food level drops below threshold). Werner's architecture is also persistent because "the sensory input to the agent is proportional to the ... distance of the object being sensed. This causes nearby objects

to be more important in behavioral selection than distant objects, unless the internal motivation to approach the distant objects is much stronger than for close objects"[Werner, 1994]. This "internal motivation" is controlled by the "Find Food" node (figure 3.19) which converts the continuous hunger level into a binary signal. Thus, though the input and output nodes can take continuous values, the architecture only considers a discrete number of choices (hungry or not hungry).

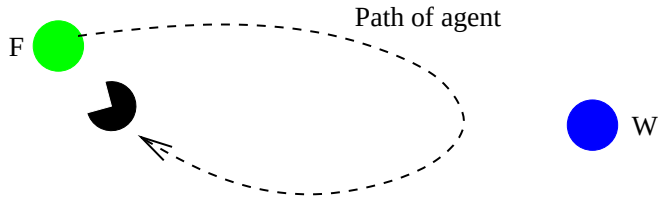


Figure 3.20: Situation when persistence of action selection is necessary. An agent that is both hungry and thirsty senses both food (F) and water (W).

2. *Length of Paths*: WTA action selection takes the agent on a direct path to the selected goal while if actions are composed then the path will be drawn toward secondary goals giving rise to sub-optimal paths. This is illustrated in figure 3.21. Combining actions works well in cases such as escaping from a predator while moving toward food. However, in cases where there are two nearby goals, the resulting path could be significantly longer than the direct path (the case where two goals are exactly equidistant from the agent causing the agent to miss the goals completely by moving between them is only of theoretical interest since in nature randomness will prevent such a situation). This is illustrated in figure 3.22.

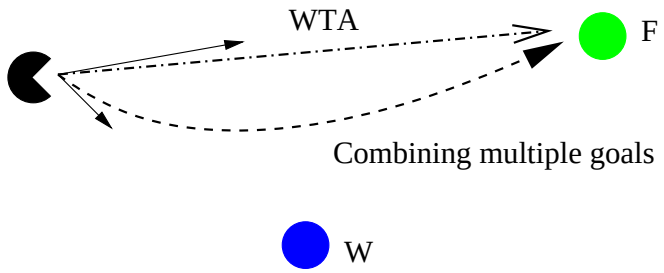


Figure 3.21: Length of path for WTA selection and when actions are composed: WTA selection takes the agent on a direct path even if there are multiple goals (food (F) and water (W)).

3. *Parallel actions*: As long as actions do not interfere with each other, they can be executed in parallel in both the ConAg and Werner's architecture. For example, a ConAg agent can Approach-Food and Drop at the same time while in Werner's architecture can enable an agent to "produce signals while moving"[Werner, 1994]. However, coordinating dependent actions is a difficult problem and is a weakness of both these architectures.

4. *Smooth actions*: When the motor output of an agent is a result of a sum of behaviors that each produce continuous valued outputs, the agent can exhibit "natural" looking behavior. However, as described for persistence, the internal motivations take on binary values in Werner's architecture. Moreover, even in WTA action selection, smooth movement of the agent is observed due to the inertia of the agent. Thus, summing motor actions from different sources is not essential for smooth movement.

5. *Opportunism*: Consider an environment in which are present predators that constantly threaten the agent, forcing the agent to constantly move away from the predators for survival. Using a WTA architecture will cause an agent to starve since the agent would always be pre-occupied with running

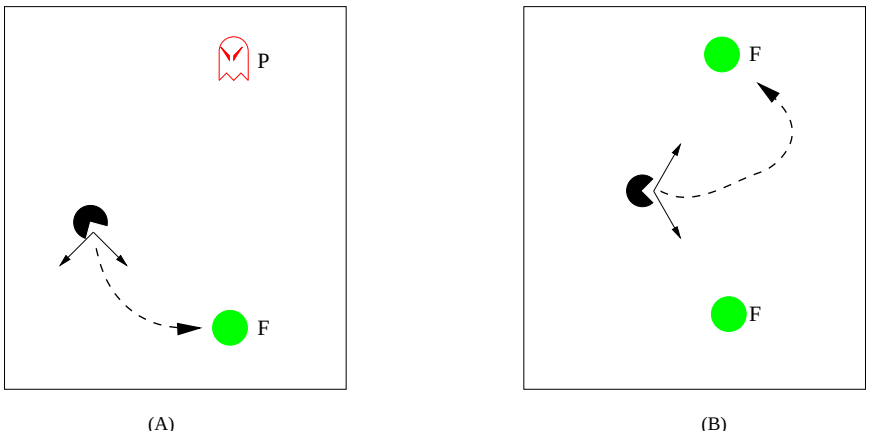


Figure 3.22: Path of agent when there are two goals. (A) Food (F) and predator (P): both the goals take the agent in the same direction. (B) Two food sources: the goals pull the agent in different actions resulting in an inefficient path.

away from the predators. However, combining the escape behavior with the approach food behavior will enable the agent to move toward food if it on an escape route. In these situations, Werner's architecture out-performs the ConAg architecture.

3.7.2 Spatial Representation

The spatial maps enable the agent to construct a representation of the environment and use it for path planning. All the computations involved in path planning by spreading activation are local and hence they can be carried out in parallel. Localization (locating oneself on the map) is easy on a ESM since the center of the map always corresponds to the agent's current location. This is particularly important for construction tasks, since the agent has to match the given target pattern (the Configuration ESM) to the actual locations of the building blocks (the Brick ESM) to identify what discs are not part of the structure and what parts of the structure are yet to be built. Since the maps are always aligned due to their egocentricity, the aforementioned matching can be done easily.

The spatial map uses only a prefixed number of nodes and interconnections. In this respect, it shares a common feature with biological systems. This method also provides a reasonable solution to the issue of how should a fixed number of neurons be allocated. All the neurons are allocated to the space that immediately surrounds the agent and this is the space that is most crucial to the survival of the agent. In fact, the density of neurons (the number of neurons allocated per unit area of space) in an egocentric map could be designed to gradually decrease with distance from the agent. The drawback to this cell-based representation is that the number of nodes is proportional to the area mapped irrespective of the actual density of discs in the environment. This is an inefficient use of neurons since sparse or less important areas use equal amount of nodes as denser areas.

3.7.3 Exploration

An agent or robot that has to build a spatial map needs an exploration strategy. Typically, the robot should move to those areas of the environment that have not yet been mapped (or mapped with low confidence). Thus, Yamauchi *et al.* program their robot to move to a location that are on the "frontier" between mapped and unexplored areas [1999]. In case of a grid-based representation, each cell in the grid could store the confidence in the occupancy of that area. The robot could then propagate these confidence measures from each cell to the current location cell (spreading activation) and then move in the direction of the nearest unexplored cell [Thrun and Bücken, 1996].

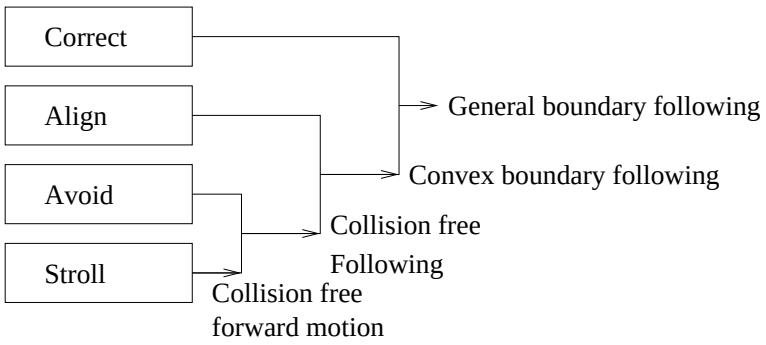


Figure 3.23: Example of Subsumption architecture. The output of a higher level behavior (for example, *Align*) can over-ride that of a lower level behavior (for example, *Stroll*). The increasing capabilities of the robot as more behaviors are added are shown to the right [Mataric, 1992].

In the ConAg architecture, the Explore reactive behavior is implemented as a random wander behavior (i.e, it does not use its ESMs). This simple behavior is sufficient since the agent starts with an initialized Configuration ESM to provide it navigation goals. Moreover, the agent has to eat and drink periodically and while the agent moves toward food and water discs, the ESMs get updated, thus mitigating the need for a sophisticated exploration strategy.

3.8 Related Work

3.8.1 Behavior-based Architectures

Behavior-based architectures for robots were first introduced to solve problems that arose due to elaborate planning on internal world models. The first behavior-based architecture was the *Subsumption* architecture of Brooks[1986]. Behaviors are arranged in layers and are implemented as augmented finite state machines. A behavior can modify the data on the inputs and outputs of its lower-level behaviors (i.e, higher-level behaviors *subsumed* the function of the lower-level behaviors). An example from [Mataric, 1992] of an architecture built using this philosophy is shown in figure 3.23. There are four behaviors: *Stroll*, *Avoid*, *Align*, and *Correct* with Stroll being the most basic behavior and Correct the most sophisticated. The Stroll behavior causes the robot to move forward and to stop and back up if an obstacle is detected in front. The Avoid behavior enables the robot to turn away from obstacles on either side. The Stroll and Avoid behaviors together enable the robot to wander without colliding into obstacles. The Align behavior is used to turn the agent such that it is parallel to any walls at its side. These three behaviors together enable the agent to follow convex boundaries. The Correct behavior prevents the robot from moving away from a wall when it is near a sharp turn in the wall. All these four behaviors together enable the robot to follow arbitrarily shaped boundaries (the increasing capabilities of the robot are shown to the right in figure 3.23).

The original formulation of the subsumption architecture did not create any internal representation [Brooks, 1991] and thus limited its applications. Mataric first integrated spatial information into the subsumption architecture [1992]. The robot represents each *landmark* as a behavior (landmarks are features such as walls, corridors in an indoor environment). Interconnections between landmarks encode neighborhood relationships (i.e, the landmarks form the nodes of a graph) and enable the agent to navigate from one landmark to another. Path planning is done by spreading activation from the goal landmark (as in our work). The collection of such landmark behaviors form a coarse-grained topological map and these behaviors are added on top of those shown in figure 3.23. The resulting architecture is shown in figure 3.24.

The subsumption architecture is not "fine-grained" in the sense that individual behaviors implement a non-trivial capability. Since the internal state of a behavior cannot be accessed, this limits the ability of the architecture to arrive at compromise solutions [Rosenblatt and Payton, 1989]. In general, a system will be unable to exploit unforeseen opportunities if relevant information is abstracted away [Payton, 1990].

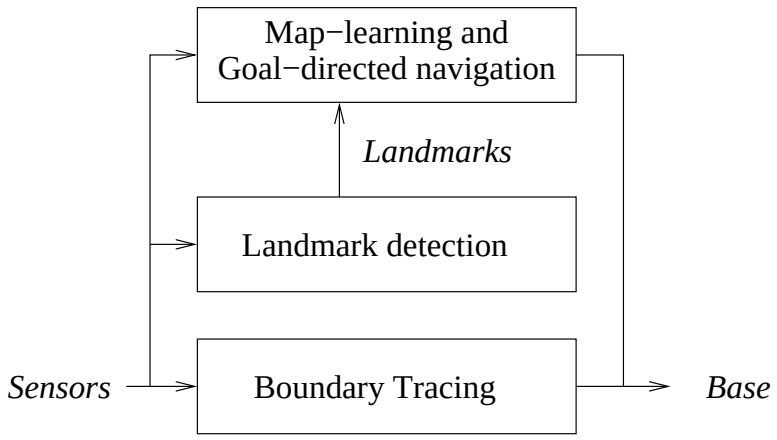


Figure 3.24: Integrating representation into a behavior-based architecture [Mataric, 1992]. The boundary tracing behaviors (described in figure 3.23) are reactive. The map-learning module contains one behavior for each detected feature such as walls or corridors. The landmark detection module activates that node (behavior) in the map which most closely matches the currently detected landmark. (figure from [Mataric, 1991]).

Rosenblatt and Payton propose a fine-grained variant of the subsumption architecture in which each behavior is composed of simple functional units (both inputs and outputs are in the range $[-1, 1]$) [1989]. The output of a unit is connected to the input of another, but unlike in a connectionist system [Rumelhart *et al.*, 1986], each unit represents a behavior-specific concept and hence links need not exist between every pair of units. Also, behaviors added later do not completely suppress behaviors already present unlike in the Subsumption architecture.

Maes presents the *Agent Network Architecture* – a distributed, non-hierarchical action-selection mechanism [1991; 1990]. It selects one behavior from a pool of candidate behaviors. The applicability of a behavior at any time is indicated by its *activation level*. When the activation level of a behavior exceeds a threshold, that behavior is executed. Every behavior has a set of pre-conditions that should hold true before it can be executed. Behaviors are connected to each other through *predecessor*, *successor*, and *conflicter* (directed) links. There is a predecessor link from behavior A to behavior B if executing B will make some of the conditions required for executing behavior A come true. Successor links work conversely and two behaviors are connected by a conflicter link if executing one will cause some of the pre-conditions for executing the other to become false. These links are used to excite and inhibit the activations of behaviors. The agent also has motivations as in the ConAg architecture and activation flows from the motivations to those behaviors that can directly satisfy these motivations. When the activation settles, the most appropriate behavior will have the highest activation.

The ability of the behaviors to directly affect each other make Maes's Agent Network Architecture a more general-purpose action selection mechanism than the ConAg architecture. However, this inter-behavior interactions reduce the reaction time of the agent since the activations will have to settle before a behavior can be selected. Tyrrell showed that Maes' action selection mechanism can get into a deadlock in situations similar to that discussed in the previous section (persistence of action selection) [1994]. Decugis and Ferber proposed extensions to overcome these problems [1998]. In particular, goals are arranged hierarchically and each such goal is associated with an underlying network of behaviors to provide persistence. This hierarchical ordering of conflicting goals eliminates the "thrashing" seen when the action selection is not persistent. Tyrrell has compared these action selection mechanisms in a simulated environment and has found that hierarchical action selection out-performs flat mechanisms [1993b; 1993a].

The PerAc (Perception-Action) architecture [Gaussier and Zrehen, 1995] is a neural architecture that can learn sensory-motor associations. The architecture consists of two pathways — perception and action (figure 3.25). Sensory input triggers a reflex action that activates the motors. The reflex system is innate

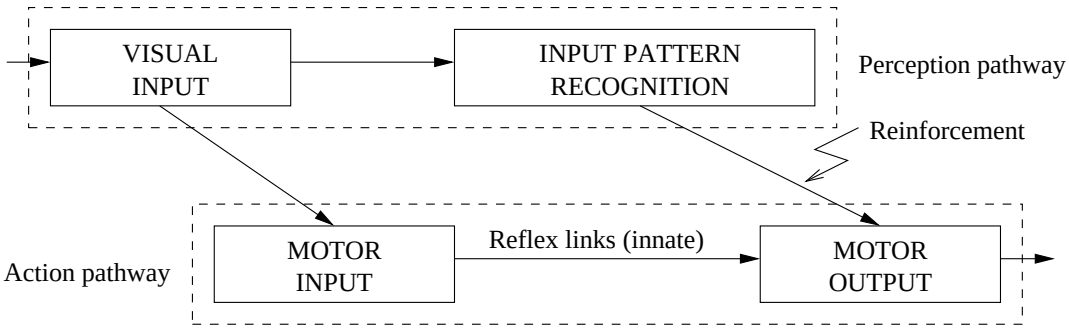


Figure 3.25: Block diagram of the PerAc architecture [Gaussier and Zrehen, 1995]. The links between the "input pattern recognition" block and the "motor output" block are modified when reinforcement is received. Figure based on [Revel *et al.*, 1998].

as in the ConAg architecture. At the same time, the perception pathway recognizes sensory input patterns. When reinforcement is received, links between these two pathways are strengthened. The PerAc architecture has been used for both landmark-based navigation [Gaussier *et al.*, 1997] and for navigation by building a topological map [Revel *et al.*, 1998].

3.8.2 Spatial Navigation

Traditional robotics has tried to represent the robot and all other obstacles in the environment using polygons and then define translations and rotations of these polygons that will move the robot to its target without hitting any of the obstacles. This is a computationally expensive process and the need for precise information make it unsuitable for situated robots. Latombe gives an overview of this approach [1991].

Configuration Space

Lozano-Perez introduced the concept of *Configuration spaces* as a means of providing a common formal model on which to plan paths for different robots and environments[1983]. The *Configuration vector* is a vector of parameters that uniquely specify the location and orientation of the robot. For example, in the case of a robotic arm (fixed to a stationary base), the components of the vector might be the angles of the joints on the arm. The configuration space is the set of all possible configuration vectors. The current location of the robot is a point in this space, as is the final goal location. Obstacles get mapped on to those areas of the space that represent the corresponding physically unrealizable configuration vectors. Path planning is now reduced to finding a path for the point representing the current location of the robot on the configuration space and that avoids the obstacle regions. To calculate a path on the configuration space efficiently, the free configuration space is first discretized into a grid and a path linking neighboring grid cells is returned [Latombe, 1991]. Other methods for fast calculation of paths use potential fields [Khatib, 1986] and randomization [Barraquand and Latombe, 1991].

The concept of configuration spaces has been extended to include movable objects. This enables the systematic study of *manipulation planning*, that is deciding on a plan of action that will re-arrange the movable objects into some desired pattern. Motion planning in the presence of movable objects is PSPACE-hard [Wilfong, 1988]. Planning algorithms have been developed to solve restricted versions of the manipulation problem [Alami *et al.*, 1995; Ben-Shahar and Rivlin, 1998].

These approaches assume complete knowledge of the surroundings at all times and the algorithms that are developed are computationally intensive. This makes these methods unsuitable for autonomous agents with limited sensing capabilities in dynamic environments. Hence this direction of research is different from that followed in this dissertation.

Cell-based Representations

Cell-based representations divide space into many small regular regions that are adjacent to each other. The cells are small enough that all relevant aspects of the environment can be represented, but large enough that the number of cells does not become prohibitively large. Moravec and Elfes presented the first successful application of a real robot that used a cell based representation (called *evidence grids*) to store the structure of the world [1985]. Each cell on the evidence grid stores the probability that that cell is occupied. The robot is equipped with sonar sensors and a probabilistic model of the sensor. Every measurement of the sonar sensors is incorporated into the evidence grid using Bayes rule (described more fully in chapter 9).

Chao and Dyer [1999] and Lagoudakis [1998] describe egocentric maps similar to the ESMs used in the ConAg architecture. *Concentric Spatial Maps* (CSMs) are egocentric maps in which the neurons representing the space around the agent are arranged in concentric circles (compared to the rectangular grids in an ESM) [Chao and Dyer, 1999]. Moreover, the agent does not have a 360° field of vision. Therefore, the agent has to explicitly rotate the CSMs to keep track of the heading of the agent. Lagoudakis also suggests other arrangements of grids (such as hexagons) [1998].

Path planning in cell-based representations is most efficiently performed using spreading activation to calculate the shortest path as in ConAg. A related method is diffusion where the activation at each cell is the average of the activation at its neighboring cells (instead of the maximum) [Stopp and Riethmüller, 1995]. These methods can also be extended to 3-dimensional grids.

Classification of Navigation Systems

Trullier *et al.* classify artificial navigation systems into a four level hierarchy [1997]. Though this classification was proposed for biologically-based navigation systems, it is used here with examples of systems that do not necessarily have a biological basis. The classification is based on the amount of information that is used to navigate to the goal and thus each class shows progressively more complex behavior.

At the lowest level is *guidance* or *local navigation* where all the information required to move to the goal is always visible. An example is the robot developed by Chesters and Hayes where a gradient of light from a single source is used as an always visible landmark and the path to the goal is stored in a recurrent network [1994].

The second level is *place recognition-triggered response* where the environment is divided into *places*. Each place is assigned an action that would take the agent to the goal from any point belonging to that place. Since this action is goal dependent, such systems can store only one goal. An example of such a system is described by Gaussier *et al.* [1997]. A robot in an indoor environment learns to navigate to a goal by storing the visual image corresponding to a few places close to the goal and associating with each such image the direction to the goal. The robot then can navigate from any place to the goal by comparing the current image with the stored images and moving toward the place corresponding to the best matched image. This is implemented on the PerAc architecture [Gaussier and Zrehen, 1995]. Navigation in maze environments often falls into this category since there is usually only one goal in the maze [Pipe *et al.*, 1994; Duchon, 1996].

To handle more than one goal, systems at the third and fourth levels do not associate each place with a goal specific action, but the places are represented in such a way that they reflect their spatial relationship to each other. Goal specific actions are then *planned* on this representation. The third and fourth levels differ depending on whether only topological or metric information is stored. Topological representations are often graph-based where nodes represent places and edges represent adjacencies between places. Since no position information is maintained, the nodes contain information that disambiguate places and the edges store action that will take the agent from one place to the other [Kuipers and Byun, 1991; Mataric, 1992]. Systems that store only topological information are incapable of discovering shortcuts in their representation.

The most straightforward metric representation is one which specifies places relative to some coordinate frame. Most navigation systems with spatial representation that are implemented on real robots use such a metric representation (for instance, evidence grids [Moravec and Elfes, 1985; Elfes, 1987], CSMs [Chao and Dyer, 1999]). Gallistel's theory of cognitive maps in animals assume a global Euclidean representation of space around the animal [1990]. Artificial navigation systems with a biological basis represent this Euclidean information in different ways. For instance, Worden divides space into triangles defined by unique landmarks

at its corners [1992]. The direction to a goal triangle can be calculated by summing the vectors that define the triangles that lie between the goal triangle and the current location of the agent (figure 3.26).

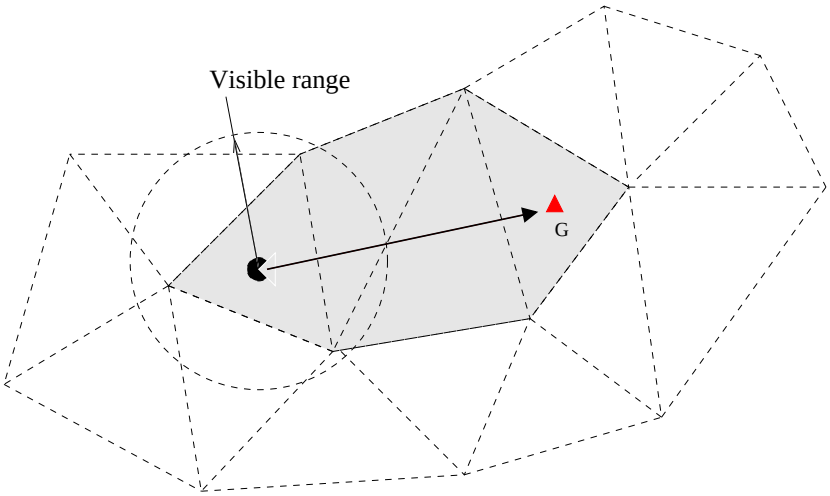


Figure 3.26: Navigating by fitting triangles [Worden, 1992]. The internal representation divides space into triangles defined by three unique landmarks. To reach a goal (G) that is not visible from the current location, the agent "fits" triangles until a continuous area is discovered from the visible to goal landmarks. The planned path is obtained by vector additions of the position vectors of the individual landmarks.

The ESMs used in the ConAg architecture store a metric map of the discs around the agent. Moreover, the representation of goals is separate from the spatial representation. Thus, the navigation in ConAg is at the highest level in the hierarchy of Trullier *et al.*. This enables the agents to exploit short-cuts that were not traversed when the map was created.

Biologically-based Navigation Systems

Biological systems have been used as the basis for many artificial navigation systems. Franz and Mallot give a survey of navigational systems based on biological models [2000]. The place cells in the hippocampus of the rat is sensitive to the location of the rat and has led to the hypothesis that these cells encode a spatial map [O'Keefe and Nadel, 1978]. Various models of the hippocampus have been proposed and demonstrated on artificial navigation systems [Mataric, 1991; Burgess *et al.*, 1994; Gerstner and Abbott, 1997; Trullier and Meyer, 1998]. For instance, Trullier and Meyer model the hippocampus as a hetero-associative network that learns sequences of visited places [1998]. These places are then represented in a topological representation. The representation of goal locations is independent of the spatial representation and the agent can navigate to multiple goals in a continuous environment containing obstacles.

Bees appear to be using *optic flow* as a measure of distance traveled [Srinivasan, 1992]. Optic flow is the amount of shift in consecutive images caused due to the movement of the observer. The closer the objects to the moving observer, the greater is the optic flow. Thus, the magnitude of optic flow can be used as an indicator of distance to obstacles and has been used for obstacle avoiding navigation in artificial systems [Sobey, 1994; Coombs *et al.*, 1995]. Duchon integrates optical flow with a spatial representation to solve maze navigation [1996]. The spatial representation stores the sequence of actions (the parameters in the optical flow equation at various points in the maze) that leads to the goal.

Foraging honey bees and ants are also capable of navigating large distances, but it is not yet clear if insects maintain an explicit cognitive map [Gould, 1986; Gallistel, 1990; Wehner and Menzel, 1990]. Roboticists have however used spatial maps when building navigation systems based on insect behaviors because of its advantages such as efficient path planning [Walker *et al.*, 1993]. In case no explicit spatial map is maintained, other strategies are used. For instance, the desert ant *Cataglyphis* uses the polarization of sunlight to keep track of direction [Wehner, 1994] (the desert ant does not lay pheromone trails). When landmarks are visible,

the direction to the nest is calculated by comparing the current landmark view with a snapshot taken when the ant was close to its nest [Wehner *et al.*, 1996] (bees perform a similar comparison when landmarks are visible [Cartwright and Collett, 1983]). Moller *et al.* simulate these strategies on a mobile robot [1998]. Navigation systems based on comparing landmark snapshot fall into the *Guidance* level of the classification of Trullier *et al.* [1997].

Various other data structures have been used to store spatial information. These include encoding the space explored by the robot as a list of production rules [Donnart and Meyer, 1996] and using fuzzy logic operators to integrate sensor information to the built map [Oriolo *et al.*, 1995]. Kuipers and Byun take a "qualitative" approach by building a topological map of only "distinctive places" (using features such as symmetry and sudden change in sensor readings) [1987]. *Control strategies* such as Follow-midline that take the agent from one place to another link these distinctive places. On the other hand, Connell take the path of minimal representation by storing only those locations where the robot has a choice of actions (such as doors) [1988].

Chapter 4

Learning Spatial and Temporal Correlations

4.1 Introduction

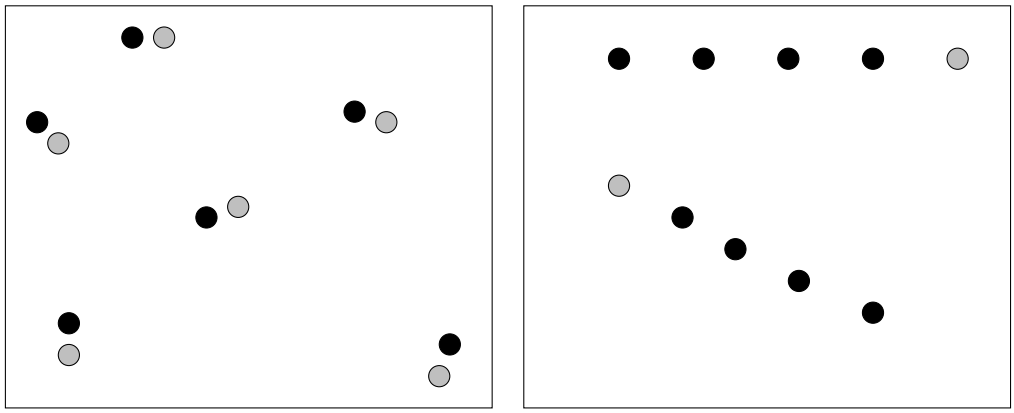
If the agent is to take advantage of features it has seen during exploration, but are not visible through its sensors from its current position, it has to build some internal representation of the world around it. [Mataric, 1992] claims that "any solution superior to random walk necessitates an internal model of the robot's current location, the desired goal location, and the relationship between the two". However, an "internal model" does not necessarily have to be a topological model of the environment. If there is some regularity in the way discs are arranged in the world, an agent can learn to move to discs outside its sensor range by exploiting the relationship between the locations of the visible discs and the goal disc(s). For instance, if food (green) discs are always situated close to red discs, then the agent should learn that moving to a red disc (Approach-Red behavior) will probably lead to food though the agent might never even have visited that part of the environment before and green discs are not visible from its current location. Once the desired food discs are in sensor range, the agent's Approach-Green behavior becomes active. Thus the agent learns to apply its innate behaviors in situations that were not foreseen when these behaviors were created. This is particularly useful in environments where it is easier to recognize locations of landmarks (the red discs) than locations of food.

The correlations between disc locations can be in many forms. For instance, red and green discs may be always located close together (figure 4.1(a)). This is an example of "spatial correlation". A more complex pattern is a "trail" of red discs leading to a food disc (shown in figure 4.1(b)). In this case, the red and green discs are not seen together, but if an agent "follows" this trail of red discs, then the Approach-Green behavior will be activated at the end of the follow behavior (assuming that it is possible to identify the correct direction). This is called "temporal correlation". In this chapter, extensions to the reactive component of the agent's architecture are described that enable the agent to take advantage of such correlations in the environment. The resulting architecture is called ConAg-ST.

4.2 Reactive Behaviors

In addition to the Approach and Avoid behaviors introduced in chapter 3, "Follow" behaviors are added to the range of innate reactive behaviors available to the agent. These are *Follow-Red*, *Follow-Green*, and *Follow-Blue*. These behaviors enable the agent to follow a "trail" of discs and are useful in learning temporal correlations. The symbols used for these behaviors in the figures and equations are listed in table 4.1.

The input to behavior B_i is the vector of sensor activations and the output is a motor activation, $\langle m_{B_i}^{speed}, m_{B_i}^{angle} \rangle$, computed from the sensor activations. In addition, *sensory excitation* s_{B_i} is computed that is a measure of the confidence in that behavior. This confidence is greatest when the activation on the sensors is strong (indicating discs are present close to the agent) and when the *current* behavior of the



(a) (b)

Figure 4.1: Examples of spatial and temporal correlations: (a) Spatial correlation: red (dark circles) and green discs (lightly shaded circles) are located together. (b) Temporal correlation: trails of red discs (dark circles) lead to green discs (lightly shaded circles).

Table 4.1: Symbols of reactive behaviors used in equations

Behavior	Symbol
Avoid-Red	VR
Avoid-Blue	VB
Avoid-Green	VG
Approach-Red	AR
Approach-Blue	AB
Approach-Green	AG
Follow-Red	FR
Follow-Blue	FB
Follow-Green	FG

agent agrees with the motor actions that *would* have been taken by behavior B_i. This comparison therefore requires examining the current sensor inputs and the direction in which the agent is currently moving. The sensory excitation of a behavior serves two purposes. Firstly, if the sensors are not activated sufficiently, then that behavior should not be performed (how much sensor activation is "sufficient" depends on the type of behavior). In this case, the sensory excitation is set to zero ($s_{B_i} = 0$). Secondly, if the motor outputs of behavior B_i matches with the current action of the agent, then behavior B_i should be reinforced. This forms the basis of the correlation learning algorithms. Thus if the actions match, then the sensory excitation of behavior B_i is set to a high value ($s_{B_i} = 1.0$).

For instance, the Approach-Green behavior produces a motor output to move the agent towards a visible green disc. Since more than one green disc may be visible, the behavior selects the closest disc. Let green sensor $i \in G$ be directed at angle θ_i and its activation be G_i, where G is the set of green sensors. Let green sensor $k \in G$ have the greatest activation:

$$G_k = \max_{i \in G}(G_i) \tag{4.1}$$

Then, the motor outputs of the Approach-Green behavior are

$$m_{AG}^{angle} = \theta_k \tag{4.2}$$

$$m_{AG}^{speed} = \begin{cases} 1, G_k < 0.8 \\ 0.2, G_k \geq 0.8 \end{cases} \tag{4.3}$$

$$s_{AG} = \begin{cases} 1.0, G_k > 0.8 \; or \; |m_{AG}^{angle} - \theta| < \frac{3\Pi}{4} \\ 0, G_k = 0 \\ 0.5, else \end{cases} \tag{4.4}$$

where θ is the current direction of the agent's movement.

Recall from chapter 2 that the activation of a sensor varies from 0 to 1 and is inversely proportional to the distance from the agent to the sensed disc. For instance, if $G_i = 0.4$, then a disc is present at a distance $0.6R$ from the agent where R is the sensor range for green sensors. The speed is high when the agent is far from the nearest green disc ($G_k < 0.8$) and reduces to 0.2 when the agent is close to a disc as it has to slow down to stop. If the agent is close to a green disc or if the agent is facing a disc ($|m_{AG}^{angle} - \theta| < \frac{3\Pi}{4}$), the sensory excitation s_{AG} is set to 1. Therefore, even if the agent is moving closer to a green disc because of a behavior other than Approach-Green, the sensory excitation of the Approach-Green behavior will be high. s_{AG} is thus an indicator of how applicable the Approach-Green behavior is in the current situation and is used in learning the spatial and temporal correlations in the environment.

The avoid behaviors output motor activations that direct the agent away from discs within sensor range. Since the Avoid-Green behavior is used for obstacle avoidance, the behavior only reacts to green discs that are very close to the agent. Let G' be the set of green sensors that indicate the presence of a disc close to the agent:

$$G' = \{i | G_i > 0.9, i \in G\} \tag{4.5}$$

However, the agent has to avoid *all* discs that are close to it, so the agent moves in a direction away from the sum of angular displacements of all nearby discs. The output motor activations of the Avoid-Green behavior are:

$$m_{VG}^{angle} = -\tan^{-1}\left(\frac{\sum_{i \in G'} \sin\theta_i}{\sum_{i \in G'} \cos\theta_i}\right) \tag{4.6}$$

$$m_{VG}^{speed} = 1 \tag{4.7}$$

The sensory excitation of the Avoid behaviors is also active only when the agent is very close to a disc:

$$s_{VG} = \begin{cases} 1.0, G_k > 0.9 \\ 0, G_k \leq 0.9 \end{cases} \tag{4.8}$$

The Follow behaviors enable an agent to move alongside a row of discs of the same color in a particular direction. A "trail" of red discs leading to a blue (water) disc is shown in figure 4.2. Since the direction

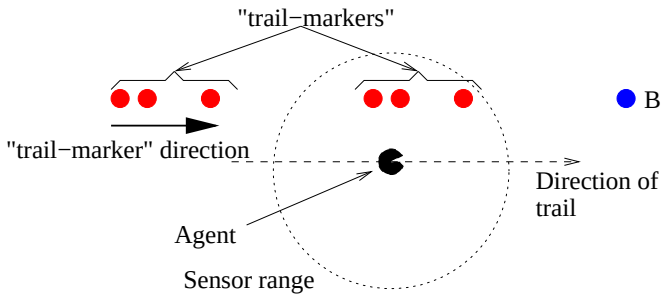


Figure 4.2: Trail of red discs leading to a blue disc (B). Every group of three red discs is a "trail-marker" indicating the direction of the trail.

of the trail has to be distinguished, every set of three discs may be considered as a "trail marker" with the direction of the trail being from the two closely spaced discs towards the third disc (two discs of the same color are sufficient to uniquely specify a line on the 2-dimensional world and a third disc is necessary to give the line a direction). The follow behavior checks the activations on the sensors for the presence of three such collinear discs (the "trail marker") and if they are present, outputs motor activations to take the agent in the direction specified by this trail marker. Since discs can be occluded, it is necessary for the agent to be in a position from which all three discs are visible (for instance, if the agent is collinear with the discs, the trail marker cannot be identified). Testing for the presence of a trail marker given the activations on the sensors is implemented procedurally. The sensory excitation of the follow behavior is set to 1 if the angle between the agent's direction of motion and the detected trail marker is less than $\frac{\Pi}{4}$, and set to 0.5 otherwise. This sensitivity to the agent's direction of motion is important when learning temporal correlations.

Since the needs of the agent are regulated by its motivations, the output of a behavior is gated by these motivations before being sent to the action selection module, i.e, there are second-order connections (excitatory and inhibitory) between motivations and behaviors (as shown in figure 3.3). Let the activations of excitatory and inhibitory motivations of behavior B_i be $m_{B_i}^{excitor}$ and $m_{B_i}^{inhibitor}$ respectively. The motivations take only positive values:

$$0 \leq m_{B_i}^{excitor}, m_{B_i}^{inhibitor} \leq 1 \tag{4.9}$$

Then, the total gating for behavior B_i, g_{B_i} is

$$g_{B_i} = m_{B_i}^{excitor} \times (1 - m_{B_i}^{inhibitor}) \tag{4.10}$$

$$a_{B_i} = s_{B_i} \times g_{B_i} \tag{4.11}$$

where a_{B_i} is the activation of behavior B_i that is sent to the action selection module. Thus, a behavior is active only if it is activated by the sensors, excited by some motivation, and not inhibited by any motivation. As described in chapter 3, the avoid behaviors are excited by the avoid-obstacle motivation, Approach-Green is excited by hunger and Approach-Blue is excited by thirst. The Avoid-Green behavior is also inhibited by hunger and the Avoid-Blue behavior is inhibited by thirst. For example, the Avoid-Green behavior is active only if a green disc is visible (sensory activation), the avoid-obstacle motivation is present (excitatory motivation), and the agent is not "hungry" (inhibitory motivation). There are no excitatory motivations to the Follow behaviors and hence these behaviors will never be selected by the action selection module. The learning algorithm that is described next introduces weighted links *between behaviors* which enable these Follow behaviors to be activated even in the absence of direct motivation.

4.3 Learning Correlations between Behaviors

As described above, the activations of the behaviors are regulated only by the gating connections from the motivations. These connections are innate and do not change over the lifetime of the agent. To enable the agent to incorporate any correlations present in its environment into its behaviors, weighted second-order

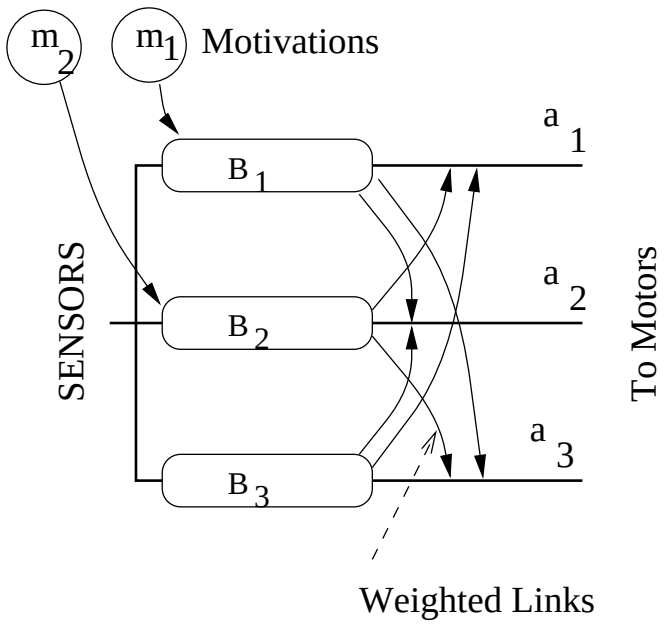


Figure 4.3: Weighted second-order links (indicated by curved arrows) between three behaviors, B_1, B_2, and B_3.

links are introduced between every pair of behaviors except the Avoid behaviors. This allows the activation of one behavior to regulate the activations of other behaviors, in addition to the motivations.

Let w_{ij} be the weight on the second-order link from behavior B_j to B_i. Behavior B_i may be excited by motivation $m_{B_i}^{excitor}$ also. Since there are no inhibitory motivations to an Approach or Follow behavior B_i, the total gating g_{B_i} and activation a_{B_i} can be now redefined as (removing inhibitory motivations and adding second-order links from other behaviors):

$$g_{B_i} = threshold(\max_j(w_{ij} g_{B_j}), m_{B_i}^{excitor}), T^g) \tag{4.12}$$

$$threshold(x, T) = \begin{cases} 1, x \geq T \\ 0, x < T \end{cases} \tag{4.13}$$

$$a_{B_i} = s_{B_i} \times g_{B_i} \tag{4.14}$$

where T^g is some threshold. The maximum activation on the second-order links from other behaviors B_j and the excitatory motivation $m_{B_i}^{excitor}$ is used to gate the sensory activation s_{B_i} (earlier, only the excitatory motivation could gate the sensory activation). Figure 4.3 shows the second-order links between three behaviors.

The weights are learned to enable the agent to change its behavior to take advantage of both spatial and temporal proximity of discs in the environment.

4.3.1 Spatial Proximity

If two features occur together in the environment, then moving towards one of the features will take it to the other feature as well. For example, food discs may always be present along with bricks. So, if the agent is hungry it should select Approach-Red behavior even though it might not initially perceive green discs near it.

The agent discovers spatially proximal features of the environment when the corresponding behaviors are simultaneously active. Thus if one of the behaviors is excited (through some motivation), then it should spread its activation to the other behavior too. This can be accomplished by increasing the weight on the

second order link between the two behaviors using a Hebbian learning rule. Let B_i and B_j be two behaviors and the weight of the second-order link from B_j to B_i at time t that represents the spatial correlation between i and j be $w_{ij}^{sp}(t)$ (w_{ij}^{tmp}, the weight that corresponds to temporal correlations is introduced later). Let the sensory excitations of the behaviors be s_{B_i} and s_{B_j}, T^{sp} be some threshold and η^{sp} the learning rate. Then the spatial proximity learning rule is as given in figure 4.4.

Spatial proximity learning rule:
1: if $(s_{B_i} \geq T^{sp})$ and $(s_{B_j} \geq T^{sp})$ then
2: $\quad w_{ij}^{sp}(t+1) = w_{ij}^{sp}(t) + \eta^{sp}(1 - w_{ij}^{sp}(t))$

3: if $((s_{B_i} \geq T^{sp})$ and $(s_{B_j} < T^{sp}))$ or
$\quad ((s_{B_i} < T^{sp})$ and $(s_{B_j} \geq T^{sp}))$ then
4: $\quad w_{ij}^{sp}(t+1) = w_{ij}^{sp}(t) + \eta^{sp}(0 - w_{ij}^{sp}(t))$

Figure 4.4: Spatial proximity learning rule.

4.3.2 Temporal proximity

If a goal is reached by executing a sequence of behaviors, then the agent should enable all these behaviors if it wants to attain the goal. For example, the agent may have stumbled upon food by following a trail of red discs. Therefore, to satisfy hunger, the agent should enable the Follow-Red behavior along with the Approach-Green behavior.

In the case of temporally proximal features, the corresponding behaviors are not simultaneously active. Instead, they are active one after the other. Hence, the Hebbian learning rules are different for learning temporal proximity and a different weight, w_{ij}^{tmp}, on the second order link between behaviors B_i and B_j is used when learning temporal correlations. w_{ij}^{tmp} should be increased only when behavior B_j is excited τ time steps after behavior B_i. Thus, the sensory excitation of behavior B_i is *delayed* before being compared with the sensory excitation of behavior B_j. The temporal proximity learning rule is given in figure 4.5.

Temporal proximity learning rule:
1: if $(s_{B_i}(t-\tau) \geq T^{tmp})$ and $(s_{B_i}(t) < T^{tmp})$ and $(s_{B_j}(t) \geq T^{tmp})$ then
2: $\quad w_{ij}^{tmp}(t+1) = w_{ij}^{tmp}(t) + \eta^{tmp}(1 - w_{ij}^{tmp}(t))$

3: if $(s_{B_i}(t-\tau) \geq T^{tmp})$ and $(s_{B_i}(t) < T^{tmp})$ and $(s_{B_j} < T^{tmp})$ then
4: $\quad w_{ij}^{tmp}(t+1) = w_{ij}^{tmp}(t) + \eta^{tmp}(0 - w_{ij}^{tmp}(t))$

Figure 4.5: Temporal proximity learning rule.

The condition $s_{B_i}(t-\tau) \geq T^{tmp}$ and $s_{B_i}(t) < T^{tmp}$ (lines 1 and 3 in figure 4.5) is true when the sensory excitation of behavior B_i is decreasing (activation of B_i was above threshold at time $t-\tau$, but is below threshold at time t). If sensory excitation of behavior B_j is also active at time t (line 2), then the weight w_{ij}^{tmp} is increased, else decreased (line 4). η^{tmp} is the learning rate.

Figure 4.6 shows the sensory excitations of behaviors B_i and B_j with time and the period when temporal learning occurs. The figure also shows how learning rate is affected by the choice of τ. A larger value of τ allows for a longer learning period but there is also a period when the weights are erroneously decreased ($\tau = 5$ compared to $\tau = 1$). Thus, as the interval between the end of the first behavior and the beginning of the second increases, so should the value of τ. This also means that learning becomes slower as the interval

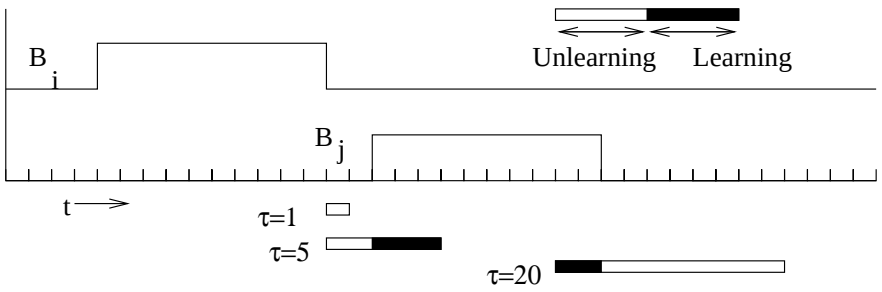


Figure 4.6: Temporal learning with $\tau = 1, 5, 20$. The bar shows the period when learning or unlearning can occur ($s_{B_i}(t-\tau) \geq T^{tmp}$ and $s_{B_i}(t) < T^{tmp}$). The filled portion shows learning ($s_{B_j}(t) \geq T^{tmp}$), while the unfilled portion shows unlearning ($s_{B_j}(t) < T^{tmp}$).

between the two behaviors increases (because of the erroneous decrease in weights involved with large τ). However, an excessively large value of τ will lead to large periods of erroneous decrease in weights as shown in the figure for $\tau = 20$.

The spatial learning rule is not a special case of the temporal learning rule for $\tau = 0$. This is because for temporal correlation, learning occurs only during the short period when the activation of behavior B_i is decreasing. For spatial correlation, learning or unlearning occurs when the activations of at least one of the behaviors B_i and B_j are above the threshold. Moreover, the spatial learning rule is symmetric because w_{ij}^{sp} and w_{ji}^{sp} increase or decrease at the same time. In the case of temporal learning, w_{ij}^{tmp} is independent of w_{ji}^{tmp}. For these reasons, the agent maintains both the weights w_{ij}^{sp} and w_{ij}^{tmp} separately on the link between behaviors B_i and B_j. To calculate the activation on the link, the weight used is the maximum of these two components:

$$w_{ij} = \max(w_{ij}^{sp}, w_{ij}^{tmp}) \tag{4.15}$$

The maximum is used because behavior B_j can either be spatially or temporally correlated with behavior B_i.

4.4 Experiments with Learning Rules

This section describes experiments that show how the agent can use the spatial and temporal learning rules to adapt to regularities in its environment. These experiments also show learning correlations improve the ability of the agent to satisfy its hunger and thirst motivations. The usefulness is measured by the average number of time-steps during which the agent was hungry or thirsty after learning compared to the case when the sensor range is infinite (optimal situation) and to random exploration.

The world is restricted to 100×100 units. The agent is provided with 60 distance sensors of each color that are spaced evenly all around it (each sensor is sensitive over a sector of angle 6 degrees). A random error of $\pm 1\%$ is added to the sensor activations. The range of the green and blue sensors are 5 units while that of the red sensors is 10 units. The thresholds are $T^{sp} = T^{tmp} = 0.6$, $T^{g} = 0.6$, learning rates $\eta^{sp} = 0.005$, $\eta^{tmp} = 0.01$ and the delay time for temporal learning, $\tau = 5$. Initially, only the weights from hunger motivation to Approach-Green and from thirst motivation to Approach-Blue are above threshold and all other weights are set to zero.

If some feature of the environment (for example, a green disc) is always present at the end of a trail of discs, then the temporal learning rule can associate the corresponding Follow behavior to that feature only if the agent follows the trail to the end (until the feature is reached). During random exploration, a trail will often not be followed to its end and thus the temporal learning rule will *decrease* the weights to the Follow behavior. To learn such correlations that require a behavior to be sustained over a period of time, an omniscient "parent" is used to guide the learning agent to the nearest green or blue disc when hungry or thirsty respectively. The "parent" does not physically exist in the environment. Learning temporal correlations

require that the right sequence of behaviors be performed repeatedly and omniscience when hungry or thirsty is used to reduce the time spent randomly moving in the environment. Note that random explorations (when the agent is not hungry or thirsty) still cause the learning agent to follow trails incompletely and that the path followed by the "parent" does not necessarily follow a trail - the agent moves along a straight line to the closest food or water disc. The use of a "parent" just increases the chances that the correct behavior (following a trail to its end) will be exhibited.

4.4.1 Spatial Proximity

To test the spatial proximity learning rule, the agent was placed in an environment in which a red disc was always present close to a green disc (figure 4.7). The positions of the (20 green and 20 red) discs in the world were set randomly. The internal food thresholds are set such that the agent's hunger motivation becomes active every 350 steps ($T_0^h = 0.1$, $T_1^h = 0.7$, $f_0 = -0.002$, $f_1 = 0.05$) and at this time it is led by its "parent" to the nearest green disc until the agent's own Approach-Green behavior becomes active (green disc in sensor range). The spatial proximity learning rule is applied to every pair of behaviors at every time-step. Figure 4.8 shows the increase in $w^{sp}_{AR,AG}$ over 8000 time-steps (averaged over 50 trials). This spatial correlation between red and green discs can also be learned through random exploration (without the need for a "parent") and the increase in weight of $w^{sp}_{AR,AG}$ in this learning scenario is also shown in figure 4.8. The rate of learning is slower compared to the case when the "parent" was present.

Once $w^{sp}_{AR,AG}$ has increased beyond the threshold, the agent can approach a red disc when its hunger motivation becomes active even if no green disc is within sensor range. The performance after learning is compared to the case when an omniscient parent is present to lead the agent to food when hungry (optimal) and to random exploration (the situation before learning) in figure 4.9. The graph shows the average number of time-steps out of 8000 during which the agent was "hungry" for the three cases (the actual values are dependent on the environment and agent parameters such as average distance between discs, sensory range, and how frequently hunger motivation becomes active; the numbers in the graph are for comparison to each other).

To show that spatial correlations can be learned even when the distance between the correlated discs varies by larger amounts, the spatial correlation rule was applied in environments where the distance between red and green discs varies randomly between 2 and 3 units (figure 4.10). The corresponding changes in the weight from Approach-Green to the Approach-Red behavior is shown in figure 4.11. Since the sensory range of the green sensors is only 5 units, the rate of increase of weights decreases as the distance between the correlated discs increase. When the distance between red and green discs can vary as much as 3 units, the weight $w^{sp}_{AR,AG}$ just goes above the threshold. Further increases in distance between the correlated discs will not cause spatial correlations to be learned. Notice also that since the disc pairs are randomly distributed there are instances when the agent can perceive more than one correlated pair at the same time. In fact, it was observed that increasing the density of correlated disc pairs did not have a significant impact on the learning of the spatial correlations (in other words, it is more important that a red disc is perceived along with every green disc compared to whether more than one pair is simultaneously visible).

4.4.2 Temporal Proximity

To test the temporal proximity learning rule, the agent was placed in an environment where "mini-trails" (each consisting of only two sets of trail markers) of red discs led to a blue disc as shown in figure 4.12. The distance between the last red disc of a mini-trail and a blue disc is too large for the Follow-Red and Follow-Blue behaviors to be simultaneously active. Thus the spatial proximity learning rule cannot be applied here. However, the application of the temporal learning rule increases the weight of the link between these two behaviors. The agent's thirst motivation becomes active every 70 steps ($T_0^t = 0.1$, $T_1^t = 0.7$, $w_0 = -0.01$, $w_1 = 0.1$) and at this time it is led by its "parent" to the nearest blue disc until the agent's own Approach-Blue behavior becomes active (blue disc in sensor range). The link between Follow-Red and Approach-Blue behaviors cannot be learned through random exploration (as in the spatial proximity case) because during a random walk the number of times a mini-trail is partially followed, without leading to a blue disc, is greater than the number of times a mini-trail is followed to its end (leading to a blue disc).

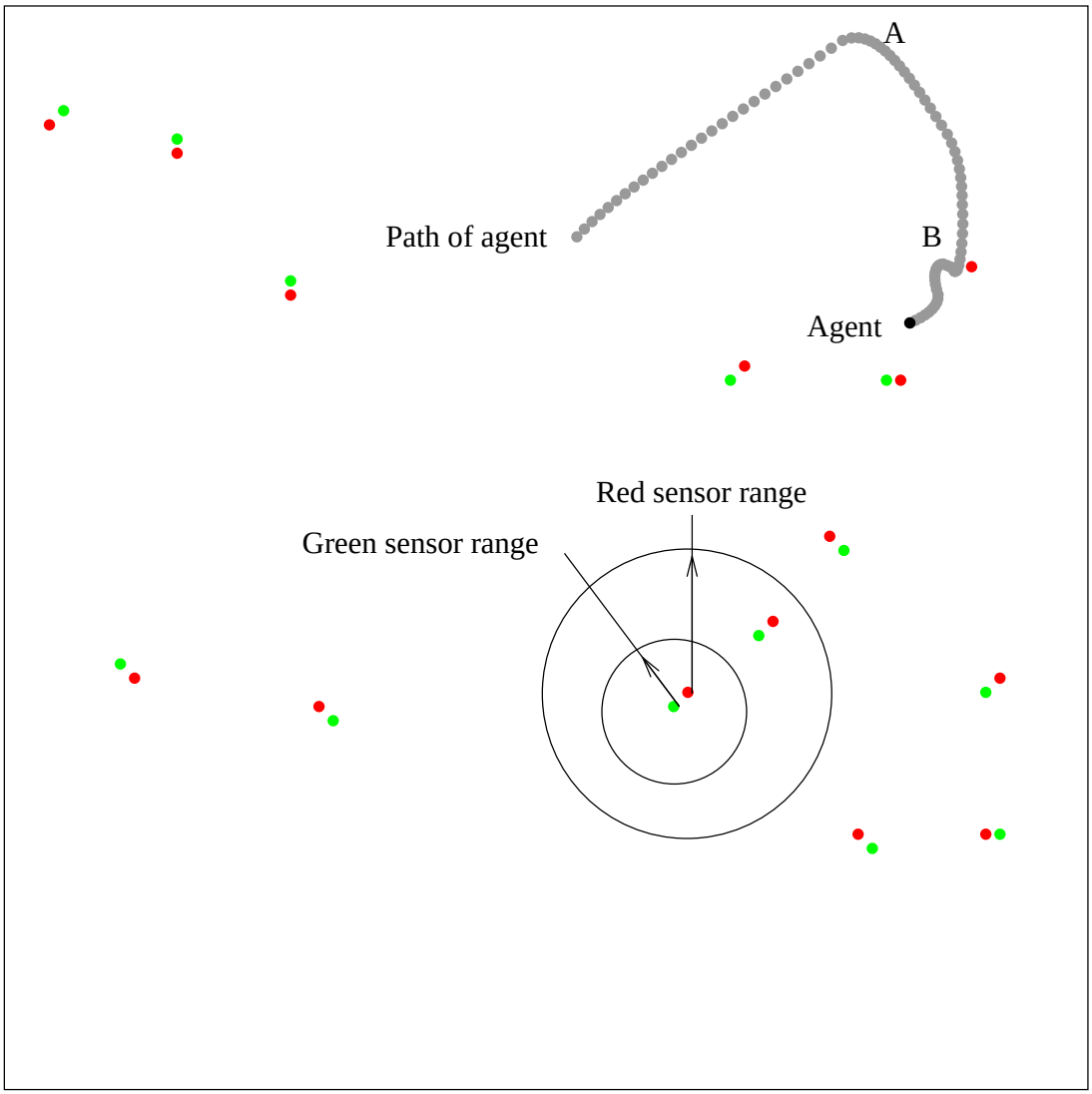


Figure 4.7: Spatial proximity: portion of the environment showing green and red discs that appear together. A portion of the agent's path is also shown. The agent becomes hungry at A and moves to the closest food (green) disc (at B) under the influence of the omniscient "teacher").

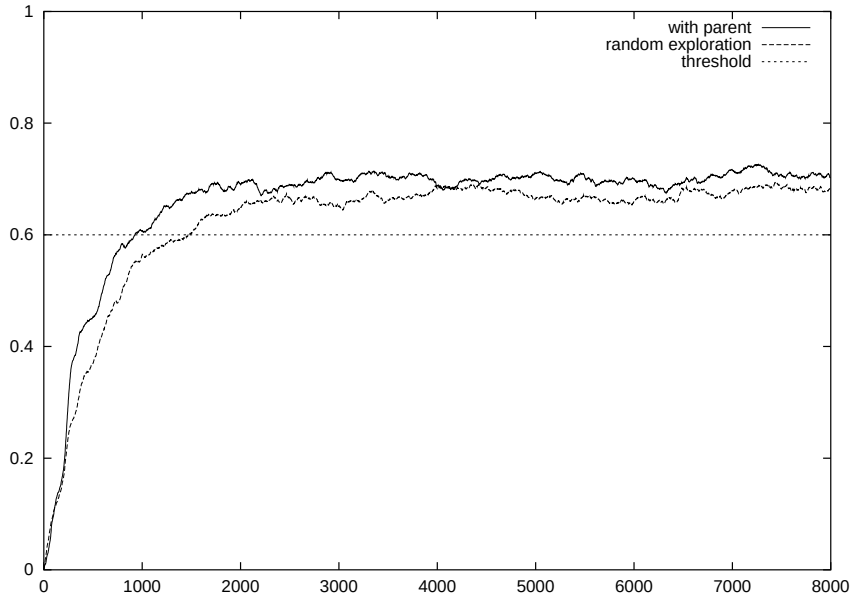


Figure 4.8: Spatial proximity results: increase of weights $w^{sp}_{AR,AG}$ when "parent" is available and during random exploration. Data averaged over 50 trials.

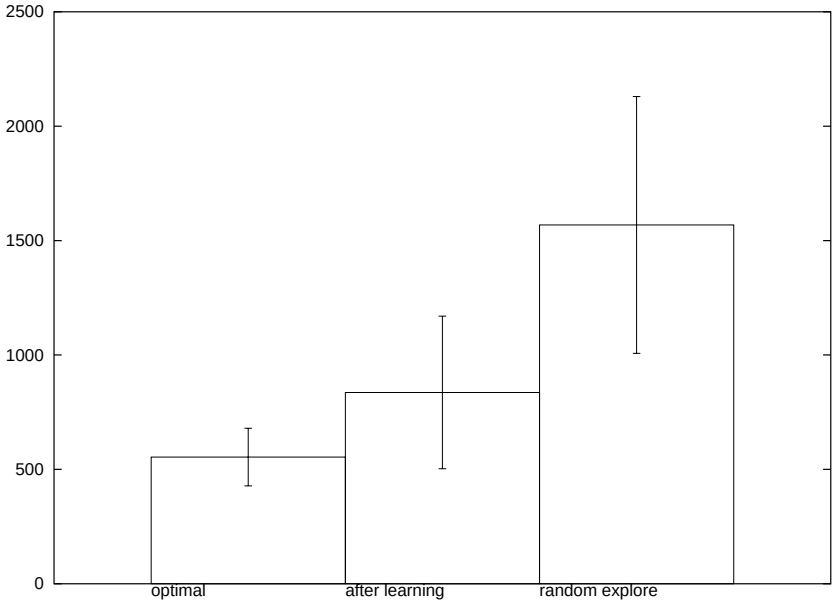


Figure 4.9: Spatial proximity results: Performance of agent with omniscient "parent", after learning, and random exploration (before learning). Data averaged over 50 trials; Error bars indicate 1 standard deviation.

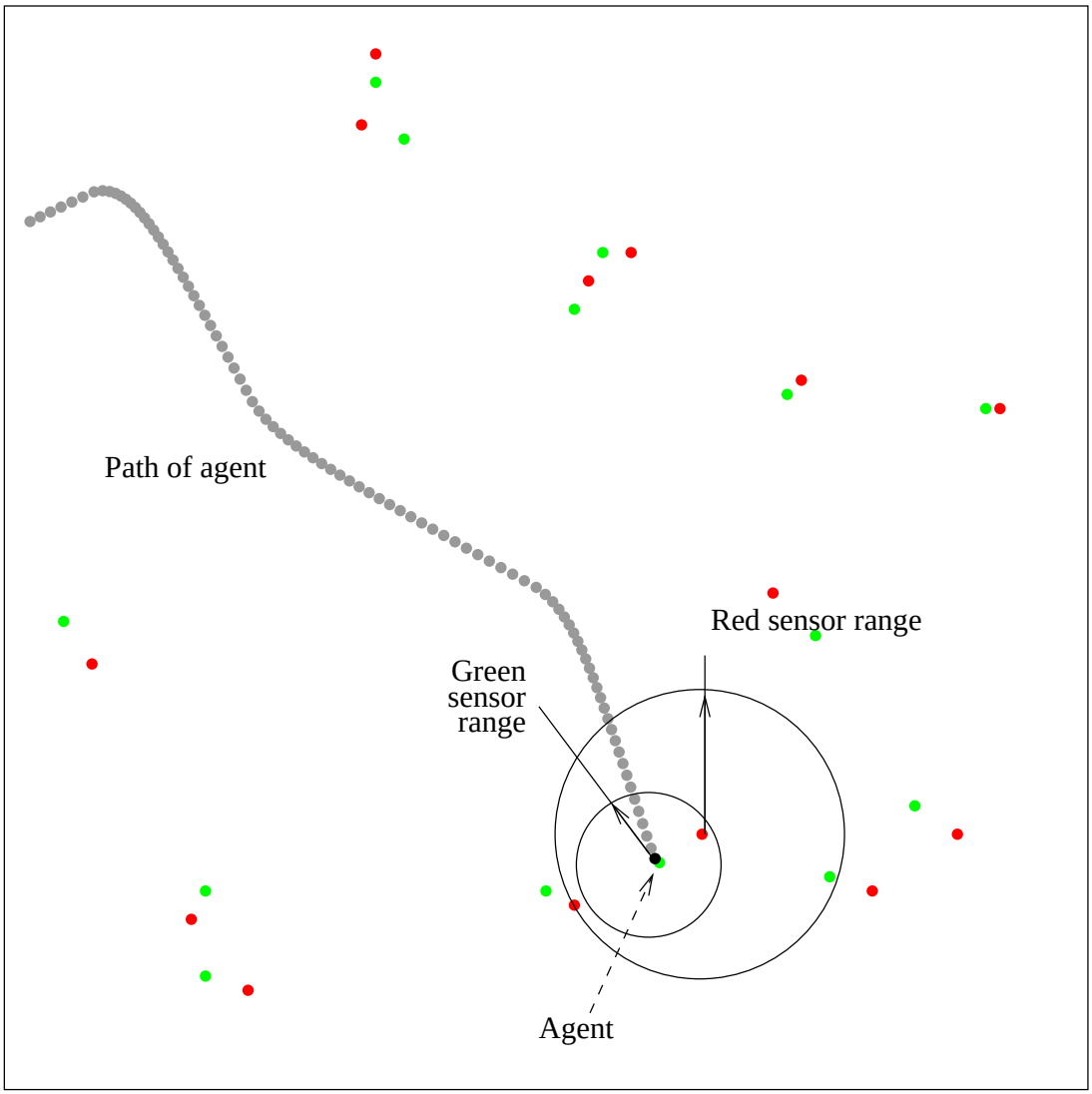


Figure 4.10: Spatial proximity (varying distance): portion of the environment showing green and red discs that appear together. The distance between the red and green discs varies between 1 and 3 units (the sensory range of the green sensors is only 5 units).

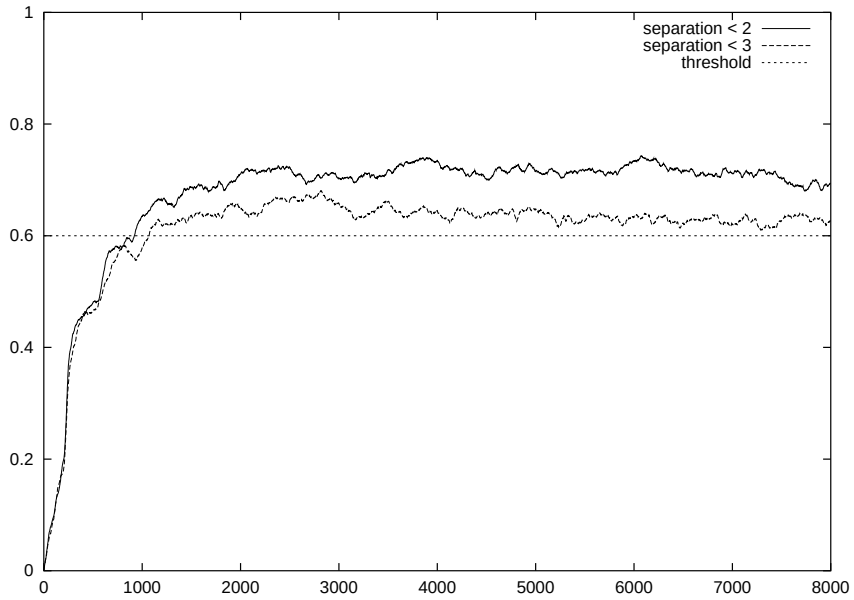


Figure 4.11: Spatial proximity weights (with varying distance): increase of weight $w^{sp}_{AR,AG}$ when the distance between green and red discs is less than 2 units and less than 3 units. Data averaged over 50 trials.

Figure 4.13 shows the second-order weights $w^{tmp}_{FR,AB}$, $w^{tmp}_{AR,AB}$, and $w^{tmp}_{AR,FR}$ at 8000 time-steps (averaged over 50 trials). Only $w^{tmp}_{FR,AB}$ increases over threshold. The weights stop increasing after they reach a certain value because with larger weights the rate of increase of a weight decreases (and conversely the rate of decrease goes up). At equilibrium, the rate of weight increase (due to activating the corresponding behaviors sequentially) is equal to the rate of decrease (during random exploration, the mini-trail is not followed to the end).

Once $w^{tmp}_{FR,AB}$ has increased beyond the threshold, the agent has learned to follow a trail of red discs when its thirst motivation becomes active even if no blue disc is within sensor range. The performance after learning is compared to the optimal case (when an omniscient parent is present to lead the agent to a blue disc when thirsty) and to random exploration (situation before learning) in figure 4.14. The graph shows the average number of time-steps out of 8000 during which the agent was "thirsty" for the three cases.

Figure 4.15 shows an environment where the trails leading to blue discs are longer and curved. The agent learns the temporal correlation between the Follow-Red behavior and the Approach-Blue behavior in this environment too though the rate of learning is slower compared to the mini-trails environment (corresponding weight increases shown in figure 4.16). To test the robustness of the Follow behavior, the agent after having learned the temporal correlation, was placed in an environment where a long trail with many curves led to a water disc (figure 4.17). The agent is able to follow the trail and reach the water disc at the end. The behavior is robust only if the discs are placed at regular specific distances. If they are too far apart, then the agent will lose sight of trail markers and will not be able to complete the trail (the distances between the three discs that make up a trail marker cannot be changed since the relative positions encode the direction in which the agent has to move). These rigid constraints arise due to the simplicity of the environment (only discs are present) and the limited sensing capabilities of the agent and not due to any inherent limitation in the temporal correlation learning algorithm.

4.4.3 Spatial and Temporal Proximity Together

Figure 4.18 shows an environment where green discs are always close to red discs and blue discs are present at the end of a trail of red discs. To take advantage of this peculiarity of the environment, the agent has to link the Follow-Red and the Approach-Blue behaviors together (through temporal learning, since these are

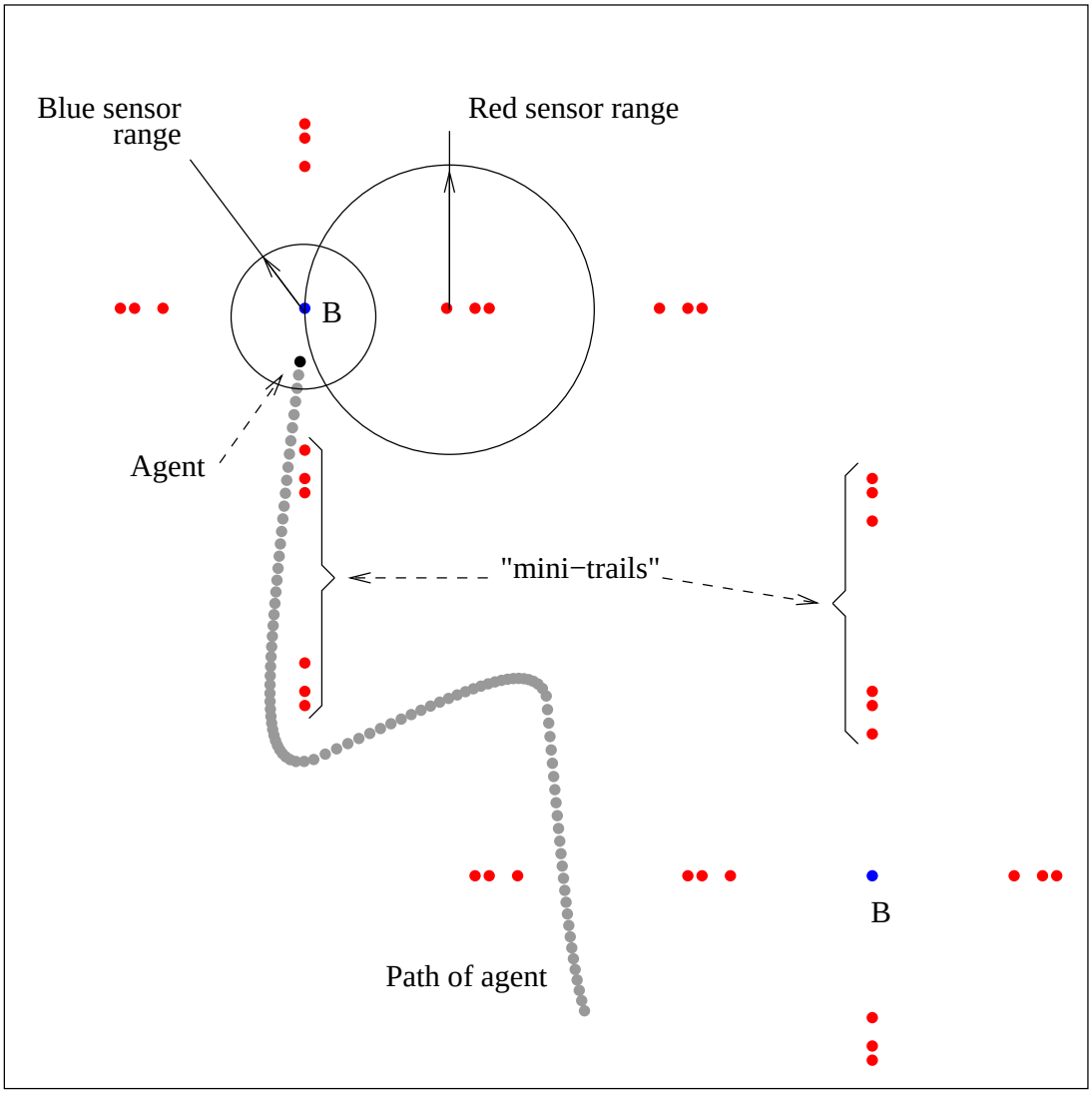


Figure 4.12: Temporal proximity results: environment contains “mini-trails” (consisting of two sets of trail markers made of red discs) leading to a blue disc (B).

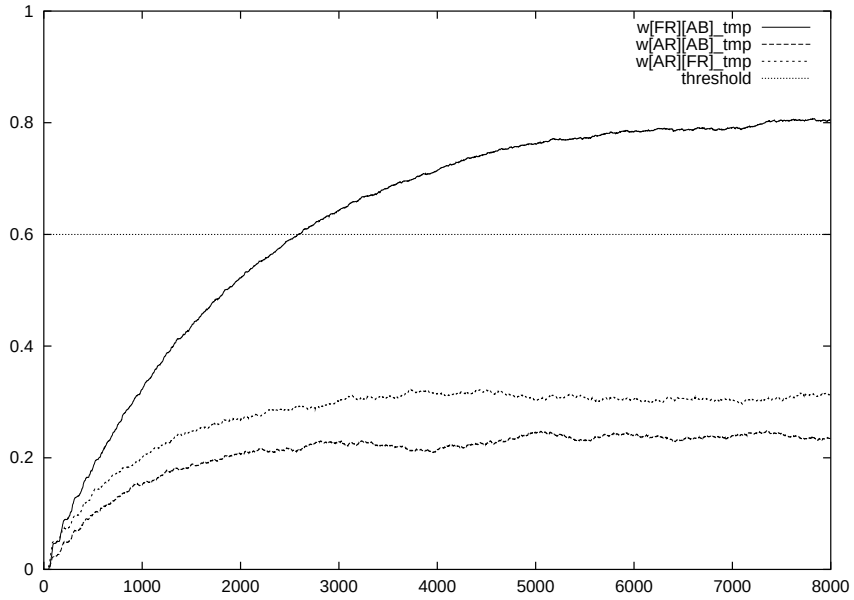


Figure 4.13: Temporal proximity results: weights $w^{tmp}_{FR,AB}$, $w^{tmp}_{AR,AB}$, and $w^{tmp}_{AR,FR}$ at 8000 time-steps. Data averaged over 50 trials.

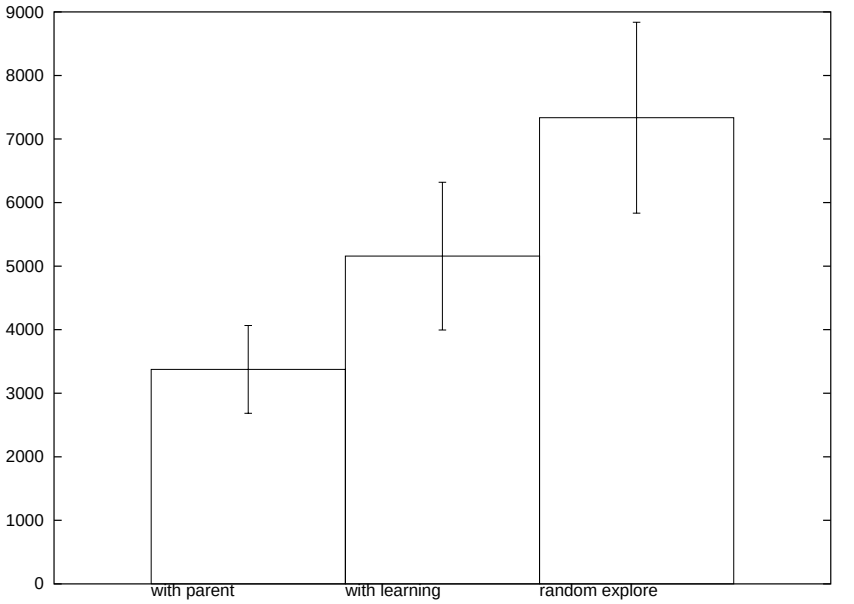


Figure 4.14: Temporal proximity results: Performance of agent in optimal case (with omniscient "parent"), after learning temporal correlation, and random exploration (before learning). Data averaged over 50 trials; Error bars indicate 1 standard deviation.

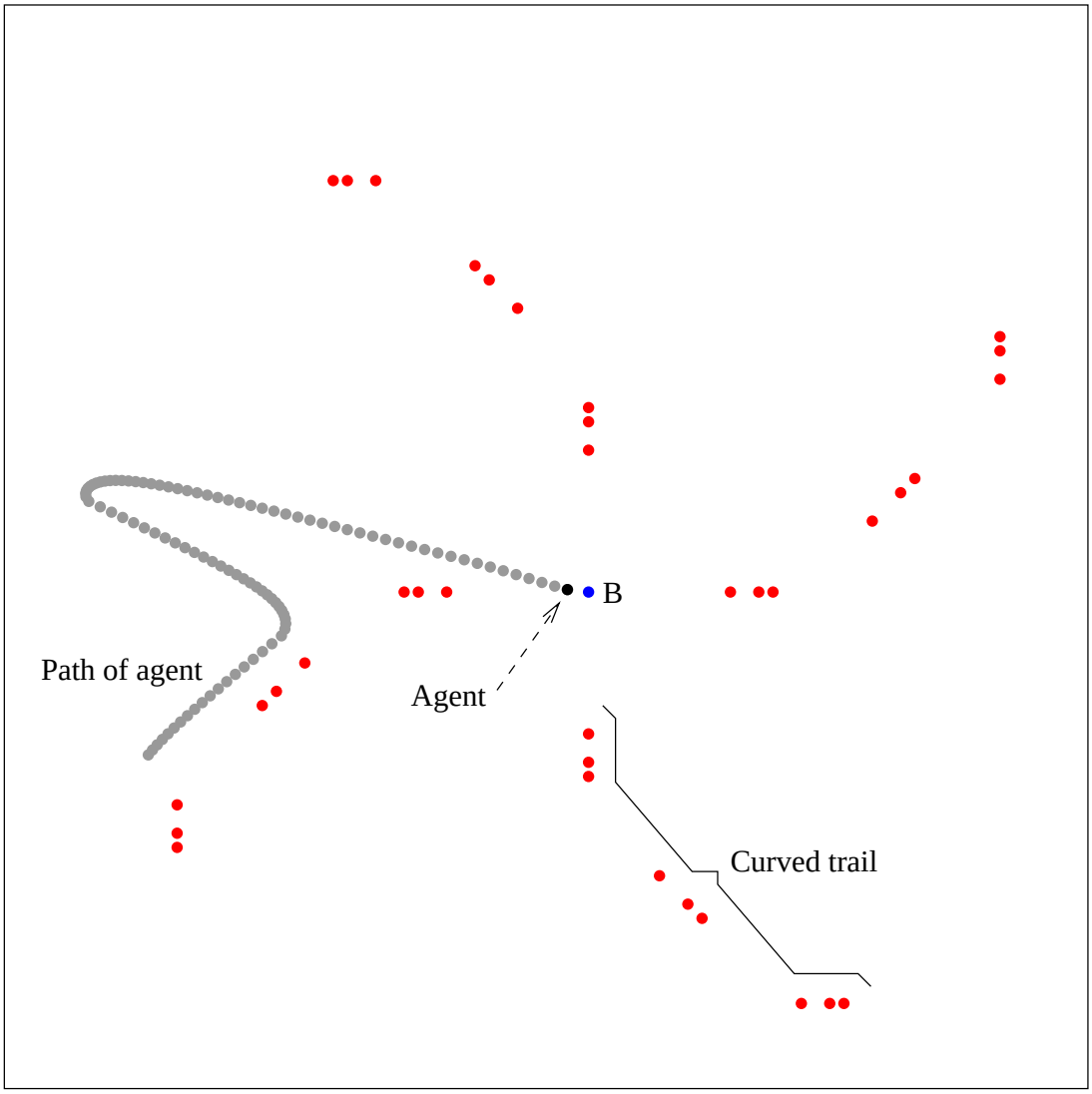


Figure 4.15: Temporal proximity results: environment contains trails (made of three sets of trail markers) that curve before leading to a blue disc (B).

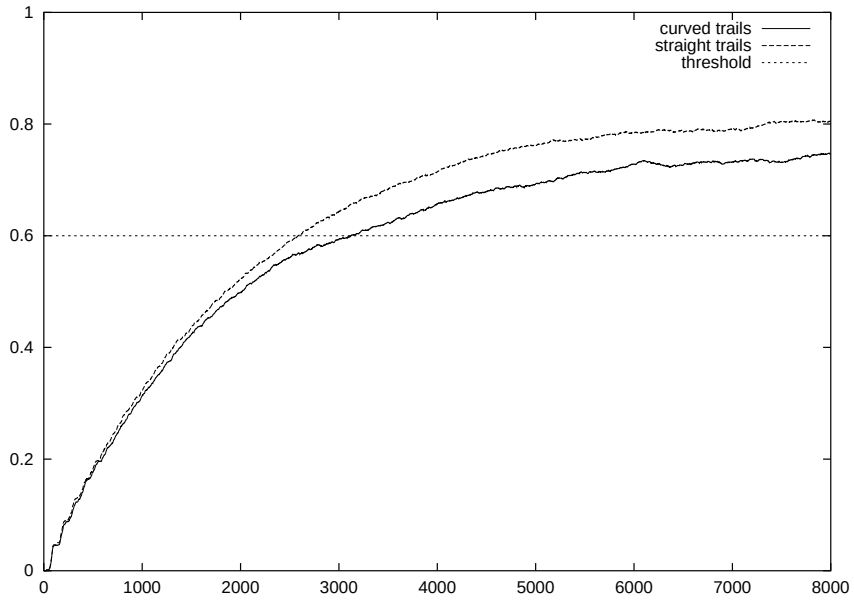


Figure 4.16: Temporal proximity: weights learned from curvy trails. Increase in $w^{tmp}_{FR,AB}$ when the trails are straight ("mini-trails" containing two trail markers) and when trails are curvy (containing three trail markers) over 8000 time-steps. Data averaged over 50 trials.

not simultaneously active) and also link the Approach-Red and Approach-Green behaviors (through spatial learning). The agent's hunger and thirst motivations become active every 350 and 70 steps respectively (the agent is made to drink more often since the temporal correlation between red trails and blue discs is learned slower than the spatial correlation). When either of these motivations become active, it is led by its "parent" to the nearest green or blue disc until the disc appears within the agent's sensor range. Figure 4.19 shows the weights $w^{sp}_{AR,AG}$, $w^{tmp}_{FR,AB}$, $w^{sp}_{AG,FR}$, $w^{tmp}_{AG,FR}$, and $w^{tmp}_{AR,AB}$ at 8000 time-steps (averaged over 50 trials). Only $w^{sp}_{AR,AG}$ and $w^{tmp}_{FR,AB}$ increases over threshold ($w^{sp}_{AR,AG}$ oscillates because red and green discs appear at periodic intervals on the "trails"). The weights between Follow-Red and Approach-Green also increase since red and green discs appear together. However, these weights do not reach the threshold because they decrease during the time the agent moves along the "trail" in a direction opposite to that indicated by the "trail-markers" (the sensor activation of Follow-Red is dependent on the direction in which the agent is moving with respect to the trail). $w^{tmp}_{AR,AB}$ also does not increase beyond the threshold for the same reason - when the agent moves along a "trail" in the opposite direction, Approach-Red is active but this does not lead to a blue disc.

Figure 4.20 shows those links of the agent that increased to over the threshold value after learning. The performance of such an agent is compared to the case when an omniscient parent is present to lead the agent to a green or blue disc when hungry or thirsty respectively and to random exploration (situation before learning) in figure 4.21. The graph shows the average number of time-steps out of 8000 during which the agent was "hungry" or "thirsty" for the three cases. Comparing this graph with that in figure 4.14 indicates that the agent spends most of its time searching for water discs when thirsty (which can be reached after the Follow-Red behavior is triggered).

4.4.4 Dynamic Environments

To test the ability of the agent to adapt its weights to a changing environment, the positions of the discs were changed during learning. Initially, red discs were close to green discs (as in figure 4.7). After 3000 time-steps, the positions were changed so that green discs were located at the end of red trails (similar to figure 4.12). The agent's hunger motivation becomes active every 150 steps. When this motivation becomes

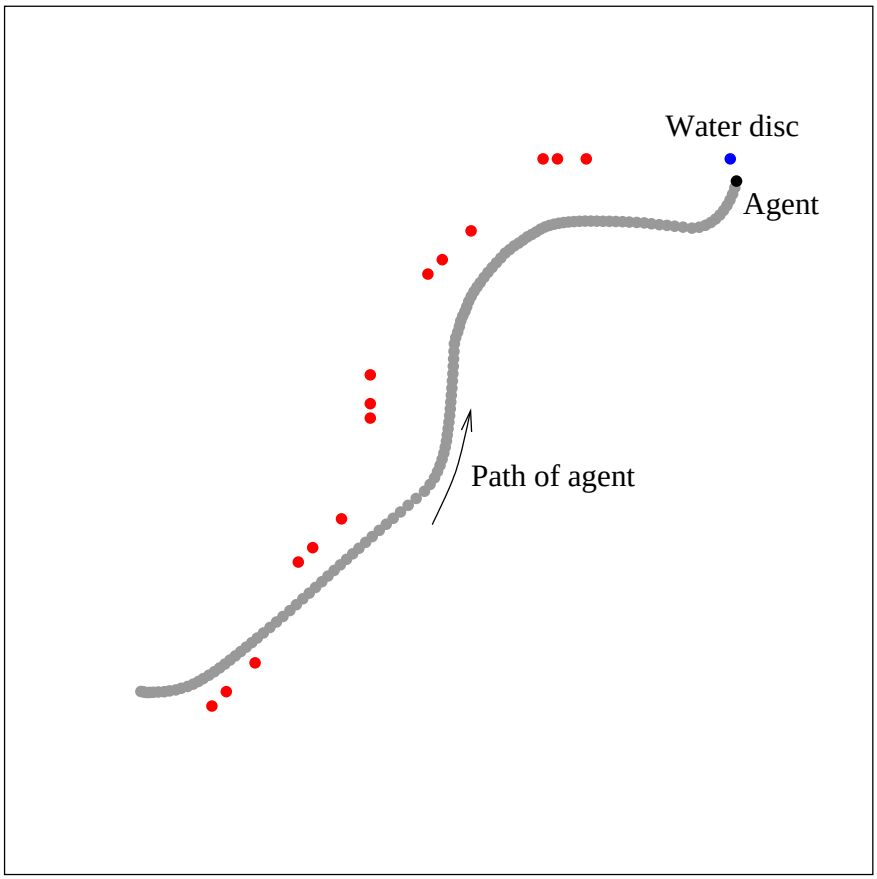


Figure 4.17: Temporal proximity: agent following a long curvy trail to reach a water (blue) disc.

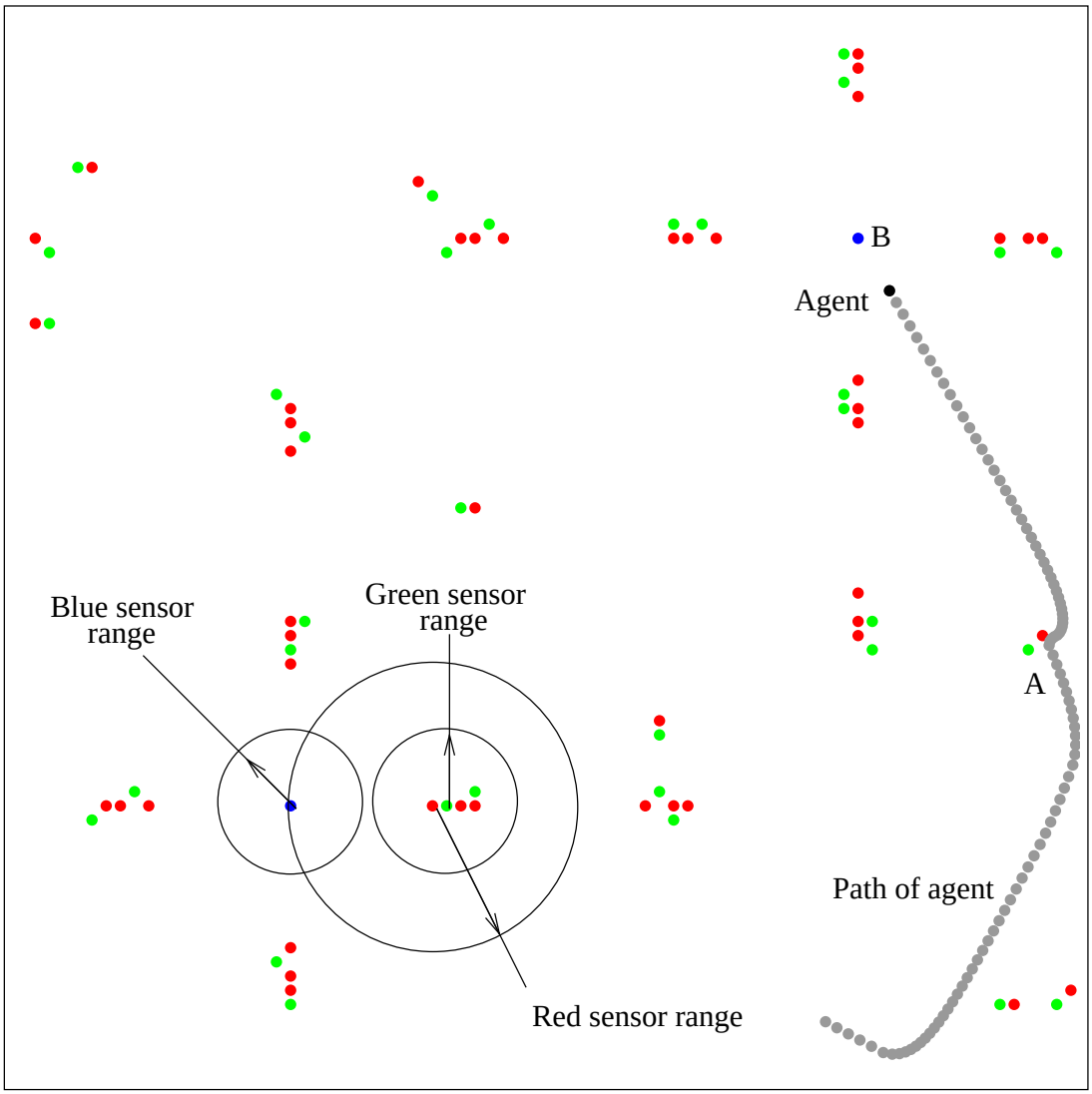


Figure 4.18: Learning Spatial and Temporal proximity together: portion of the environment containing trails of red discs leading to a blue disc; green discs are located close to red discs. A portion of the agent's path is also shown. The agent "ate" at point A and is moving toward water (blue disc at point B).

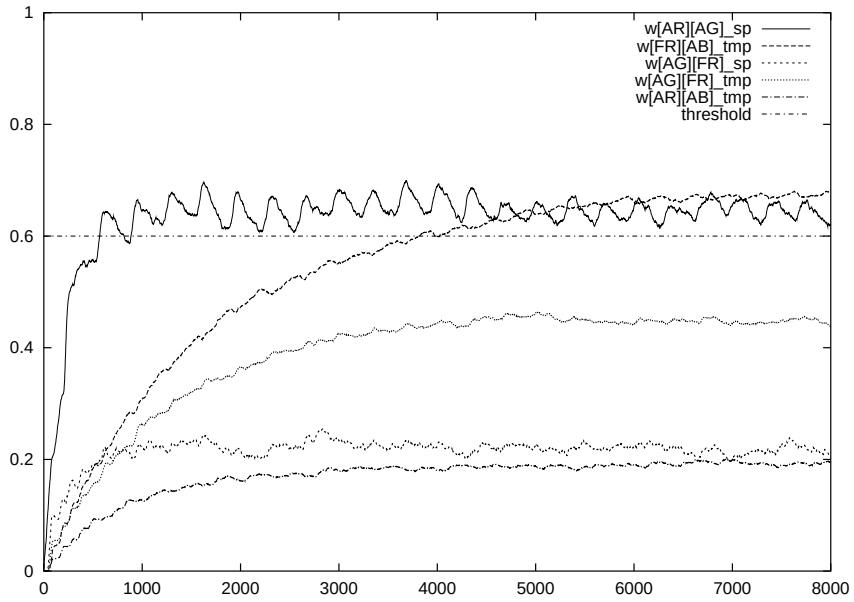


Figure 4.19: Learning Spatial and Temporal proximity together: weights $w^{sp}_{AR,AG}$, $w^{tmp}_{FR,AB}$, $w^{sp}_{AG,FR}$, $w^{tmp}_{AG,FR}$, and $w^{tmp}_{AR,AB}$ at 8000 time-steps. Data averaged over 50 trials.

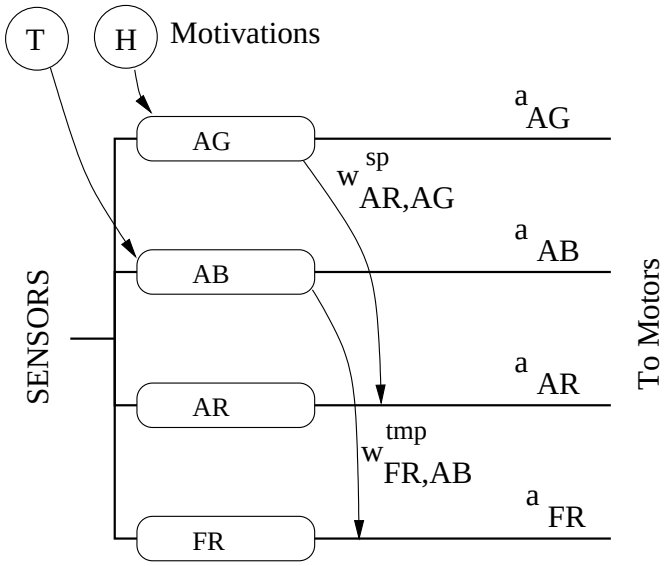


Figure 4.20: Learning Spatial and Temporal proximity together: links with weights over threshold ($T^g = 0.6$) at the end of spatial and temporal learning phase.

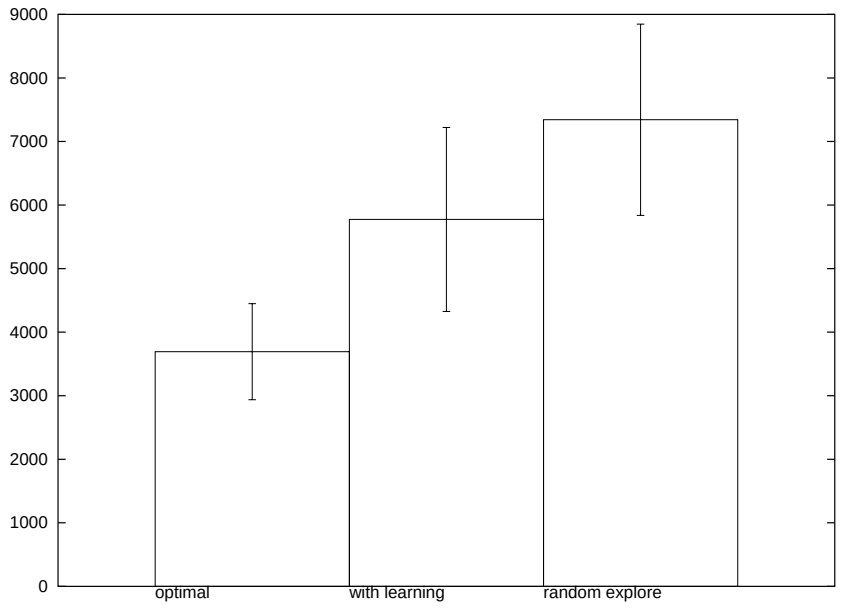


Figure 4.21: Learning Spatial and Temporal proximity together: performance of agent after spatial and temporal learning compared with omniscient "parent" (optimal) and random exploration (before learning). Data averaged over 50 trials; Error bars indicate 1 standard deviation.

active, it is led by its "parent" to the nearest green disc. Figure 4.22 shows the weights $w^{sp}_{AR,AG}$, $w^{tmp}_{FR,AG}$, and $w^{tmp}_{FR,AG}$ at 8000 time-steps (averaged over 50 trials). Initially, $w^{sp}_{AR,AG}$ increases to reflect the spatial correlation between red and green discs. After the environment changes, this weight decreases and $w^{tmp}_{FR,AG}$ increases due to the new temporal correlation between red and green discs. Thus, the agent will initially learns to move toward red discs when hungry. But when the world changes, it learns to follow trails of red discs to reach green discs when hungry.

4.5 Discussion

Since the learning rules are Hebbian, the agent can learn every time the behaviors are activated by the sensors without any kind of reinforcement. This is particularly important for agents that have to survive in a real world environment, since positive reinforcement occurs rarely and negative reinforcement can be fatal.

The learning rules do not create new behaviors, it only strings together innate behaviors. [Steels, 1997] presents a method of creating new behaviors by considering all combinations of possible perceptions, actions and the relationship between the two. Each new behavior that is created is then tested for fitness and is retained if it improves the overall health of the agent. [McFarland and Spier, 1997] provides a theoretical basis for multiplying motivations and activations. These are called "deficit" and "cues" respectively in that work.

The weights between links settle into a steady state that is dependent on environmental parameters. Thus, the threshold used to gate behaviors is also dependent on the environment. An alternative to using a threshold is to only gate that behavior with the maximum gating of all behaviors. The learning rule does not change weights on the inhibitory links. The addition of "pain" motivations could be used to learn these weights.

An agent that has learned to exploit correlations in the environment using the ConAg-ST architecture and learning rules will fail to survive in the presence of exceptions to these correlations. For instance, in an environment where most, but not all, green discs are located close to red discs, the agent will learn to move toward the nearest red disc when hungry even if that particular red disc happened to be far from a green

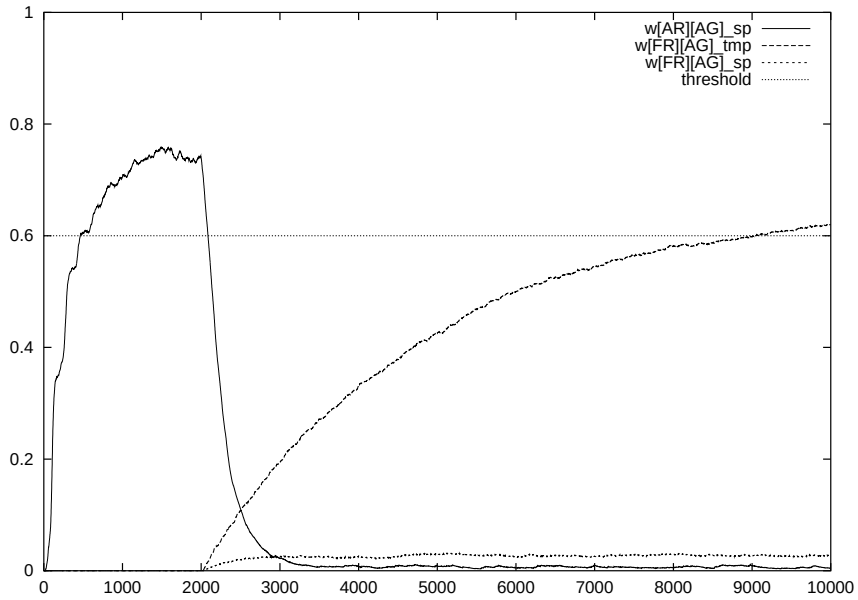


Figure 4.22: Learning in dynamic environments: weights $w^{sp}_{AR,AG}$, $w^{tmp}_{FR,AB}$, $w^{sp}_{AG,FR}$, $w^{tmp}_{AG,FR}$, and $w^{tmp}_{AR,AB}$ at 8000 timesteps. Data averaged over 50 trials.

disc. A mechanism that enables the agent to detect when it is not making progress toward satisfying its goals would be useful for handling exceptions (chapter 6 describes an unsupervised learning algorithm that uses the distance moved as reinforcement to learn to escape deadlocks).

4.6 Related Work

Chapter 3 described the Agent Network Architecture (ANA) of Maes[1990]. This architecture links behaviors through predecessor, successor, and conflicter links which spread activation among behaviors. In this initial formulation, the links between behaviors were unweighted. A learning mechanism was later introduced that enabled these links to be learned [Maes, 1992]. The links were given weights which represented the certainty with which performing the first behavior would affect the second behavior. Every behavior monitors the activation of its pre-conditions. When the activation on a pre-condition changes, the link between the pre-condition and behavior is modified. Jung and Zelinsky provides an alternate formulation of the ANA [1999]. The learning rules is slightly more sophisticated since each link keeps track of four kinds of activation changes and the weight on the link is dependent on these four counts. This kind of learning is similar to the ConAg-ST architecture in that learning identifies correlations between behaviors but does not change the behaviors themselves. Moreover, the learning is based on the changes in the activation levels of the behaviors. The activation also has two kinds of sources: behaviors and motivations in ConAg-ST, competence modules and conditions respectively in [Maes, 1992], competence modules and feature detectors respectively in [Jung and Zelinsky, 1999].

However, there are significant differences between the ANA and the ConAg-ST architecture:

1. The issue of identifying active behaviors: ANA was demonstrated with behaviors such as "wandering around", "recharging", and "ask human". Since these behaviors are very different from each other (they are expressed using different effectors), it is easy to identify an active behavior from an inactive one. In the ConAg-ST architecture, all the behaviors access the same set of sensors and produce outputs that can affect the same motors (for instance, both the behaviors Follow-Red and Approach-Red are activated when red discs are in sensor range). Hence, it is necessary to explicitly compute the Sensory excitation of a behavior before the learning algorithm can be invoked. The Sensory excitation also

enables the ConAg-ST architecture to learn correlations between behaviors, several of which may be active and applicable at any time.

2. ConAg-ST separates the learning of spatial and temporal correlations (ANA's learning rules may be considered to be only temporal). Having a separate spatial learning rule (and associated weights) improves the learning efficiency of the agent since spatial correlations may be learned faster than temporal correlations.

3. In ConAg-ST, the spreading activation is thresholded at every behavior and the activation of every behavior is the maximum of the activations that arrive at it. This ensures that the activation of a behavior cannot affect itself by following the weighted links through a path of behaviors, i.e, activations settle down after one pass through the behaviors. In ANA, activation at a behavior can accumulate from different sources and the time to settle down is significant (there could be a maximum of $(2n+1)m^2$ links where m is the number of behaviors and n is the average number of pre-conditions of a behavior [Maes, 1992]).

4. The conflictor links in ANA connect behaviors that inhibit one another. There is no equivalent inhibitory links in ConAg-ST. However, learning of inhibitor links in ANA makes the strong assumption that the when a behavior becomes inactive it is because of the behavior that was active most recently. This assumption becomes less reliable as the number of behaviors increases.

The ability of place cells in the hippocampus of the rat to respond to spatial locations has led to the hypothesis that they may be used as a topological map [O'Keefe and Nadel, 1978]. Artificial navigational systems have been built to exploit the temporal correlations between the firing of these place cells [Gerstner and Abbott, 1997]. [Trullier and Meyer, 1998] also stores the topological map in a model that is based on the rat's hippocampus. [Singh, 1991] uses reinforcement learning techniques to learn to satisfy sequences of subgoals of a structured navigational task. [Mataric, 1994] gives a criticism of such techniques in unstructured real-world domains. Sequences of actions have been learned in robot systems using imitation learning [Billard and Mataric, 2000] and vicarious learning [Crabbe and Dyer, 2001].

Reinforcement learning [Kaelbling *et al.*, 1996b] has been used to learn sequences of actions. Most of this work has been performed in synthetic domains (for instance, the "box-pushing" world [Mahadevan and Connell, 1992] and grid worlds [Singh, 1991]). However, reinforcement learning has met with limited success in unstructured real-world domains [Mataric, 1994]. The main problems include the difficulty for a robot to unambiguously identify its state with inherently faulty sensors, and the fact that the environment can change independent of the robot's actions (especially in multi-agent systems). These factors invalidate the Markov property which is assumed for reinforcement learning techniques like Q-learning [Watkins and Dayan, 1992]. Moreover, Q-learning requires that the state space be discretized while sensor data is often continuous. The large dimension of the state space (due to the number of sensors on a robot) also makes reinforcement learning perform slowly in such environments. Shackleton and Gini give a more detailed description of the application of reinforcement learning in behavior-based robots [1997].

Suggestions to speed up reinforcement learning include grouping similar states [Connell, 1988], partitioning the state space based on discovered features [Drummond, 1998], and dividing the task into sub-tasks [Stone and Veloso, 2000]. Learning correlations in the environment may be used as another means of reducing this large state space before applying reinforcement learning techniques. Another method to exploit structure in the environment is to consider only *partial views* instead of all the sensor readings [Porta and Celaya, 2000]. A partial view of the environment is one in which only a subset of the sensors are considered. The assumption is that in most real-world environments, only a few of the sensors will be relevant for deciding on an action at a time. A different unsupervised learning approach is that of treating the sensor data as being generated by a Markov chain and then clustering this data based on similar transition probabilities [Ramoni *et al.*, 2001]. Rao and Fuentes also maintain only a sparse subset of the sensor-motor space using sparse distributed memory (they use a behavior-based architecture and a "teaching-by-showing" approach similar to the temporal learning in ConAg-ST).

It has been found that temporal correlations encoded in the firing patterns of the place cells in the hippocampus may be used as the basis for a spatial map [Gerstner and Abbott, 1997]. For instance, unsupervised Hebbian learning is used to construct such a spatial map composed of neurons with overlapping

place fields (modeling place cell activity) [Arleo and Gerstner, 2000]. These models use temporal correlations in the firing patterns to build an explicit topological map, unlike in the ConAg-ST architecture where no topological map is built and only correlations between behaviors are identified for navigation.

Chapter 5

Sequence Learning

5.1 Introduction

The links in the action selection mechanism described in chapter 3 enable an agent to perform the construction task. 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, i.e, the interconnections between the internal state nodes and the navigational planning behaviors do not have to be set *a priori*. The spatial maps also indicate in which phase of the construction task the agent is currently in, including whether the task has been completed. This chapter describes how the weights in the action selection module can be learned and the internal state nodes extended to identify the end of the construction task.

5.2 Internal State Nodes

In addition to the Holding-Brick, At-Drop-Site, and At-Brick internal state nodes introduced in chapter 3, three other state nodes are added:

1. *Near-Brick*: Becomes active when a brick is sensed close to the agent.
2. *Brick-Available*: Is active if the spatial maps indicate a brick that is not part of the structure being built.
3. *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.

The Brick-Available and Drop-Site-Available state nodes indicate if the construction task is complete or that there are no more bricks that can be moved to drop-sites. The activations on these state nodes can also be set directly from the navigation maps. For instance, if the structure is complete, then there will be no active node on the Configuration navigation map. Thus, the Drop-Site-Available state node is set from the sum of activations on the nodes of the Configuration navigation map. Similarly, if there are no bricks that are not part of the structure, then none of the nodes on the Brick navigation map will be active. Thus the activation of the Brick-Available state node is set from the sum of activations on the nodes of the Brick navigation map. The activation of the Near-Brick internal state node is set from the activations of the red disc sensors.

5.3 Construction Sequence

If none of the self-preservation goals are active, then the agent can attend to the construction task. This task requires an orchestration of both Sensory/Motor and Navigational Planning actions, summarized in the action sequence given in figure 5.1.

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

Figure 5.1: Sequence of actions required to perform construction.

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 5.2. Each node in the action selection module may have both excitatory (arrows) and inhibitory (dots) connections from the internal state nodes. Weights range from 0 to 1 for excitatory and 0 to -1 for inhibitory, and all activations are summed and thresholded:

$$a_j = \psi(\sum_i (w_{ij}^{excite} \times a_i + w_{ij}^{inhibit} \times a_i), T^B) \tag{5.1}$$

where a_j is the activation at node j, ψ is the step function and T^B is the threshold.

$$\psi(x, T) = \begin{cases} 1, & x > T \\ 0, & x \leq T \end{cases} \tag{5.2}$$

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 assigned to both weights to ensure that *Grab* is active only when both input nodes are firing to surpass the threshold T^B of 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 weight of -1.0, 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 of -1.0 from the Near-Disc internal state node to the Avoid-Red behavior.

By using this combination of excitatory and inhibitory connections, shown in figure 5.2, the Action Selection module selects the correct action based on its currently active goals and the construction status. The issue of learning this construction sequence through weight adjustment is first discussed, followed by the demonstration of this capability in the Results section.

5.4 Learning the Construction Sequence

An agent adjusts the weights on the links in its action selection module through a process of imitation 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 ideal action selection network shown in figure 5.2. 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.

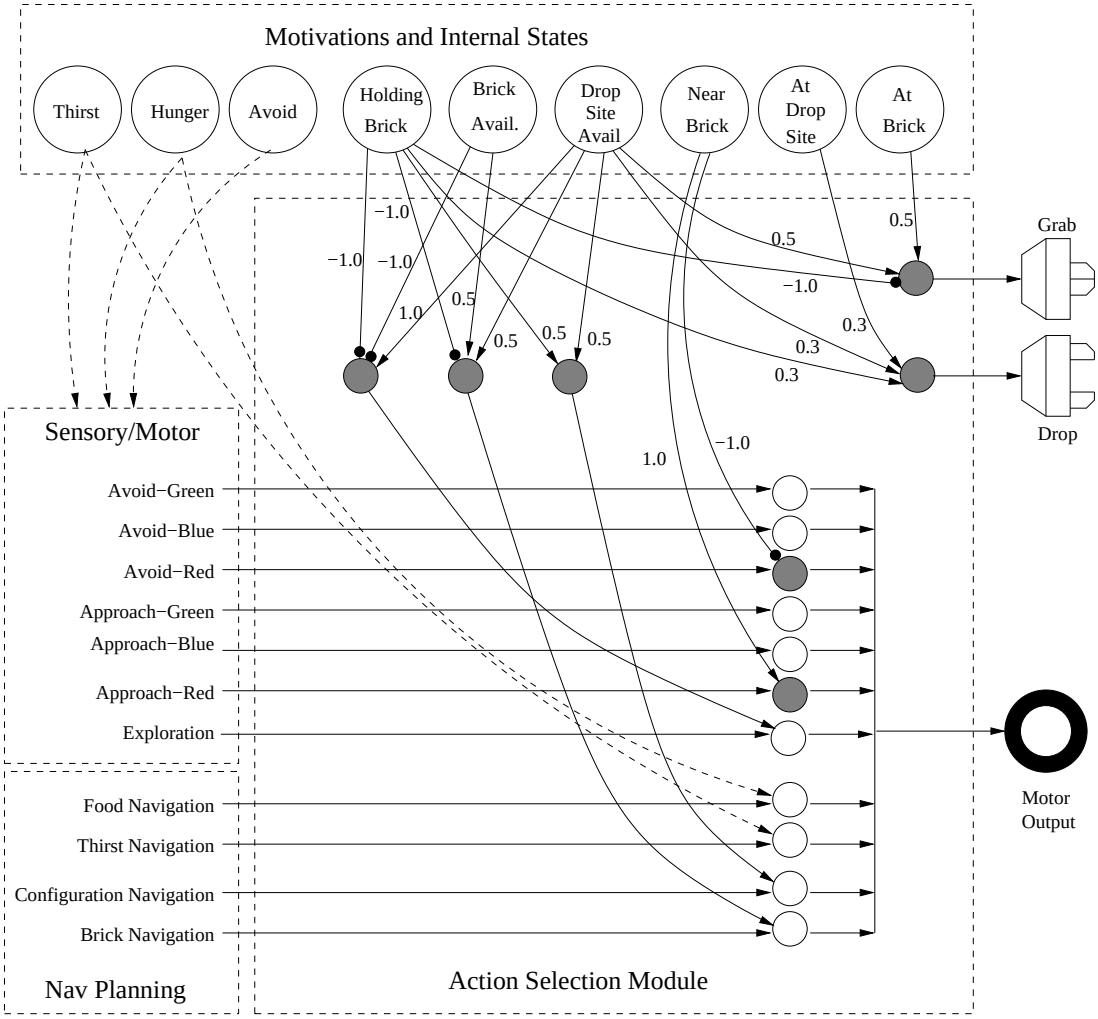


Figure 5.2: The Action Selection module with ideal connections and weights for construction. The shaded circles indicate the output nodes, the weights to which are learned (described in section 5.4). The outputs from the Sensory/Motor and Navigational Planning components are fed in from the left and regulated by the Motivation and Internal State nodes. The actions chosen are then sent to the motors on the right.

The student begins with a fully connected network, and 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 equation for weight adjustments for both the excitatory and inhibitory weights are the same and are as follows:

$$w_{ij}^{excite}(t+1) = w_{ij}^{excite}(t) \times (1 + R_j^t \times w_{ij}^{excite}(t) \times a_i \times e^{-|w_{ij}^{excite}(t)|} \times \eta^{seq}) \tag{5.3}$$

$$w_{ij}^{inhibit}(t+1) = w_{ij}^{inhibit}(t) \times (1 + R_j^t \times w_{ij}^{inhibit}(t) \times a_i \times e^{-|w_{ij}^{inhibit}(t)|} \times \eta^{seq}) \tag{5.4}$$

where w_{ij}^t is the weight between 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. The exponential term is used to restrict the range of the weights to between -1 and 1. η^{seq} is the learning rate (a real number between 0 and 1). The duration of a learning trial is determined by the size of the structure to be built and in successive learning trials η^{seq} is reduced by a decay factor ρ^{seq}:

$$\eta^{seq}(n+1) = \eta^{seq}(n) \times \rho^{seq} \tag{5.5}$$

where $\eta^{seq}(n)$ is the learning rate at trial n. The reinforcement signal R generated by the teacher takes only two values: -1 and $+1$. If a positive reinforcement of 1 is received, the excitatory weights between a pair of firing neurons at time t is strengthened, while the complementary inhibitory weights are weakened. However, a negative reinforcement of -1 causes the opposite adjustments to occur, reducing the likelihood of firing given the same inputs. Due to the nature of having both weights adjusted simultaneously, this learning rule is referred to as Positive/Negative (P/N) Hebbian learning. With a positive reinforcement, all excitatory weights that activated j are increased and the inhibitory weights decreased. However, 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. To learn the action selection sequence for construction, P/N Hebbian learning is used on two sets of weight matrices of 6 inputs - Holding-Brick, At-Brick, Near-Brick, Brick-Available, Drop-Site-Available, and At-Drop-Site by 7 outputs - Grab, Drop, Configuration navigation, Brick navigation, Explore, Avoid-Red, and Approach-Red, representing the fully connected network of both excitatory and inhibitory links.

If the student performs the correct actions, the weights that resulted in these actions are reinforced. If a wrong action is performed, one could simply weaken the weights that produced the erroneous activation. However, this form of "blame assignment" is flawed since the agent could potentially never learn because the correction action is never activated and thus never positively reinforced. The correct node that is supposed to fire should also be trained. However, the reinforcement signal does not identify the correct node. Instead of having the student agent randomly guess the correct action, all actions except the one just executed are treated as correct (even though only one is correct). As a result, the incorrect node receives a negative reinforcement while all others receive a positive one. With sufficient training, the correct node will prevail while all the incorrect nodes cease to activate under the same inputs, increasing the chance of future activation of the correct node with the same inputs. Through P/N Hebbian learning, the student's weights are adjusted to activate the right nodes in the right sequence, thus resulting in the student performing the construction task as specified by the teacher.

5.5 Results and Discussion

Learning to construct a configuration of discs was evaluated by having one student agent trained in the environment shown in figure 3.12(a), where five discs were used to construct a structure composed of 4 widely separated discs. 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, i.e., the activations on the Configuration ESM. All other ESMs are initially empty and 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).

The two learning parameters, learning rate η^{seq} and decay factor ρ^{seq} of the learning rate between trials, were varied to test their effects. The results from two sets of parameters are shown in figure 5.3 (as well as the

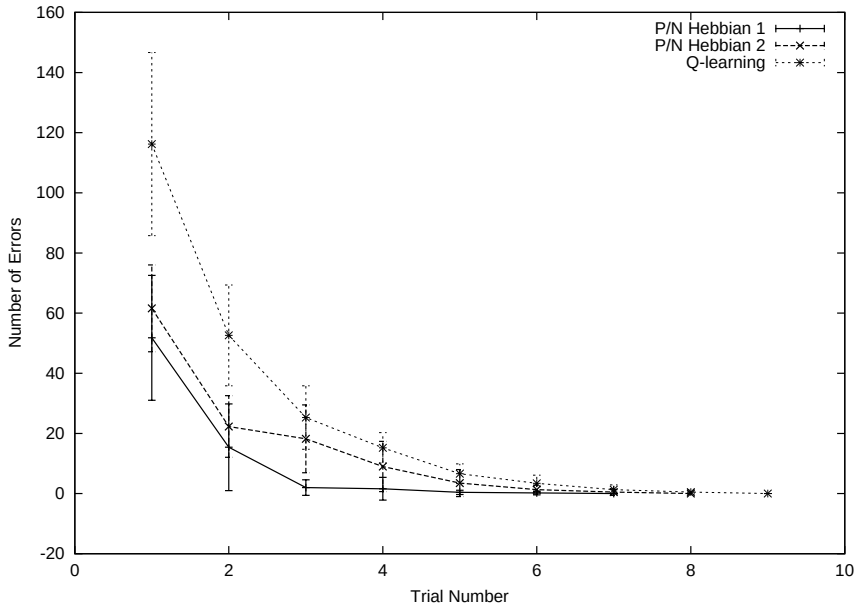


Figure 5.3: Errors during sequence learning: the number of errors made by the student agent versus the learning trial using two different learning algorithms. The two P/N Hebbian learning (parameters: $\eta_1^{seq} = 0.8$, $\rho_1^{seq} = 0.4$, and $\eta_2^{seq} = 0.6$, $\rho_2^{seq} = 0.8$) results are compared to Q-learning (parameters: $\alpha = 0.3$, $\gamma = 0.5$, $\epsilon = 0.05$)

results from using Q-learning [Watkins, 1989; Sutton and Barto, 1998] to learn and perform the same task). The number of errors (calculated as the number of time-steps when the output actions of the learner did not agree with those of the teacher, i.e, the time-steps during which a negative reinforcement was generated) during training by the student agent versus the trial number is shown, with the error bars showing the standard deviation from 10 independent runs. Each trial consists of 1000 time-steps, and between each trial the learning rate η^{seq} is reduced by the decay ρ^{seq} for P/N Hebbian learning.

The Q-learning used in this evaluation learns two action-value functions, one for Grab/Drop actions and the other for motor actions. Instead of using function approximators, the two Q-functions are implemented using tables since the state space is small (2^6). The same training and evaluation scenario, teacher-agent, and reinforcement signals are used for Q-learning, repeating the learning trials until no mistakes are made. The learning rule at every step is given by [Sutton and Barto, 1998]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{5.6}$$

where $Q(s,t)$ is the learned value of taking action a from state s, r is the reinforcement received, α is the step-size, and γ is the discount factor. Actions are chosen at every step using a ϵ-greedy method (with probability ϵ a random action is performed, otherwise the action suggested by the current Q value is performed). The ϵ parameter is halved between trials to improve the rate of convergence.

As shown in figure 5.3, P/N Hebbian learning makes fewer mistakes and converges faster than Q-learning. For P/N Hebbian, the higher learning and decay rates produced faster learning and convergence. In 7 out of 10 runs the student learned the task in 4 trials or less. With a lower learning and decay rate, the learning is slower and takes longer to converge, as expected. For Q-learning, the student agent makes more mistakes and requires more time to converge. The parameters for Q-learning were chosen to achieve a balance between the number of errors and the convergence rate.

A benefit of P/N Hebbian learning is that the weight matrices can be easily interpreted for the action sequence learned. As an example, learning of the weights to Brick navigation behavior is shown in figure 5.4. An agent should activate this behavior when there are bricks missing from the structure (state node Drop-

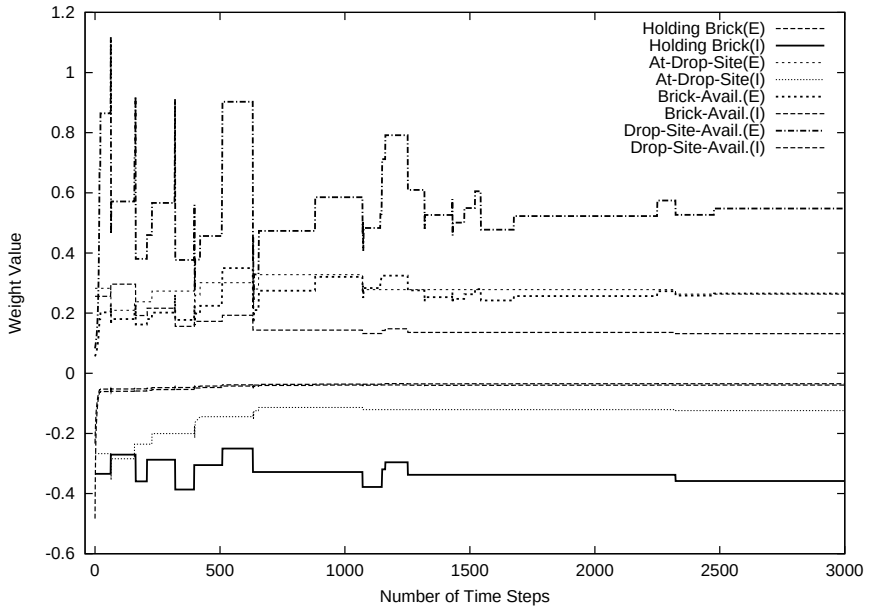


Figure 5.4: Weights during sequence learning: P/N Hebbian learning of the weight pairs between four of the input nodes to the Go to Disc node versus the number of time steps

Site-Available is active) and there are bricks available within the environment (Brick-Available is active). However, if the agent is holding a brick (Holding-Brick is active), this node should be inhibited and the agent should proceed to the location where a disc is missing from the structure (Configuration navigation behavior). These three pairs of weights are plotted over time using thick lines in figure 5.4. The excitatory weight from Drop-Site-Available state node is strengthened while the inhibitory one approaches zero, as expected. Similarly, the inhibitory weight from Holding-Brick increases and the excitatory one decreases with time. The weights from At-Drop-Site and Brick-Available state nodes are similar (unlike the teacher's weights). This is due to the following scenario: the Brick navigation behavior is activated immediately after the agent drops off a brick so it would continue the building process. However, the At-Drop-Site state node is still active from the *last* goal since it has not yet moved away far enough to stop this state node from firing. Therefore, the weight from At-Drop-Site to Brick navigation behavior tends to be strengthened alongside the weight from Brick-Available. Nevertheless, the student does learn to properly activate the Brick navigation behavior, even though it is with weights different from the human-engineered teacher network.

The complexity of P/N Hebbian learning is linear for space and time (for each update), both requiring $O(n_i \times n_o)$, where n_i and n_o are the number of inputs and outputs, respectively. However, Q-learning, when implemented using tables as in our model, is exponential with respect to n_o for time per update and n_i and n_o for space (since the table has to store Q values for every possible state-action combination) This is significant for scalability, since if the agent is to learn other complex tasks that require more input and output nodes, using Q-learning with tables can become too cost prohibitive. However, if function approximators are used in place of tables, then Q-learning would not perform nearly as well since the ideal Q-functions that describe the construction task consist of multiple sharp discontinuities and hence are difficult to learn and approximate accurately. Hebbian learning is more suited to this task since in a single-layer network the links to an output node do not affect the activation of other output nodes and therefore the weights can be adjusted independently of the weights on links to other output nodes. However, a single layer network can only threshold a linear combination of the input activations and cannot learn more complex relations between inputs and outputs. Moreover, in this learning environment, the agent was given reinforcement at every step and thus an error term could be provided to the action selection network immediately after an action was taken. Q-learning can be used even in situations where only a delayed reinforcement is available.

5.6 Related Work

Crabbe and Dyer use higher order links to control the sequence of actions required to perform construction [1999a]. Figure 5.5 shows the network structure used to perform sequences. The *detection* nodes are similar to the internal state nodes in ConAg and are set from the sensors. An example of a detection node is "Touching-Red-Disc" which would be active if the agent was currently touching a red disc. Detection nodes in turn set *Goal* nodes which encode steps of the construction sequence (such as "Approach-Red-Disc"). These goal nodes then gate the links between *sensor* and *motor* nodes. The order in which these goal nodes become active is controlled by *sequence* sub-networks (for example, "Scavenge").

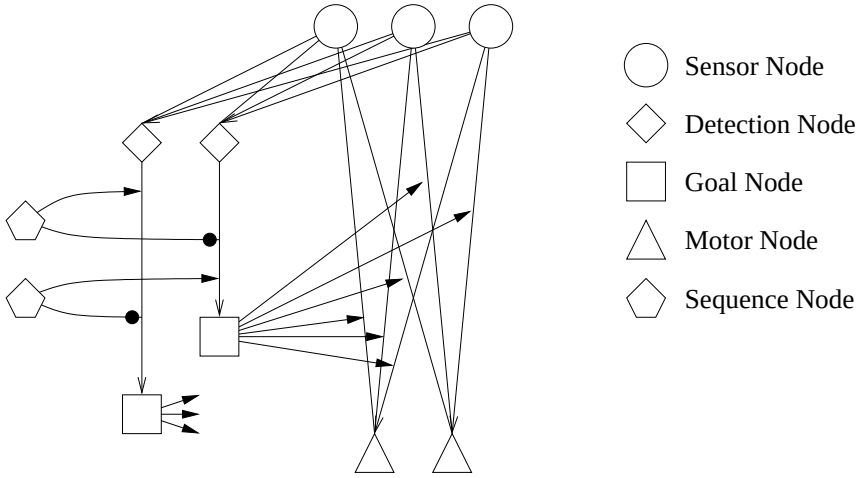


Figure 5.5: Network of nodes for a construction agent [Crabbe and Dyer, 1999a]. Goal nodes modulate the links between sensor and motor nodes using second-order links. The Goal nodes are set from the detection nodes which are in turn set from the sensor nodes. Sequence nodes control the order in which goal nodes are excited.

This network of nodes accomplish the same sequencing performed by the ConAg architecture. Figure 5.6 shows the corresponding flow of activations in both systems. The ConAg architecture is simpler since each behavior internally implements how the motors respond to sensor activation. This enables a single layer of higher order links from the internal state nodes to implement the construction sequence. In both the ConAg architecture and that of Crabbe and Dyer, the transition from one behavior (or goal) to another is brought about by a change in the environment (as encoded by the internal state or detection nodes). Thus, an unexpected failure of a behavior will cause the agent to repeat its behaviors. For instance, the agent will perform the Configuration navigation behavior only when the Holding-Brick internal state node is active. If the agent unexpectedly drops the brick, Holding-Brick becomes inactive, causing the agent to repeat the Brick navigation behavior (the agent will move to the nearest brick which is the unexpectedly dropped brick).

Crabbe and Dyer present the MAXSON (MAX-based Second Order Network) architecture that is capable of learning to associate correct responses to objects in a 2-dimensional environment that consists of food, water and poison discs [1999b; 2001]. For instance, a MAXSON agent could learn to avoid eating poison. The architecture contains two networks: a second-order *policy* network and a first-order *value* network (figure 5.7). The policy network is responsible for activating the motors based on the current sensor readings. The agent receives reinforcement whenever it eats or drinks. The value network calculates reinforcement for the policy network and the policy network weights are adjusted based on this reinforcement. The weights in the value network are itself adjusted based on the external reinforcement. The purpose of having a separate value network to calculate reinforcement (instead of directly applying external reinforcement to the policy network) is to distribute the rarely received reinforcement over time. In the ConAg architecture, the second-order links from the motivations to the reactive behaviors responsible for survival (such as Approach-Green) are innate.

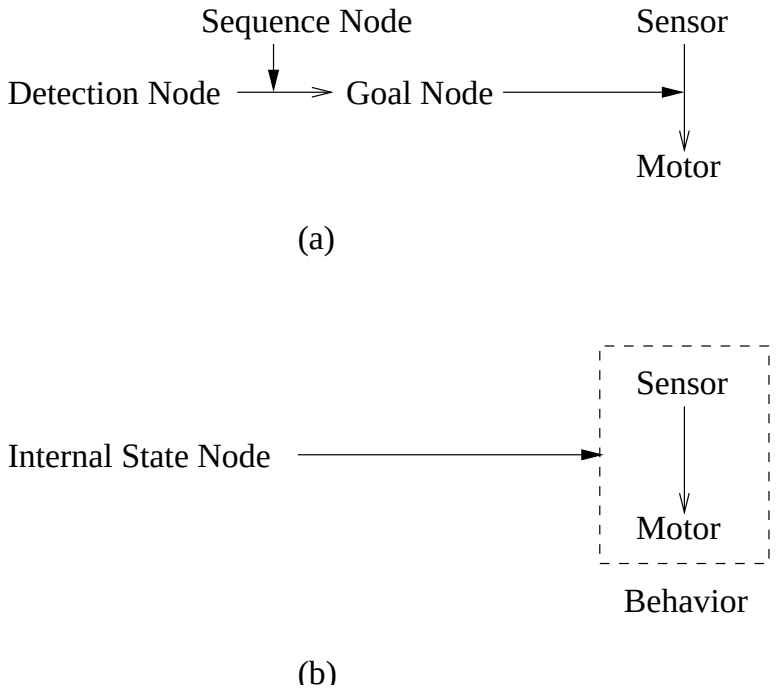


Figure 5.6: Corresponding nodes in (a) [Crabbe and Dyer, 1999a] and (b) ConAg architectures.

This enables a ConAg agent to learn the construction sequence while carrying out its survival behaviors.

The MAXSON architecture was extended to the MAXSON-VI architecture that can learn sequences of goals through imitation [Crabbe and Dyer, 2000]. Teacher agents in the environment are already endowed with the ability to respond correctly to the different colored discs. Learner agents observe the actions of these teacher agents and consider the teacher's actions as reinforcement in adjusting their internal sequence networks so that after learning is complete, the learner will replicate the teacher's responses. One of the main issues in imitation learning is the *perspective* problem or associating the teacher's actions (sensed through the learner's sensors) to the learner's actions [Crabbe and Dyer, 2000]. The MAXSON-VI architecture solves this problem by having dedicated nodes within the learner that can detect the teacher's actions and state and linking these special nodes with the corresponding motor or detection node (figure 5.8). For instance, there is a link from the "Teacher-Eat" node to the learner's "Eat" motor node. In this way, a teacher's eating is equivalent to the student performing the same action. However, this approach works only for those actions that can be unambiguously distinguished by the sensors. For example, it is reasonable to expect the sensors to detect another agent eating but it is difficult for the sensors to determine to which particular disc an agent is moving to (and hence there can be no "Teacher-Approach-Red" sensor node). Thus, a MAXSON-VI agent can learn only during the detection of these distinguishable *interaction* events (such as "eating").

In a ConAg agent, all links between sensors and motors are encapsulated within behaviors and an agent's motor output at every time-step is the output of exactly one behavior. Thus, the actions of the teacher and the learner can be compared directly by checking if their respective actions agree or not (i.e, Teacher Detection nodes are not required in the learner). This comparison of actions then provides reinforcement at every step which is used for sequence learning (unlike learning only during interaction events as in MAXSON-VI). However, the ConAg learning mechanism assumes that the teacher has the same motivations and sensor information as the learner at every instant. This approach of the "learner riding on the teacher's back" so that they share the same sensor information has been used in other imitation learning work [Cecconi *et al.*, 1995]. Moreover, the sequence learning in MAXSON-VI is "one-shot", i.e, one observable sequence of a teacher with a particular colored disc is sufficient for the student to learn the appropriate response. This is possible because the agent always recruits an unused goal node for every new interaction. In ConAg,

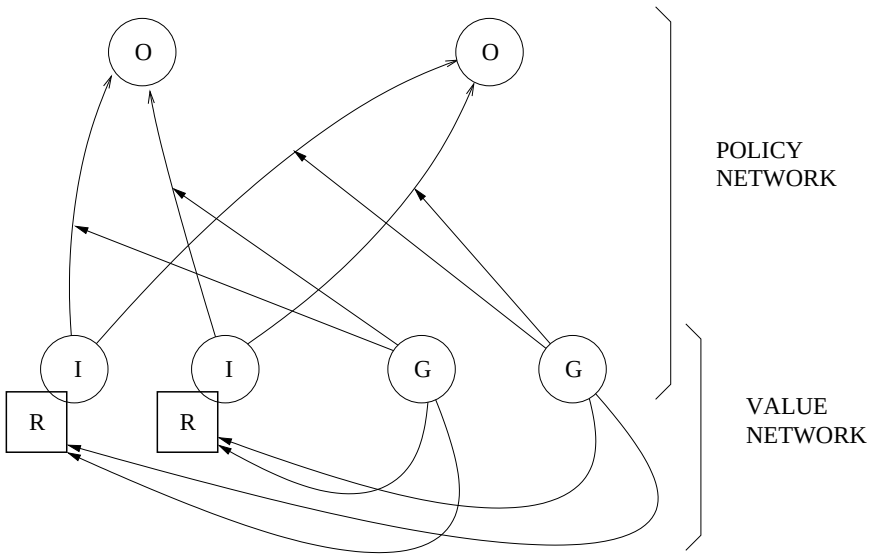


Figure 5.7: The MAXSON architecture [Crabbe and Dyer, 2001]. The architecture contains two networks: the Policy network that activates motor nodes from sensor activations through second-order connections and the Value network that generates reinforcement for the Policy network Crabbe and Dyer.

multiple training instances are required before the student can learn the entire sequence. However, ConAg does not recruit any new nodes which is more biologically plausible. Also, in MAXSON-VI, the learner learns the sequence of *goals*, not just the sequence of actions as there are nodes that explicitly corresponds to every goal. In ConAg, there is no explicit representation of goals. The internal state nodes and behaviors are present innately and the agent learns sequences of actions by changing the weights between these nodes.

Imitation has been used elsewhere as a means of sequence learning especially to learn fixed trajectories in robots [Hayes and Demiris, 1994; Dautenhahn, 1995; Gaussier *et al.*, 1998]. The Per-Ac [Gaussier and Zrehen, 1995] architecture that was briefly described in chapter 3 was extended to facilitate imitation learning [Moga and Gaussier, 1999]. Bakker and Kuniyoshi give a review of imitation learning in robots [1996].

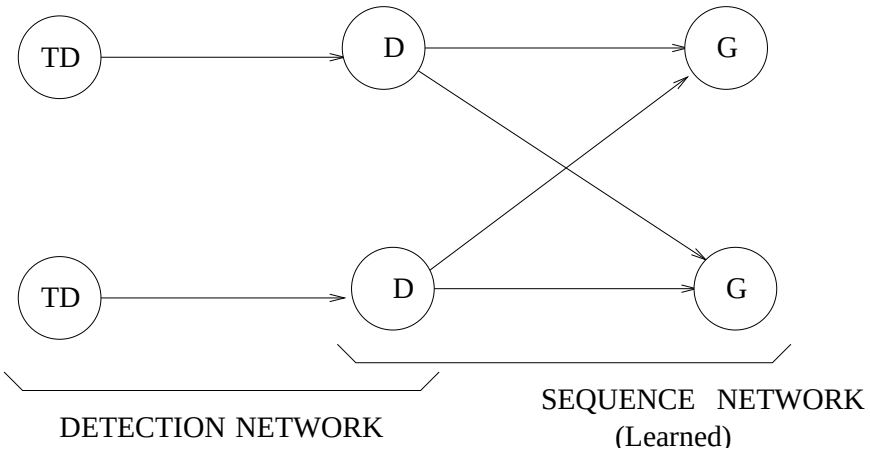


Figure 5.8: Linking teacher's and learner's actions in the MAXSON-VI architecture [Crabbe and Dyer, 2000]. The Teacher Detection (TD) nodes in the learner detect *interaction* events (for example, "eating") performed by the teacher. There are innate one-to-one links between these TD nodes and the corresponding Detection (D) nodes. Thus, actions performed by the teacher can be used as reinforcement to adjust the weights in the Sequence network (which set the Goal (G) nodes).

Chapter 6

Learning Social Rules

6.1 Introduction

The ConAg architecture described in chapter 3 was designed for the single agent environment. There is no mechanism to detect or predict the locations of other agents in the environment. In this situation, individual paths could interfere with one another and in the extreme case, there could arise a deadlock between two or more agents. For instance, an agent tries to move to a location occupied by another agent. If the other agent is similarly trying to move to the location of the first agent then a deadlock occurs. This is illustrated in figure 6.1. Agent B (holding a brick) is trying to move to the drop site at the left. At the same time, agent A is trying to get to the source of bricks at the right. Since the paths that each agent is trying to follow to get to its goal cross each other, both the agents get into a deadlock. Such a situation can be common in narrow passageways where the chance of encountering another agent is higher.

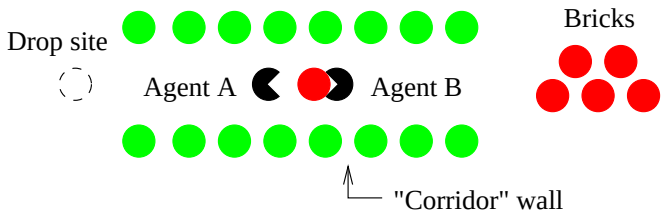


Figure 6.1: An example of a deadlock: Each agent is trying to move across the other to reach the other side (to pick a brick or to drop the brick that it is holding). This has resulted in a deadlock in the narrow corridor.

A solution to this situation is for agent B to drop its brick at its current location for agent A to pick up. Agent A can then go to the left to drop it off and B can move to the right to pick up another brick. If more than one agent is involved in a deadlock then this strategy will lead to a "bucket brigade". If more than two agents are trapped in a narrow "corridor", the agents will exhibit a bucket brigade behavior effectively moving bricks from one point to another without the agents themselves moving from their positions.

In this chapter, a learning mechanism is introduced to the ConAg architecture, resulting in the "ConAg-DL" architecture. The ConAg-DL architecture can be used in the multi-agent scenario *without* extending the sensory capabilities of the agents. The learning algorithm enables the agents to learn to drop any "brick" being carried in case of a deadlock so that the other agent can pick it up and replan its path, thus breaking the deadlock. The learning is unsupervised as an agent uses the distance that it has moved since a deadlock was detected to provide a reinforcement signal.

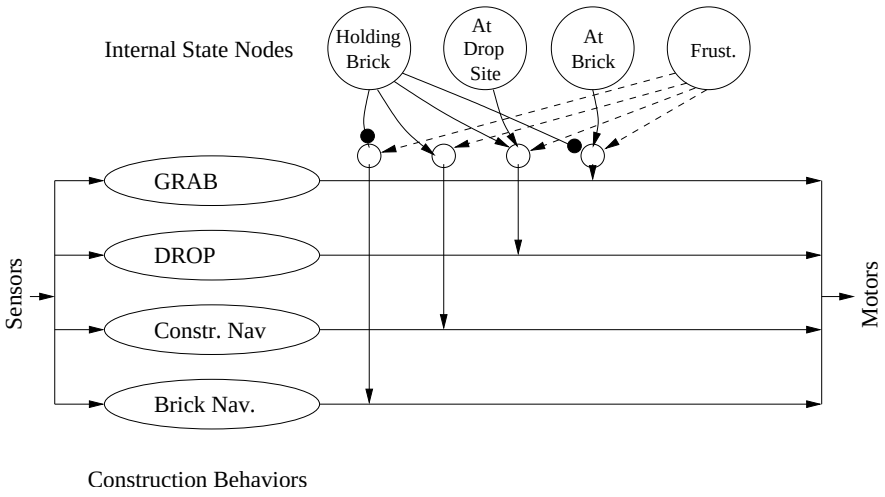


Figure 6.2: New "frustration" internal state node regulates the four behaviors required for construction along with the Holding-Brick, At-Drop-Site, and At-Brick state nodes. The dashed lines indicate the newly added links.

6.2 Learning to Break Deadlocks

The idea is to have a new internal state node that measures the "frustration" of the agent and which will trigger the learning phase. The modified action selection part of the architecture is shown in figure 6.2. Only the navigational planning, grab, and drop behaviors and the Holding-Brick, At-Drop-Site, and At-Brick internal state nodes that are essential for construction are used. If the agent is unable to move in a time-step, the activation on the Frustration node increases. When this activation exceeds a threshold, then the agent tries to perform a random behavior (grab, drop, move toward brick, move toward drop site). Like all other internal state nodes, there are weighted links between the Frustration node and the construction behaviors. If the agent is able to get out of the deadlock, then the weights of the connections are changed to reinforce this action. If the agent remains deadlocked, then that action is penalized. Since the action selection network is a single layer network, the Perceptron learning rule can be used.

Let I denote the set of internal state nodes and a_i the activation on internal state node $i \in I$. Let $f(t)$ denote the amount of frustration at time t and T^F represent the threshold of frustration that triggers a random behavior to be performed. Let a_F denote the activation of the Frustration internal state node. Then,

$$a_F = \begin{cases} 1, & f(t) \geq T^F \\ 0, & f(t) < T^F \end{cases} \tag{6.1}$$

Let B denote the set of possible behaviors and let $w_{ij}(t)$ denote the weight on the link from internal state node $i \in I$ to behavior node $j \in B$ at time t. Then, the activation of behavior node $j \in B$, a_j, at time t is set by the state nodes (analogous to equation 5.1):

$$a_j = \psi(\sum_{i \in I} w_{ij}(t) a_i, T^B) \tag{6.2}$$

where $\psi()$ is the step function defined in equation 5.2 and T^B is the threshold of behavior activation. As in the ConAg architecture, these behavior activations are sent to the action selection module which gives higher priority to the grab and drop behaviors over the navigation planning behaviors.

Let $\vec{d}(t)$ represent the position of the agent at time t. The algorithm, *Frustration_Learning*, to learn to escape from deadlocks is given in figure 6.3. The $sign()$ function is defined as:

$$sign(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{6.3}$$

Algorithm *Frustration_Learning*

1: At time t: if $(a_F = 1)$
2: then perform a behavior $b \in B$ randomly with probability p_b
3: set $f(t+1) \leftarrow 0$
4: At time $(t+\delta)$: if $|\vec{d}(t) - \vec{d}(t+\delta)| > d$
5: then $w_{ib}(t+1) = w_{ib}(t) + sign(1-a_b)\eta^F a_i e^{-kw_{ib}^2(t)}$
6: $w_{ij}(t+1) = w_{ij}(t) + sign(0-a_j)\eta^F a_i e^{-kw_{ij}^2(t)}$, $j \in B, j \neq b$
7: else $w_{ib}(t+1) = w_{ib}(t) + sign(0-a_b)\eta^F a_i e^{-kw_{ib}^2(t)}$

Figure 6.3: Algorithm *learnFrustrated* to learn to escape from deadlocks.

In step 2 of algorithm Frustration_Learning (figure 6.3), a behavior b is randomly selected with probability p_b. Once a random behavior is performed, the frustration is reset to 0 (line 3). To reduce the chance that both agents are taking random actions at the same time, an agent can choose not to perform any behavior at all in step 2 (i.e, an agent does not perform any behavior with probability $1 - \sum_{b \in B} p_b$). This is useful because if both agents perform a random behavior at the same time and the deadlock gets broken, one of the agents will erroneously reinforce its action though the deadlock was resolved by the other agent's action. Line 4 is the test of whether this random action was successful. This test is carried out at some later time $t+\delta$ by checking if the change in the agent's position is greater than some distance d. δ is the time taken to determine if the deadlock was broken or not (it is not possible to determine if a deadlock has been broken immediately after a behavior was performed since there could be many actions that only take the agent temporarily out of deadlock, such as moving back only to move forward in the next step). However, if δ is large, then the agents will have to wait longer before trying a different random action. This results in a longer time spent in trying to break deadlocks during the learning stage. Lines 5–7 are the learning rules for a successful and unsuccessful behaviors. η^F is the learning rate. $sign()$ indicates the direction in which the weights are adjusted. The exponential term is present to bound the weights $w_{ij}(t)$. Moreover, since the base construction sequence is encoded with weights (set *a priori*) that are close to the threshold, the rate at which these weights change is lower than that of the weights that start at 0.

6.3 Generalizing Learned Behaviors

Applying the learning algorithm described in the previous section, an agent learns that if it is holding a brick and is in a deadlock, then it should drop the brick. However, this behavior is triggered *only* when the agent is caught in a deadlock. Since the agent also has a spatial representation of the world around it, it is possible for it to associate the deadlock breaking behavior with the environmental conditions that existed when it entered a deadlock. It could then apply this behavior anytime its spatial map indicates that the world around the agent is similar to this learned pattern, even if the agent is not currently in a deadlock.

For instance, if deadlocks tend to occur in "corridors" (two closely placed parallel rows of bricks), then the agent should learn to drop bricks whenever it finds itself in a narrow passageway (provided that a sufficient number of such examples were available during learning). If this brick is later picked up by another agent, then a mechanism develops whereby agents pass bricks from one to another in constricted spaces (as opposed to trying to pass each other). This behavior is beneficial only if there are at least two agents participating in the constriction task and these agents are of two types: those responsible for moving bricks to narrow corridors and those that move bricks from the corridors to the drop-sites. The agents can automatically divide themselves into these two types of agents due to the nature of the environment: the agents nearer the drop-sites will try to pick up bricks from the corridor (which is closer to it than the original source of bricks) while the agents situated nearer the bricks will move them to the corridor (which is closer to it than the drop-sites).

Since ESMs are used as the spatial representation, the agent does not need to have any pre-defined notion of a "corridor". The egocentric nature of the spatial maps ensures that approximately the same ESM

neurons get activated every time the agent finds itself between two rows of bricks. Thus, the agent only has to learn to associate the pattern of activations on these ESM neurons to deadlocks and then apply the deadlock escape behavior whenever the ESM activations match the learned pattern. The learned pattern is stored in a *Deadlock Pattern Map* (DPM). Let a_i denote the activation of Brick ESM cell i and let d_i be the activation on the corresponding node of the DPM (the activations on the DPM are not shifted with the movement of the agent unlike the ESM activations since the DPM represents the status of the environment around the agent only during deadlocks). A simple update rule to learn the pattern of DPM neurons is given below:

$$\Delta d_i = \eta^D (a_i - d_i) e^{-k d_i{}^2} \tag{6.4}$$

where Δd_i is the change that is added to d_i at the end of every application of this rule. This rule is applied each time the agent's frustration is above the threshold T^F (i.e, it is in a deadlock). η^D is the learning rate and the exponential term (k is a constant) is used to retain values learned from previous time steps (with high d_i).

The pattern stored in the DPM can be matched with the current set of activations on the ESM neurons using the following match function, $match(\underline{d}, \underline{a})$:

$$match(\underline{d}, \underline{a}) = \frac{\sum_i \psi(d_i, T^D) \max_{i' \in nb(i)}(a'_i)}{\sum_i \psi(d_i, T^D)} \tag{6.5}$$

where $\psi()$ is the step function defined in equation 5.2 and T^D is the threshold used to identify those nodes in the DPM that indicate a disc. $nb(i)$ is the set of all neighboring cells of an ESM cell i. The summation is over all cells i in the central portion of the ESM (since discs farther away from the agent are less likely to be a part of any constriction).

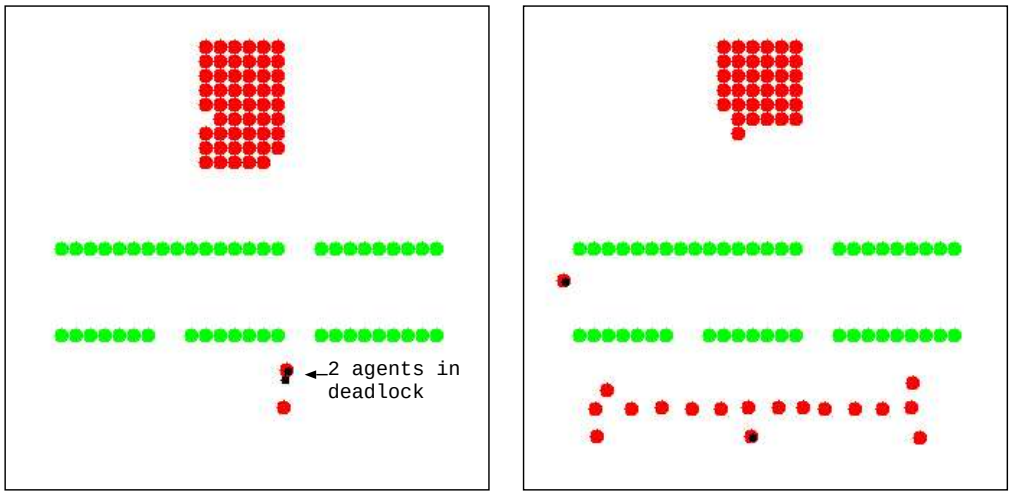
After the agent has learned a deadlock escape behavior, it applies equation 6.5 to match DPM activations ($\underline{d}$) with the ESM activations ($\underline{a}$) at every step. The result of this match is used to set the central node of the Configuration navigation map which encodes the locations where bricks are to be dropped. Thus the agent selects the drop behavior not only when it is at a drop-site specified by the Configuration ESM, but also when the agent is in a location where deadlocks are likely to occur (as indicated by $match(\underline{d}, \underline{a})$).

6.4 Results

The performance of the learning algorithms was analyzed and the effect of varying the parameters is described in this section. The world was restricted to a square of 100 × 100 units. The ESMs are grids consisting of 100 × 100 cells with each cell covering a unit square. The size of the ESM relative to the size of the world does not affect the frustration learning algorithm or the match function provided the ESM is not so small that it cannot represent the pattern of activations that encode a "passageway" or other structure where deadlocks are likely to occur.

The sensor range of an agent corresponds to a square of 30 × 30 units centered at the location of the agent, i.e, the central 30 × 30 square of the ESM is set from the sensor activations in every step (a square sensor range was used instead of the circular sensor range as in the ConAg architecture solely for ease of implementation— none of the behaviors that are used in ConAg-DL are reactive and hence they are relatively insensitive to the shape and size of the sensor range). An agent can move up to 0.5 units in one time-step (slows down to drop bricks).

Agents are initially set at random locations within the world. The ESMs were initialized with the positions of the discs around each agent to remove the need for an exploration phase (the ESMs get updated normally during construction and learning). Moreover, the deadlock escape mechanism of dropping bricks to be picked up by the other agent in the deadlock works only if the location of the closest drop-site/brick is the same in both the agents' ESMs (otherwise, two agents both trying to move toward different brick sources can get into a deadlock which cannot be broken by the actions of either agent). Initializing the ESM with the initial positions of the discs ensures that different agents carry out their path planning on similar representations of the world when they are caught in a deadlock. The initial location of the discs in the world is shown in figure 6.4(a) and the world after the structure has been built is in figure 6.4(b).



(a) (b)

Figure 6.4: Environment for learning deadlock breaking behavior: The world (a) near the beginning of construction and learning. The two agents are in a deadlock and are performing random behaviors. (b) after construction is complete.

Table 6.1: Initial weights of links

INITIAL	Grab	Drop	Config. Nav.	Brick Nav.
Holding-Brick	-0.6	0.6	0.6	-0.6
At-Drop-Site	0	0.6	-0.6	0
At-Brick	0.6	0	0	-0.6
Frustration	0	0	0	0

The initial weights of the links in the Action Selection module responsible for sequencing the construction task are shown in table 6.1. Though there are many values of weights that capture the sequence of behaviors for construction, values close to the threshold ($T^B = 0.7$) were chosen so that the pre-defined construction sequence that the agent performs when not frustrated would not be unlearned. The activations of Holding-Brick, At-Drop-Site, and At-Brick internal state nodes are ± 1, except for Frustration which is 0 or 1. The weights from the "frustration" internal state node to the output nodes are initially zero.

6.4.1 Two learning agents

Two agents were released in the world to simultaneously learn an appropriate response to deadlocks. The weights at the end of one learning run is shown in table 6.2. The learning rate η^F was kept constant at 0.05. The agent retains the normal construction sequence when the *frustration* state node is inactive. When *frustration* is active and the agent is holding a brick, it will drop it even if it is not at a drop site (indicated by increasing $w_{Frust,Drop}$ in column 2 of table 6.2). If the agent is not holding a brick and is "frustrated", then it will not choose any behavior (indicated by decreasing $w_{Frust,BrickNav}$ in column 4 of table 6.2).

The evolution of the weights from the Frustration state node to each of the four behaviors over time is shown in figure 6.5. The time in the data includes that spent in performing construction though learning occurs only during deadlocks. When choosing a random behavior, an agent selects one of the four behaviors with probability 0.1 (i.e, it will not select any behavior with probability 0.6). This conservative choice is to

Table 6.2: Final weights of links

FINAL	Grab	Drop	Config. Nav.	Brick Nav.
Holding-Brick	-0.598	0.613	0.580	-0.591
At-Drop-Site	-0.05	0.585	-0.590	0.251
At-Brick	0.598	-0.344	0.233	-0.591
Frustration	0.05	0.344	-0.284	-0.251

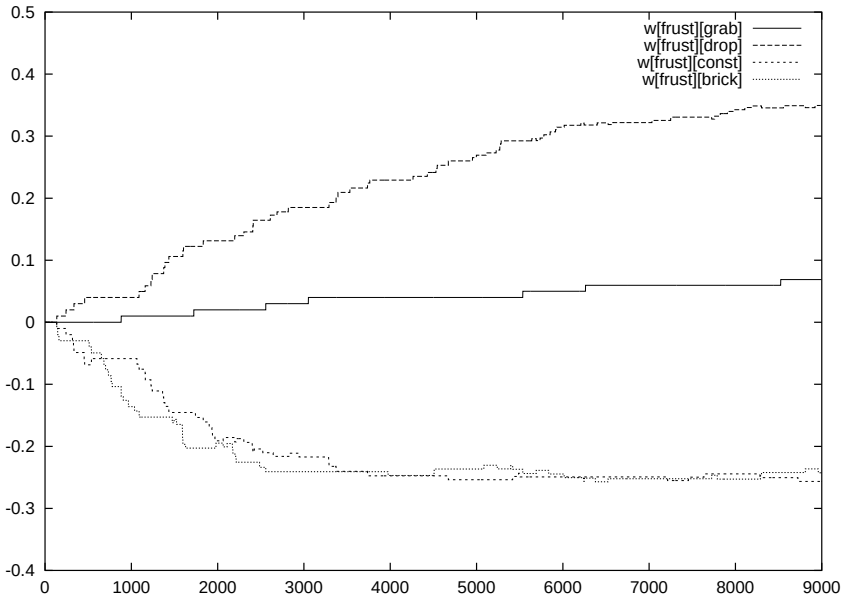


Figure 6.5: The weights from Frustration state node to the four output behaviors when there are two agents in the environment (averaged over 5 trials).

reduce the chance that an agent erroneously reinforces its behavior when it was another agent's behavior that broke the deadlock. As expected, the weight on the link from Frustration to the Drop behavior is the highest because performing the other behaviors in a deadlock will not break the deadlock always and thus the corresponding weights will not be reinforced.

Figure 6.6 shows the case when an agent chose each behavior with probability 0.25 when it had to perform a random behavior. Since less time is now spent performing no behavior, learning occurs faster. However, the weight $w_{Frust,Grab}$ has also increased since an agent is now likely to reinforce its selected behavior even when it did not contribute to breaking the deadlock.

6.4.2 Five learning agents

Figure 6.7(a) shows the same learning algorithm performed with five agents in the same environment. The weights learned are similar to that of the two agent case. This is because the environment is relatively unconstrained except for the short corridor and most deadlocks involve only two agents. Figure 6.7(b) shows the weights when five agents learned in a more constrained environment (shown in figure 6.9). In this case, more than two agents are often involved in deadlocks and the time to break such deadlocks will depend on the number of agents in it. Since the learning algorithm tests the progress of the agent after a fixed number of steps, the proper reinforcement will not be given even if the agent has performed the correct behavior. Therefore, the learning occurs slower than in unconstrained environments.

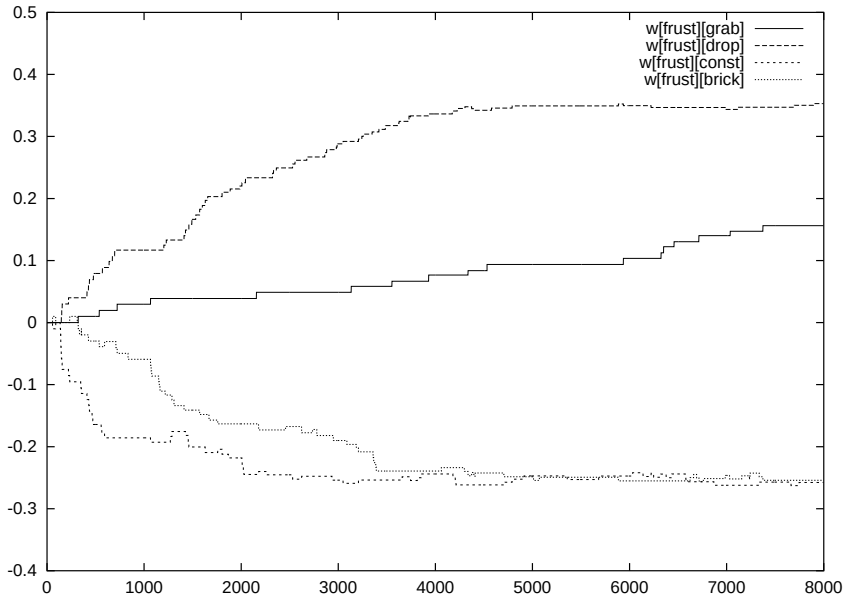


Figure 6.6: The weights from Frustration state node to the four output behaviors when each behavior is chosen with probability 0.25 when a random behavior has to be performed (averaged over 5 trials with two learning agents).

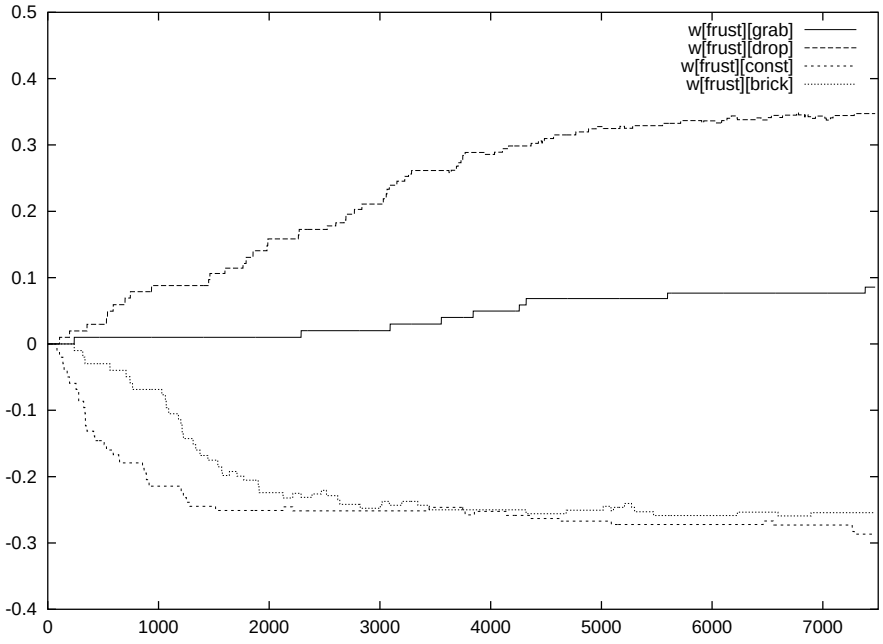
6.4.3 A learner and a previously trained agent

In the next experiment, an agent was released into the environment along with an agent that was already trained as described above. The results are shown in figure 6.8. The weight $w_{Frust,Drop}$ increases, but $w_{Frust,BrickNav}$ does not decrease as before. As a result, the agent learns to drop its brick when frustrated and holding a brick, but continues to attempt the brick navigation behavior when it is not holding a brick and deadlocked. This is because the already trained agent immediately performs the Drop behavior when it is holding a brick, and therefore does not give the learning agent an opportunity to explore its choice of behaviors (the learning agent continued to perform the pre-defined brick navigation behavior in this circumstance).

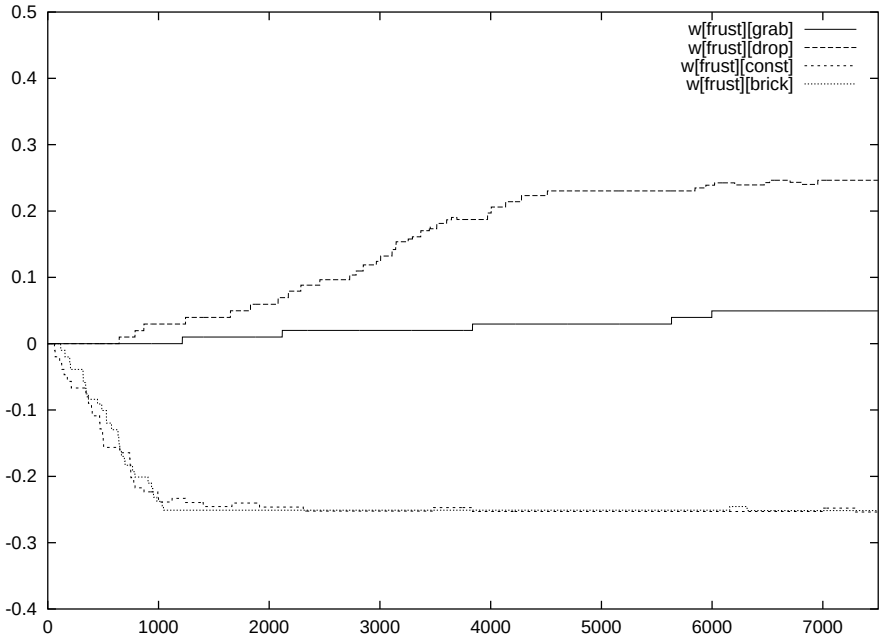
6.4.4 Bucket Brigading behavior

To study if the behavior that was learned to get out of deadlocks can exhibit a "bucket brigade" among many agents, the following experiment was conducted. Five agents, that were trained in pairs to exhibit the deadlock escape mechanism, were placed in an environment where the source of bricks and drop sites were separated by a long "corridor". Figure 6.9 shows the five agents deadlocked in the corridor ($t = 270$) with the three agents holding the bricks (A, B, and C) at the north end of the corridor trying to move south, while the other agents are trying to move north to reach the source of bricks. The Frustration state node of B is active which causes it to drop its brick to be picked up by D ($t = 272$). B and D have now changed their goals and are free to move north and south respectively. Meanwhile C is in a deadlock with E and A is deadlocked with C. C drops its brick which is picked up by E ($t = 276$). In the next few time-steps, the deadlock between A and C is also resolved similarly ($t = 280$).

Disc exchanges can take place simultaneously at more than one point in a long corridor. This is exhibited in the run shown in figure 6.10. Four agents are involved in moving red discs from the top to the bottom of the corridor whose walls are made of green discs. At $t = 5$, there are two deadlocks (between agents A and B, and between C and D). At $t = 8$, the deadlock between A and B is broken (A picks the disc dropped by B) and at $t = 9$, the deadlock between C and D is similarly broken. At $t = 13$, a new deadlock develops



(a)



(b)

Figure 6.7: The weights from *frustration* state node to the four output behaviors when there are five agents (averaged over 5 trials) learning in (a) an unconstrained environment and (b) a constrained environment.

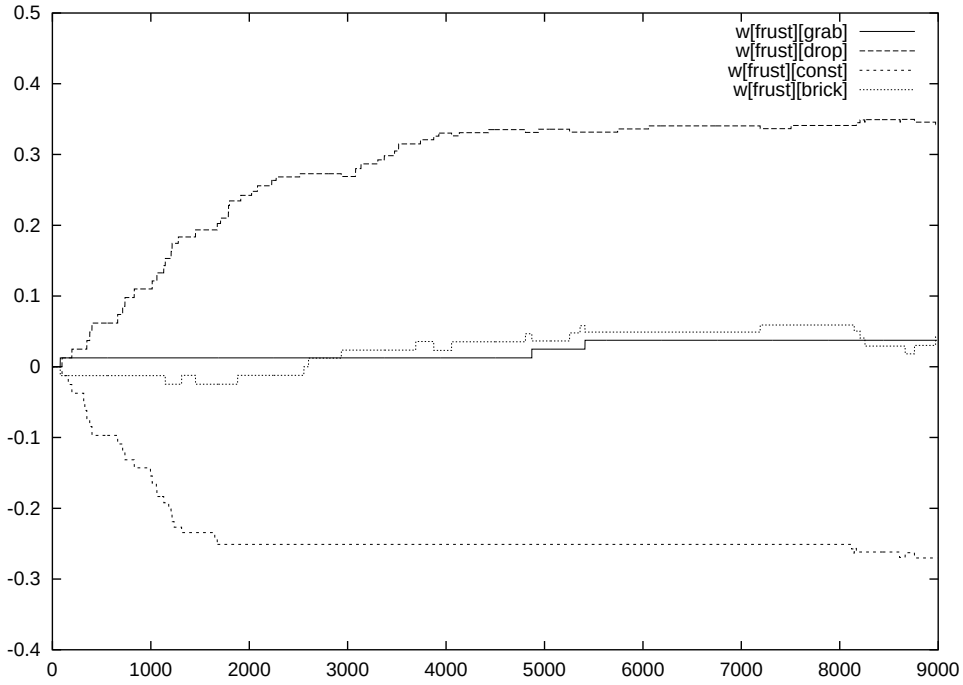


Figure 6.8: The weights from Frustration state node to the four output behaviors when only one of the agents is learning and the other has already been trained (averaged over 5 trials).

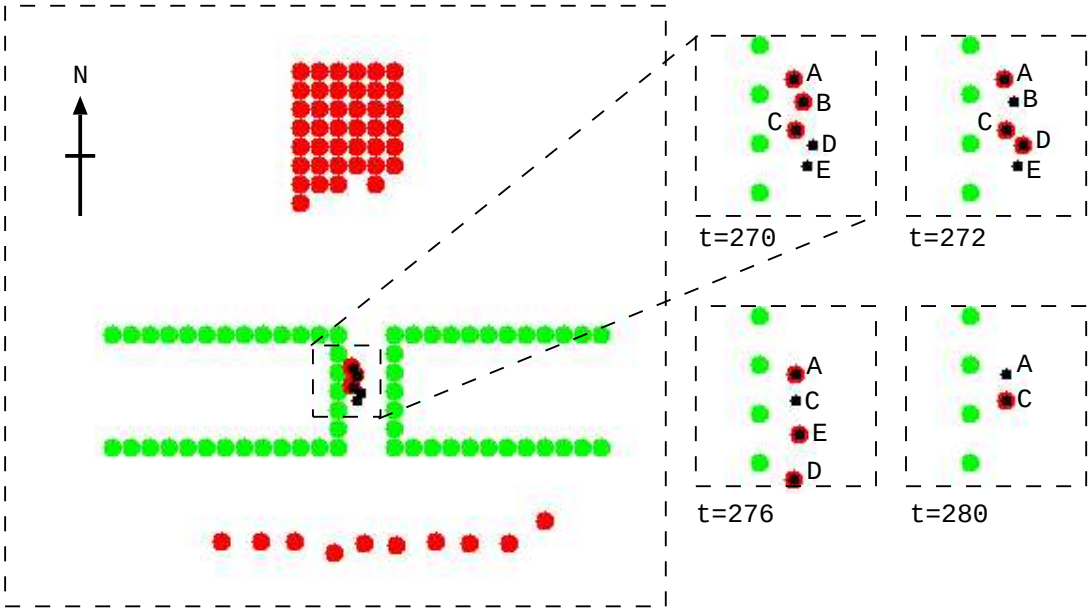


Figure 6.9: "Bucket brigade" behavior exhibited by 5 agents (A, B, C, D, and E) confined in a long corridor made of green discs. Agents are indicated by small black circles; a black circle within a red disc indicate that the agent is carrying a disc. The sequence of drops and pick ups that eventually move the three bricks down the corridor is shown to the right.

between B and C (agents A and D are free to move to drop and pick a disc respectively). This deadlock is broken (not shown) but A and D get into two separate deadlocks at $t = 24$. The deadlock between C and D is broken at $t = 31$ (allowing D to move to the top to pick up another disc) but C joins the deadlock between A and B at $t = 33$. B drops its disc at $t = 34$ to be picked up by A which then moves to drop it off (B and C continue to be in a deadlock). Notice that during this time only agent A moved to drop bricks at drop-sites and only agent D moved to pick up discs from the initial locations at the top. Thus, the deadlocks between agents ensured that the agents were confined in their own "territory" within the corridor while still enabling the transport of discs through the corridor. Though this reduced the total distance traveled by an agent (agents did not have to move around one another as would have been necessary if the disc passing mechanism were not available), the time taken to move a disc from its initial location to a drop-site is long since most of the time is spent waiting for the frustration level to increase to the point where a deadlock is detected and the deadlock escape behavior is triggered.

6.4.5 Map association

In the environment in figure 6.11(a), most deadlocks occur within the narrow corridor. Figure 6.11(b) shows DPM readings learned by an agent after 10,000 steps (learned only when "frustrated", i.e, $a_F = 1$). Two agents were then released into the environment of which one had the learned DPM pattern. This agent moved bricks from the source and dropped them in the shaded region in figure 6.11(a) (those locations its sensor readings matches the learned activation pattern). The other agent picks up these bricks from the shaded region since it is closer to the structure being built than the source of bricks. Thus, the agents do not try to cross each other in the narrow corridor. As discussed in section 6.3, at least two agents are necessary to exhibit this behavior. Since ConAg-DL agents do not have the ability to detect other agents, the designer has to ensure that there are at least two agents in the environment.

6.5 Increasing the number of Behaviors

The ConAg-DL architecture that was used for the experiments described earlier contained four behaviors and there was only one action (Drop) that could be performed that would break the deadlock. However, if one of the agents that is in a deadlock has a goal that cannot be satisfied by the picking up of a brick that was dropped by the other agent, then the deadlock cannot be broken by performing the drop behavior. This is the case when agents are allowed to become thirsty and blue (water) discs are also present in the environment. For instance, consider two agents, neither of which are holding a brick but one is thirsty. In this case, the thirsty agent will try to move toward the nearest water disc while the other agent tries to move toward the nearest brick. If their paths cross one another, these agents will enter in to a deadlock which cannot be broken by dropping a brick (since neither agent is even holding a brick).

To study how the ConAg-DL architecture can be extended to take into account situations where agents might be trying to satisfy unrelated goals, two new behaviors, "LieDown" and "ClimbOver" are introduced. Performing the LieDown behavior causes an agent to crouch in its place. Performing the ClimbOver behavior causes an agent to move past an agent in front of it, provided the other agent is crouching (i.e, performing the LieDown behavior). These behaviors do not need input from the sensors. If an agent performs these behaviors in the absence of another agent, there is no change in position. Since agents in a deadlock face each other, a deadlock can be broken if one of the agents performs the LieDown behavior and the other performs the ClimbOver behavior simultaneously. The agents no longer block each other and they can continue on their planned paths. If both agents perform the same behavior simultaneously (both crouch, or both try to climb over each other), the deadlock remains in place. Also, if one of the agents performs these behaviors while the other agent remains stationary, the deadlock is still not broken These cases are illustrated in figure 6.12. The two new behaviors are contrived behaviors since it is not physically possible to crouch or climb over in a 2-dimensional world. Thus, the simulation environment explicitly shifts the agents' positions if these behaviors are used to break a deadlock.

The environment in which agents are placed to learn the use of LieDown and ClimbOver behaviors is shown in figure 6.13. The water discs are spread throughout the environment while the source of bricks and the location of the structure to be built is at opposite ends of the corridor made of green discs. In such an

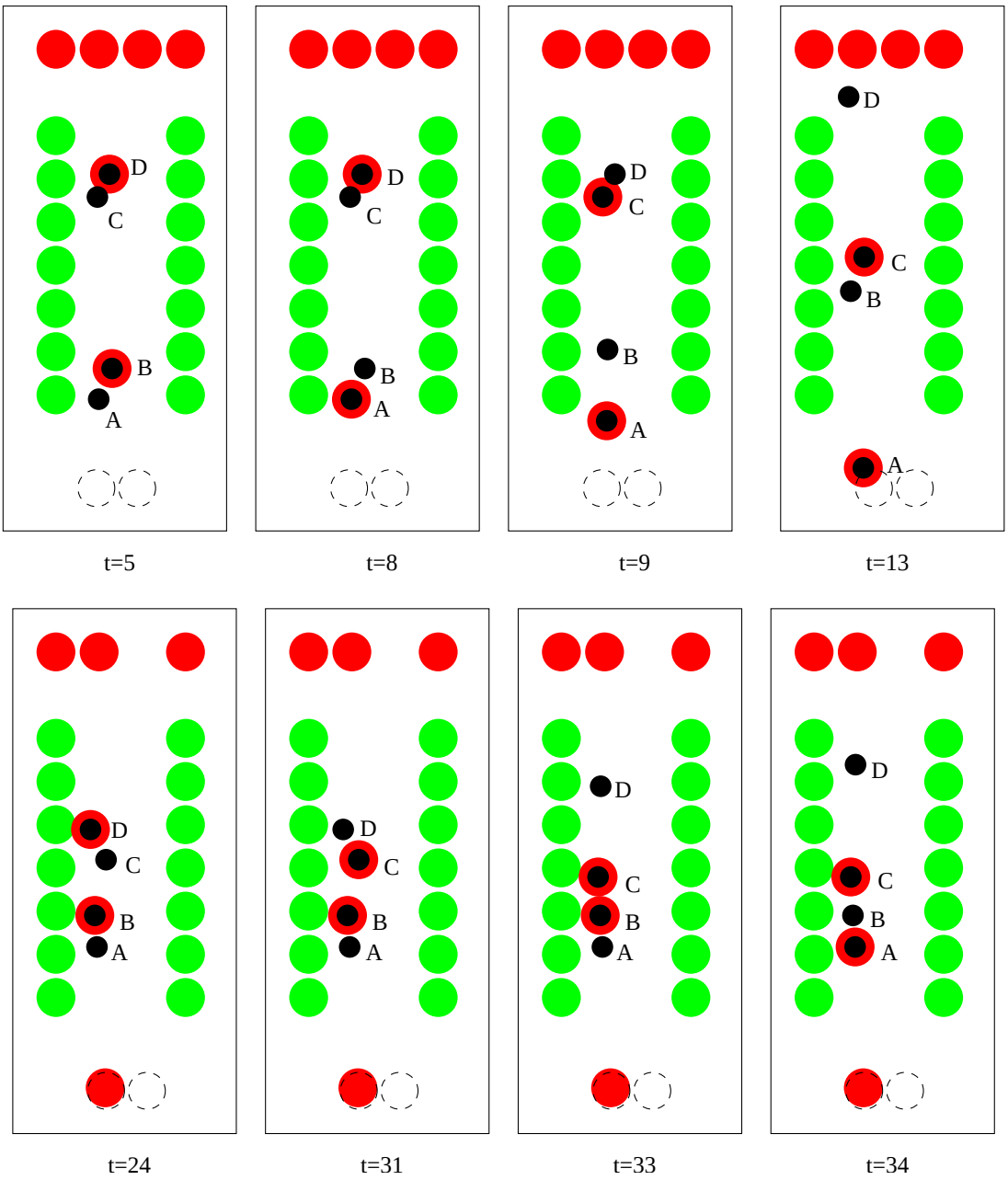
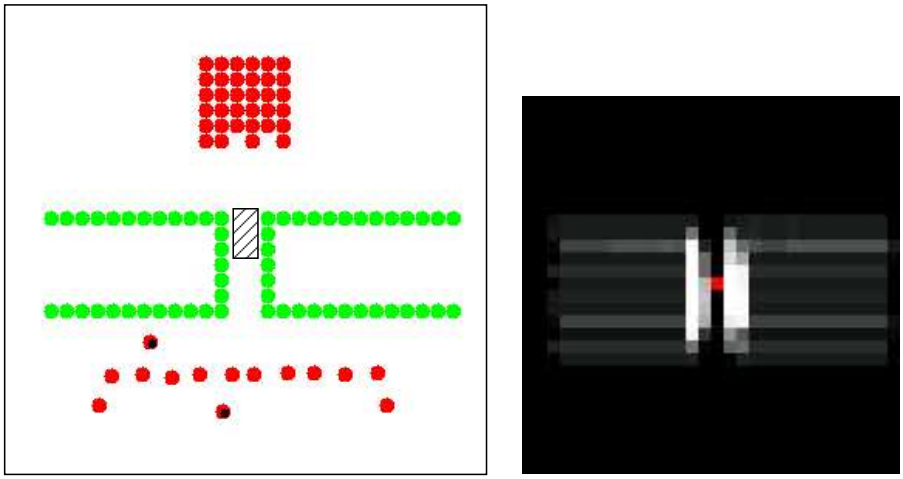


Figure 6.10: Disc exchange at more than one point: Four agents (small dark circles) A, B, C, and D are confined in a corridor made of green walls. The dashed circles indicate drop-sites. A black disc within a red disc indicates that that red disc is being carried by an agent. The positions of the discs and agents are shown at times $t = 5, 8, 9, 13, 24, 31, 33, 34$. Two simultaneous deadlocks are visible at $t = 5$ and $t = 24$.



(a) (b)

Figure 6.11: (a) Environment with narrow corridor used for associating sensor readings with deadlocks. The shaded area indicates the locations where sensor inputs will match the learned readings. The source of bricks are at the top and the built structure is at the bottom. (b) The DPM activations after 10,000 steps.

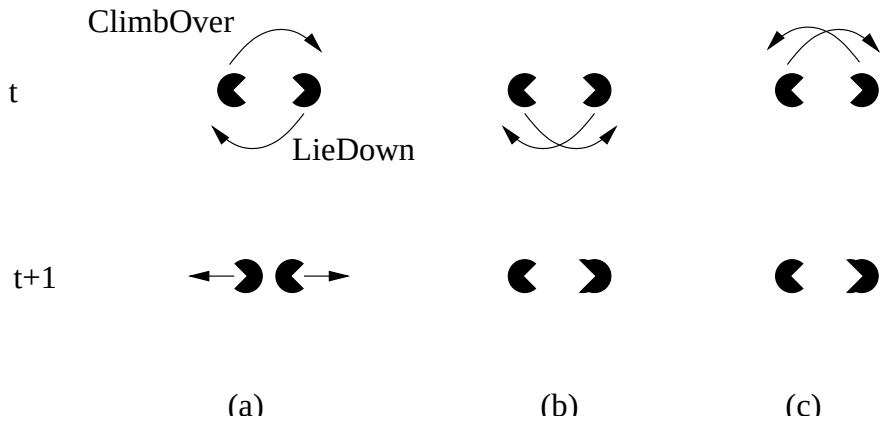


Figure 6.12: Breaking deadlocks with LieDown and ClimbOver behaviors. In figure a, one of the agents performs the LieDown behavior while the other performs the ClimbOver behavior and the deadlock is broken. In figures b and c, both agents perform the same behavior simultaneously and the deadlock continues.

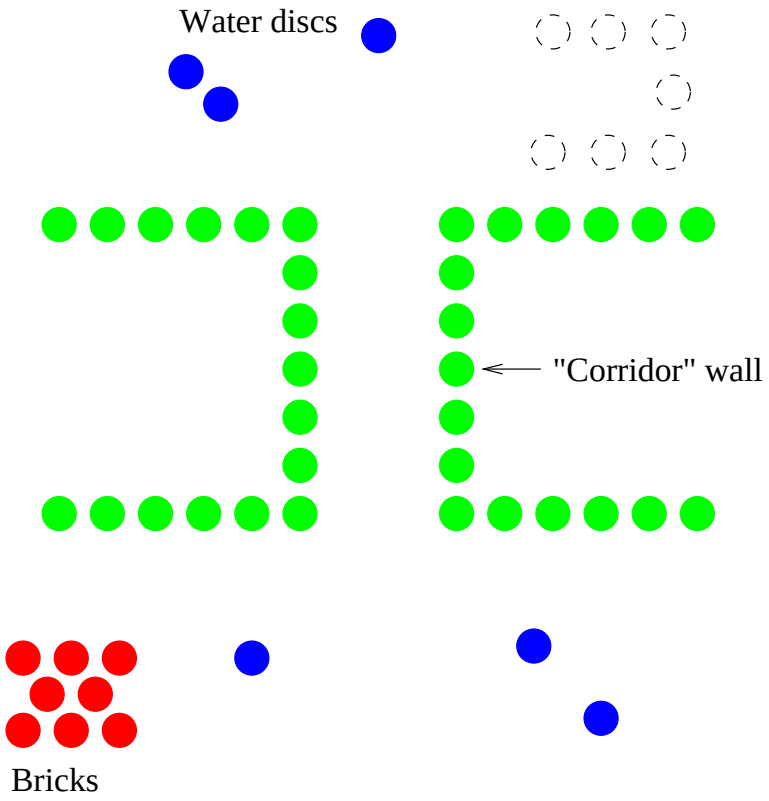


Figure 6.13: Environment used for learning LieDown and ClimbOver behaviors. Dashed circles indicate the location where bricks are to be placed. Water (blue) discs are scattered throughout the environment.

environment, agents can be moving to either a water disc (when thirsty), a brick (when it is not holding a brick) or a drop-site (when it is holding a brick). Two agents that are trying to move toward different types of bricks can get into a deadlock if their paths cross. Some of these deadlocks can be broken by one agent dropping a brick and the other picking it up (in the case of agents moving toward bricks and drop-sites), while others can be broken only with coordinated execution of the LieDown and ClimbOver behaviors. Table 6.3 lists the different behaviors that can be performed by two agents in different kinds of deadlocks. Each cell lists the possible pairs of simultaneous behaviors (Crouch and Climb represent LieDown and ClimbOver respectively) that will break the deadlock for different internal states of the agents (T, B, $\overline{T}$, and $\overline{B}$ indicate if the agent is thirsty, holding brick, not thirsty, and not holding brick respectively). Note that performing the LieDown and ClimbOver behaviors will break any deadlock including those between agents trying to move toward drop-sites and bricks (where the Drop behavior can break deadlocks as described earlier). Deadlocks are not possible between two agents moving toward the same type of disc or drop-site location. These cases are indicated by an "x" in table 6.3.

The agent has to learn to apply the correct deadlock breaking behavior depending on its current internal state and also that of the other agent. For this purpose, the agent is given three new internal state nodes:

1. *Thirsty* becomes active only when the agent becomes thirsty (as described in chapter 2).
2. *OtherAgentThirsty* becomes active only when the other agent in a deadlock is thirsty.
3. *OtherHoldingBrick* becomes active only when the other agent in a deadlock is holding a brick.

The OtherAgentThirsty and OtherHoldingBrick state nodes are inactive when the agent is not in a deadlock. One can think of these state nodes as "touch" sensors that can sense the state of another agent only when it is extremely close.

Table 6.3: Simultaneous behaviors that can break deadlocks

Agent 1	Agent 2			
	$\overline{T}\overline{B}$	$\overline{T}B$	$T\overline{B}$	TB
$\overline{T}\overline{B}$	x	Grab, Drop Crouch, Climb Climb, Crouch	Crouch, Climb Climb, Crouch	Grab, Drop Crouch, Climb Climb, Crouch
$\overline{T}B$	Drop, Grab Crouch, Climb Climb, Crouch	x	Drop, Grab Crouch, Climb Climb, Crouch	Crouch, Climb Climb, Crouch
$T\overline{B}$	Crouch, Climb Climb, Crouch	Grab, Drop Crouch, Climb Climb, Crouch	x	x
TB	Drop, Grab Crouch, Climb Climb, Crouch	Crouch, Climb Climb, Crouch	x	x

The agent has to use its OtherAgentThirsty and OtherHoldingBrick internal state nodes and LieDown and ClimbOver behaviors only when it is caught in a deadlock. Therefore, a separate network ("Frustration Network") is used to learn which behaviors are to be performed when in a deadlock depending on its internal state. The modified ConAg-DL architecture is shown in figure 6.14. The "Construction Network" is a single layer network that enables the internal state nodes to control the behaviors using second-order links from its outputs to the behaviors. The weights on this network are innate and are used to perform the construction sequence and move toward water when thirsty (as described in chapter 3). Since this sequence is to be performed only when the agent is not in a deadlock, all the outputs from the Construction Network are inhibited by the Frustration internal state node. The outputs of the Frustration Network control the behaviors through second-order links when the agent is in a deadlock and hence the outputs are excited by the Frustration internal state node. The Frustration Network is a single layer network and the weights are learned (described below) to enable the agent to break deadlocks.

After learning, the agents should be able to simultaneously perform one of the pairs of deadlock breaking behaviors listed in table 6.3 depending on the internal states of the two agents. There is no unique behavior that an agent has to perform to escape the deadlock; the behavior that has to be performed is dependent on that performed by the other agent. Thus, the agents have to learn to agree on one pair of behaviors that they will perform for each possible combination of internal state node activations.

To ensure that the agents converge to perform a coordinated set of behaviors, the agents perform both random behaviors and the currently learned behaviors (chosen by the Frustration Network) during deadlocks in the learning phase (as opposed to trying out *only* random behaviors in the Frustration_Learning algorithm of figure 6.3). As the learning phase progresses, the agent reduces the number of times it performs a random behavior when frustrated. This learning algorithm, called *Coordination_Learning*, is shown in figure 6.15. When the agent becomes frustrated (frustration level goes over threshold, line 2), it first tries the behavior output by the Frustration Network (line 5). If this behavior leads to the deadlock being broken at later time $t + \delta$ (line 11), then the weights contributing to this behavior in the Frustration Network is reinforced (lines 12-13). η^F is the learning rate. If however the agent continues to remain in a deadlock, the weights are negatively reinforced (line 16) and the agent performs a random behavior with probability p_r (line 7), or with probability $1 - p_r$, perform the behavior output by the Frustration Network (line 8). The value of p_r determines the number of random trials explored compared to using the output of the Frustration Network. The probability p_r is decreased with decay rate ρ_r in the case of the behavior breaking the deadlock (line 15). Thus, as learning progresses, the learned behaviors are performed more often compared to random behaviors. This process of trying a learned behavior and then a random behavior is repeated until the deadlock is broken. The variable *nAttempts* is used to keep track of the number of attempts (performing

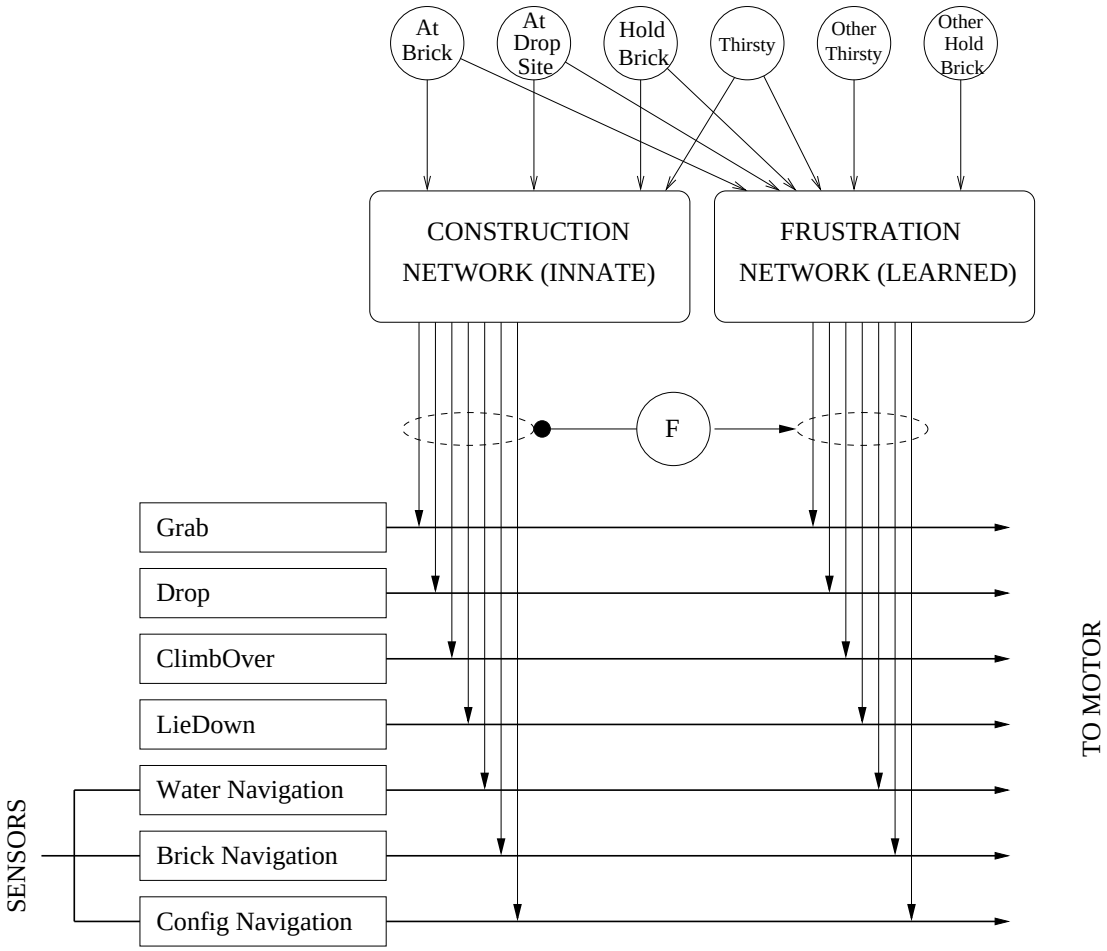


Figure 6.14: Extended ConAg-DL architecture with new behaviors and internal state nodes. The LieDown and ClimbOver behaviors can be used to escape deadlocks if used in a coordinated manner with another agent. The Water Navigation behavior takes the agent toward the nearest water (blue) disc. The two new internal state nodes, OtherAgentThirsty and OtherHoldingBrick, indicate if the other agent in the deadlock is thirsty or holding a brick respectively. The Frustration state node selects the outputs of either the Construction or the Frustration Network to gate the behaviors depending on whether the agent is in a deadlock or not.

both random and learned behaviors) that the agent has made to try to break the deadlock. *nAttempt* is initialized in line 1 and reset after the deadlock has been broken (line 14).

Algorithm *Coordination_Learning*
1: Initialize: nAttempts $\leftarrow 0$
2: At time t: if $(a_F = 1)$
3: nAttempts $\leftarrow$ nAttempts + 1
4: if (nAttempts % 2 = 1) {odd attempts}
5: $b \leftarrow$ behavior output by Frustration Network
6: else {even attempts}
7: with probability p_r, $b \leftarrow$ randomly selected behavior
8: with probability $1 - p_r$, $b \leftarrow$ behavior output by Frustration Network
9: set $f(t+1) \leftarrow 0$
10: perform behavior b
11: At time $(t+\delta)$: if $|\vec{d}(t) - \vec{d}(t+\delta)| > d$
12: then $w_{ib}(t+1) = w_{ib}(t) + sign(1 - a_b)\eta^F a_i$
13: $w_{ij}(t+1) = w_{ij}(t) + sign(0 - a_j)\eta^F a_i,\ j \in B, j \neq b$
14: nAttempts $\leftarrow 0$
15: $p_r \leftarrow \rho_r \times p_r$
16: else $w_{ib}(t+1) = w_{ib}(t) + sign(0 - a_b)\eta^F a_i$

Figure 6.15: Algorithm *Coordination_Learning* to learn to escape from deadlocks.

6.5.1 Results with Two Agents

Two agents were released into the environment shown in figure 6.13. The weights in their Frustration Network are initialized to zero. The initial value of p_r is 0.5 and $\rho_r = 0.99$. The learning rate $\eta^F = 0.2$. Trials were run for 10,000 time-steps. Each trial converges to one of the multiple sets of coordinated actions that the agents can take to escape deadlocks (listed in table 6.3) depending on the random behaviors that were performing during learning. Table 6.4 gives two solutions that were arrived at by the agents from two trials (each cell in the table lists the first and second agent's behaviors for the corresponding internal state). Figure 6.16 shows the number of attempts the first agent made to escape a deadlock at that time-step (*nAttempts* in algorithm Coordination_Learning). After about 6000 time-steps, the agent has learned to escape a deadlock as soon as it is detected. The corresponding decrease in the value of p_r is plotted in figure 6.17.

Learning to Drop and Grab Bricks

In the trials performed above, the agents failed to ever learn to perform the Drop behavior even in those circumstances where dropping a brick to be picked up by the other agent in the deadlock would have broken the deadlock. This is because performing the LieDown and ClimbOver behaviors simultaneously always breaks a deadlock and the Frustration Network generalizes to perform these behaviors even in those situations where dropping a brick would have broken a deadlock. To demonstrate that the Frustration Network can learn to perform different kinds of behaviors depending on the context (internal state of the agent and that of the other agent in the deadlock), the interaction between deadlocked agents was explicitly modified so that simultaneously performing the LieDown and ClimbOver behaviors would break deadlocks only if both agents were holding bricks or both agents were not holding bricks (i.e, if only one agent is holding a brick, then only dropping that brick to be picked up by the other agent can break the deadlock). After this modification in the environment, the Coordination_Learning algorithm was applied to both the agents using the same parameters as in the previous trial. The agents are now able to learn the Grab/Drop behaviors along with the LieDown/ClimbOver behaviors as shown in table 6.5. The complementary Grab behavior is

Table 6.4: Simultaneous behaviors learned using Coordination_Learning

Agent 1	Agent 2			
	$\overline{T}\overline{B}$	$\overline{T}B$	$T\overline{B}$	TB
$\overline{T}\overline{B}$	x	Climb, Crouch	Climb, Crouch	Climb, Crouch
$\overline{T}B$	Crouch, Climb	x	Crouch, Climb	Climb, Crouch
$T\overline{B}$	Crouch, Climb	Crouch, Climb	x	x
TB	Crouch, Climb	Crouch, Climb	x	x

Agent 1	Agent 2			
	$\overline{T}\overline{B}$	$\overline{T}B$	$T\overline{B}$	TB
$\overline{T}\overline{B}$	x	Crouch, Climb	Climb, Crouch	Crouch, Climb
$\overline{T}B$	Climb, Crouch	x	Climb, Crouch	Climb, Crouch
$T\overline{B}$	Crouch, Climb	Crouch, Climb	x	x
TB	Crouch, Climb	Crouch, Climb	x	x

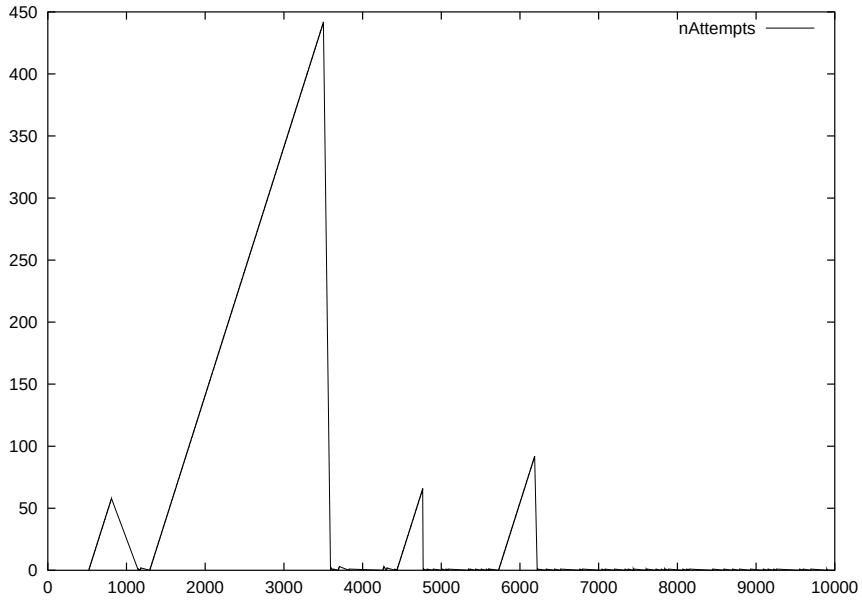


Figure 6.16: Number of attempts the first agent made to escape a deadlock at that time-step (*nAttempts* in algorithm Coordination_Learning).

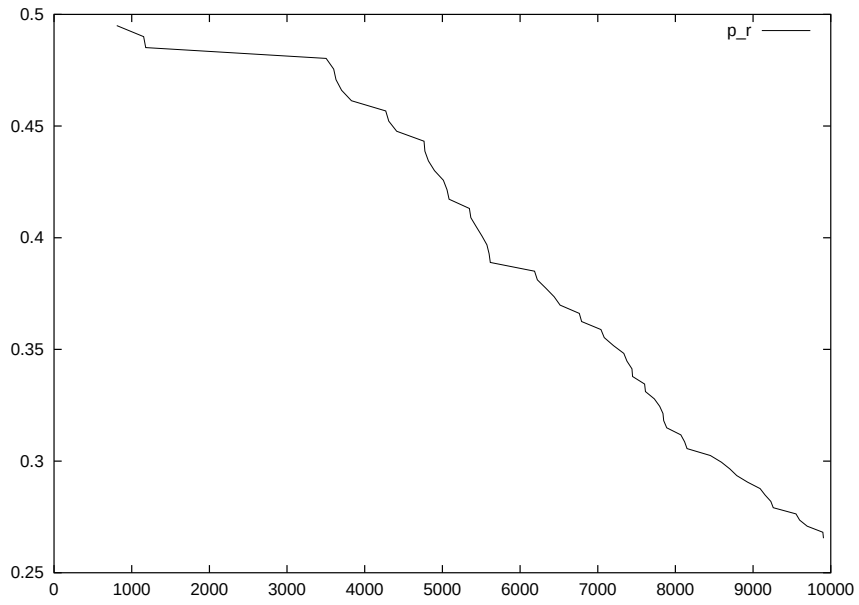


Figure 6.17: Decrease in the value of p_r during Coordination_Learning. Initially, $p_r = 0.5$ and decay rate $\rho_r = 0.99$.

not learned as a response to the Drop behavior because the innate connections in an agent already cause it to grab the dropped disc in the time-step after it is dropped (provided it is not already holding a brick).

6.5.2 Results with Three Agents

When three agents are present in the environment, deadlocks may involve all three agents, i.e, the third agent can be blocked by the first two agents caught in a deadlock. In such a case, any action of the third agent will not contribute to breaking the deadlock (since it is between the other agents). Therefore, all behaviors learned earlier by the third agent are unlearned and the agents do not converge to any set of behaviors that are capable of breaking deadlocks between any pair of agents. In the following experiment, deadlocks between three agents were explicitly prevented to enable the behaviors performed by an agent to directly affect the deadlock. During learning with three agents, the behaviors learned by an agent while

Table 6.5: Learning Drop and Step behaviors using Coordination_Learning

Agent 1	Agent 2			
	$\overline{T}\overline{B}$	$\overline{T}B$	$T\overline{B}$	TB
$\overline{T}\overline{B}$	x	Climb, Drop	Climb, Crouch	Climb, Drop
$\overline{T}B$	Drop, Climb	x	Drop, Climb	Crouch, Climb
$T\overline{B}$	Climb, Crouch	−, Drop	x	x
TB	Drop, Climb	Crouch, Climb	x	x

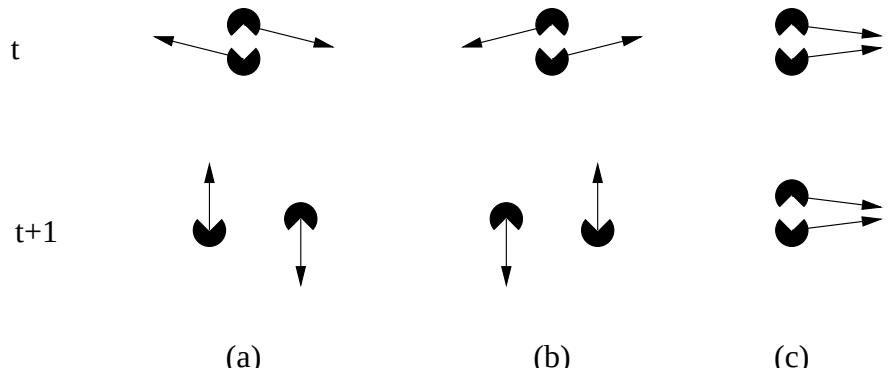


Figure 6.18: Breaking deadlocks with StepLeft and StepRight behaviors. In figure a (b), both the agents perform the StepLeft (StepRight) behavior and the deadlock is broken. In figure c, the agents perform different behaviors and the deadlock continues.

caught in a deadlock with one agent might not be able to break deadlocks with the other agent. Moreover, the agents do not have the ability to detect the identity of the other agent in a deadlock and hence cannot learn a different set of behaviors for interacting with different agents. This implies that an agent will have to relearn behaviors during the course of the learning phase. For instance, consider three agents A, B, and C. Assume that agents A and B learned to perform the LieDown and ClimbOver behavior respectively when caught in a deadlock. If agent C performs the LieDown behavior in a deadlock, then if agents A and C are caught in a deadlock, one of these agents will have to relearn its behavior. On the other hand if agent C performs the ClimbOver behavior in a deadlock, then if agents B and C are caught in a deadlock, one of these agents will have to relearn its behavior. Hence, this set of three agents can never agree on a fixed set of behaviors that will break all deadlocks between any pair of agents.

To demonstrate a fixed set of behaviors in the case of three agents, two behaviors "StepLeft" and "StepRight" replace the LieDown and ClimbOver behaviors in this experiment. If two agents are in a deadlock, then this deadlock can be broken if both agents perform StepLeft (or StepRight) behaviors simultaneously. These behaviors may be thought of as causing the agents to "step aside" creating sufficient space for the agents to pass each other (however, if only one agent performs this behavior, there is no change in position). This is summarized in figure 6.18

To enable the learning to converge, the learning rate (η^F in algorithm in figure 6.15) is decreased every time an agent performs a behavior while in a deadlock. The Coordination_Learning algorithm for multiple agents is presented in figure 6.19. The difference between this algorithm and that in figure 6.15 is the use of different learning rates depending on whether a behavior is successful in breaking a deadlock or not (η^{F+} and η^{F-} respectively) in lines 12, 13 and 16 and the decrease of these learning rates after each behavior is performed (by decay rates ρ^{F+} and ρ^{F-} respectively) in lines 13a and 16a.

As in the case with two agents, the weights in their Frustration Network are initialized to zero and the initial value of $p_r = 0.5$ and $\rho_r = 0.99$. The initial values of the learning rates $\eta^{F+} = \eta^{F-} = 0.2$ and the decay rates are $\rho^{F+} = 0.9995$ and $\rho^{F-} = 0.9999$. Trials were run for 200,000 time-steps. Table 6.6 lists the behaviors that were learned by the three agents after one such learning trial ("Left" and "Right" denote StepLeft and StepRight respectively). For example, consider agents 1 and 2 are caught in a deadlock and that agent 1 is thirsty but does not hold a brick (and hence is trying to move toward the nearest water disc) while agent 2 is not thirsty and holds a brick (and hence is trying to move toward the nearest drop-site). Thus, agent 1's internal state nodes are Thirsty=1, Holding-Brick=-1, OtherAgentThirsty=-1, and OtherHoldingBrick=1. Upon detecting a deadlock Agent 1 will perform the StepLeft behavior after learning (row 3, column 2 of Agent 1's table of behaviors in table 6.6). Similarly, agent 2's internal state nodes are Thirsty=-1, Holding-Brick=1, OtherAgentThirsty=1, and OtherHoldingBrick=-1 and it will also perform the StepLeft behavior thus breaking the deadlock (row 2, column 3 of Agent 2's table of behaviors in table 6.6). Any behavior of an agent when performed simultaneously with the corresponding behavior of *either* of the other agents breaks a deadlock between these two agents. For this reason, the three matrices

Algorithm *Coordination_Learning* for three agents

1: Initialize: nAttempts $\leftarrow 0$
2: At time t: if $(a_F = 1)$
3: nAttempts $\leftarrow$ nAttempts + 1
4: if (nAttempts % 2 = 1) {odd attempts}
5: $b \leftarrow$ behavior output by Frustration Network
6: else {even attempts}
7: with probability p_r, $b \leftarrow$ randomly selected behavior
8: with probability $1 - p_r$, $b \leftarrow$ behavior output by Frustration Network
9: set $f(t+1) \leftarrow 0$
10: perform behavior b
11: At time $(t+\delta)$: if $|\vec{d}(t) - \vec{d}(t+\delta)| > d$
12: then $w_{ib}(t+1) = w_{ib}(t) + sign(1 - a_b)\eta^{F+}a_i$
13: $w_{ij}(t+1) = w_{ij}(t) + sign(0 - a_j)\eta^{F+}a_i,\ j \in B, j \neq b$
13a: $\eta^{F+} \leftarrow \eta^{F+} \times \rho^{F+}$
14: nAttempts $\leftarrow 0$
15: $p_r \leftarrow \eta_r \times p_r$
16: else $w_{ib}(t+1) = w_{ib}(t) + sign(0 - a_b)\eta^{F-}a_i$
16a: $\eta^{F-} \leftarrow \eta^{F-} \times \rho^{F-}$

Figure 6.19: Algorithm *Coordination_Learning* for three agents to learn to escape from deadlocks.

of behaviors shown in table 6.6 are symmetric matrices (as explained in section 6.5.1, the complementary Grab behavior is not learned as a response to the Drop behavior).

Figure 6.20 shows the number of attempts the first agent made to escape a deadlock at that time-step (*nAttempts* in algorithm Coordination_Learning for three agents in figure 6.19). After about 160,000 time-steps, the first agent has learned to escape a deadlock as soon as it is detected, irrespective of the identity of the other agent in the deadlock. The corresponding decrease in the value of p_r for the first agent is plotted in figure 6.21, and the decrease in the learning rates (η^{F+} and η^{F-}) are shown in figure figure 6.22.

6.6 Discussion

The connectionist nature of the action selection mechanism enables the agent to apply simple learning rules to adapt its behaviors (that were learned in a single agent environment) to novel situations (deadlocks are a result of the agent existing in an autonomous multi-agent system). This adaptation took place as the agent continued to perform its construction task. The agents only used information regarding the goals of other agents (such as if they were holding a brick or were thirsty) with which it is physically in contact. However, the agents are unable to detect the actions of other agents and they have to use the distance moved as a measure of the outcome of their joint actions. Thus, adaptation took place solely by utilizing the feedback provided by the environment and does not require external supervision.

The "bucket brigade" mechanism is an example of a socially useful behavior that is learned from purely local interactions. It is exhibited only when there is close interactions between the agents (the agents have to be close enough to pick up another agent's dropped brick) and the environment (if the world was not constraining there would not be a global direction to the movement of bricks). Variations of this behavior may arise if costs are attached to each action. While adapting to novel situations (deadlocks), the agent has to make sure that previously learned behaviors (construction sequence) are not forgotten. Connectionist systems are suited to such relearning tasks as they are better able to degrade gracefully.

Learning was faster when there were two agents learning simultaneously compared to the case when one agent was already trained. Also, in the experiments with the learned sensor readings, only one of the agents

Table 6.6: Behaviors learned using Coordination_Learning for three agents. HdBrck, OThirsty, and OHdBrck denote Holding-Brick, OtherAgentThirsty, and OtherHoldingBrick internal state nodes respectively.

Agent 1	OThirsty=−1 OHdBrck=−1	OThirsty=−1 OHdBrck=1	OThirsty=1 OHdBrck=−1	OThirsty=1 OHdBrck=1
Thirsty=−1 HdBrck=−1	x	Left	Left	Left
Thirsty=−1 HdBrck=1	Drop	x	Left	Left
Thirsty=1 HdBrck=−1	Left	Left	x	x
Thirsty=1 HdBrck=1	Drop	Left	x	x
Agent 2	OThirsty=−1 OHdBrck=−1	OThirsty=−1 OHdBrck=1	OThirsty=1 OHdBrck=−1	OThirsty=1 OHdBrck=1
Thirsty=−1 HdBrck=−1	x	Drop	Left	Left
Thirsty=−1 HdBrck=1	Drop	x	Left	Left
Thirsty=1 HdBrck=−1	Left	Left	x	x
Thirsty=1 HdBrck=1	Drop	Left	x	x
Agent 3	OThirsty=−1 OHdBrck=−1	OThirsty=−1 OHdBrck=1	OThirsty=1 OHdBrck=−1	OThirsty=1 OHdBrck=1
Thirsty=−1 HdBrck=−1	x	Drop	Left	Left
Thirsty=−1 HdBrck=1	Drop	x	Left	Left
Thirsty=1 HdBrck=−1	Left	Left	x	x
Thirsty=1 HdBrck=1	Drop	Left	x	x

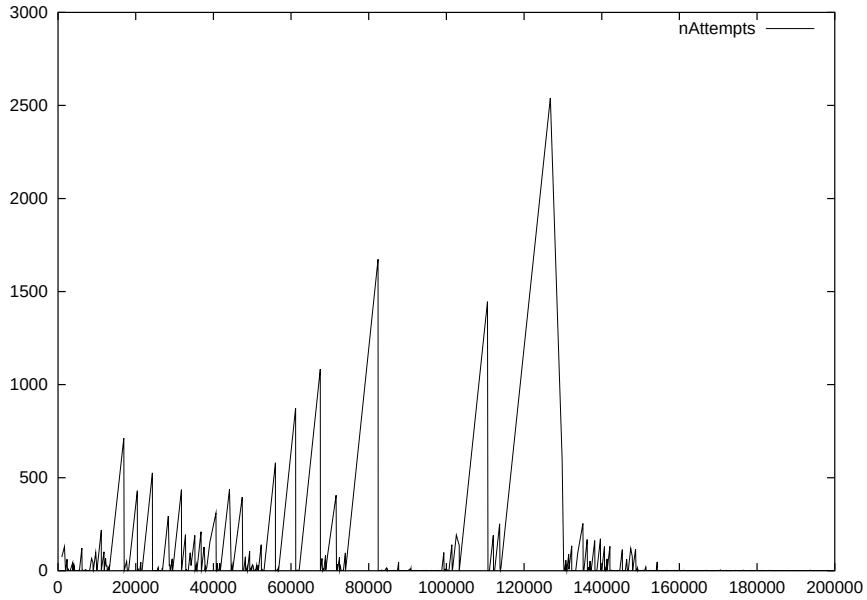


Figure 6.20: Number of attempts the first agent made to escape a deadlock at that time-step (*nAttempts* in algorithm Coordination_Learning for three agents). After about 160,000 time-steps, the agent has learned to escape a deadlock as soon as it is detected.

could use the learned readings (as otherwise both agents would try to fill the intermediate goals dictated by the learned sensor readings). These conditions were explicitly satisfied by the experimenter. Learning to exhibit diverse behavior has been studied by Balch [2000]. Teams of simulated soccer agents were provided with either local or global reinforcement. In local reinforcement, an agent's actions are reinforced only if it scores a goal, while in global reinforcement all agents in a team are rewarded for a goal scored by any member in the team. Teams which received only local reinforcement remained homogeneous, but members of teams which received global reinforcement diversified to perform different behaviors. Such a heterogeneous team also performed better than a homogeneous team of agents. Balch also gives an information-theoretic metric for quantifying the heterogeneity of a group of agents [2000].

The drawbacks of this method of social learning concern scalability and efficiency. All deadlocks arising in the system cannot be resolved using the above approach. If an agent is trying to perform a life-preserving action (for instance moving to a food disc) and is blocking the path of another agent, then dropping or picking discs will not resolve the conflict. It is an interesting issue to study if altruistic behavior can be learned in such situations. As the number of possible motor actions increases, randomly trying all possible actions (and their combinations) will become infeasible.

The map activation learning rule (to recognize environments that cause deadlocks) and the associated matching rule did not consider different orientations of the sensed world compared to the learned activation pattern. For instance, the agent considers two corridors that are oriented at different angles as two completely different features. This arises because all sensor readings are aligned in one global direction. If on the other hand, the ESMs changed orientation along with the heading of the agent, the learned activation pattern would match "corridors" aligned in any direction.

The agents involved in the "bucket brigade" behavior spend a significant amount of time waiting for their "frustration" to increase before they drop their brick. Moreover, when a brick is dropped by an agent in the corridor it could be picked up by any agent near it - even the agent that passed the brick in the previous time-step. There is no directionality in space in the actions of the agents. Hence, efficiency is an issue for long lines of agents.

The task of two agents performing the LieDown and ClimbOver behaviors together (or either the StepLeft and StepRight behaviors) is an instance of a *purely collaborative matrix game*. In a matrix game, each agent

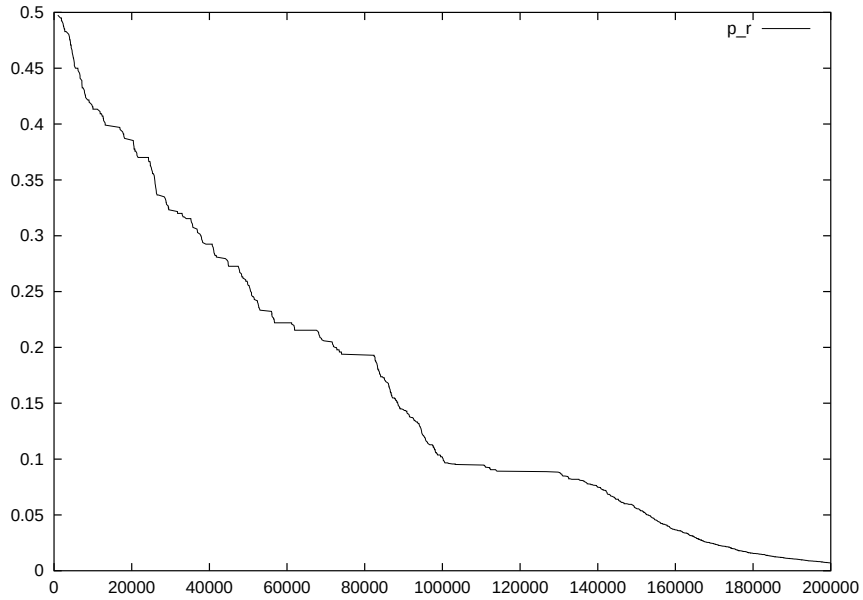


Figure 6.21: Decrease in the value of p_r of the first agent during Coordination_Learning among three agents. Initially, $p_r = 0.5$ and decay rate $\rho_r = 0.99$.

Table 6.7: Pay-off matrices to escape from deadlocks

	LieDown	ClimbOver
LieDown	0	1
ClimbOver	1	0

	StepLeft	StepRight
StepLeft	1	0
StepRight	0	1

(or *player*) can perform one of a finite set of actions and associated with each agent is its *pay-off matrix*. The pay-off matrix gives the reward obtained by an agent for every pair of actions performed by the two agents. A matrix game is called *purely collaborative* if the pay-off matrices for both the agents are the same. A matrix game is called *purely competitive* if the two players' matrices are negatives of each other; these are also called zero-sum games. Other kinds of games are called *general sum games*. The pay-off matrix for the two agents to escape from a deadlock is shown in table 6.7. The agents in fact have to collaborate on multiple instances of this game: one for each possible combination of the internal state nodes of the two agents that can exist during a deadlock (for example, one agent is thirsty and holding a brick, while the other is not thirsty and holding a brick). The adaptation of the behaviors of the ConAg-DL agents can thus be studied from a game-theoretic perspective. Myerson presents detailed descriptions of the major results in game theory including those in repeated games [1991].

In a static environment (where there is only one learning agent), the task of the learner is to discover the action that needs to be performed that will lead to maximum pay-off. However, in a multi-agent environment with multiple learners, the "optimal" actions of an agent depend on the actions that are being performed by the other agents and hence the optimal actions will change as other agents change their actions during learning. Thus, a new definition for "optimality" in multi-agent environments is required. A *strategy* for an agent determines its action. Strategies can be *pure* or *mixed*. A pure strategy is one in which the action is chosen deterministically while a mixed strategy selects an action from a probability distribution [Bowling and Veloso, 2002]. A *Nash equilibrium* for a matrix game is a set of strategies for each agent such that no agent can improve its pay-off by changing its strategy while other agents keep their strategies unchanged. All matrix games have at least one Nash equilibrium [Nash, 1997; Myerson, 1991]. Converging to a Nash

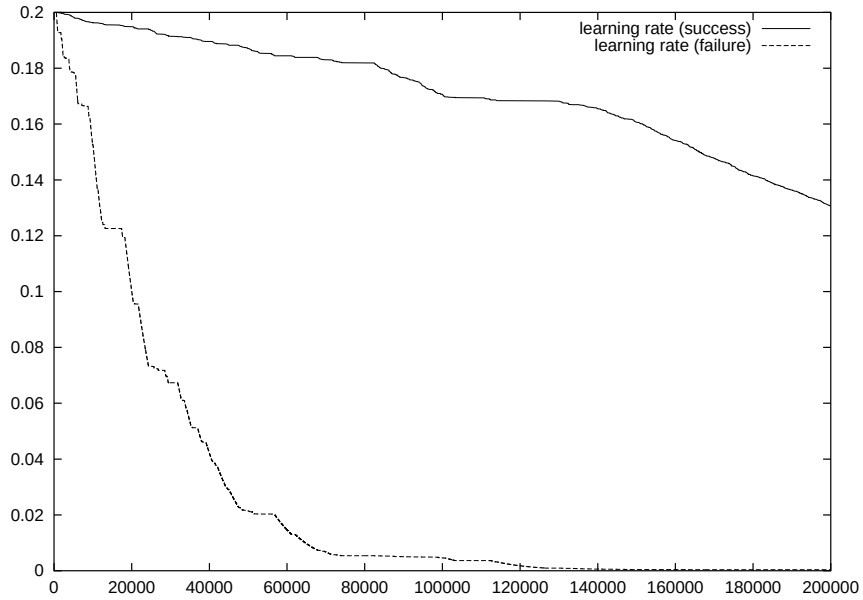


Figure 6.22: Decrease in the learning rates η^{F+} (success) and η^{F-} (failure). Initially, $\eta^{F+} = \eta^{F-} = 0.2$. The decay rates are $\rho^{F+} = 0.9995$ and $\rho^{F-} = 0.9999$.

equilibrium strategy with the maximum pay-off is used as the optimality criterion for multi-agent learning algorithms.

A purely competitive (zero-sum) game has a unique Nash equilibrium, possibly a stochastic (mixed) strategy [Myerson, 1991]. A purely collaborative game, such as the problem for construction agents that need to escape deadlocks with their LieDown/ClimbOver behaviors, has deterministic (pure) equilibrium strategies [Claus and Boutilier, 1998]. In a neural network implementation as in the ConAg-DL architecture, the actions of an agent are chosen deterministically (the output nodes depend only on the weights in the Frustration Network and the activations on the input nodes). The learning algorithm should converge to the equilibrium strategy with maximum payoff. The main difficulty in converging to a solution is that there could be more than one set of actions that give equal pay-off. For instance, the agents could either decide to perform the LieDown or ClimbOver behaviors.

Reinforcement learning algorithms such as Q-learning assume a static environment for convergence [Watkins and Dayan, 1992]. For reinforcement learning to converge in a multi-agent setting, the environment (i.e, the state of all the learners) should gradually settle into a steady state. This requirement translates into a trade-off between *exploration* and *exploitation*. Initially, the agents are more likely to explore different behaviors (high p_r) during deadlocks. As the learning phase progresses, p_r decreases and the agents are more likely to exploit their learned behaviors. Thus, the "steady" state is one in which both the agents perform their learned behaviors. However, the probability of performing a random behavior should not become zero to enable the agents to break out of non-optimal combinations of actions [Claus and Boutilier, 1998]. Note that for Q-learning to converge in a static environment, it is sufficient if the exploration strategy tries every possible action eventually; no "exploitation" of the learned Q-values is necessary [Watkins and Dayan, 1992].

6.7 Related Work

Østergaard *et al.*, study the performance of the bucket brigading behavior with variations on the structure of the environment [2001]. The task that the simulated robots have to perform is foraging - robots first search their world for certain objects that when found have to be moved to a pre-specified "home" area. In any environment, there is a certain optimum number of robots that can perform the foraging task most

efficiently. If the number of robots is greater than this number, then the effects of interference between robots over-weigh the benefits of performing the task in parallel. This notion of "critical mass" is shown for the foraging task by Fontán and Mataric[1996]. Østergaard *et al.* use bucket brigading to reduce the interference between robots and thus improve the efficiency of foraging [2001]. Both maze-like and open environments are studied. The size and number of the simulated robots is also changed. The bucket brigading behavior significantly improves the efficiency of foraging in maze-like environments but does not show any advantage in open environments (this agrees with the results observed in our work). The bucket brigading behavior is also relatively insensitive to the size of the robots compared to the case when foraging is performed without the bucket brigading behavior. Bartholdi III *et al.* studies the use of a bucket brigading system to coordinate workers on an assembly line [2001]. Workers are arranged along the assembly line from the most to the least efficient. A worker (w_i) carries work down the assembly line until she meets the next worker on the line (w_{i+1}) at which time worker w_i transfers her job to w_{i+1} (provided w_{i+1} does not have a job). Worker w_i then moves up the assembly line until she meets her predecessor (w_{i-1}) and receives a job from w_{i-1} (the first worker picks up a new job). This results in a *self-balancing* distribution of jobs in that more efficient workers cover a larger area of the assembly line resulting in minimal time being wasted waiting for a slow worker to finish a job. This leads to the maximum possible rate of production in both a deterministic model of job arrivals [Bartholdi III and Eisenstein, 1996] and a stochastic model [Bartholdi III *et al.*, 2001] (provided there is a sufficient number of jobs and workers).

Østergaard *et al.* also provide a taxonomy of foraging tasks [2001]. This taxonomy distinguishes foraging tasks based on the number of agents, sources (objects to be picked up; bricks), sinks (locations where objects are to be dropped; drop-sites) and on the number of different types of agents and items to be collected, distribution of items in the environment, layout of the environment, and the ability to communicate. If foraging is considered to be a special case of the construction task, our work using ConAg-DL agents would fall into "multiple robot, multiple sources, multiple sinks, constrained space environment, single type of item to be collected, items sprinkled around environment, homogeneous agents, without communication". Fontán and Mataric uses bucket brigading as a means to spatially separate robots in a foraging task [1998]. Bucket brigading in this context has the advantage that the separation between agents adapts to failure of an individual robot. In all these works, the bucket brigading behavior was hard-coded into the robots. The issue of learning to perform bucket brigading was not studied as in our work. Moreover, since the ConAg-DL agents have an internal spatial representation, an explicit search for objects to be picked up is not necessary. Arkin *et al.* consider communication between foraging agents [1993]. Communication allows multiple agents to carry a single object, increasing the speed at which that object can be moved. Communication between ConAg agents is introduced in chapter 9; however this communication is used only to improve the accuracy of the spatial maps and not for coordination.

Methods other than bucket brigading to reduce interference between agents have also been proposed. Vaughan *et al.* resolves interference between two simulated robots whose paths cross each other in a narrow corridor by choosing an "aggressive" policy - the robots have an internal *aggression* trait that is communicated to the other agent in the narrow corridor; the robot with the lower aggression backs away while the other robot continues on its way. Thus, symmetry between the robots is broken and deadlocks are prevented. If resource contests are always won by the same agents, then the group exhibits a dominance hierarchy or *caste* [Goldberg and Mataric, 1997]. There is no benefit to using a caste system to break resource contention between identical agents [Mataric, 1993]. Aggressive behavior is observed in nature (for example, leaping at the same spot by gazelles on sensing a predator) presumably as a signal of the fitness of the signaler [Zahavi, 1975; Enquist, 1985].

Mataric describes a more general approach to reducing interference between greedy homogeneous agents by learning *social rules*: rules that are beneficial to the group but not necessarily to an individual immediately. Each agent receives *social reinforcement* while foraging. This social reinforcement is a mixture of individual reinforcement (received when an agent makes some progress in foraging) and vicarious reinforcement (obtained by observing the actions of other agents). Using only individual reinforcement was not shown to converge. This work is similar to the ConAg-DL architecture in that agents *learn* to reduce interference. However, vicarious reinforcement is not possible for ConAg-DL agents since they lack the ability to sense other agents. Since, deadlocks could always be broken only by performing a single action (Drop brick), the ConAg-DL learning converges though it uses only individual reinforcement. Measuring the distance moved to

reinforce an agent's actions during a deadlock is an instance of a *progress estimator* - an internal critic associated with a single goal [Mataric, 1994]. Distance measurements are used as progress estimators in [Mataric, 1994] to learn a group foraging task. Using a critic associated with a single goal was shown to increase the rate of group learning compared to the use of a single critic for all the agent's goals (as in [Whitehead, 1992]). This is because global reinforcement occurs less frequently than self-generated reinforcement.

Impasse driven learning in the Soar cognitive architecture [Laird *et al.*, 1987] is similar to the use of "frustration" to trigger learning in ConAg-DL agents. An *impasse* in a cognitive architecture occurs when there is insufficient knowledge to make a decision. In a general purpose cognitive architecture such as Soar, an impasse triggers an examination of the cause of the impasse and the creation of new subgoals. The ConAg-DL architecture does not have the ability to reason about its internal representation (planning efficient paths for construction is introduced in chapter 8). Hence during deadlocks an agent can only try out behaviors randomly and reinforce those behaviors that break the deadlock. The appropriate behavior for an agent when given a stimulus depends on the surrounding *context*.

The ConAg-DL architecture represents context on the activations of the learned map, $\underline{l}$, and this enabled an agent is able to learn to associate occurrence of deadlocks with narrow corridors. However, there can be only one such context at a time. Balkenius and Morén describe a computational model that learns to recognize contexts from a sequence of sensor stimuli [2000]. The model can learn multiple contexts associated with different locations. A matching function similar to equation 6.5 is used to compare the input stimulus to the expected stimulus.

Claus and Boutilier compare two methods of applying Reinforcement Learning to purely collaborative games [1998]. In the first method, called *Independent learning*, agents do not consider the actions of the other agent while applying reinforcement. This is similar to the approach taken in ConAg-DL since an agent reinforces any action that breaks a deadlock irrespective of the action of the other agent in the deadlock. In the other method, called *Joint Action learning*, Q-values are learned for every possible state-action pair where the action is a combination of both the agent's actions. Convergence is shown in both cases provided the probability of exploration is gradually reduced such that the agents' actions settle into a steady state. Reinforcement learning has also been applied to purely competitive *stochastic* games [Littman, 1994]. A stochastic game is a matrix game in which the state of an agent is not deterministically determined by its actions but rather from a probability distribution.

Cooperation among more than two agents is demonstrated by having n robots solve the iterated prisoner's dilemma for n players [Birk and Wiernik, 2000] (the prisoner's dilemma has often been used to study cooperation in more general contexts such as arms races [Axelrod, 1984]). However, learning still proceeds by repeated interactions between different pairs of robots. Each of the robots use strategies such as "Tit-for-Tat" [Sandholm and Crites, 1995] and *raise-the-stakes* where the amount of cooperation offered by an agent is increased gradually if reciprocated [Roberts and Sheratt, 1998]. In our environment, deadlocks can occur between more than two agents and hence these techniques cannot be directly applied. Moreover, strategies like raise-the-stakes assume that agents can offer a variable amount of cooperation, a requirement that does not hold in the case of LieDown/ClimbOver behaviors (an agent can only choose to either perform or not perform these behaviors).

Chapter 7

Improving Construction Efficiency with Randomness

7.1 Introduction

The ConAg architecture of an agent described in chapter 3 does not take into consideration the presence of other agents or the order in which bricks have to be added to the structure being built. If this architecture is used for every agent in a multi-agent environment, each agent always moves to the nearest brick not part of the structure (to pick it up) and moves to the nearest section of the structure that is still incomplete (to drop a brick) - a greedy approach. However, interference between agents and the order in which bricks are placed do affect the *efficiency* with which construction is carried out. Efficiency is measured as the time taken by the agents to place bricks at all the unfilled parts of the structure.

For instance, two agents might try to drop a brick at the same location. However, only one of these agents will succeed, forcing the other agent to move towards another unfilled part of the structure. Thus, the time spent by this agent to navigate toward the original drop-site was wasted. Interference between agents can also occur on the path to dropping or picking up discs. If the paths of two agents cross, then the agents will have to "side-step" each other thus increasing the distance traveled by each agent.

The order of placing bricks determines the total distance traveled by the agents while performing construction because after a disc is dropped at a drop-site, it can block direct paths to other drop-sites. An example is shown in figure 7.1. The fastest way to build the three walls is to build the right-most wall first, then the middle wall and finally the wall closest to the source of discs. If the nearest wall was built first, this would block straight paths from the disc sources to the other two walls.

In this chapter, randomization is used in two ways to select those unfilled drop-sites where the agent will

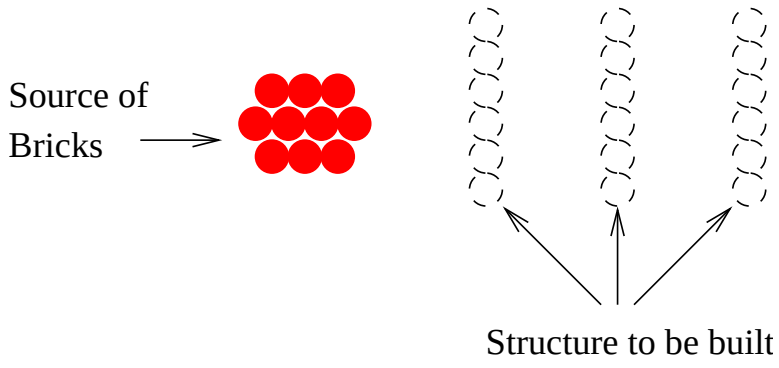


Figure 7.1: Structure where order of placing discs is important.

drop a brick. The two random placement strategies are implemented by selectively activating the neurons on the Configuration Navigation map. The performance of these randomized placement strategies is compared with the greedy approach.

7.2 Disc Placement Strategies

The three goal placement strategies that are studied are:

- *Greedy Placement:* The agents pick the closest available brick and place it at the closest drop-site. This is the placement strategy that is achieved with the interconnections between the ESMs as described in chapter 3 (the ConAg architecture). To implement this, the agent activates *all* the goal nodes in the Configuration Navigation Map. Thus, the Navigation map always plans the path to the nearest drop-site as a result of the gradients produced by spreading activation.

- *Random Placement:* The agents pick the closest available brick and place it at a random drop-site requiring a brick. This placement strategy is implemented by randomly activating *one* of the goal nodes in the Configuration Navigation Map. Thus, the Navigation map plans the path to that particular drop-site, ignoring closer drop-sites. The resulting architecture is called ConAg-R.

- *Localized Placement:* Each agent places bricks only in a small randomly chosen area of the environment. An agent activates only those goal nodes that are *neighbors of a randomly chosen goal node* in the Configuration Navigation map. (The neighborhood size is fixed *a priori*.) Thus the agent plans paths only to drop-sites within the area represented by these chosen nodes. This placement strategy adds some of the features of greedy drop-site selection to that of random placement. Unlike random placement, in which an agent can decide to drop a brick at a site far away from the first drop-site, the neighborhood criterion forces the agent to drop bricks close to each other (unless there are no nearby drop-sites in which case a new random location is chosen). The aim is to get the agents to localize themselves (and thus reduce interference) but without blocking off direct paths to goal locations. Each agent counts the number of times it comes into contact with other agents in that area and if the count exceeds a threshold it chooses a new area in which to build. In this way, there is no overcrowding of agents in a given area. The resulting architecture is called ConAg-L.

7.3 Results

A series of simulation runs was carried out to compare the performances of the different goal placement strategies. The parameters that determine the time to complete construction are:

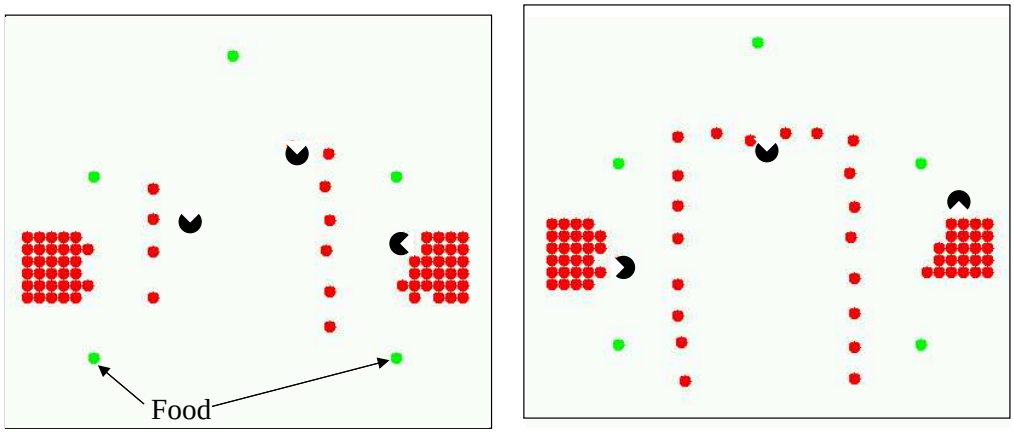
1. The number of agents: varied from 1 to 4 agents

2. The shape of the structure to be built: The structure to be built is shown in figure 7.2(b). This particular shape was chosen since building the rows at the sides first forces agents to move around them to place subsequent bricks for the top row.

3. Initial arrangement of bricks: The bricks are initially clustered together in one group, two groups, or are scattered all around (figure 7.2 shows an environment with two clusters of bricks).

4. Initial positions of agents: Agents are introduced at a random location in the environment and initially explore their world until a brick becomes visible. From that point, construction begins. The initial activations on the Configuration ESM were offset manually for each agent to account for the different starting positions of agents (as otherwise, the agents will each try to build the specified structure at a different location).

5. Positions of Food and Water discs: The hunger and thirst thresholds (T_0^h, T_1^h, T_0^t, and T_1^t) and the rates affecting the internal food and water level of an agent (f_0, f_1, w_0, and w_1) are set such that an agent becomes hungry/thirsty every 2000 time-steps.



(a) (b)

Figure 7.2: Environment for comparing brick placement strategies: (a) Environment at $T = 1000$. Initially all bricks are clustered at the two sides. (b) The three agents have completed construction at $T = 2471$. The bricks are in the desired configuration (the width of an agent is approximately equal to the gap between two neighboring bricks; however an agent cannot pass through these gaps because all neighboring nodes of an obstacle node in a ESM are inhibited while path planning to account for the size of the robot and the inaccuracy in the spatial representation as mentioned in chapter 3 as the Avoid-Red behavior becomes active when close to a brick).

The reported data for each case is obtained from one simulation run. Results from different simulation runs vary slightly depending on the initial positions of the agents (and hence the time taken to explore), but the trends across parameters are similar. Figures 7.3-7.6 shows the number of time-steps taken to complete the construction task using the three placement strategies for each of the three initial environments: building blocks in one cluster, two clusters or randomly distributed. When the building blocks are in one cluster (figure 7.3), the greedy placement blocks off direct paths to drop-sites by constructing the side "walls" first (figure 7.2(a)). Thus, randomly choosing goal locations works better since gaps exist for the agents to move directly to their goal locations. When the number of agents is small, the localized placement strategy is similar to random placement. However, as the number of agents increases to 4, localized placement works significantly better as it tries to keep the goal locations of different agents separate.

As the brick sources get more widely distributed (figures 7.4-7.5), the greedy placement strategy does not suffer from the blocked path problem. Agents do not have to move around a wall of already placed discs to approach any goal location - there is almost always a direct path. Thus, the difference in performance between the strategies is less marked in this case. The time taken to complete the task decreases as the number of clumps increases because the total distance that has to be covered decreases. The speedup obtained by adding more agents increases when the bricks are scattered because the agents tend to localize around the clumps of building blocks leading to reduced interference.

Figure 7.6 shows the number of bricks placed over time by four agents when the bricks were initially in one cluster. The number of bricks placed per time-step (the slope of the curves) decreases with time for greedy placement, but is relatively constant for the other two strategies.

7.4 Discussion and Related Work

The relative performance of the different placement strategies varies with the nature of the environment. In unstructured environments (such as those in which the bricks are scattered), the greedy approach works best

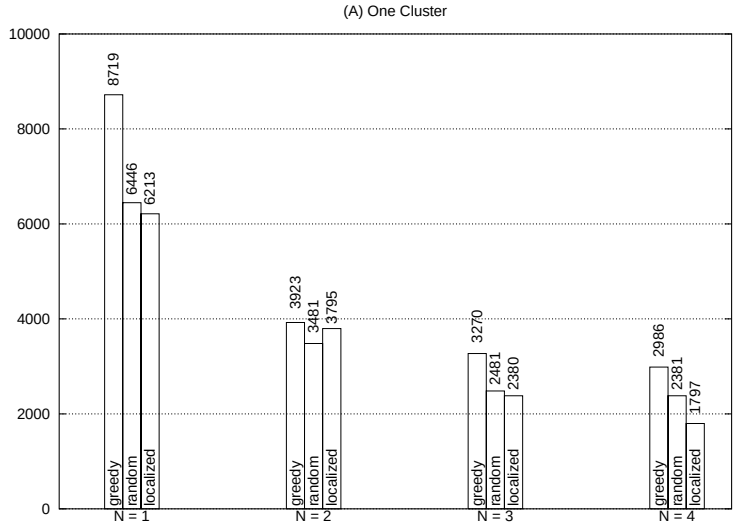


Figure 7.3: Comparison of disc placement strategies - single source: The time to complete construction task when all bricks are initially in one cluster; N = number of agents.

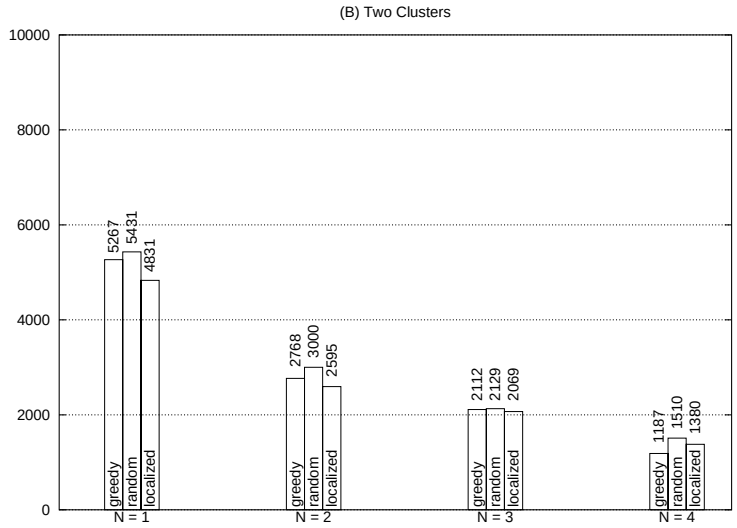


Figure 7.4: Comparison of disc placement strategies - two sources: The time to complete construction task when all bricks are initially in two clusters; N = number of agents.

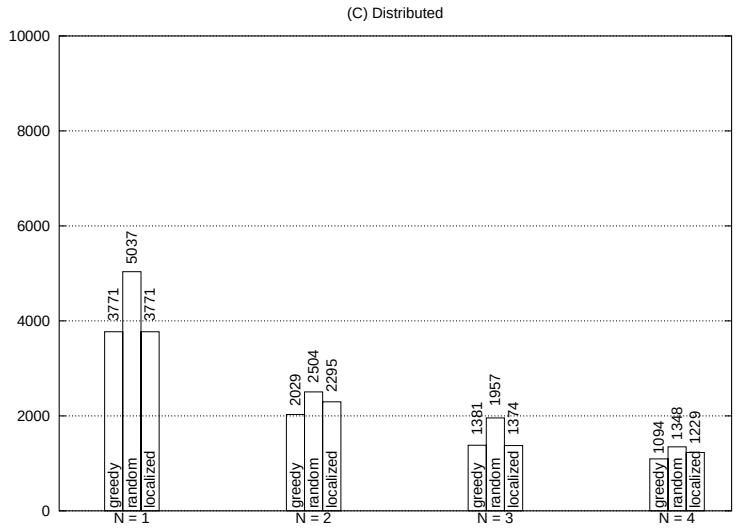


Figure 7.5: Comparison of disc placement strategies - scattered sources: The time to complete construction task when all bricks are initially scattered; N = number of agents.

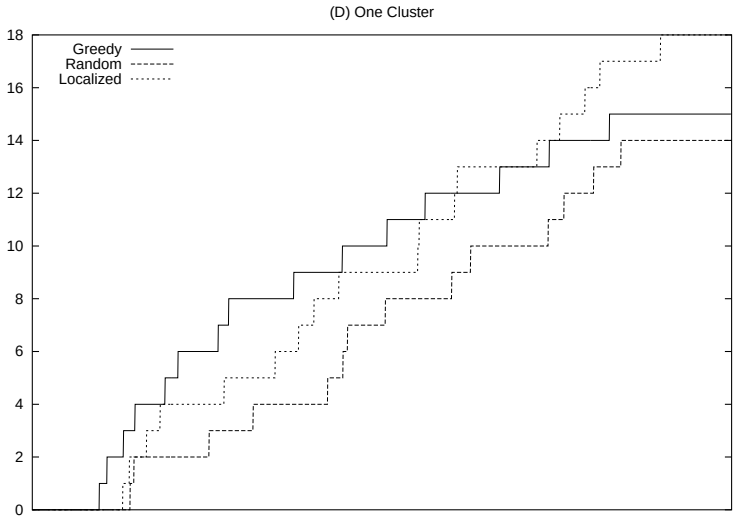


Figure 7.6: Number of bricks placed over time N=4, and bricks are initially in one cluster; N = number of agents.

since dropping a brick does not block of all paths to other drop-sites. In structured environments (such as those having only a single source of bricks and the structure is composed of rows of bricks), the randomized algorithms work best since these approaches do not construct an entire row of bricks that can block off direct paths to other drop-sites, i.e, "gaps" are left between the bricks being dropped.

As the number of agents increases, the interference between agents becomes an important factor in determining efficiency. Thus, the randomized placement strategy works well even in structured environments with increasing number of agents. The localized placement strategy, resulting in the ConAg-L architecture, is the best strategy overall since it incorporates the benefits of both greedy (successive bricks are dropped at nearby locations) and randomized (bricks are not placed to become walls in the beginning that block direct paths to other drop-sites) placement strategies.

Distributed construction is a complex task and simple local strategies will not work for all cases. Thus it is important for the architecture to allow for easy implementation of different placement strategies. All the placement strategies were implemented using ESMs, thus demonstrating the benefits of a simple grid-based representation of space.

Several architectures have been proposed to enable explicit coordination in multi-robot systems. Cao *et al.* provide a general survey of cooperative multi-robot systems [1997]. Some of these architectures are described below and compared to the ConAg-R architecture.

- **ALLIANCE**

  The ALLIANCE architecture is a distributed behavior-based architecture that was designed for fault-tolerant coordination among a group of robots [Parker, 1998]. The robots have similar capabilities to perform a "hazardous waste cleanup" task in an indoor environment). Each robot maintains internal *impatience* and *acquiescence* motivations. These motivations enable an agent to keep track of the progress that is made on the task by *other* robots. If progress is not detected, "impatience" increases and the robot takes over that task. The set-up is similar to the ConAg-R architecture in that each agent or robot is capable of completing the task by itself (construction or waste cleanup) and coordination is used for efficiency purposes. However, in ALLIANCE, robots broadcast messages about their current actions to let other robots know of the current tasks they are trying to perform. In ConAg-R, there is no communication; agents do not explicitly consider the actions of others and it is only when a change in the environment is sensed (and ESMs updates) is the behavior of an agent altered.

- **Basis behaviors**

  Mataric proposes a methodology to design *basis behaviors* for multi-robot systems [1995]. Basis behaviors are those minimal behaviors of a robot that can be combined with those of others to enable the group of robots to exhibit interesting social coordination and reduce interference. This is an extension of the principle of behavior-based robotics to multi-agent systems. Five basis behaviors were described for spatial interactions:

  1. *Safe-wandering*: moving while avoiding collisions
  2. *Following*: follow behind another agent
  3. *Dispersion*: move while maintaining a minimum distance from other agents
  4. *Aggregation*: converse of dispersion; move while maintaining a maximum distance from other agents
  5. *Homing*: move to a particular region

  These were combined to exhibit group behaviors such as flocking and herding. The agents do not explicitly broadcast their actions (as in ALLIANCE); each behavior is implemented by considering the position of only the neighboring robots. In this respect (amount of communication among agents), the use of basis behaviors is closer to ConAg-R than ALLIANCE. However, though the basis behaviors can be used for navigating without interference, the choice and order of goal locations is not specified. In ConAg-R, the ESMs are used to provide these goal locations.

- **Motor-schema**

  Arkin proposed navigation using *motor schemas* [1989]. Motor schemas are similar to the reactive behaviors in ConAg-R as they produce a motor action based on current sensor input. However, unlike in ConAg-R, the outputs are summed before being sent to the motors (i.e, it is not winner-take-all). The motor schema approach was used to enable a group of four robots to move to a particular goal while avoiding obstacles and maintaining a particular formation (such as a line or a column) [Balch and Arkin, 1995; 1999]. The formations are comparable to those exhibited by using basis behaviors but each robot determines its position based on the locations of *all* robots in the group (for instance, based on the centroid of all the robot positions).

The main difference between these architectures and ConAg-R is that they do not have an internal spatial representation and hence the exhibited behaviors are purely reactive. In fact, these architectures are used for the foraging task where it is not necessary that objects be moved to precise locations in a particular order. ConAg-R agents, on the other hand, have an internal spatial map and and an agent can use its map not just for determining the locations of bricks and drop-sites but also to select these navigation goals in such a way to reduce interference between agents. Thus, the ConAg-R architecture uses the internal spatial representation and a randomized approach for multi-robot coordination.

Chapter 8

Improving Construction Efficiency with Path Planning

8.1 Introduction

The disc placement strategies studied in chapter 7 do not consider the shape of the particular structure being built or its relation to the initial positions of bricks. Certain orders of placing bricks can make it impossible to complete the construction task. For instance, in figure 8.1, the structure to be built consists of a square inside a larger square. The greedy approach to placing bricks would cause the agents to build the outer square before completing the inner one, blocking *all* paths from the bricks source to the remaining unfilled positions of the inner square.

Besides the order concerning which parts of the structure are to be built, the assignment of bricks to drop-sites also affects the distance traveled by the agents. For instance, in figure 8.2 the agent(s) should use the upper source of discs to build the upper parts of the walls and the lower source for the lower parts of the walls. This minimizes the total distance between the initial and final positions of the discs. The agents should also build the rightmost wall first.

In this chapter, an algorithm that explicitly computes which drop-sites block paths to others is described so that agents can drop bricks at those drop-sites that block the least number of paths. This algorithm is implemented using spreading activations on ESMs and hence automatically takes into account the presence of obstacles. Implementation using ESMs also enables this drop-site selection algorithm to be integrated into the existing ConAg architecture. The resulting path planning architecture is termed ConAg-TS (for temporal sequencing) because each ConAg-TS agent plans the temporal order in which it will fill goal locations.

8.2 Theoretical Formulation of the Construction Task

The construction environment can be described as a weighted undirected bipartite graph. The vertices of one partition represent the initial locations of bricks and the vertices of the other partition represent the final drop-site locations (the pattern of the structure to be built). The weights on the edges represent the distances of the paths between these two kinds of locations. If there is only one agent, the task is to determine the shortest path that visits all the vertices representing drop-sites and no vertex is visited more than once. Since the graph is bipartite, the edges of this path represents a route that alternates between an initial location of a brick and a construction site. If the number of vertices in the two partitions are the same (the number of bricks is equal to the number of drop-sites) this problem becomes one of computing the shortest Hamiltonian path (a path that visits every node exactly once) in this graph. The problem of calculating the shortest Hamiltonian path in an undirected graph is NP-complete [Garey and Johnson, 1979].

This characterization assumes that the length of the path between two locations (the edge weight) will not change during the construction task. However, in the general case, the order in which bricks are placed affects the distance traveled by agents to place the remaining bricks. Since bricks dropped off earlier in

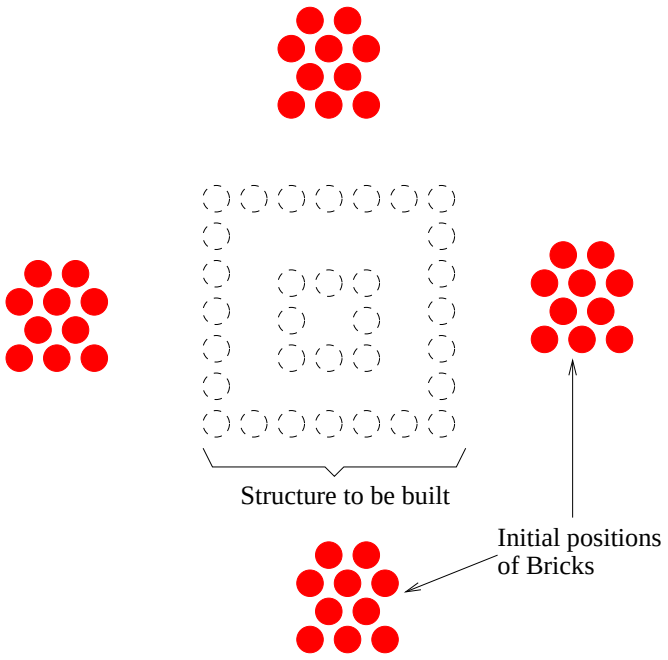


Figure 8.1: Concentric squares to be built: the order of placing bricks is important; if the outer walls are built first, the inner square becomes inaccessible.

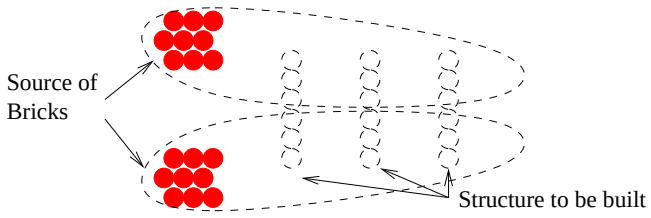
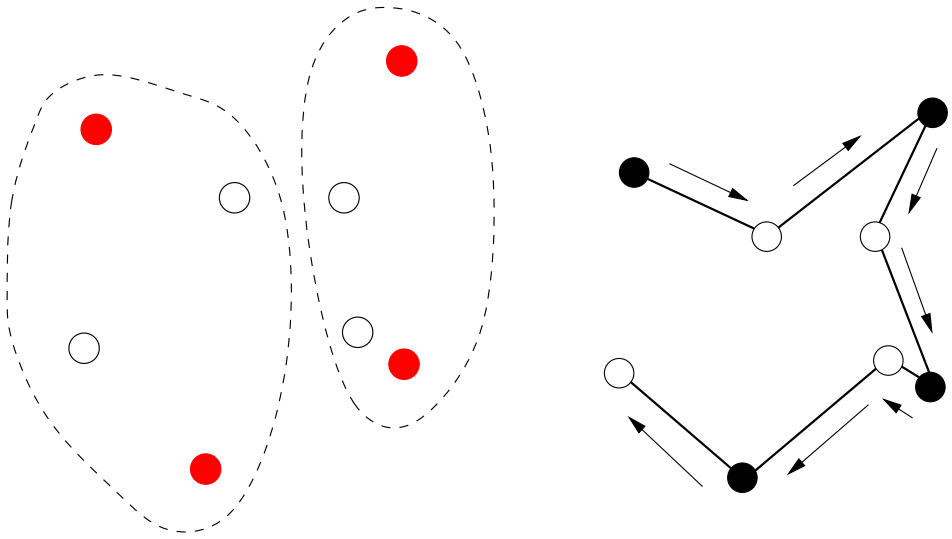


Figure 8.2: Assigning bricks to drop-sites: the agent(s) should move the upper source of discs to the upper parts of the walls and the lower source to the lower parts of the walls to minimize total distance traveled.



(a) (b)

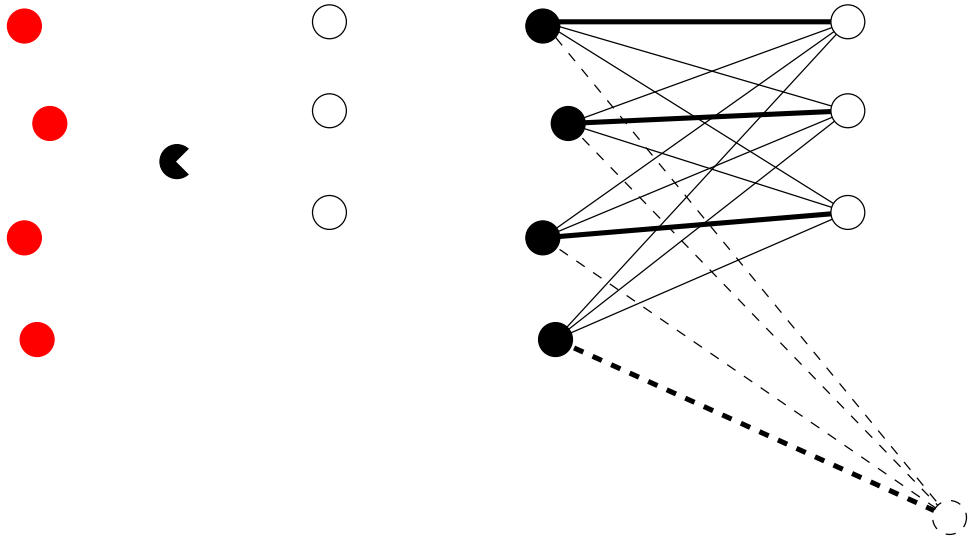
Figure 8.3: Shortest Hamiltonian path for a construction task: (a) 4 bricks (filled red discs) and 4 drop-sites (unfilled circles). If there were two agents, they might divide the construction task among themselves as indicated by the dashed areas. (b) Shortest Hamiltonian path of the construction task. Filled nodes correspond to bricks and unfilled nodes correspond to the drop sites.

the construction task can block direct paths to other drop-sites. The characterization for multiple agents is even more complex. Moreover, in the case of multiple agents, there is interference between agents since only one agent can occupy a location at any given time. If the paths of agents cross each other, the chances of such interference increase. Thus, for efficient construction with multiple agents, the agents have to plan the temporal sequence of drop-site locations where bricks have to be moved to and also work on different parts of the environment. However, the path indicated by the shortest Hamiltonian path will take the agent to all parts of the environment thus increasing the chances for interference. An example is shown in figure 8.2.

Another approach to characterizing the construction task is to model it as a minimum weighted perfect matching (MWPM) or assignment problem for bipartite graphs. The MWPM problem is to compute a one-to-one mapping between the vertices in the two sets such that the sum of the edge weights between matched pairs is minimum. If the number of vertices in the two partitions (number of bricks and drop-sites) are not equal, the graph can be augmented with dummy nodes with a very large weight. An example is shown in figure 8.4. In an optimal solution to the MWPM problem with Euclidean weights, edges between matched pairs do not cross, reducing interference between agents. This is illustrated in figure 8.5. If a matching of the graph contains two edges AB and CD that cross each other at Z, then those edges can be removed and replaced with edges AD and BC with smaller total weight since all edge weights are Euclidean distances and the triangle inequality holds.

$$\begin{aligned} AB + CD &= (AZ + ZB) + (CZ + ZD) \\ &= (AZ + ZD) + (CZ + ZB) \\ &\geq AD + BC \end{aligned} \tag{8.1}$$

An optimal solution to the MWPM problem can be calculated in $O(n^3)$ time where n is the number of vertices in each set using the Hungarian algorithm [Kuhn, 1955] or more efficiently by using an augmenting path approach to solve a maxflow formulation of the assignment problem [Jonker and Volgenant, 1987]. The greedy approach as implemented by the ESMs is a distributed neural implementation of the "matrix scan" heuristic for the MWPM [Kurtzberg, 1962]. Avis gives a comparison of the average-case bounds for the



(a) (b)

Figure 8.4: MWPM characterization of a construction task: (a) 4 bricks (red discs) and 3 drop-sites (unfilled circles). (b) Bipartite graph of the construction task. Filled nodes correspond to bricks and unfilled nodes correspond to the drop sites. The dashed circle indicates a node representing a dummy drop-site. The lines represent edges between nodes (dashed lines indicate edges with a large weight to the dummy node). The edges that are in the solution to the MWPM are shown by thick lines.

greedy MWPM heuristic with others [1983].

Unlike the shortest path solution, the MWPM characterization does not determine the order in which bricks are to be moved to drop-sites. 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 the path from every disc to its drop site is a straight line (not blocked by any previously dropped disc). The algorithm described below determines such a sequence in two steps. Each agent first computes a matching (an approximate solution to the MWPM) between discs and drop sites represented in its ESMs using a greedy heuristic. It then determines which of the drop sites are not on any of the paths (between a disc and its matching drop site) and proceeds to fill these first.

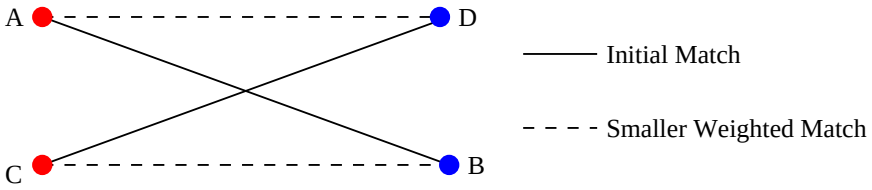


Figure 8.5: Edge crossings in a solution to the MWPM: an optimal solution contains no edges that cross each other.

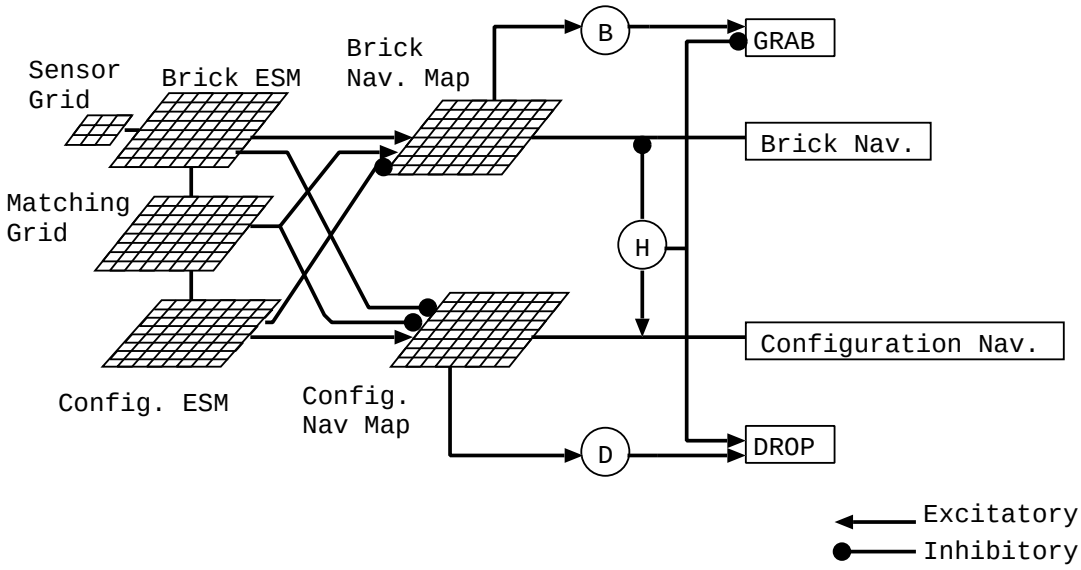


Figure 8.6: ConAg-TS architecture: the lines between the matching grid and the ESMs represent links between every corresponding pair of neurons. Three internal state nodes are also shown: At-Brick (marked "B") is set from the Brick navigation map, At-Drop-Site (marked "D") is set from the Configuration navigation map, and Holding-Brick (marked "H").

8.3 Matching Algorithm

The algorithm, *calculate_matching()*, is implemented on a grid of neurons called the *Matching grid* that can spread activations among neighboring cells. There is a one-to-one correspondence between the neurons on the matching grid and those on the Brick and Configuration ESMs. The obstacles for brick or configuration navigation (the locations of bricks that are already part of the structure) inhibit activations on the matching grid. Currently, activations from Food and Water ESM neurons that can represent food and water disc obstacles have not been incorporated into the matching grid. The interconnections between the matching grid and the ESMs in the ConAg-TS architecture are shown in figure 8.6 and are described in more detail in figure 8.12.

The calculate_matching algorithm is shown in figure 8.7. At the end of the algorithm, $match[n] = 1$ for all nodes n that are on a path between a matched disc and any drop-off location. Let N be the set of all nodes on the Matching grid and S, T be the set of nodes representing initial brick locations and unfilled drop sites respectively. The matching grid spreads two different activations: the a activation spreads from nodes representing bricks while the b activation spreads from nodes representing drop-sites. Both the activations together determine the match value at a node. Denote by a_n and b_n the a and b activation respectively at an arbitrary node n. Define $succ(n)$ to be the neighboring node of n along the strongest gradient of a activation:

$$succ(n) = argmax_{n' \in nb(n)}(a'_n) \tag{8.2}$$

In line 2, the a activation is initiated from all nodes representing brick locations. This activation flows around nodes representing bricks that are already placed as these are obstacles. In lines 4-6, every drop-off location t spreads activation b *against* the gradient of the a spreading activation initiated in line 2. Thus $b_n = 1$ for all nodes n that lie on a path between some two nodes that represent a brick and drop-site location. When this b activation reaches a brick location s (condition in line 7), another b activation is started (distinguished from the first by setting b_n to 2) from s that is spread along those nodes n that already have $b_n = 1$ (line 9). At the same time, a_s is reset to 0 (step 11) so that the spread of the b activation initiated in step 6 is stopped (since the $b_n = 1$ activation spreads against the gradient created by the a activation). Thus, though activation from more than one drop-site might reach s in step 7, only the activation that reaches s first is

```
calculate_matching (S, T)
     returns match[n] ∈ {0, 1}, ∀n ∈ N
1:   match[n] := 0∀n ∈ N
2:   spread activation a from all s ∈ S (a_s = 1)
3:   while there are unmatched nodes in T
4:      for all unmatched nodes t ∈ T
5:         {set b_n := 1 for all nodes n on path from t to nearest disc}
6:         b_succ(t) := 1; if (b_n = 1) then set b_succ(n) := 1
7:         if (b_s = 1) ∧ (a_s = 1), s ∈ S
8:            {match s to t, where t initiated b activation that reached s}
9:            b_s := 2; if (b_n = 2) then set b_n' := 2, ∀n' ∈ nb(n) ∧ b_n' = 1
10:           set match[n] := 1 for all nodes n on path from s to t
11:           a_s := 0
12:        end if
13:     end for
14:  end while
```

Figure 8.7: Algorithm calculate_matching

reinforced with the $b_n = 2$ activation. This ensures that $b_n = 2$ only for those nodes that lie on a path from a brick source to the nearest drop-site. In this way, a node representing a brick is "matched" to the nearest drop-site (indicated by setting $match[n] = 1$ only for those nodes n where $b_n = 2$ in line 10).

8.3.1 Example of calculate_matching Algorithm

An example illustrating the calculate_matching algorithm is shown in figures 8.8-8.9. Figure 8.8(a) shows an environment containing three bricks and three drop-sites and figure 8.8(b) shows the cells on the matching grid corresponding to these bricks and drop-sites. The spread of activation a from the three nodes representing bricks is shown in figure 8.8(c). The activation $b_n = 1$ that spreads from the nodes representing drop-sites against the gradient of the a activation is shown in figure 8.8(d). Note that though drop-site D2 is closer to brick B1 than to B2, the activation from D2 did not reach B1 since nodes representing B1 and B3 stopped spreading a activation when the b activation reached them (line 11 in figure 8.7). Thus, $b_n = 2$ only for those nodes that lie between B1 and D1, and between B3 and D3 (figure 8.8(e)). The new a activation (only from the remaining unmatched brick B2) is shown in figure 8.9(a). Now the $b_n = 1$ activation from D2 reaches B2 (figure 8.9(b)) and the $b_n = 2$ activation from D2 back to B2 matches this brick and drop-site to each other. The final values in the matching grid are shown in figure 8.9(c) indicating the three matched brick and drop-site pairs.

8.4 Sequencing Algorithm

In the example described above, the paths between the three pairs of matched bricks and drop-sites did not cross each other and hence the three bricks may be moved to their corresponding drop-sites in any order. This section describes how to determine 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. Note that the spread of the b activation starts from a neighboring node of a drop-site node ($b_{succ(t)} := 1$ in line 6 of the calculate_matching algorithm (figure 8.7), 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 8.9(c)), then its match value is 0. However, if a drop-site D_1 is present on the path between a drop-site D_2 and its matched brick, then $match[D_1] = 1$ and D_1 must be filled with a brick only after D_2 is filled as otherwise the brick dropped at D_1 will block the path to D_2. In other words, bricks must be dropped at drop-sites whose corresponding match value is 0.

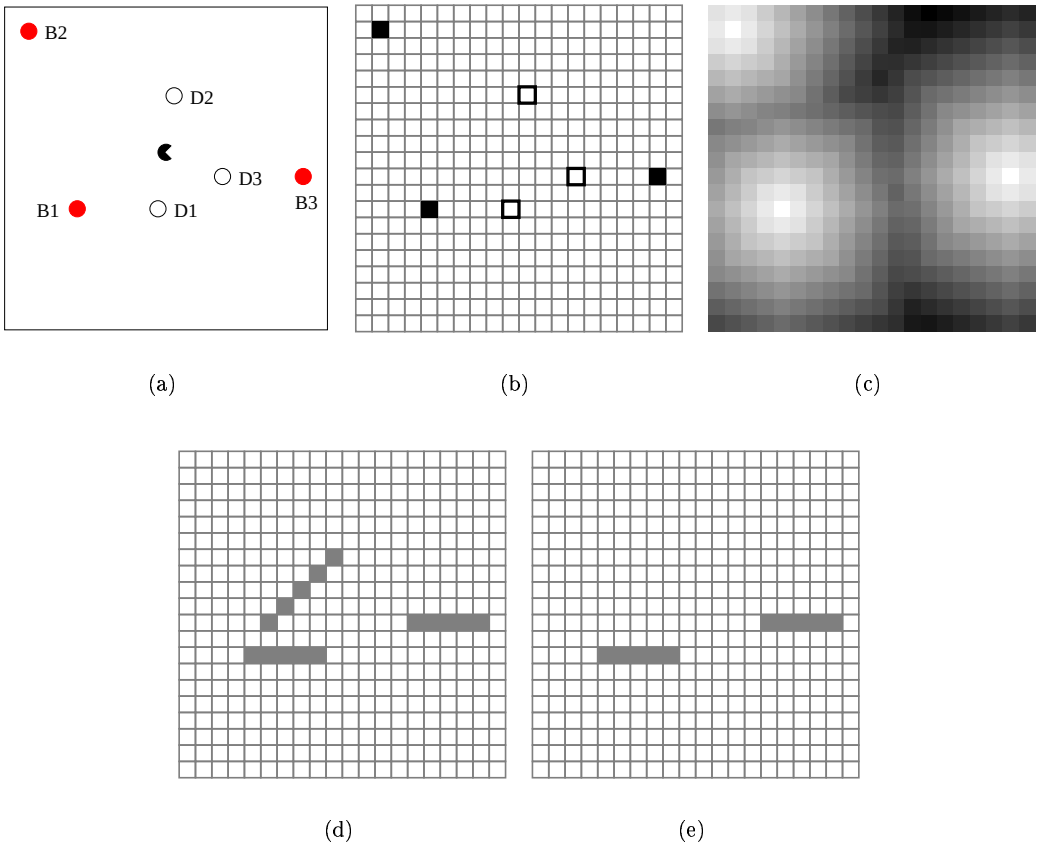


Figure 8.8: Example illustrating algorithm calculate_matching. (a) Environment containing three bricks (B1, B2, and B3) and three drop-sites (D1, D2, and D3). (b) The cells in the matching grid corresponding to the bricks (filled squares) and drop-sites (squares with dark edges) (c) Activation a spreading from nodes representing bricks (d) Activation b: nodes n where $b_n = 1$ (e) Activation b: nodes n where $b_n = 2$.

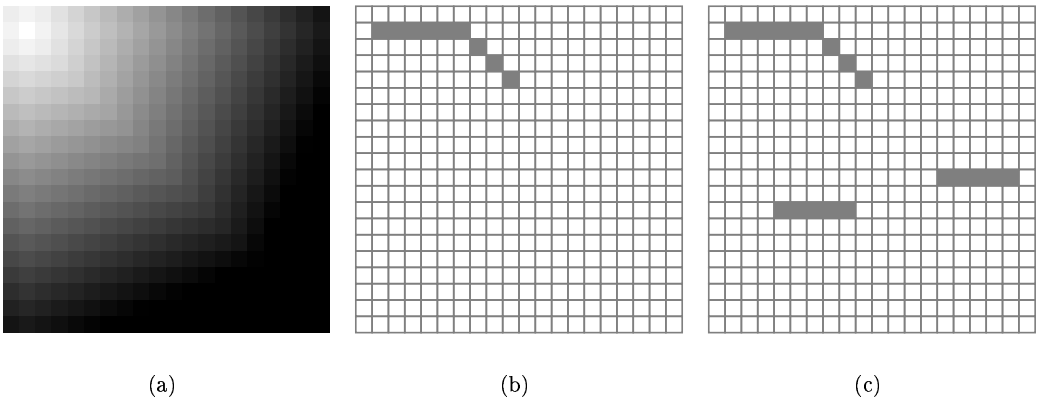


Figure 8.9: Example illustrating algorithm calculate_matching (contd.). (a) Activation a spreading from last unmatched brick (b) Activation b: nodes n where $b_n = 1$ (c) Activation b: nodes n where $b_n = 2$. These are also the nodes where $match[n] = 1$ and shows which bricks should be moved to which drop-sites.

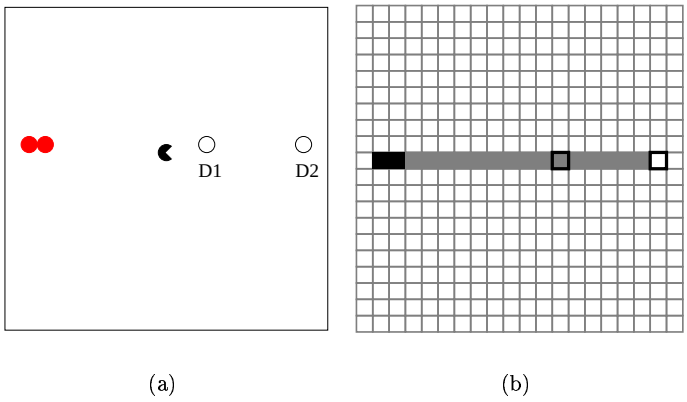


Figure 8.10: Example illustrating goal sequencing. (a) Environment containing two bricks and two drop-sites, D1 and D2. (b) The corresponding matching grid. The path from the bricks to drop-site D2 passes through the drop-site, D1. Filled black squares indicate cells representing bricks, squares with thick edges indicate drop-sites D1 and D2, filled gray squares indicate cells with match value of 1.

For instance, consider the two bricks and two drop-sites in figure 8.10(a) and the nodes in the corresponding matching grid that have a match value of 1 (figure 8.10(b)). 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, in this case, drop-site D2 should be filled before D1.

The agent also has to decide which brick to pick up to be moved to a drop-site. Bricks that are matched to some drop-site have their corresponding nodes on the matching grid set to one (dark filled squares in figure 8.10(b)). Also, given a brick, the drop-site matched to that brick can be identified by following the path of nodes with match value 1 originating from the node on the matching grid corresponding to that brick. For example, starting from the node corresponding to brick B2 (of the world shown in figure 8.8(a)) in figure 8.9(c) and following the shaded squares, one reaches the cell corresponding to drop-site D2. In general, one may reach more than one drop-site by following an activated path on the matching grid from a brick node since paths between different matched brick and drop-site pairs may cross. For instance, in figure 8.10(b), one reaches the nodes corresponding to both D1 and D2 starting from the brick nodes at the left. However, only those drop-site nodes with match value 0 are filled first.

After a brick is dropped, it becomes an obstacle for all future path planning and the activations on the matching grid change; algorithm calculate_matching is run again, the match values are recomputed, and the next set of drop-sites with match value 0 is chosen to be filled. This sequencing algorithm that determines which bricks are picked up and moved to which drop-sites is described in figure 8.11. Line 3 identifies the closest brick node t that is matched to some drop-site. Line 4 spreads activation from node t to identify its matching drop-site. Activation may reach more than one drop-site node (due to crossed paths), but only a drop-site with match value 0 is chosen (represented by node t).

The sequencing algorithm is implemented by having inhibitory connections from the nodes in the matching grid to the corresponding nodes in the Configuration navigation map and excitatory connections to the corresponding nodes in the Brick navigation map. Recall that the Configuration navigation map is used for path planning to drop-sites and therefore 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 navigation map ensures 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 navigation maps, the agent moves toward the closest selected brick or drop-site. These interconnections are shown in figure 8.12.

sequence ()
1: while there are unfilled drop-sites ($T \neq \{\}$)
2: match = calculate_matching (S, T)
3: pick up closest brick with corresponding matching grid node $s \in S$ such that $match[b] = 1$
4: spread activation from s along nodes with match value 1.
5: drop brick at nearest drop-site with corresponding matching grid node $t \in T$ such that $match[t] = 0$ and activation has reached a neighboring cell of t

Figure 8.11: Drop-site sequencing algorithm

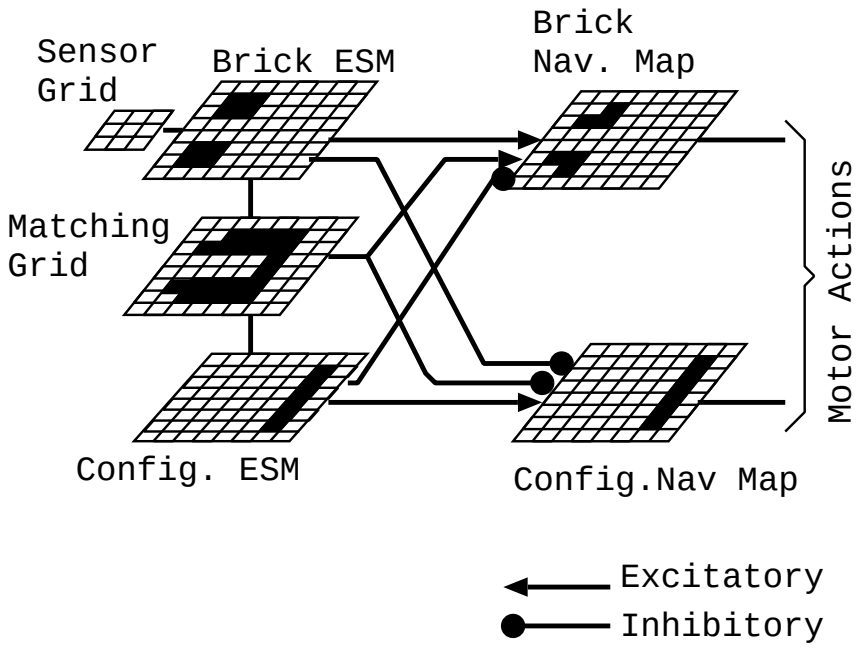
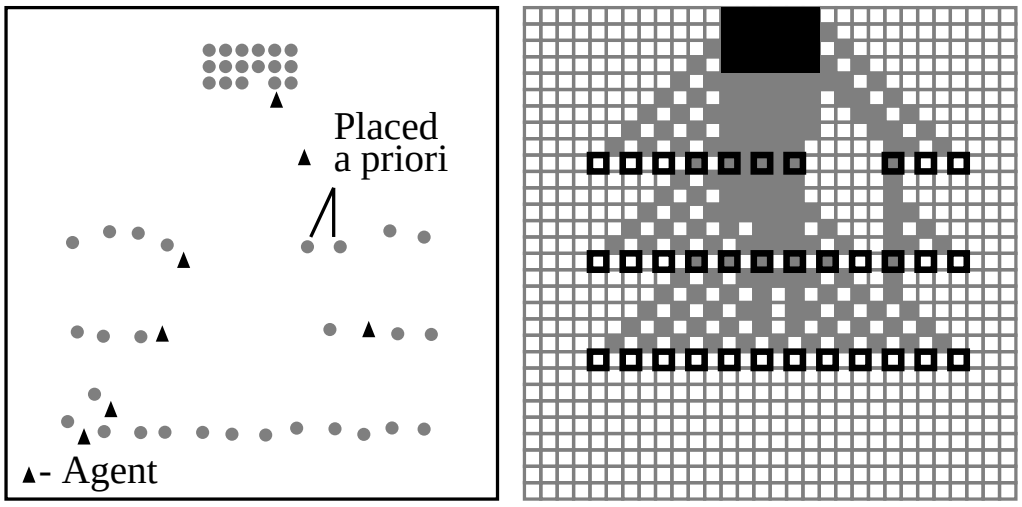


Figure 8.12: Interconnections between Matching grid, ESMs, and navigation maps: the lines between the matching grid and the ESMs represent links between every corresponding pair of neurons. The Configuration ESM shows that a row of bricks have to be built, while the Brick ESM shows that bricks are in two clusters. The activations on the matching grid show the paths between the clusters of bricks and the row of drop-sites. These activations inhibit the corresponding nodes in the Configuration navigation map and excite the corresponding nodes in the Brick navigation map.



(a) (b)

Figure 8.13: Environment with one brick source and three walls to be built. (a) The agents have already built the bottom-most wall, and parts of the other two walls. (b) match values on the nodes of the matching grid of an agent at the 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 $match[n] = 1$.

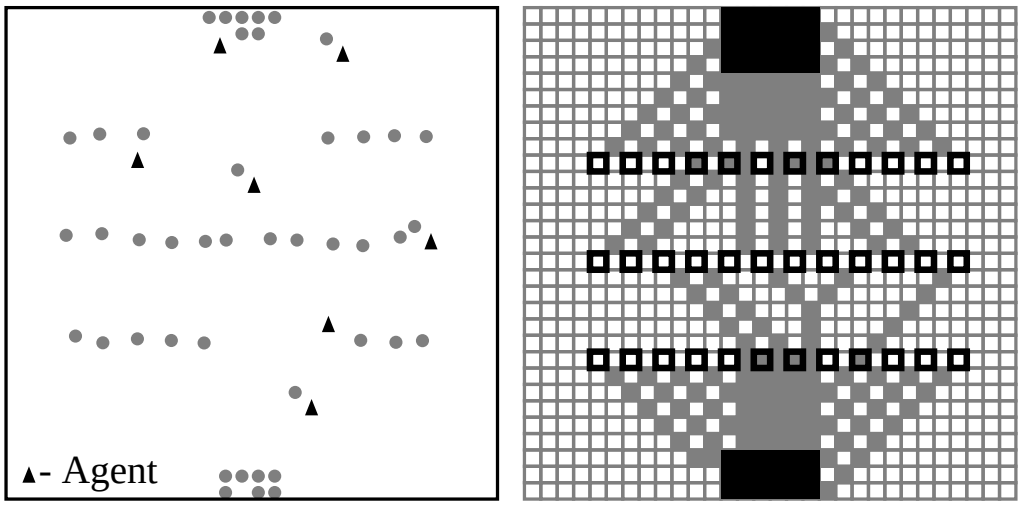
8.4.1 Example 1 of Sequencing Algorithm

Figure 8.13(a) shows an environment where three parallel walls are to be built and there is one source of bricks. Two of the bricks were placed *a priori*. Figure 8.13(b) shows the final match values after running the calculate_matching algorithm on the matching grid of an agent at the start of construction. The match 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.

The sequencing algorithm fills the farthest wall first (since all the nodes corresponding to this wall have match value 0 as indicated by the unfilled squares with thick edges in figure 8.13(b)), then the middle wall and finally the nearest wall (nodes corresponding to this wall have match value 1 as indicated by the filled squares with thick edges in figure 8.13(b)). The greedy algorithm would have filled these rows of drop-sites in the reverse order requiring the agents to move around the walls placed first. Figure 8.13(a) shows that the agents have already built the bottom-most wall, and parts of the other two walls.

8.4.2 Example 2 of Sequencing Algorithm

Figure 8.14(a) shows an environment where three parallel walls are to be built and there are two sources of bricks on either side. Figure 8.14(b) shows the final 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 match values set to 1 (indicated by filled squares with dark edges in figure 8.14(b)) while the match value of the nodes corresponding to the middle wall is 0 (indicated by unfilled squares with dark edges in figure 8.14(b)). Therefore, in this environment, the sequencing algorithm fills the middle wall first, runs the calculate_matching algorithm again to recompute the activations on the matching grid and then fills the top and bottom walls (in figure 8.14(a) the agents have already built the middle wall, and parts of the other two



(a) (b)

Figure 8.14: Environment with two brick sources and three walls to be built. (a) The agents have already built the middle wall, and parts of the other two walls. (b) Activations on the matching grid of an agent at the 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 $match[n] = 1$.

walls). The greedy algorithm would have filled the two outer walls first requiring the agents to move around these outer walls to reach the middle wall.

The simulation results show that planning the sequence of disc placement reduces the total distance traveled to complete the construction task. Implementing good heuristic solutions to the MWPM problem with Euclidean weights directly on the spatial representation is a convenient way of reducing interference between agents without the need for communication. The algorithm also takes into account obstacles on paths between discs and drop-sites. All computations are performed using only local interactions between neighboring nodes and hence this algorithm can be efficiently implemented in parallel systems. In this work, the environment contained only bricks (no food or water discs were present). An extension of this work could include food and water discs as obstacles when spreading activation in the calculate_matching algorithm of figure 8.7.

8.5 Related Work

Neural networks have been proposed to solve combinatorial optimization problems such as the matching problem and the Traveling Salesman problem. The neural networks are in the form of continuous valued Hopfield nets [Tank and Tank, 1985], stochastic simulated annealing [Kirkpatrick *et al.*, 1983], and mean-field annealing [Peterson and Anderson, 1987]. For example, to solve the minimum matching problem, each node in a Hopfield network would represent a possible matched pair of elements [Kung, 1993]. The weighted links between the nodes encode the constraints of the problem. The dynamics of the networks then minimize a function that contain terms corresponding to the total weight of matched pairs and the constraints (an element can be matched to exactly one element from the other set). These techniques require the weights to be calculated *a priori* for every problem instance. The number of nodes in such networks is equal to the square of the number of elements in each set (since a node represents a possible mapping between two elements). In the system presented in this report, each ESM has $100 \times 100 = 10^4$ neurons and hence it is infeasible to solve the perfect matching problem by any of the above methods. Moreover, these networks

do not give good solutions as the problem size increases – Wilson and Pawley show that Hopfield networks often give solutions far from optimal [1988]. This is due to the presence of hard constraints in the energy function [Gee *et al.*, 1993]. Also, none of these techniques are guaranteed to give optimal or even feasible solutions.

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 are robust to unexpected changes to the structures being built and to individual failures [Deneubourg and Goss, 1989; Bonabeau *et al.*, 1997]. The main features of such groups that make these tasks possible are positive feedback (good solutions like 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]. These principles have 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 *et al.*, 1999], and for other problems like sorting [Deneubourg *et al.*, 1991; Holland and Melhuish, 1999] and network routing [Caro and Dorigo, 1998]. (Sorting was demonstrated by having a group of robots with minimal sensing capability separate randomly scattered objects into separate piles based on their type. This is similar to the behavior of ants who group their larvae together). However, these are all stochastic approaches (similar to simulated annealing) and will not always lead to optimal solutions.

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 more than longer ones and the gradient of pheromone gives a path leading to home. Spreading activation initially in all directions and then reinforcing the most desirable paths is used in applications that require some form of path planning such as data dispersion in sensor networks [Intanagonwiwat *et al.*, 2000]. Similar to the ConAg-TS architecture, activation is spread in the forward direction from a single source node and a reverse activation is then directed against the gradient of the forward activation.

Spreading activation is also used when large numbers of agents (physical or simulated robots) are to be controlled. For instance, Lewis and Bekey present a simulated 2-dimensional grid world which is an abstraction of a biological brain [1992]. *Micro-robots* in this world have to detect an irregularly shaped "tumor". This "tumor" is initially recognized during a random walk by an agent which then spreads a chemical enabling other agents to move to the tumor site directly. In [Payton *et al.*, 2001], physical robots first spread themselves in an unknown environment. The robots then spread activation to neighboring robots by broadcasting messages until the activation reaches a human who is then able to navigate through the environment. Explicit communication between robots was used because of the distance between robots. However, if robots are very close together, then only minimal sensing capabilities are sufficient for communication. Werger and Mataric describe a society of robots that has only contact sensors and uses each others' physical bodies to forage – robots arrange themselves to form chains between "home" and "food" destinations so that other robots can follow this chain to move the food to home [1996]. The robots forming the chain also maintain statistics on the number of food objects that have been foraged so that the position of the chain can be modified over time.

A shared *localization space* between agents is a representation of an agent's position within the shared environment such that the representations of different agents are correlated [Vaughan *et al.*, 2000b]. The ESMs in the ConAg-TS architecture are a shared localization space because each agent maintains its ESM independently of the other agents but the positions of discs in the ESMs of different agents are correlated since they share the same space (the positions do not correspond exactly due to the limited sensing range and sensor and odometry errors). Though the ConAg-TS architecture uses spreading activation to coordinate the actions of the agents, the main difference from the systems described earlier is that the activation is spread on the ESMs (the shared localization space) and not directly with other agents or the environment. These two approaches are compared below:

1. If agents do not have the ability to explicitly communicate with each other, then they will have to modify the environment as a form of stigmergetic communication (for example, laying pheromone trails or coloring discs [Crabbe and Dyer, 1999a]). Moreover, in some cases it is disadvantageous to mark the environment (for example, coloring a disc requires new behaviors from the agent; robots with touch sensors will have to be close to each other) and in such cases it is preferable to spread activation on

the ESM rather than among the agents themselves.

2. Complex structures can be built by distributed agents without a spatial representation. Deneubourg *et al.* present a simulated 2-dimensional grid world where a *swarm* of agents build structures whose shapes are inspired by wasp's nests. These construction agents have purely reactive behaviors that depend only on the occupancy of the four neighboring cells of an agent (there is no internal representation). In this respect, these creatures are simpler than ConAg-TS agents. However, when the shape of the structure to be built is encoded within each agent (the Configuration ESM), the resulting structure is not dependent on the number of agents. But if only local rules are used, then the shape of the structure varies with the number of agents [Therauluz and Bonabeau, 1995; Deneubourg *et al.*, 1992].

3. In ConAg-TS, it is easy for the designer to specify the shape of the structure to be built by just exciting the corresponding nodes in the Configuration ESM. However, in systems that rely only on local interactions it is not clear what individual behaviors will collectively lead to a particular structure. Moreover, using the Configuration ESM any arbitrary 2-dimensional shape can be specified.

4. The greedy heuristic that is used in stigmergetic systems leads to good solutions only in certain cases (such as collectively finding shortest paths in an ant colony). In the case of construction, a greedy approach leads to inefficient construction for certain shapes (such as that in figure 8.1). Planning the order in which bricks have to be placed requires knowledge of the current positions of all the discs and this is available in the internal spatial representation.

Chapter 9

Communication to Reduce Map Errors

9.1 Introduction

All the algorithms described for construction depend on the accuracy with which discs in the environment are represented in the environment. An agent with inaccurate information in its maps will not be able to navigate safely, i.e, without colliding with discs present around it. Moreover, for the construction task in a multi-agent scenario, accuracy of the spatial maps is important for two reasons:

1. To identify parts of the structure that are missing bricks, or bricks that are not part of the structure, an agent matches its Configuration and Brick ESM at every time-step. If these ESMs are not aligned (corresponding cells on the ESMs do not represent the same location in the environment), then not all matching brick and drop-site pairs will be identified. Thus, the agent will erroneously pick up bricks that are already part of the structure or try to drop bricks at locations that already contain a brick.

2. In a multi-agent system, all the agents are trying to build the same structure. Hence, the Configuration ESMs (that encode the shape of the structure to be built) of the different agents should always agree in the locations of the drop-sites of the bricks. Thus it is not enough for an agent to maintain the relative shape of the structure to be built in its Configuration ESM, but the locations of each of the drop-sites will have to match with the corresponding locations of other agent's Configuration ESMs.

Agents get information about their surroundings from two kinds of sources: sensors and odometry information from their motors. However, both these sources supply spatial information with a certain amount of error. Some of these errors can be reduced by careful calibration of the sensors and motors, but this reduces the agent's ability to perform well in new environments. Also the physical aspects of real world sensors make it impossible to eradicate errors from their readings completely. Thus, if the raw sensory and odometry information is incorporated into the spatial maps, these errors will be added to the locations of discs represented in the maps. Hence, a mechanism that reduces the effect of sensor and motor errors on the accuracy of the spatial map is necessary.

The problem of map building and localization in noisy environments has been extensively studied in the robotics field [Elfes, 1987; Kaelbling *et al.*, 1996a; Burgard *et al.*, 1996]. However, most of this work has been done for static environments. The approaches dealing with dynamic environments [Fox *et al.*, 1999b] assume that the bulk of the sensor readings are obtained from interaction with static objects and that the dynamic objects introduce noise and ambiguity into the system. However, the construction environment is highly dynamic since all discs may be moved by agents. In this scenario, it is not possible to globally recalibrate the position of the agent based on the stored map. Moreover, a "fuzzy" representation of space (such as evidence grids [Moravec and Elfes, 1985]) cannot be used since the positions of discs have to be continuously matched to the specified pattern to identify goal locations *to* which the agent has to navigate (i.e, they are not used only for obstacle avoidance).

In this chapter, communication between agents is added so that agents can exchange their spatial maps. The spatial maps received from other agents are incorporated into the agent's own maps to improve accuracy. The locations of the discs are stored as precise points on the map with an associated expected deviation. The sensor and odometry information is incorporated into the map using a Kalman filter. The resulting architecture is termed ConAg-COM.

9.2 Random Errors

The sensors add random errors that are normally distributed around the true position of the sensed disc. The deviation increases linearly with the distance. The information returned by the sensors is represented in a grid. Let the space that can be sensed by the sensors be divided into a grid of size $S \times S$ of uniform square cells. Let $S = 2s + 1$ and the cells be labeled from $(-s, -s), \ldots, (0, 0), \ldots, (s, s)$ where cell $(0, 0)$ contains the agent's current position. The occupancy of each cell (i, j) is denoted by o^s_{ij}. $o^s_{ij} = 1$ if a disc is sensed in the space represented by cell (i, j) and 0 otherwise. The position of the disc in each cell is returned as a two-dimensional vector $\vec{p}^s_{ij}$ (if $o^s_{ij} = 0$, $\vec{p}^s_{ij}$ is the midpoint of the cell). The sensors cannot return the position of more than one disc within a cell; thus the cell size specifies how finely space is sensed. The expected standard deviation of the position is $\sigma^s_{ij} = \sigma_s|\vec{p}^s_{ij}|$ where σ_s is a measure of the accuracy of the sensors. Thus, three pieces of information are returned for each cell (i, j): o^s_{ij}, $\vec{p}^s_{ij}$, and σ^s_{ij}.

The motors return the distance and direction, $\vec{d}$, moved in every time step (dead reckoning input). The dead reckoning data also has random errors normally distributed around the true distance moved by the agent. The uncertainty increases with the distance moved: the deviation is $\sigma_d|\vec{d}|$ where σ_d is a measure of the accuracy of the motors.

Agents can exchange their spatial maps with other agents that are within communication range. The distance and direction of the sending agent (B) is measured by the receiving agent (A). This measurement has a normally distributed random error with deviation σ_{BA} that is proportional to the distance between the agents: $\sigma_{BA} = \sigma_c|\vec{p}_{BA}|$, where $\vec{p}_{BA}$ is the position of B as measured by A and σ_c is a measure of the accuracy with which the distance and direction to B can be sensed.

9.3 Updating Spatial Maps

Let the size of an ESM grid be $M \times M$, where $M = 2m+1$ and the cells be labeled $(-m, -m), \ldots, (0, 0), \ldots, (m, m)$ (cell $(0, 0)$ corresponds to the center of the space represented by the map and the agent's current location). Each cell of an ESM m in a ConAg-COM agent stores three pieces of information o^m_{ij}, $\vec{p}^m_{ij}$, and σ^m_{ij} corresponding to those on the sensors. σ^m_{ij} is thus a measure of the confidence in the occupancy of cell (i, j).

The spatial map is updated from three sources: sensors, dead reckoning data and exchange of maps with other agents. Each of these sources has an associated normally distributed random error with known deviations. The uncertainty in the locations of discs in the ESM is also represented as a deviation. If the three sources of data are assumed to be independent, a Kalman filter [Kalman, 1960] can be used to merge this data into the spatial map. Let two independent measurements of the location of a disc return vectors $\vec{p_1}$ and $\vec{p_2}$ with corresponding standard deviations, σ_1 and σ_2. Then the best estimate of the position $\vec{p}$ given these two measurements also has a normal distribution with deviation σ given by [Maybeck, 1979]:

$$\vec{p} = merge(\vec{p_1}, \sigma_1, \vec{p_2}, \sigma_2) = \frac{\sigma_2^2\vec{p_1} + \sigma_1^2\vec{p_2}}{\sigma_1^2 + \sigma_2^2} \tag{9.1}$$

$$\sigma = decrease(\sigma_1, \sigma_2) = \frac{\sigma_1\sigma_2}{\sqrt{\sigma_1^2 + \sigma_2^2}} \tag{9.2}$$

The update rules for each source of spatial information are described in this section. The difference between these update rules and those described for ESMs in chapter 3 is the use of the standard deviation in the sensor, odometry reading, and ESM cells as a measure of confidence in the measured or stored positions. In other words, in chapter 3 the ESM update procedures assumed that the sensors and odometers returned data with no errors (deviation of 0) and hence the ESM used these values directly. Since random errors

are added in to the sensor readings, the accuracy of the ESMs decreases with time. The errors in the ESM could increase to a point where the activations in the Brick and Configuration ESMs are not aligned with each other causing the agent to move to bricks that already form part of a structure. The techniques described in this section reduce the effect of random sensory and odometry errors on the accuracy of the ESMs (the effect of sensory errors is less pronounced on reactive behaviors since these do not depend on an internal representation that can accumulate errors over time and readings from different color sensors are not compared to each other).

9.3.1 Sensor Update

The environment for the multi-agent construction task is inherently highly dynamic since agents actively move discs. Thus, sensory readings may not match the stored position information either due to sensor and odometry errors or due to an actual change in the position of a disc. Any procedure that updates the spatial map has to distinguish between these two cases. A simple update procedure is obtained if it is assumed that the errors are small enough that if a disc has not been moved then the location of that disc returned by the sensors at different time-steps will all fall in neighboring cells and this is described below for all three sources of information. Denote by $nb((i,j))$ the set of cell (i,j) and all its eight neighboring cells:

$$nb((i,j)) = \{(i-1,j-1),\ldots,(i,j),\ldots,(i+1,j+1)\} \tag{9.3}$$

At every time-step, the sensors take a snapshot of the world around the agent and the activations on the sensor grid, s, are integrated into the center portion of a disc ESM, m. The procedure for integrating the sensor activations is given in figure 9.1 and illustrated in figure 9.2. The updated values of o_{ij}^m, σ_{ij}^m, and $\vec{p}_{ij}^{\,m}$ after sensing are denoted by $o_{ij}^{m'}$, $\sigma_{ij}^{m'}$, and $\vec{p}_{ij}^{\,m'}$ respectively. The position of a disc whose coordinates are stored in cell (i,j) of ESM m is updated with the position vector that is closest to $\vec{p}_{ij}^{\,m}$ from all the sensor grid position vectors in $nb((i,j))$ (line 9) and σ_{ij}^m is decreased (confidence in occupancy of cell increases; line 10). If all the sensor grid cells in $nb((i,j))$ are unoccupied, then this implies that the disc was moved (line 3). o_{ij}^m is set to 0 and σ_{ij}^m is set to the sensor deviation, σ_{ij}^s (lines 5, 6). Similarly, if $o_{ij}^s = 1$ but all ESM cells in $nb((i,j))$ are unoccupied (line 20), then a disc was moved into the cell and o_{ij}^m, $\vec{p}_{ij}^{\,m}$, and σ_{ij}^m are set from the corresponding sensor grid cell to reflect this change (lines 21-23).

9.3.2 Odometry Update

The dead reckoning information from each time-step, $\vec{d}$, is used to maintain egocentricity of both the disc and Configuration ESMs. $\vec{d}$ is subtracted from $\vec{p}_{ij}^{\,m}$ for all cells (i,j) in m. If $\vec{p}_{ij}^{\,m} + \vec{d}$ is not contained in (i,j), then it is assigned to the appropriate neighboring cell. Thus, the activations on each cell are passed to neighboring cells in a direction opposite to which the agent has moved. This is illustrated in figure 9.3. To account for the uncertainty in $\vec{d}$, the variances are added ($\sigma_{ij}^{m'}$ is the confidence after the movement update):

$$(\sigma_{ij}^{m'})^2 = (\sigma_{ij}^m)^2 + (\sigma_d|\vec{d}|)^2 \tag{9.4}$$

9.3.3 Communication Update

Including a spatial map sent from another agent into an ESM is similar to incorporating sensory data. However, the areas represented by the received and stored maps are different and the cells are offset by an amount proportional to the distance between the two agents. Let the receiving agent be labeled A and the sending agent, B. Let $\vec{p}_{BA}$ denote the position of B with respect to A as sensed by A. The deviation in this measurement is $\sigma_{BA} = \sigma_c|\vec{p_{BA}}|$. Consider a disc whose location is included in both A's ESM, m_A and B's ESM, m_B. Let this disc be present in the area represented by cell (i,j) of m_B and in the area represented by cell (k,l) of m_A (figure 9.4).

Let the updated values of $o_{ij}^{m_A}$, $\sigma_{ij}^{m_A}$, and $\vec{p}_{ij}^{\,m_A}$ after including m_B be denoted by $o_{ij}^{m'_A}$, $\sigma_{ij}^{m'_A}$, and $\vec{p}_{ij}^{\,m'_A}$ respectively. The update procedure is given in figure 9.5. It is assumed that the error in $\vec{p}_{BA}$ is small enough that the location of a disc as calculated from both the received (m_B) and stored map (m_A) fall in

Input: o^m_{ij}, σ^m_{ij}, $\vec{p}^m_{ij}$, $(i,j) \in \{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$
o^s_{ij}, σ^s_{ij}, $\vec{p}^s_{ij}$, $(i,j) \in \{(-s,-s),\ldots,(0,0),\ldots,(s,s)\}$
Output: $o^{m'}_{ij}$, $\sigma^{m'}_{ij}$, $\vec{p}^{m'}_{ij}$, $(i,j) \in \{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$

1: for (i,j) in $\{(-s,-s),\ldots,(0,0),\ldots,(s,s)\}$ do
2: if $(o^m_{ij} = 1)$
3: if $(o^s_{i'j'} = 0 \forall (i',j') \in nb((i,j)))$
4: {all neighboring cells are empty; remove disc}
5: $o^{m'}_{ij} := 0$
6: $\sigma^{m'}_{ij} := \sigma^s_{ij}$
7: else {update $\vec{p}^m_{ij}$ from closest vector in $nb((i,j))$}
8: $o^{m'}_{ij} := 1$
9: $(i_m, j_m) := argmin_{(i',j') \in nb((i,j))}(|\vec{p}^m_{ij} - \vec{p}^s_{i'j'}|)$
10: $\sigma^{m'}_{ij} := decrease(\sigma^m_{ij}, \sigma^s_{i_m j_m})$
11: $\vec{p}^{m'}_{ij} := merge(\vec{p}^m_{ij}, \sigma^m_{ij}, \vec{p}^s_{i_m j_m}, \sigma^s_{i_m j_m})$
12: end if
13: end if
14: if $(o^m_{i'j'} = 0 \forall (i',j') \in nb((i,j)))$ {surrounded by unoccupied cells}
15: if $(o^s_{i'j'} = 0 \forall (i',j') \in nb((i,j)))$
16: {corresponding sensor grid cells are unoccupied}
17: $o^{m'}_{ij} := 0$
18: $\sigma^{m'}_{ij} := decrease(\sigma^m_{ij}, \sigma^s_{i_m j_m})$
19: end if
20: if $(o^s_{ij} = 1)$ {disc in sensor grid cell; add disc}
21: $o^{m'}_{ij} := 1$
22: $\sigma^{m'}_{ij} := \sigma^s_{ij}$
23: $\vec{p}^{m'}_{ij} := \vec{p}^s_{ij}$
24: end if
25: end if
26: end for

Figure 9.1: Procedure for update from sensors.

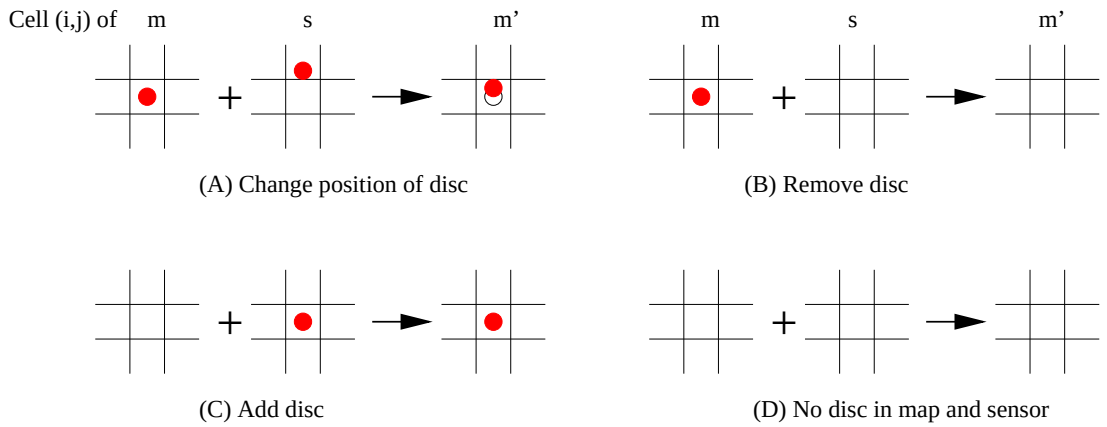


Figure 9.2: Sensory update rules: Occupancy of corresponding cells in the ESM (m), sensor grid (s) and updated ESM (m') obtained from incorporating s into m are shown. (A) Cell (i, j) of m contains a disc and a neighboring cell in s: the position is updated to their average. (B) Cell (i, j) of m contains a disc, but cell (i, j) of s is empty: the disc is removed. (C) Cell (i, j) of m is empty, but cell (i, j) of s contains a disc: add a disc. (D) Cell (i, j) of m is empty and so are the cells in s: the confidence is increased.

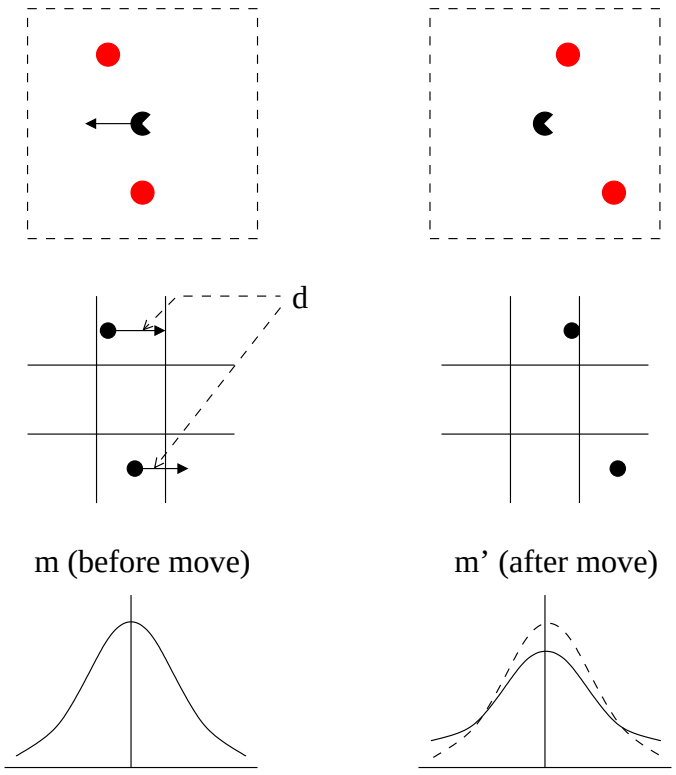


Figure 9.3: Odometry update rules: An agent and two discs. The corresponding ESM is shown below (darkened circles indicate occupied cells). As the agent moves to the left the activations on the ESM are shifted proportionally to the right ($\vec{d}$). The new activation may move to a neighboring cell (top disc) or remain in the same cell (bottom disc). The central node always represents the location of the agent. The Gaussians drawn below the ESM is an indication of the confidence before and after the odometry update. The deviation increases after the update to show the reduced confidence.

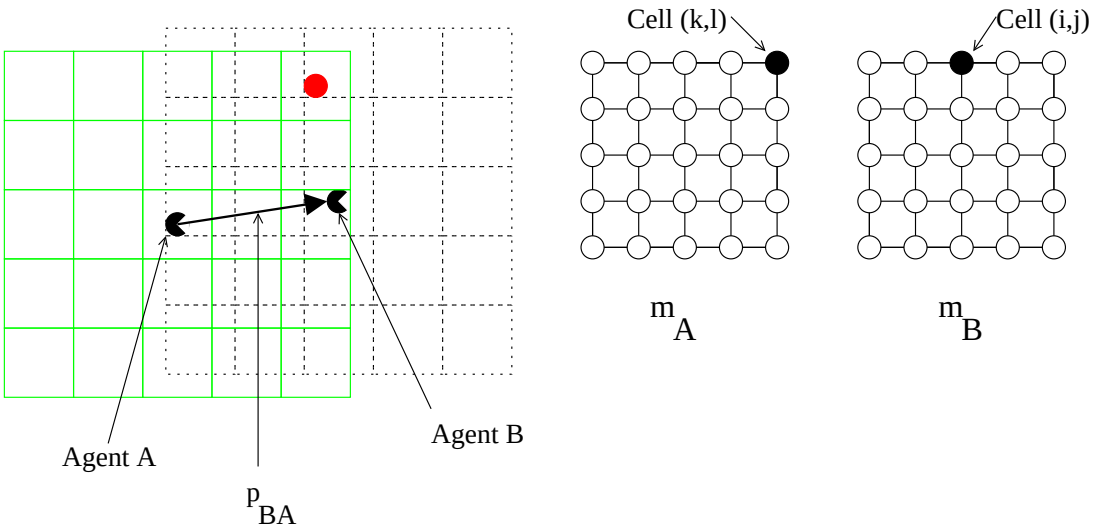


Figure 9.4: Communication update: Two agents A and B store the location of the same disc in different cells of their ESMs. The solid (dashed) grid shows how space is divided into uniform squares by agent A (B). The activations on the corresponding ESMs, m_A and m_B, are shown to the right.

neighboring cells ($\vec{p}_{ij}^{\,m_B} + \vec{p}_{BA}$ is located in a neighboring cell of (k,l) in m_A; line 2). In this case, $\vec{p}_{kl}^{\,m_A}$, is updated to bring it closer to $\vec{p}_{ij}^{\,m_B} + \vec{p}_{BA}$ (line 7). To account for the error in measuring $\vec{p}_{BA}$, σ^2_{BA} is added to all the variances stored in m_B (using function $f(x,y) = \sqrt{x^2+y^2}$). If all neighboring cells of (k,l) are labeled unoccupied by m_A (line 16), then it means m_B contains the coordinates of a disc not present in m_A. Cell (k,l) is updated with this new disc only if the confidence in the received map is greater than that in the stored map ($(\sigma_{ij}^{m_B})^2 + \sigma^2_{BA} < (\sigma_{kl}^{m_A})^2$; line 23).

The above communication update rule does not address the problem of "double counting": using another robot's ESM repeatedly. If two robots exchange their maps often without incorporating sensor data (using the same evidence more than once), then they will reinforce each other's confidence in their maps without regard to the accuracy of the positions stored in them. This problem is alleviated by keeping track of when a map was received from an agent and accepting that agent's map only after a certain number of time-steps have passed. This approach has also been taken in [Fox *et al.*, 2000].

9.4 Results

The performance of a ConAg-COM agent is measured by counting the average number of mismatches between the ESM and the actual location of the discs. A mismatch is an occupied cell in the ESM which does not have a corresponding disc in the environment or a disc that is within the ESM range but the corresponding cell is represented as unoccupied.

All the discs are located within an area of 100×100 units. In this experiment, the size of the ESM is 51×51 and the sensor grid is 11×11. Each cell corresponds to a unit square in the world centered around the agent. The agents move a distance of approximately 1 unit per time-step (slows during turns). Both the initial positions of the discs and the pattern to be constructed are generated randomly. The world contains 125 discs (at random locations throughout the world) while the pattern contains 25 locations from the middle of the world. In this way, even agents at the edge of the pattern location area have approximately the same number of discs within ESM range. A portion of one random instance of the world is shown in figure 9.6.

Figure 9.7(a) shows the decrease in map error with time for 1 (no communication), 5, 10 and 15 agents. $\sigma_s = \sigma_d = \sigma_c = 0.05$ (the sensory, odometry and communication distance errors are within $\pm 15\%$ of the measured distances 99.7% of the time). The maximum distance between agents within which they can

Input: $o_{ij}^{m_A}$, $\sigma_{ij}^{m_A}$, $\vec{p_A}_{ij}^{m}$, $(i,j) \in \{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$
$\vec{p}_{BA}$, $o_{ij}^{m_B}$, $\sigma_{ij}^{m_B}$, $\vec{p_B}_{ij}^{m}$, $(i,j) \in \{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$
Output: $o_{ij}^{m'_A}$, $\sigma_{ij}^{m'_A}$, $\vec{p}_{ij}^{m'_A}$, $(i,j) \in \{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$

1: for (i,j) in $\{(-m,-m),\ldots,(0,0),\ldots,(m,m)\}$ do
2: (k,l) is cell in m_A that contains $\vec{p}_{ij}^{m_B} + \vec{p}_{BA}$
3: if $(o_{kl}^{m_A} = 1)$
4: if $(o_{i'j'}^{m_B} = 0 \forall (i',j') \in nb((i,j)))$
5: {all neighboring cells empty; remove disc if confidence is greater}
6: if $(\sigma_{kl}^{m_A} > f(\sigma_{ij}^{m_B}, \sigma_{BA}))$
7: $o_{kl}^{m'_A} := 0$; $\sigma_{kl}^{m'_A} := f(\sigma_{ij}^{m_B}, \sigma_{BA})$; $\vec{p}_{kl}^{m'_A} := \vec{p}_{ij}^{m_B} + \vec{p}_{BA}$
8: endif
9: else {update $\vec{p}_{kl}^{m_A}$ from closest vector in $nb((i,j))$}
10: $o_{kl}^{m'_A} := 1$
11: $(i_m, j_m) := argmin_{(i',j') \in nb((i,j))}(|\vec{p}_{kl}^{m_A} - (\vec{p}_{i'j'}^{m_B} + \vec{p}_{BA})|)$
12: $\sigma_{kl}^{m'_A} := decrease(\sigma_{kl}^{m_A}, f(\sigma_{ij}^{m_B}, \sigma_{BA}))$
13: $\vec{p}_{kl}^{m'_A} := merge(\vec{p}_{kl}^{m_A}, \sigma_{kl}^{m_A}, \vec{p}_{i_m j_m}^{m_B}, f(\sigma_{ij}^{m_B}, \sigma_{BA}))$
14: end if
15: end if
16: if $(o_{k'l'}^{m_A} = 0 \forall (k',l') \in nb((k,l)))$
17: {cell (k,l) is surrounded by unoccupied cells}
18: if $(o_{i'j'}^{m_B} = 0 \forall (i',j') \in nb((i,j)))$
19: {the corresponding cells in m_B are unoccupied}
20: $o_{kl}^{m'_A} := 0$; $\sigma_{kl}^{m'_A} := decrease(\sigma_{kl}^{m_A}, f(\sigma_{ij}^{m_B}, \sigma_{BA}))$
21: end if
22: if $(o_{ij}^{m_B} = 1)$ {disc in m_B; add disc if confidence is greater}
23: if $(\sigma_{kl}^{m_A} > f(\sigma_{ij}^{m_B}, \sigma_{BA}))$
24: $o_{kl}^{m'_A} := 1$; $\sigma_{kl}^{m'_A} := f(\sigma_{ij}^{m_B}, \sigma_{BA})$; $\vec{p}_{kl}^{m'_A} := \vec{p}_{ij}^{m_B} + \vec{p}_{BA}$
25: end if
26: end if
27: end if
28: end for

Figure 9.5: Procedure for update from communication. Define $f(x,y) = \sqrt{x^2 + y^2}$.

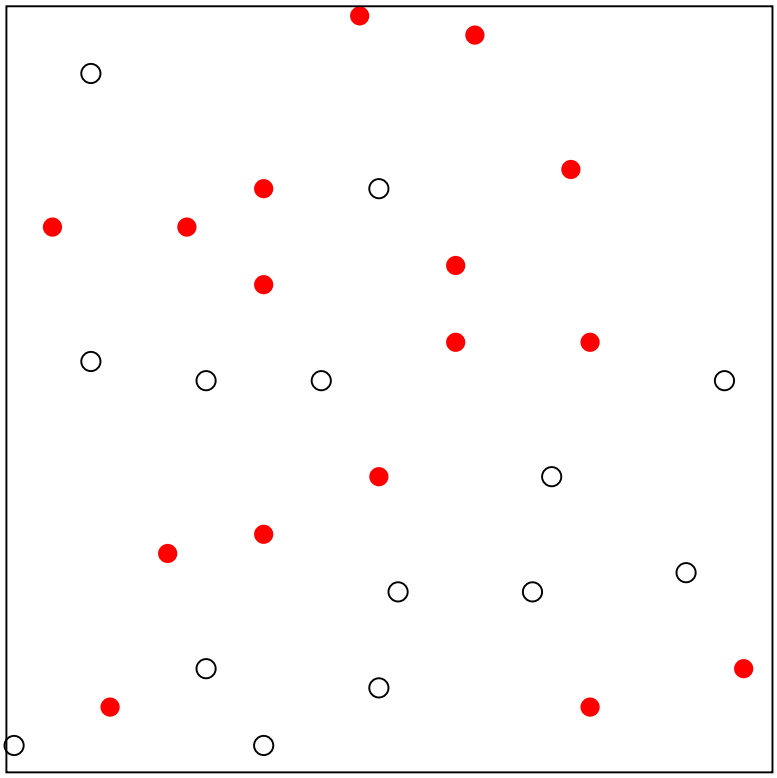


Figure 9.6: A portion of one random instance of the environment and pattern of the structure to be built. Filled red circles indicate bricks and unfilled black circles indicate a drop-site.

exchange maps was 10 units. A pair of agents do not exchange maps more than once in 10 time-steps. The errors are averaged over 20 trials and figure 9.7(b) shows one standard deviation error bars.

Figure 9.8 shows the affect of the range of communication on the map error. The map error over 100 time-steps is shown when the range is 10, and 15, and 20 units. The number of agents is fixed at 10 and $\sigma_s = \sigma_d = \sigma_c = 0.05$. As the communication range increases, the error in estimating the distance between agents predominates over the benefits of exchanging maps with a larger number of agents.

Figure 9.9 shows the affect of the deviation in the sensory, motor and communication errors. The average map error is plotted for 10 agents when $\sigma_s = \sigma_d = \sigma_c = 0.025, 0.05, 0.05$. All other parameters are kept constant. Since no global recalibration of the odometry error is performed, the map error increases when the error deviation reaches 0.075.

9.5 Discussion

Communication of spatial maps between agents has two advantages:

1. *Reducing sensor and odometry errors:* If the errors introduced by the sensors and motors are random, then increasing the number of measurements and averaging the results will yield a more accurate map. Using a Kalman filter is the optimal method of integrating these measurements. Incorporating the map of another agent is equivalent to including all the sensor readings made by that agent (and other agents with which it has communicated earlier) into the spatial map.

2. *Extending the sensor range:* In the absence of communication, the space represented by an agent is limited by the range of its sensors and the speed at which it moves (to cover new areas). By exchanging spatial maps with other agents, agents can incorporate spatial information of areas that it has not directly sensed.

The simulation results show that the spatial map errors increase rapidly with increase in communication range (since the error in measuring the distance to the sending agent increases with distance). The spatial map errors however decrease with larger number of agents in the environment (since this is effectively increasing the number of measurements available to an agent). Thus, decreasing the communication range while increasing the number of agents prevents the errors in the spatial map from increasing.

9.6 Related Work

The ESMs used in this work are similar to evidence grids [Moravec and Elfes, 1985; Elfes, 1987]. An evidence grid represents space as a grid of uniform cells and was initially formulated to convert sonar scan data into a spatial map. Each cell in an evidence grid stores the probability (or equivalently the odds) that the corresponding space is occupied by some object. The probability of the sensors detecting obstacles is calculated beforehand. Sensor readings can then update the probabilities in each grid cell using Bayes rule and the conditional independence of different sensor measurements:

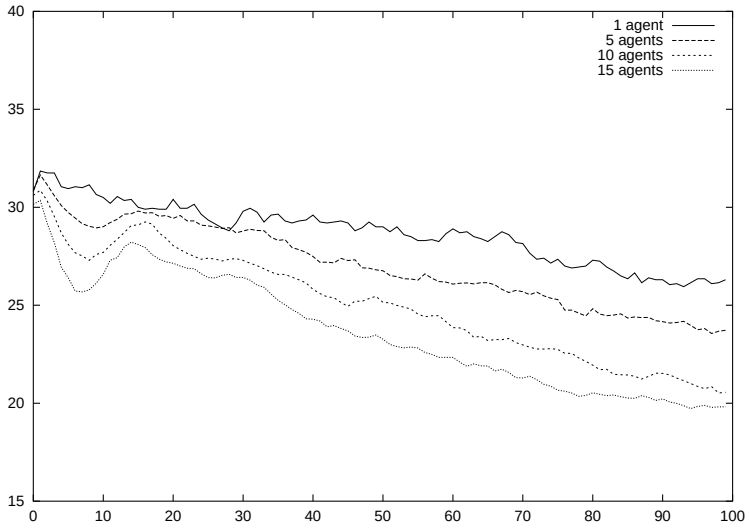
$$\frac{p(o|M_1 \wedge M_2)}{p(\overline{o}|M_1 \wedge M_2)} = \frac{p(o|M_1)p(M_2|o)}{p(\overline{o}|M_1)p(M_2|\overline{o})} = \frac{p(o|M_1)p(o|M_2)p(\overline{o})}{p(\overline{o}|M_1)p(\overline{o}|M_2)p(o)}$$

where M_1 represents all *a priori* measurements, M_2 the latest sensor measurement, and $p(o|M_1)$ and $p(\overline{o}|M_1)$ are the probabilities that the cell is occupied or unoccupied respectively (stored in a cell). If the probabilities are initialized such that $p(o) = p(\overline{o}) = 0.5$, then the sensor update equations can be rewritten as:

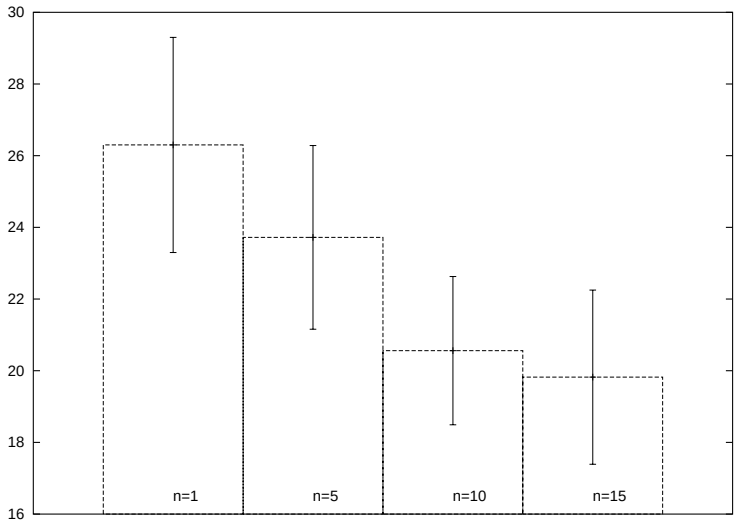
$$O(o|M_1 \wedge M_2) = O(o|M_1)O(o|M_2) \tag{9.5}$$

using the odds form of writing probabilities

$$O(o|M) = \frac{p(o|M)}{p(\overline{o}|M)}$$



(a)



(b)

Figure 9.7: (a) Map error against time for 1, 5, 10 and 15 agents (b) Map error at the end of 100 time-steps with errorbars (at one standard deviation).

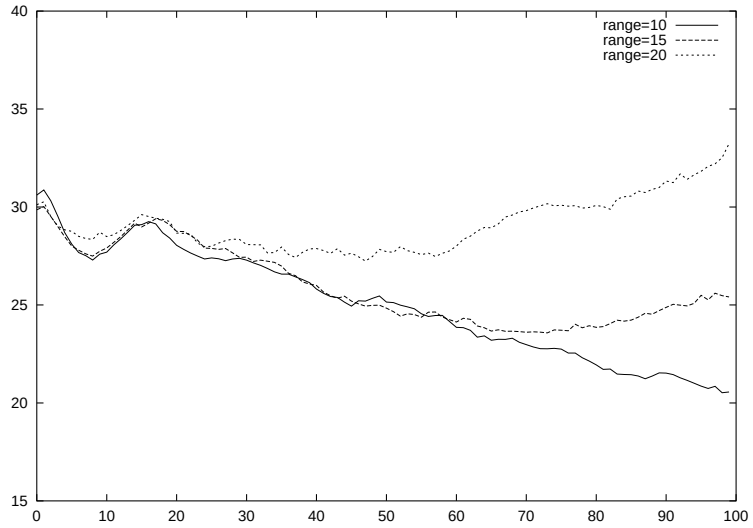


Figure 9.8: Effect of communication range on map error: Map error against time for 10 agents when the range of communication is 10, 15, and 20 units. Averaged over 20 trials.

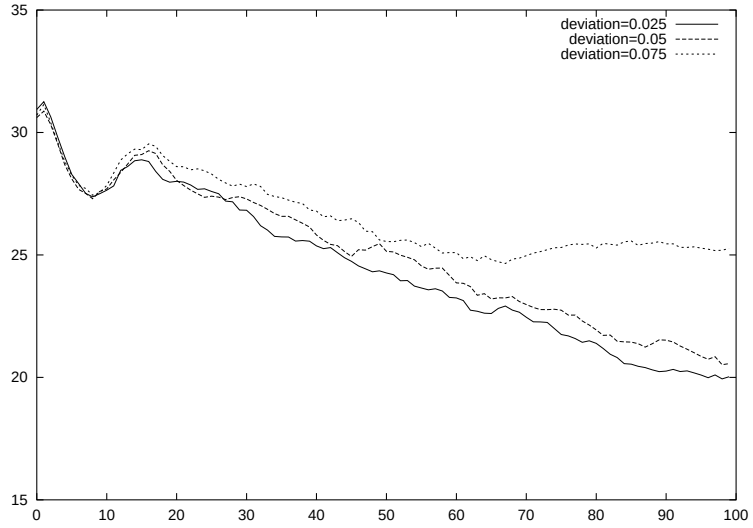


Figure 9.9: Effect of sensor, odometry, and communication errors: Map error against time for 10 agents when $\sigma_s = \sigma_d = \sigma_c = 0.025, 0.05, 0.075$. Averaged over 20 trials.

Equation 9.5 gives an incremental update rule which can be implemented efficiently even for large grid sizes. Since sensor readings are directly incorporated into the spatial map, the location of an obstacle is not restricted to a single cell. Thus matching the map with the construction pattern will not always be possible. However, entire maps can be compared to each other to measure how well they agree [Moravec and Blackwell, 1993] and current sensor readings can be aligned with the stored grid readings to implement place learning [Yamauchi and Langley, 1997].

Evidence grids only specify the location of the objects and not that of the robot itself. Hence, robot localization has to be implemented separately. The most common method is *position tracking* where the initial position of the robot is known and this position is updated at every step. In static environments, the steps involve matching the stored evidence grid with the current sensory input (usually converted into grid form) and using the best match location as the estimate of the robot's position [Yamauchi, 1996]. These systems always maintain a single map and estimate of the robot and hence cannot disambiguate between places that give the same sensor readings. Moreover, past estimates of the map cannot be changed based on newer sensor readings. Therefore, the accuracy of the robot position on the map degrades over time. This matching can be reliably performed by first extracting line segments from the evidence grids Schiele and Crowley compare different techniques for position tracking using evidence grids [1994].

Kalman filtering techniques [Kalman, 1960] represent the position of the robot and landmarks as a Gaussian probability distribution. Since it is necessary to match the sensor readings with stored landmarks, this technique has been used extensively in environments with easily recognizable features such as beacons [Durrant-Whyte and Leonard, 1991]. Lu and Milios give a more sophisticated algorithm that maintains all past sensor information [1997]. All the past sensor readings are then optimally combined together using a maximum likelihood criterion to create a global spatial map.

The Gaussian probability distribution used in Kalman filtering is uni-modal and hence these methods cannot solve the *kidnapped robot* problem - to recompute the position of the robot after it has been moved without its knowledge. Markov localization overcomes this limitation. The internal spatial representation can store either only large-scale topological features [Kaelbling *et al.*, 1996a; Nourbakhsh *et al.*, 1995] or it can be a fine-grained grid [Burgard *et al.*, 1996]. These two kinds of representation have been integrated by partitioning a fine-grained grid into a small number of regions that are connected by topological links [Thrun and Bücken, 1996]. Another method of representing a multi-modal distribution is using particle filters [Fox *et al.*, 1999a]. In this method, called Monte-Carlo localization, the density of particles at a location represents the probability of the robot being situated there. A modification of this method for rapid re-localization is Sensor-Resetting Localization [Lenser and Veloso, 2000].

In a static world, given the robot's current location, future sensor readings are independent of past readings. This is assumed in all the localization methods described above. However this is not true in a dynamic world as in the case of the construction task. Transient changes can be handled in an evidence grid by reducing the occupancy probability [Yamauchi, 1996; Yamauchi and Langley, 1997]. Fox *et al.* describe a filtering technique for Monte-Carlo localization to detect which sensor readings are due to a changed environment [1999b].

Markov localization was extended for map formation among multiple robots [Fox *et al.*, 2000]. A method similar to that used in ConAg-COM was used by Sugiyama and Murata for distributed map building [1995]. Two robots that were in communication range would measure the distance to each other and the maps are communicated to each other to reduce the error in them.

In an egocentric spatial representation like ESMs, the issue of localization is slightly different since the center of the map always corresponds to the current location of the agent. Therefore, changes in position of the robot is reflected by adjusting the position of objects in the spatial map. The localization method used in ConAg-COM is a form of position tracking. The environment for which ConAg-COM was designed for is highly dynamic and comparing corresponding grids of the ESMs of different agents enables an agent to decide which discs have been moved. This will not generally be possible if raw range measurements are stored in the spatial map. However the discs in the ConAg-COM environment can be distinguished by their color and thus the readings stored in an ESM are more "high-level" than that in an evidence grid storing sonar readings. Color has been used elsewhere as a basis of selecting landmarks from a real-world scence [Dodds and Hager, 1997].

Chapter 10

Conclusions and Future Work

This dissertation presents the ConAg series of architectures that enable a group of autonomous simulated agents to construct arbitrary structures in their simulated 2-dimensional environment. The main features of this architecture are that it is behavior-based and includes a spatial representation. The main capabilities of this system are that it can learn the construction sequence, and after learning it can build arbitrary two-dimensional structures specified by the designer. Moreover, the agents build this structure in an efficient way.

The behavior-based architecture integrates both low-level reactive behaviors and higher-level planning behaviors. The fast reactive behaviors enable an agent to fulfill its survival needs, namely eating and drinking periodically. The spatial representation is a grid-based egocentric map that maintains the locations of all the discs that the agent has sensed. This map is used for path planning by spreading activation, an efficient method to calculate a path to the nearest goal and which avoids obstacles. The spatial representation also generates the navigational goals of the agent (locations of bricks, drop-sites, etc.). This is performed by comparing the spatial map (the current state of the world) with the blueprint of the structure to be built. Locations where they differ become goal locations. In this way, even unexpected changes in the structure being built will result in an agent repairing the structure automatically. Thus, this approach has significant advantages over the reactive, stigmergetic approaches to construction which do not easily generalize to arbitrary structures and whose success depend on the initial configuration of objects [Deneubourg *et al.*, 1992; Therauluz and Bonabeau, 1995; Bonabeau *et al.*, 1998; Karsai, 1999].

The architecture has a connectionist action selection and this facilitates learning. An agent can learn the construction sequence by imitating a teacher agent. It can also learn to exploit correlations between discs in the environment through unsupervised Hebbian learning. Learning occurs at every step which is important for an autonomous agent since in the real world reinforcement occurs rarely. The agent is also able to learn an emergent bucket-brigading behavior that reduces interference between agents. This behavior is learned while attempting to break deadlocks between agents with crossing paths.

An agent selects its navigational goals in such a way that it reduces interference with other agents. It also builds the structure in such an order that the bricks placed will not obstruct paths to future goal sites. This also enables it to build certain structures (for example, concentric circles) which would be impossible to build if a certain order is not followed (inner circle has to be completed before the outer circle). These goals are selected either by randomized strategies or by running algorithms that spread activation on the spatial maps. Spreading activations can be performed in parallel on a grid based spatial map for fast implementations.

The sensors and motors found on the simulated agent reflect the properties of physical robots. A method to reduce the impact of random sensory errors on the accuracy of the spatial map is presented. This method also takes advantage of the multiple agents: agents exchange their spatial maps with each other improving the accuracy of the individual maps and also increasing the area of the world that is covered by each map.

There are several directions in which the ConAg architecture may be extended:

1. **Imitation Learning**

   The ConAg sequence learning algorithm imposes strong limitations on the teacher and student (for instance, both have to receive the same sensory input at all times). Relaxing these conditions would

require a more sophisticated mechanism to enable the learner to infer the teacher's actions and goals even if they do not share the same sensory input.

2. **Correlation Learning**

   The ConAg-ST architecture uses an unsupervised Hebbian learning. Comparing this with reinforcement learning for the same task could give useful insights. The large state space that is induced by a continuous real-world environment makes reinforcement learning using the sensor data directly infeasible for a physical robot. A solution would therefore be to use the correlation learning as a means of speeding up reinforcement learning.

3. **Scale-up**

   All the architectures and algorithms demonstrated in this dissertation were carried out with at most 20 agents. The performance of these algorithms have to be studied in the case where the number of interacting agents is large (in the hundreds and thousands). This could also lead to unexpected emergent behaviors.

4. **Construction order**

   The algorithm presented in chapter 8 that enabled the ConAg-TS architecture to compute the order of dropping discs uses the internal spatial representation to spread activations. This algorithm could be modified to instead spread activations among the agents themselves instead of on their spatial maps as in the system of Payton *et al.* [2001]. This would be useful in environment embedded computing application.

5. **Map building with sensor errors**

   Sophisticated probabilistic approaches to map building have recently met with success on physical robots in static environments [Fox *et al.*, 2000; 1999a]. However, approaches to highly dynamic environments such as a construction world are not yet available. This would therefore be a useful line of research.

6. **Integration of architectures**

   This dissertation demonstrated the individual architectures, but did not present a unified architecture. A single architecture could exhibit all the advantages of the individual architectures. However, some of the architectures were designed for different environments. For instance, the ConAg-ST architecture learns to exploit correlations between discs while construction changes the layout of the discs.

7. **Physical Implementation**

   It would be very useful to have a group of physical robots build any structure autonomously after they were given a blueprint of the structure. Current, "construction" robots can only do simpler tasks such as cooperatively carry a beam from one place to another [Pirjanian *et al.*, 2000]. The primary impediment to realizing the ConAg architecture in a physical robot is the quality of currently available sensors and motors.

Appendix

List of Symbols

Agent parameters	
G	Set of green sensors
G_i	Activation of green sensor i
R	Maximum sensor range
f_0	Rate at which internal food level decreases while moving
w_0	Rate at which internal water level decreases while moving
f_1	Rate at which internal food level increases while eating
w_1	Rate at which internal water level increases while drinking
T_0^h	Threshold of food level that activates hunger motivation
T_0^t	Threshold of water level that activates thirst motivation
T_1^h	Threshold of food level that de-activates hunger motivation
T_1^t	Threshold of water level that de-activates thirst motivation
ESM	
N	Number of neurons in an ESM
M	Number of neurons per row in an ESM. $N = M \times M$
a_n	Activation of neuron n
$nb(n)$	Set of neighboring neurons of neuron i
$d(n, m)$	Distance between areas represented by neurons n and m
$gradient(n, m)$	Gradient of activation from n to m. $gradient(n, m) = \frac{a_m - a_n}{d(m,n)}$

ConAg-ST	
m_B^{speed}	Speed motor output of behavior B
m_B^{angle}	Turn angle motor output of behavior B
$m_B^{excitor}$	Excitation at behavior B from motivations
$m_B^{inhibtor}$	Inhibition at behavior B from motivations
s_B	Sensory excitation of behavior B
a_B	Motor activation of behavior B
g_B	Gating at behavior B
T^g	Gating threshold
w_{ij}^{sp}	Weight of spatial correlation link from behavior i to j
w_{ij}^{tmp}	Weight of temporal correlation link from behavior i to j
w_{ij}	$\max(w_{ij}^{sp}, w_{ij}^{tmp})$
T^{sp}	Threshold of sensory excitation in spatial learning rule
T^{tmp}	Threshold of sensory excitation in temporal learning rule
η^{sp}	Spatial correlation learning rate
η^{tmp}	Temporal correlation learning rate
Sequence Learning	
w_{ij}^{excite}	Excitatory weight on link from node i to node j
$w_{ij}^{inhibit}$	Inhibitory weight on link from node i to node j
T^B	Threshold of behavior motor activation
R_j	Reinforcement signal at node j
η^{seq}	Learning rate for sequence learning
ρ^{seq}	Decay factor of learning rate η^{seq}

ConAg-DL	
$f(t)$	Frustration level at time t
T^F	Threshold of frustration level
a_F	Activation of Frustration internal state node
$\vec{d}(t)$	Position of agent at time t
δ	Time taken to test if deadlock is broken or not
d	Minimum distance agent must have moved if deadlock is broken
w_{ib}	Weight on link from internal state node i to behavior b
p_b	Probability of performing behavior b in algorithm learnFrustrated
η^F	Learning rate in algorithm learnFrustrated
d_i	Activation of neuron i of Deadlock Pattern Map
η^D	Learning rate for Deadlock Pattern Map
T^D	Threshold for identifying occupied cells in Deadlock Pattern Map
p_r	Probability of performing random behavior in algorithm Coordination_Learning
ρ_r	Decay rate of p_r
η^{F+}	Learning rate in algorithm Coordination_Learning when deadlock is broken
η^{F-}	Learning rate in algorithm Coordination_Learning when deadlock continues
ρ^{F+}	Decay rate of η^{F+}
ρ^{F-}	Decay rate of η^{F-}
ConAg-TS	
$match[i]$	Activation of node i in Matching Grid

ConAg-COM	
σ_s	Standard deviation of error in sensors per unit distance
σ_d	Standard deviation of error in motors per unit distance
σ_c	Standard deviation of error in measuring position of sender per unit distance
$\vec{d}$	Distance moved by agent in a time-step
$\vec{p}_{BA}$	Position of agent B as measured by agent A
σ_{BA}	Standard deviation of measurement of $\vec{p}_{BA}$
S	Number of cells per row in Sensor grid ($S = 2s + 1$)
o^s_{ij}	Occupancy of sensor grid cell ij
$\vec{p}^s_{ij}$	Position of disc in sensor grid cell ij
σ^s_{ij}	Expected standard deviation of $\vec{p}^s_{ij}$
o^m_{ij}	Occupancy of cell ij in ESM m
$\vec{p}^m_{ij}$	Position of disc in cell ij of ESM m
σ^m_{ij}	Expected standard deviation of $\vec{p}^m_{ij}$

Bibliography

[Alami *et al.*, 1995] R. Alami, J. P. Laumond, and T. Simeon. Two manipulation planning algorithms. In K. Goldberg, D. Halpern, J. C. Latombe, and R. Wolson, editors, *Algorithmic Foundations of Robotics*. A. K. Peters, Boston, Massachusetts, 1995.

[Arkin *et al.*, 1993] Ronald C. Arkin, Tucker Balch, and Elizabeth Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE Press, 1993.

[Arkin, 1989] Ronald C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[Arkin, 1998] R.C. Arkin. *Behavior-based Robotics*. MIT Press, Cambridge, Massachusetts, 1998.

[Arleo and Gerstner, 2000] Angelo Arleo and Wulfram Gerstner. Spatial cognition and neuro-mimetic navigation: a model of hippocampal place cell activity. *Biological Cybernetics*, 83(3):287–299, 2000.

[Avis, 1983] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[Axelrod, 1984] Robert Axelrod. *The evolution of cooperation*. Basic Books, New York, 1984.

[Bakker and Kuniyoshi, 1996] P. Bakker and Y. Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB Workshop on Learning in Robots and Animals*, 1996.

[Balch and Arkin, 1995] Tucker Balch and Ronald C. Arkin. Motor schema-based formation control for multiagent robot teams. In *First International Conference on Multiagent Systems*, pages 10–16, Menlo Park, California, June 1995. AAAI Press.

[Balch and Arkin, 1999] Tucker Balch and Ronald C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, XX(Y), 1999.

[Balch, 2000] Tucker Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3), June 2000.

[Balkenius and Morén, 2000] Christian Balkenius and Jan Morén. A computational model of context processing. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation Of Adaptive Behavior*, pages 256–265, Cambridge, Massachusetts, 2000. MIT Press.

[Barraquand and Latombe, 1991] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

[Bartholdi III and Eisenstein, 1996] John J. Bartholdi III and Donald D. Eisenstein. A production line that balances itself. *Operations Research*, 44(1):21–34, 1996.

[Bartholdi III *et al.*, 2001] John J. Bartholdi III, Donald D. Eisenstein, and Robert D.Foley. Performance of bucket brigades when work is stochastic. *Operations Research*, 49(5):710–719, 2001.

[Beckers *et al.*, 1994] R. Beckers, O.E. Holland, and J. L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Artificial Life IV: Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 181–189, Cambridge, Massachusetts, 1994. Bradford Books/MIT Press.

[Ben-Shahar and Rivlin, 1998] Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation*, 14(4):549–565, 1998.

[Billard and Mataric, 2000] Aude Billard and Maja J. Mataric. Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture. In *Proceedings of the First IEEE-RAS International Conference on Humanoid Robotics*, 2000.

[Birk and Wiernik, 2000] Andreas Birk and Julie Wiernik. Partner-based strategies for a real-world instance of the N-player prisoner's dilemma with continuous degree of cooperation. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, pages 399–405, Cambridge, Massachusetts, 2000. MIT Press.

[Bonabeau *et al.*, 1997] E. Bonabeau, G. Theraulaz, J. L. Deneubourg, S. Aron, and S. Camazine. Self-organization in social insects. *Trends in Ecology and Evolution*, 12:188–193, 1997.

[Bonabeau *et al.*, 1998] E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, N. Franks, O. Rafelsberger, J.-L. Joly, and S. Blanco. A model for the emergence of pillars, walls and royal chambers in termite nests. *Philosophical Transactions of the Royal Society of London B*, 353:1561–1576, 1998.

[Borenstein *et al.*, 1996] J. Borenstein, H. R. Everett, and L. Feng. *Navigating Mobile Robots: Sensors and Techniques*, chapter Odometry and Other Dead-Reckoning Methods. A. K. Peters, Ltd., Wellesley, Massachusetts, 1996.

[Bowling and Veloso, 2002] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[Braitenberg, 1984] Valentino Braitenberg. *Vehicles : Experiments in Synthetic Psychology.* MIT Press, Cambridge, Massachusetts, 1984.

[Brooks *et al.*, 1990] Rodney A. Brooks, Pattie Maes, Maja J Mataric, and Grinnell Moore. Lunar base construction robots. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS-90)*, pages 389–392, 1990.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[Brooks, 1991] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[Burgard *et al.*, 1996] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, 1996.

[Burgess *et al.*, 1994] N. Burgess, M. Recce, and J. O'Keefe. A model of hippocampal function. *Neural Networks*, 7:1065–1081, 1994.

[Cao *et al.*, 1997] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23, 1997.

[Caro and Dorigo, 1998] Gianni Di Caro and Marco Dorigo. AntNet: Distributed stigmergetic control for communication networks. *Journal of Artificial Intelligence Research*, 9:317–265, 1998.

[Cartwright and Collett, 1983] B. Cartwright and T. Collett. Landmark learning in bees. *Journal of Comparitive Physiology A*, 151:521–543, 1983.

[Castano *et al.*, 2000] A. Castano, W.-M. Shen, and P. Will. CONRO: Towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, 2000.

[Cecconi and Parisi, 1992] Federico Cecconi and Domenico Parisi. Neural networks with motivational units. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation Of Adaptive Behavior*, pages 346–355, Cambridge, Massachusetts, 1992. MIT Press.

[Cecconi *et al.*, 1995] F. Cecconi, F. Menczer, and R. K. Belew. Maturation and the evolution of imitative learning in artificial organisms. *Adaptive Behavior*, 4(1):29–50, 1995.

[Chao and Dyer, 1999] Gerald Chao and Michael G. Dyer. Concentric spatial maps for neural network based navigation. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, 1999.

[Chesters and Hayes, 1994] William Chesters and Gillian Hayes. Connectionist environment modelling in a real robot. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 189–197, Cambridge, Massachusetts, 1994. MIT Press.

[Claus and Boutilier, 1998] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Menlo Park, California, 1998. AAAI Press.

[Connell, 1988] Jonathan Connell. Navigation by path remembering. In *Proceedings of SPIE Mobile Robots III*, volume 1007, 1988.

[Coombs *et al.*, 1995] D. Coombs, M. Herman, T. Hong, and M. Nashman. Real-time obstacle avoidance using central flow divergence and peripheral flow. In *Proceedings of the 5th International Conference on Computer Vision*, 1995.

[Cormen *et al.*, 1990] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[Crabbe and Dyer, 1999a] Frederick Crabbe and Michael G. Dyer. Second-order networks for wall-building agents. In *Proceedings of the International Joint Conference on Neural Networks*, 1999.

[Crabbe and Dyer, 1999b] Frederick Crabbe and Michael G. Dyer. Vicarious learning in mobile neurally controlled agents: The V-MAXSON architecture. In *Proceedings of the International Conference on Artificial Neural Networks*, 1999.

[Crabbe and Dyer, 2000] Frederick Crabbe and Michael G. Dyer. Observation and imitation: Goal sequence learning in neurally controlled construction animats: VI-MAXSON. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation Of Adaptive Behavior*, pages 373–382, Cambridge, Massachusetts, 2000. MIT Press.

[Crabbe and Dyer, 2001] Frederick Crabbe and Michael G. Dyer. Goal directed adaptive behavior in second-order neural networks: Leaning and evolving in the MAXSON architecture. *Adaptive Behavior*, 8(2), 2001.

[Crabbe, 2000] Frederick Crabbe. *DiscoTech: Construction and Learning among Neurally controlled agents.* PhD thesis, University of California, Los Angeles, 2000.

[Dautenhahn, 1995] Kerstin Dautenhahn. Getting to know each other – artificial social intelligence for autonomous robots. *Robotics and Autonomous Systems*, 16:333–356, 1995.

[Decugis and Ferber, 1998] Vincent Decugis and Jacques Ferber. An extension of Maes' action selection mechanism for animats. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 153–158, Cambridge, Massachusetts, 1998. MIT Press.

[Deneubourg and Goss, 1989] J. L. Deneubourg and S. Goss. Collective patterns and decision making. *Ecology, Ethology, and Evolution*, 1:295–311, 1989.

[Deneubourg *et al.*, 1991] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 356–365, Cambridge, Massachusetts, 1991. MIT Press.

[Deneubourg *et al.*, 1992] J.-L. Deneubourg, G. Theraulaz, and R. Beckers. Swarm-made architectures. In *Proceedings of the First European Conference on Artificial Life*, pages 123–133, Cambridge, Massachusetts, 1992. Bradford Book/MIT Press.

[Dodds and Hager, 1997] Zachary Dodds and Gregory D. Hager. A color interest operator for landmark-based navigation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. AAAI Press/MIT Press, 1997.

[Donnart and Meyer, 1996] Jean-Yves Donnart and Jean-Arcady Meyer. Spatial exploration, map learning and self-positioning with MonaLysa. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 204–213, Cambridge, Massachusetts, 1996. MIT Press.

[Dorigo *et al.*, 1996] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[Drummond, 1998] C. Drummond. Composing functions to speed up reinforcement learning in a changing world. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 370–381. Springer-Verlag, 1998.

[Duchon, 1996] Andrew P. Duchon. Maze navigation using optical flow. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 224–232, Cambridge, Massachusetts, 1996. MIT Press.

[Durrant-Whyte and Leonard, 1991] H. F. Durrant-Whyte and J.J. Leonard. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3), 1991.

[Elfes, 1987] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3), 1987.

[Enquist, 1985] M. Enquist. Communication during aggressive interactions with particular reference to variation in choice of behavior. *Animal Behavior*, 33:1152–1161, 1985.

[Fontán and Mataric, 1996] Miguel Schneider Fontán and Maja J Mataric. A study of territoriality: the role of critical mass. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 553–561, Cambridge, Massachusetts, 1996. MIT Press.

[Fontán and Mataric, 1998] Miguel Schneider Fontán and Maja J Mataric. Territorial multi-robot task division. *IEEE Transactions on Robotics and Automation*, 14(5), 1998.

[Fox *et al.*, 1999a] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[Fox *et al.*, 1999b] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[Fox *et al.*, 2000] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3), June 2000.

[Franz and Mallot, 2000] Matthias O. Franz and Hanspeter A. Mallot. Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30:133–153, 2000.

[Gallistel, 1990] Charles R. Gallistel. *The Organization of Learning*, chapter The Cognitive Map. MIT Press, Cambridge, Massachusetts, 1990.

[Gambardella *et al.*, 1999] L. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50:167–176, 1999.

[Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and intractability : a guide to the theory of NP-completeness.* W. H. Freeman, San Fransisco, 1979.

[Gaussier and Zrehen, 1995] Phillipe Gaussier and Stephane Zrehen. PerAc: A neural architecture to control artificial animals. *Robotics and Autonomous Systems*, 16:291–230, 1995.

[Gaussier *et al.*, 1997] P. Gaussier, C. Joulain, S. Zrehen, J. P. Banquet, and A. Revel. Visual navigation in an open environment without map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 545–550, 1997.

[Gaussier *et al.*, 1998] P. Gaussier, S. Moga, J. P. Banquet, and M. Quoy. From perception-action loops to imitation processes: A bottom-ip approach of learning by imitation. *Applied Artificial Intelligence Journal*, 12(7-8):701–727, 1998.

[Gee *et al.*, 1993] A. H. Gee, S. V. B. Aiyer, and R. Prager. An analytical framework for optimizing neural networks. *Neural Networks*, 6:79–97, 1993.

[Gerstner and Abbott, 1997] Wulfram Gerstner and L. F. Abbott. Learning navigational maps through potentiation and modulation of hippocampal place cells. *Journal of Computational Neuroscience*, 4:79–94, 1997.

[Goldberg and Mataric, 1997] Dani Goldberg and Maja J. Mataric. Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 637–642, Menlo Park, California, 1997. AAAI Press.

[Gould, 1986] James L. Gould. The locale map of honey bees: Do insects have cognitive maps? *Science*, 232:861–863, 1986.

[Hayes and Demiris, 1994] Gillian Hayes and John Demiris. A robot controller using learning by imitation. In *Proceedings of the International Symposium on Intelligent Robotic Systems*, pages 198–204, Grenoble, France, 1994. Lifia Imag.

[Holland and Melhuish, 1999] Owen Holland and Chris Melhuish. Stimergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

[Intanagonwiwat *et al.*, 2000] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6^{th} Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*, 2000.

[Jonker and Volgenant, 1987] Roy Jonker and Ton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[Jung and Zelinsky, 1999] Jung and Zelinsky. An architecture for distributed cooperative planning in a behavior-based multi-robot system. *Robotics and Autonomous Systems*, 26:149–174, 1999.

[Kaelbling *et al.*, 1996a] L. P. Kaelbling, A. R. Cassandra, and J. A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[Kaelbling *et al.*, 1996b] L. P. Kaelbling, M. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kalman, 1960] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82, 1960.

[Karsai, 1999] Istvan Karsai. Decentralized control of construction behavior in paper wasps: An overview of the stimergy approach. *Artificial Life*, 5(2):117–136, 1999.

[Kawauchi *et al.*, 1993] Y. Kawauchi, M. Inaba, and T. Fukuda. A principle of distributed decision making of cellular robotic system (CEBOT). In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 833–838, 1993.

[Khatib, 1986] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *IEEE Journal of Robotics and Automation*, 5(1):90–98, 1986.

[Khatib, 1999] O. Khatib. Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26:175–183, 1999.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecci. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Kuhn, 1955] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(83-97), 1955.

[Kuipers and Byun, 1987] Benjamin Kuipers and Yung-Tai Byun. A qualitative approach to robot exploration and map-learning. In *Proceedings of the Spatial Reasoning and Multi-Sensor Fusion Workshop*, 1987.

[Kuipers and Byun, 1991] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8:47–63, 1991.

[Kung, 1993] S. Y. Kung. *Digital Neural Networks*, chapter 9. PTR Prentice Hall, 1993.

[Kurtzberg, 1962] J. M. Kurtzberg. On approximation methods for the assignment problems. *Journal of the ACM*, 9:419–439, 1962.

[Lagoudakis, 1998] Michail G. Lagoudakis. Mobile robot local navigation with a polar neural map. Master's thesis, University of Southwestern Louisiana, 1998.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Latombe, 1991] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, Massachusetts, 1991.

[Lenser and Veloso, 2000] Scott Lenser and Maria Manuela Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[Lewis and Bekey, 1992] M. A. Lewis and G. A. Bekey. The behavioral self-organization of nanorobots using local rules. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[Lipson and Pollack, 2000] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(31):974–978, 2000.

[Littman, 1994] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, San Fransisco, 1994. Morgan Kaufmann.

[Lozano-Perez, 1983] Tomas Lozano-Perez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, C-32:108–120, 1983.

[Lu and Milios, 1997] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[Maes, 1990] Pattie Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[Maes, 1991] Pattie Maes. A bottom-up mechanism for action selection in an artificial creature. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 238–246, Cambridge, Massachusetts, 1991. MIT Press.

[Maes, 1992] Pattie Maes. Learning behavior networks from experience. In *Proceedings of the First European Conference on Artificial Life*, Cambridge, Massachusetts, 1992. MIT Press.

[Mahadevan and Connell, 1992] S. Mahadevan and J Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[Mataric *et al.*, 1995] M.J. Mataric, M. Nilsson, and Simsarin K.T. Cooperative multi-robot box-pushing. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[Mataric, 1991] Maja J. Mataric. Navigating with a rat brain: A neurobiologically-inspired model for robot spatial representation. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 169–175, Cambridge, Massachusetts, 1991. MIT Press.

[Mataric, 1992] Maja J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8:333–344, 1992.

[Mataric, 1993] Maja J. Mataric. Kin recognition, similarity, and group behavior. In *Proceedings of the Fifteenth Annual Cognitive Science Society Conference*, pages 705–710. Lawrence Erlbaum Associates, 1993.

[Mataric, 1994] Maja J. Mataric. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, San Fransisco, 1994. Morgan Kaufmann.

[Mataric, 1995] Maja J Mataric. Designing and understanding adaptive group behavior. *Adaptive Behavior*, 4(1), 1995.

[Mataric, 1997] Maja J. Mataric. Learning social behaviors. *Robotics and Autonomous Systems*, 20, 1997.

[Maybeck, 1979] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 1. Academic Press, 1979.

[McFarland and Spier, 1997] D. McFarland and E Spier. Basic cycles, utility and opportunism in self-sufficient robots. *Robotics and Autonomous Systems*, 20:179–190, 1997.

[McFarland, 1981] David McFarland. *The Oxford companion to Animal Behaviour*. Oxford University Press, New York, 1981.

[Moga and Gaussier, 1999] S. Moga and P. Gaussier. A neural structure for learning by imitation. In *Proceedings of the Fifth European Conference on Artificial Life*, pages 314–318. Springer, 1999.

[Moller *et al.*, 1998] Ralf Moller, Dimitrios Lambrinos, Rolf Pfeifer, Thomas Labhart, and Rudiger Wehner. Modeling ant navigation with an autonomous agent. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 185–194, Cambridge, Massachusetts, 1998. MIT Press.

[Moravec and Blackwell, 1993] Hans Moravec and Mike Blackwell. Learning sensor models for evidence grids. In *CMU Robotics Institute 1991 Annual Research Review*, pages 8–15. 1993.

[Moravec and Elfes, 1985] H.P. Moravec and A.E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1985.

[Myerson, 1991] Roger B. Myerson. *Game theory: analysis of conflict*. Harvard University Press, Cambridge, Massachusetts, 1991.

[Nash, 1997] J. F. Jr. Nash. Equilibrium points in n-person games. In H. W. Kuhn, editor, *Classics in game theory*. Princeton University Press, Princeton, New Jersey, 1997.

[Nilsson, 1984] N. J. Nilsson. Shakey the robot. Technical Report 323, SRI AI Center, 1984.

[Nourbakhsh *et al.*, 1995] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office navigating robot. *AI Magazine*, 16(2), 1995.

[O'Keefe and Nadel, 1978] J. O'Keefe and L. Nadel. *The Hippocampus as a Cognitive Map*. Clarendon, London, 1978.

[Oriolo *et al.*, 1995] Giuseppe Oriolo, Marilena Venditelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 2900–2906, 1995.

[Østergaard *et al.*, 2001] Esben Østergaard, Gaurav Sukhatme, and Maja J Mataric. Emergent bucket brigading. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[Parker, 1998] Lynne E. Parker. ALLIANCE: an architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[Payton *et al.*, 2001] D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone robotics. *Autonomous Robots*, 11(3), 2001.

[Payton, 1990] David W. Payton. Internalized plans: A representation for action resources. *Robotics and Autonomous Systems*, 6:89–103, 1990.

[Peterson and Anderson, 1987] C. Peterson and J. R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[Pipe *et al.*, 1994] A. G. Pipe, T. C. Fogarty, and A. Winfield. A hybrid architecture for learning continuous environmental models in maze problems. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 198–205, Cambridge, Massachusetts, 1994. MIT Press.

[Pirjanian *et al.*, 2000] P. Pirjanian, T.L. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, H. Das, S. Joshi, and P.S. Schenker. 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*, volume 4196, pages 221–230, 2000.

[Porta and Celaya, 2000] Josep M Porta and Enric Celaya. Learning in categorizable environments. In *From animals to animats 6: Proceedings of the sixth International Conference on Simulation of Adaptive Behavior*, pages 342–352, Cambridge, Massachusetts, 2000. MIT Press.

[Ramoni *et al.*, 2001] M. Ramoni, P. Sebastiani, and P. Cohen. Bayesian analysis of sensory inputs of a mobile robot. In *Proceedings of the Fifth International Workshop on Case Studies in Bayesian Statistics*, pages 363–379, New York, 2001. Springer.

[Rao and Fuentes, 1996] R. Rao and O. Fuentes. Learning navigational behaviors using a predictive sparse distributed memory. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 382–390. MIT Press, 1996.

[Revel *et al.*, 1998] A. Revel, P. Gaussier, S. Lepretre, and J. P. Banquet. Planification versus sensory-motor conditioning: what are the issues? In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 129–138, Cambridge, Massachusetts, 1998. MIT Press.

[Roberts and Sheratt, 1998] Gilbert Roberts and Thomas N. Sheratt. Development of cooperative relationships through increasing investment. *Nature*, 334:175–179, July 1998.

[Rosenblatt and Payton, 1989] Kenneth Rosenblatt and David Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, 1989.

[Rothemund and Winfree, 2000] P. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Symposium on the Theory of Computation (STOC)*, 2000.

[Rumelhart *et al.*, 1986] David E. Rumelhart, James L. McClelland, and The PDP Reasearch Group. *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[Sandholm and Crites, 1995] Tuomas W. Sandholm and Robert H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147–166, 1995.

[Schenker *et al.*, 2000] P.S. Schenker, T.L. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das, S. Joshi, H. Aghazarian, A.J. Ganino, B.A. Kennedy, and M.S. Garrett. 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*, volume 4196, pages 210–220, 2000.

[Schiele and Crowley, 1994] B. Schiele and J.L. Crowley. A comparison of position estimation techniques using occupancy grids. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1628–1634, 1994.

[Shackleton and Gini, 1997] John Shackleton and Maria Gini. Measuring the effectiveness of reinforcement learning for behavior-based robots. *Adaptive Behavior*, 5(3-4):365–390, 1997.

[Singh, 1991] S. P. Singh. Transfer of learning across compositions of sequential tasks. In *Proceedings of the Eighth International Conference on Machine Learning*, San Fransisco, 1991. Morgan Kaufmann.

[Sobey, 1994] P. Sobey. Active navigation with a monocular robot. *Biological Cybernetics*, 71:433–440, 1994.

[Srinivasan, 1992] M. V. Srinivasan. How bees exploit optic flow: Behavioral experiments and neural models. *Philosophical Transactions of the Royal Society of London B*, 337:253–159, 1992.

[Steels, 1997] Luc Steels. A selectionist mechanism for autonomous behavior acquisition. *Robotics and Autonomous Systems*, 20:117–132, 1997.

[Stone and Veloso, 2000] P. Stone and M. Veloso. Layered learning. In *Proceedings of the Eleventh European Conference on Machine Learning*, pages 369–381. Springer-Verlag, 2000.

[Stone, 2001] Peter Stone. RoboCup-2000: The fourth robotic soccer world championships. *AI Magazine*, 22(1), 2001.

[Stopp and Riethmüller, 1995] A. Stopp and T. Riethmüller. Fast reactive path planning by 2d and 3d multi-layer spatial grids. In *Proceedings of the 1995 IEEE International Symposium on Intelligent Control*, pages 545–550, August 27-29 1995.

[Sugiyama and Murata, 1995] Hisayoshi Sugiyama and Masashi Murata. A method of map building with cooperative mobile robots. In *26^{th} International Symposium on Industrial Robots*, October 1995.

[Sutton and Barto, 1998] Richard Sutton and Andrew Barto. *Reinforcement Learning*. The MIT Press, Cambridge, Massachusetts, 1998.

[Tank and Tank, 1985] John J. Tank and David W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[Theraulaz and Bonabeau, 1999] Guy Theraulaz and Eric Bonabeau. A brief history of stimergy. *Artificial Life*, 5(2):97–116, 1999.

[Therauluz and Bonabeau, 1995] G. Therauluz and E. Bonabeau. Coordination in distributed building. *Science*, 269:686–688, 1995.

[Thrun and Bücken, 1996] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*. AAAI Press/MIT Press, 1996.

[Toates, 1986] Fred Toates. *Motivational Systems.* Cambridge University Press, Cambridge, 1986.

[Trullier and Meyer, 1998] Olivier Trullier and Jean-Arcady Meyer. Animat navigation using a cognitive graph. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 213–222, Cambridge, Massachusetts, 1998. MIT Press.

[Trullier *et al.*, 1997] O. Trullier, S. Wiener, A. Berthoz, and J. Meyer. Biologically based artificial navigation systems: review and prospects. *Progress in Neurobiology*, 51, 1997.

[Tyrrell, 1993a] Toby Tyrrell. *Computational Mechanisms for Action Selection.* PhD thesis, University of Edinburgh, 1993.

[Tyrrell, 1993b] Toby Tyrrell. The use of hierarchies for action selection. *Adaptive Behavior*, 1(4):387–420, 1993.

[Tyrrell, 1994] Toby Tyrrell. An evaluation of Maes's bottom-up mechanism for action selection. *Adaptive Behavior*, 2(4):307–348, 1994.

[Vaughan *et al.*, 2000a] Richard Vaughan, Kasper Stoey, Gaurav Sukhatme, , and Maja Mataric. Go ahead, make my day: Robot conflict resolution by aggressive competition. In *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation Of Adaptive Behavior*, pages 491–500, Cambridge, Massachusetts, 2000. MIT Press.

[Vaughan *et al.*, 2000b] Richard Vaughan, Kasper Støy, Gaurav Sukhatme, and Maja Mataric. Blazing a trail: insect-inspired resource transportation by a robot team. In *Proceedings of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 111–120, 2000.

[Walker *et al.*, 1993] Ashley Walker, John Hallam, and David Willshaw. Bee-havior in a mobile robot: The construction of a self-organized cognitive map and its use in robot navigation within a complex, natural environment. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1451–1456, 1993.

[Watkins and Dayan, 1992] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[Watkins, 1989] C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, Cambridge, England, 1989.

[Wehner and Menzel, 1990] Rüdiger Wehner and Randolf Menzel. Do insects have cognitive maps? *Annual Review of Neuroscience*, 13:403–414, 1990.

[Wehner *et al.*, 1996] R. Wehner, B. Michel, and P. Antonsen. Visual navigation in insects: coupling of egocentric and geocentric information. *Journal of Experimental Biology*, 199:129–140, 1996.

[Wehner, 1994] Rüdiger Wehner. The polarization-vision project: championing organismic biology. In K. Schildberger and N. Elsner, editors, *Neural Basis of Adaptive Behavior*, pages 103–143. G. Fischer, Stuttgart, 1994.

[Werger and Mataric, 1996] Barry Werger and Maja J Mataric. Robotic "food" chains: externalization of state and program for minimal-agent foraging. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 625–634, Cambridge, Massachusetts, 1996. MIT Press.

[Werner, 1994] Gregory M. Werner. Using second order neural connections for motivation of behavioral choices. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation Of Adaptive Behavior*, pages 154–161, Cambridge, Massachusetts, 1994. MIT Press.

[Whitehead, 1992] S. D. Whitehead. *Reinforcement learning for the adaptive control of perception and action.* PhD thesis, University of Rochester, 1992.

[Wilfong, 1988] G. T. Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the ACM Symposium on Computational Geometry (ASCG)*, pages 279–288, 1988.

[Wilson and Pawley, 1988] G. V. Wilson and G. S. Pawley. On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biological Cybernetics*, 58:63–70, 1988.

[Worden, 1992] R. Worden. Navigation by fragment fitting: A theory of hippocampal function. *Hippocampus*, 2(2):165–188, 1992.

[Yamauchi and Langley, 1997] Brian Yamauchi and Pat Langley. Place recognition in dynamic environments. *Journal of Robotic Systems*, 14(2), 1997.

[Yamauchi *et al.*, 1999] Brian Yamauchi, Alan Schultz, and William Adams. Integrating exploration and localization for mobile robots. *Adaptive Behavior*, 7(2), 1999.

[Yamauchi, 1996] Brian Yamauchi. Mobile robot localization in dynamic environments using dead reckoning and evidence grids. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1401–1406, 1996.

[Zahavi, 1975] A. Zahavi. Mate selection - a selection for a handicap. *Journal of Theoretical Biology*, 53:205–213, 1975.

[Zelinsky and Yuta, 1993] Alexander Zelinsky and Shin'ichi Yuta. A unified approach to planning, sensing and navigation for mobile robots. In Tsuneo Yoshikawa and Fumio Miyazaki, editors, *Proceedings of the Third International Symposium on Experimental Robotics*, pages 444–455. Springer-Verlag, 1993.