Distributed Reinforcement Learning for Self-Reconfiguring Modular Robots

by

Paulina Varshavskaya

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author .....
Department of Electrical Engineering and Computer Science
July 27, 2007

Certified by .....
Daniela Rus
Professor
Thesis Supervisor

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

Distributed Reinforcement Learning for Self-Reconfiguring Modular Robots

by

Paulina Varshavskaya

Submitted to the Department of Electrical Engineering and Computer Science
on July 27, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis, we study distributed reinforcement learning in the context of automating the design of decentralized control for groups of cooperating, coupled robots. Specifically, we develop a framework and algorithms for automatically generating distributed controllers for self-reconfiguring modular robots using reinforcement learning. The promise of self-reconfiguring modular robots is that of robustness, adaptability and versatility. Yet most state-of-the-art distributed controllers are laboriously handcrafted and task-specific, due to the inherent complexities of distributed, local-only control. In this thesis, we propose and develop a framework for using reinforcement learning for automatic generation of such controllers. The approach is profitable because reinforcement learning methods search for good behaviors during the lifetime of the learning agent, and are therefore applicable to online adaptation as well as automatic controller design. However, we must overcome the challenges due to the fundamental partial observability inherent in a distributed system such as a self-reconfiguring modular robot.

We use a family of policy search methods that we adapt to our distributed problem. The outcome of a local search is always influenced by the search space dimensionality, its starting point, and the amount and quality of available exploration through experience. We undertake a systematic study of the effects that certain robot and task parameters, such as the number of modules, presence of exploration constraints, availability of nearest-neighbor communications, and partial behavioral knowledge from previous experience, have on the speed and reliability of learning through policy search in self-reconfiguring modular robots. In the process, we develop novel algorithmic variations and compact search space representations for learning in our domain, which we test experimentally on a number of tasks.

This thesis is an empirical study of reinforcement learning in a simulated lattice-based self-reconfiguring modular robot domain. However, our results contribute to the broader understanding of automatic generation of group control and design of distributed reinforcement learning algorithms.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

My advisor Daniela Rus has inspired, helped and supported me throughout. Were it not for the weekly dose of fear of disappointing her, I might never have found the motivation for the work it takes to finish. Leslie Kaelbling has been a great source of wisdom on learning, machine or otherwise. Much of the research presented here will be published elsewhere co-authored by Daniela and Leslie. And despite this work being done entirely in simulation, Rodney Brooks has agreed to be on my committee and given some very helpful comments on my project. I have been fortunate to benefit from discussions with some very insightful minds: Leonid Peshkin, Keith Kotay, Michael Collins and Sarah Finney have been particularly helpful. The members of the Distributed Robotics Lab and those of the Learning and Intelligent Systems persuasion have given me their thoughts, feedback, encouragement and friendship, for which I am sincerely thankful.

This work was supported by Boeing Corporation. I am very grateful for their support.

I cannot begin to thank my friends and housemates, past and present, of the Bishop Allen Drive Cooperative, who have showered me with their friendship and created the most amazing, supportive and intense home I have ever had the pleasure to live in. I am too lazy to enumerate all of you — because I know I cannot disappoint you — but you have a special place in my heart. To everybody who hopped in the car with me and left the urban prison behind, temporarily, repeatedly, to catch the sun, the rocks and the snow, to give ourselves a boost of sanity — thank you.

To my parents: спасибо вам, любимые мама и папа, за все и, особенно, за то, что вы всю жизнь давали мне любую возможность учиться, несмотря на то, что учеба завела меня так далеко от вас, thank you!

To my life- and soul-mate Luke: thank you for seeing me through this.

Contents

Chapter 1

Introduction

The goal of this thesis is to develop and demonstrate a framework and algorithms for decentralized automatic design of distributed control for groups of cooperating, coupled robots. This broad area of research will advance the state of knowledge in distributed systems of mobile robots, which are becoming ever more present in our lives. Swarming robots, teams of autonomous vehicles, and ad hoc mobile sensor networks are now used to automate aspects of exploration of remote and dangerous sites, search and rescue, and even conservation efforts. The advance of coordinated teams is due to their natural parallelism: they are more efficient than single agents; they cover more terrain more quickly; they are more robust to random variation and random failure. As human operators and beneficiaries of multitudes of intelligent robots, we would very much like to just tell them to go, and expect them to decide on their own exactly how to achieve the desired behavior, how to avoid obstacles and pitfalls along the way, how to coordinate their efforts, and how to adapt to an unknown, changing environment. Ideally, a group of robots faced with a new task, or with a new environment, should take the high-level task goal as input and automatically decide what team-level and individual-level interactions are needed to make it happen.

The reality of distributed mobile system control is much different. Distributed programs must be painstakingly developed and verified by human designers. The interactions and interference the system will be subject to are not always clear. And development of control for such distributed mobile systems is in general difficult for serially-minded human beings. Yet automatic search for the correct distributed controller is problematic — there is incredible combinatorial explosion of the search space due to the multiplicity of agents; there are also intrinsic control difficulties for each agent due to the limitations of local information and sensing. The difficulties are dramatically multiplied when the search itself needs to happen in a completely distributed way with the limited resources of each agent.

This thesis is a study in automatic adaptable controller design for a particular class of distributed mobile systems: self-reconfiguring modular robots. Such robots are made up of distinct physical modules, which have degrees of freedom (DOFs) to move with respect to each other and to connect to or disconnect from each other. They have been proposed as versatile tools for unknown and changing situations, when mis-

sions are not pre-determined, in environments that may be too remote or hazardous for humans. These types of scenarios are where a robot's ability to transform its shape and function would be undoubtedly advantageous. Clearly, adaptability and automatic decision-making with respect to control is essential for any such situations.

A self-reconfiguring modular robot is controlled in a distributed fashion by the many processors embedded in its modules (usually one processor per module), where each processor is responsible for only a few of the robot's sensors and actuators. For the robot as a whole to cohesively perform any task, the modules need to coordinate their efforts without any central controller. As with other distributed mobile systems, in self-reconfiguring modular robots (SRMRs) we give up on easier centralized control in the hope of increased versatility and robustness. Yet most modular robotic systems run task-specific, hand-designed algorithms. For example, a distributed localized rule-based system controlling adaptive locomotion (Butler et al. 2001) has 16 manually designed local rules, whose asynchronous interactions needed to be anticipated and verified by hand. A related self-assembly system had 81 such rules (Kotay & Rus 2004). It is very difficult to manually generate correct distributed controllers of this complexity.

Instead, we examine how to automatically generate local controllers for groups of robots capable of local sensing and communications to near neighbors. Both the controllers and the process of automation itself must be computationally fast and only require the kinds of resources available to individuals in such decentralized groups.

In this thesis we present a framework for automating distributed controller design for SRMRs through reinforcement learning (RL) (Sutton & Barto 1999): the intuitive framework where agents observe and act in the world and receive a signal which tells them how well they are doing. Learning may be done from scratch (with no initial knowledge of how to solve the task), or it can be seeded with some amount of information — received wisdom directly from the designer, or solutions found in prior experience. Automatic learning from scratch will be hindered by the large search spaces pervasive in SRMR control. It is therefore to be expected that constraining and guiding the search with extra information will in no small way influence the outcome of learning and the quality of the resulting distributed controllers. Here, we undertake a systematic study of the parameters and constraints influencing reinforcement learning in SRMRs.

It is important to note that unlike with evolutionary algorithms, reinforcement learning optimizes during the lifetime of the learning agent, as it attempts to perform a task. The challenge in this case is that learning itself must be fully distributed. The advantage is that the same paradigm can be used to (1) automate controller design, and (2) run distributed adaptive algorithms directly on the robot, enabling it to change its behavior as the environment, its goal, or its own composition changes. Such capability of online adaptation would take us closer to the goal of more versatile, robust and adaptable robots through modularity and self-reconfiguration. We are ultimately pursuing both goals (automatic controller generation and online adaptation) in applying RL methods to self-reconfigurable modular robots.

Learning distributed controllers, especially in domains such as SRMRs where neighbors interfere with each other and determine each other's immediate capabil-

ities, is an extremely challenging problem for two reasons: the space of possible behaviors are huge and they are plagued by numerous local optima which take the form of bizarre and dysfunctional robot configurations. In this thesis, we describe strategies for minimizing dead-end search in the neighborhoods of such local optima by using easily available information, smarter starting points, and inter-module communication. The best strategies take advantage of the structural properties of modular robots, but they can be generalized to a broader range of problems in cooperative group behavior. In this thesis, we have developed novel variations on distributed learning algorithms. The undertaking is both profitable and hard due to the nature of the platforms we study. SRMRs have exceedingly large numbers of degrees of freedom, yet may be constrained in very specific ways by their structural composition.

1.1 Problem statement

Is distributed learning of control possible in the SRMR domain? If so, is it a trivial or a hard application? Where are the potential difficulties, and how do specific parameters of the robot design and the task to be learned affect the speed and reliability of learning? What are the relevant sources of information and search constraints that enable the learning algorithms to steer away from dead-end local optima? How might individuals share their local information, experience, and rewards with each other? How does sharing affect the process and the outcome of learning? What can be said about the learned distributed controllers? Is there any common structure or knowledge that can be shared across tasks?

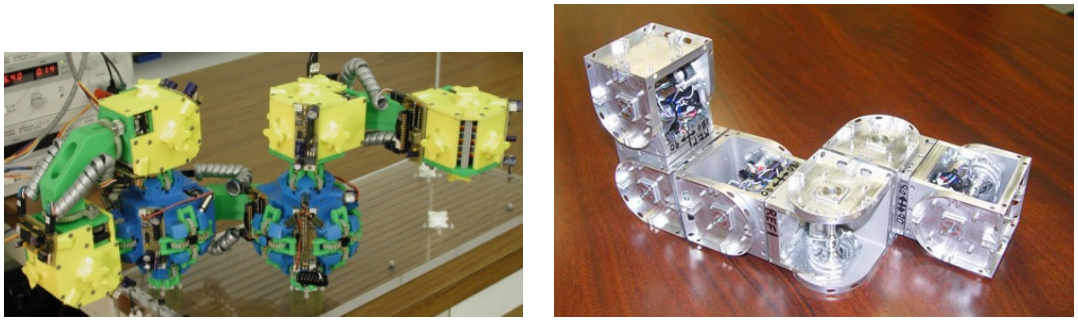
The goal of this thesis is to systematically explore the above questions, and provide an empirical study of reinforcement learning in the SRMR domain that bears on the larger issues of automating generation of group control, and design of distributed reinforcement learning algorithms.

In order to understand precisely the contributions of our work, we will now describe its position and relevance within the context of the two fields of concern to us: self-reconfiguring modular robots and reinforcement learning. In the rest of this introductory chapter, we give the background on both SRMRs and reinforcement learning, describe and resolve a fundamental conflict of assumptions in those fields, present a circumscribed case study which will enable us to rigorously test our algorithms, and give an outline of the thesis.

1.2 Background

1.2.1 Self-reconfiguring modular robots

Self-reconfiguring modular robotics (Everist et al. 2004, Kotay & Rus 2005, Murata et al. 2001, Shen et al. 2006, Yim et al. 2000, Zykov et al. 2005) is a young and growing field of robotics with a promise of robustness, versatility and self-organization. Robots made of a large number of identical modules, each of which has computational, sensory, and motor capabilities, are proposed as universal tools for unknown



(a) (b)

Figure 1-1: Some recent self-reconfiguring modular robots (SRMRs): (a) The Molecule (Kotay & Rus 2005), (b) SuperBot (Shen et al. 2006). Both can act as lattice-based robots. SuperBot can also act as a chain-type robot (reprinted with permission).

environments. Exemplifying the vision of modularity is the idea of a self-built factory on another planet: small but strong and smart modules can be packed tightly into a spaceship, then proceed to build structures out of themselves, as needed, in a completely autonomous and distributed fashion. A malfunction in the structure can be easily repaired by disconnecting the offending modules, to be replaced by fresh stock. A new configuration can be achieved without the cost of additional materials. A number of hardware systems have been designed and built with a view to approach this vision. A taxonomy of self-reconfiguring hardware platforms can be built along two dimensions: the structural constraints on the robot topology and the mechanism of reconfiguration.

Lattice vs. chain robots

SRMRs can be distinguished by their structural topology. Lattice-based robots are constructed with modules fitting on a geometric lattice (e.g., cubes or tetrahedra). Connections may be made at any face of the module that corresponds to a lattice intersection. By contrast, chain-based robots are constructed with modules forming a chain, tree or ring. This distinction is best understood in terms of the kinds of degrees of freedom and motions possible in either case. In lattice-based robots, modules are usually assumed to not have individual mobility other than the degrees of freedom required to connect to a neighboring site and disconnect from the current one. Self-reconfiguration is therefore required for any locomotion and most other tasks. On the other hand, chain-type SRMRs may use the degrees of freedom in the chain (or tree branch) as "limbs" for direct locomotion, reaching or other mobility tasks. A chain-type robot may move as a snake in one configuration, and as an hexapod insect in another one. The only connections and disconnections of modules occurs specifically when the robot needs to change configurations. But a lattice-based robot needs to keep moving, connecting and disconnecting its modules in order to move its center of mass in some direction (locomotion) as well as when specific reconfiguration is

required.

Figure 1-1 shows two recent systems. The first one, called the Molecule (Kotay & Rus 2005), is a pure lattice system. The robot is composed of male and female "atoms", which together represent a "molecule" structure which is also the basis of a lattice. The second system, in figure 1-1b, is SuperBot (Shen et al. 2006), which is a hybrid. On the one hand, it may function in a lattice-based fashion similar to the Molecule: SuperBot modules are cubic, with six possible connection sites, one at each face of the cube, which makes for a straightforward cubic grid. On the other hand, the robot can be configured topologically and function as a chain (e.g., in a snake configuration) or tree (e.g., in a tetrapod, hexapod or humanoid configuration).

While chain-type or hybrid self-reconfiguring systems are better suited for tasks requiring high mobility, lattice-based robots are useful in construction and shape-maintenance scenarios, as well as for self-repair. The research effort in designing lattice-based robots and developing associated theory has also been directed towards future systems and materials. As an example, miniaturization may lead to fluid-like motion through lattice-based self-reconfiguration. On another extreme, we might envision self-reconfiguring buildings built of block modules that are able and required to infrequently change their position within the building.

The focus of this thesis in mainly on lattice-based robots. However, chain-type systems may also benefit from adaptable controllers designed automatically through reinforcement learning.

Deterministic vs. stochastic reconfiguration

SRMRs may also be distinguished according to their mode of reconfiguration. The systems we label here as deterministic achieve reconfiguration by actively powered, motorized joints and/or gripping mechanisms within each module. Each module within such a system, when it finds itself in a certain local configuration, may only execute a finite and usually small number of motions that would change its position relative to the rest of the robot, and result in a new configuration. Both the Molecule and SuperBot (figure 1-1) are deterministic systems under this taxonomy.

On the other hand, systems such as programmable parts (Bishop et al. 2005), self-replicating robots from random parts (Griffith et al. 2005) or robots for in-space assembly (Everist et al. 2004), are composed of passive elements which require extraneous energy provided by the environment. Actuation, and therefore assembly and reconfiguration occurs due to stochastic influences in this energy-rich environment, through random processes such as Brownian motion.

The focus of this thesis is exclusively on self-actuated, deterministic SRMRs that do not require extra energy in the environment.

1.2.2 Research motivation vs. control reality

Development of self-reconfiguring modular robotic systems, as well as theory and algorithms applicable to such systems, can be motivated by a number of problems. On the one hand, researchers are looking to distill the essential and fundamental

properties of self-organization through building and studying self-assembling, self-replicating and self-reconfiguring artificial systems. This nature-inspired scientific motive is pervasive especially in stochastically reconfigurable robotics, which focuses on self-assembly.

The designers of actuated, deterministic SRMRs, on the other hand, tend to motivate their efforts with the promise of versatility, robustness and adaptability inherent in modular architectures. Modular robots are expected to become a kind of universal tool, changing shape and therefore function as the task requirements also change. It is easy to imagine, for example, a search and rescue scenario with such a versatile modular robot. Initially in a highly-mobile legged configuration that will enable the robot to walk fast over rough terrain, it can reconfigure into a number of serpentine shapes that burrow into the piles of rubble, searching in parallel for signs of life. Another example, often mentioned in the literature, concerns space and planetary exploration. A modular robot can be tightly packed into a relatively small container for travel on a spaceship. Upon arrival, it can deploy itself according to mission (a "rover" without wheels, a lunar factory) by moving its modules into a correct configuration. If any of the modules should break down or become damaged, the possibility of self-reconfiguration allows for their replacement with fresh stock, adding the benefits of possible self-repair to modular robots.

The proclaimed qualities of versatility, robustness and adaptability with reference to the ideal "killer apps" are belied by the rigidity of state-of-the-art controllers available for actuated self-reconfiguring modular robots.

Existing controllers

Fully distributed controllers of many potentially redundant degrees of freedom are required for robustness and adaptability. Compiling a global high-level goal into distributed control laws or rules based solely on local interactions is a notoriously difficult exercise. Yet currently, most modular robotic systems run task-specific, hand-designed algorithms, for example, the rule-based systems in (Butler et al. 2004), which took hours of designer time to synthesize. One such rule-based controller for a two-dimensional lattice robot is reproduced here in figure 1-2. These rules guarantee eastward locomotion by self-reconfiguration on a square lattice.

Automating the design of such distributed rule systems and control laws is clearly needed, and the present work presents a framework in which it can be done. There has been some exploration of automation in this case. In some systems the controllers were automatically generated using evolutionary techniques in simulation before being applied to the robotic system itself (Kamimura et al. 2004, Mytilinaios et al. 2004). Current research in automatic controller generation for SRMRs and related distributed systems will be examined more closely in chapter 2. One of the important differences between the frameworks of evolutionary algorithms and reinforcement learning is the perspective from which optimization occurs.

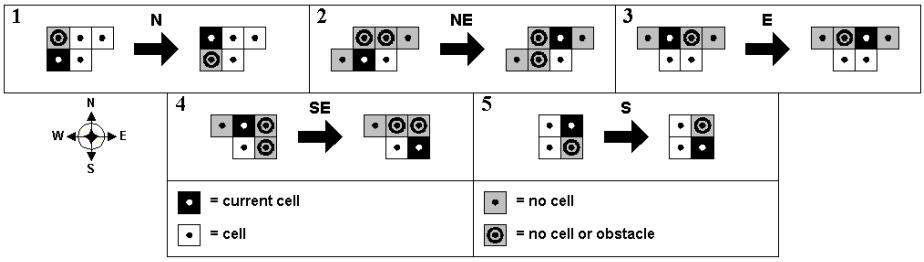


Figure 1-2: A hand-designed rule-based controller for locomotion by self-reconfiguration on a square lattice. Reprinted with permission from Butler et al. (2004).

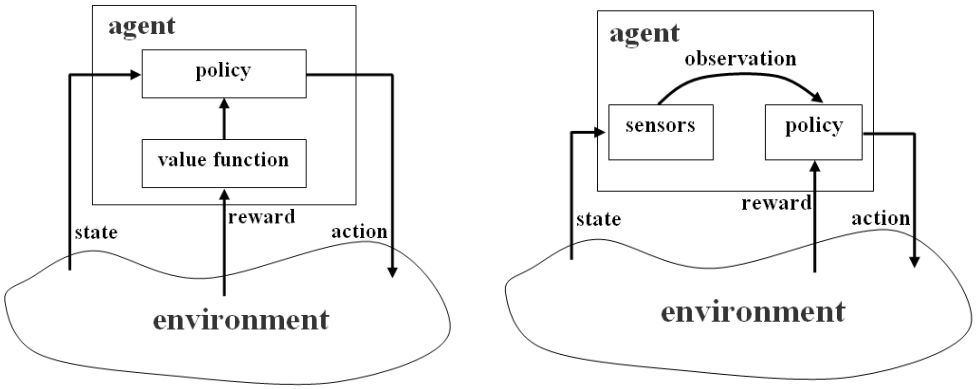
Adaptation through reinforcement learning

Let us elaborate this point. Evolutionary techniques operate on populations of controllers, each of which are allowed to run for some time after which their fitness is measured and used as the objective function for the optimization process. Optimization happens over generations of agents running these controllers. In contrast, in the reinforcement learning framework, optimization happens over the lifetime of each agent. The parameters of the controller are modified in small, incremental steps, at runtime, either at every time step or episodically, after a finite and relatively small number of time steps. This difference means that reinforcement learning techniques may be used both for automated controller generation and online adaptation to the changing task or environment. These are the twin goals of using RL algorithms in self-reconfiguring modular robots. While this thesis is primarily concerned with the first goal of automated controller generation, we always keep in mind the potential for online adaptation that is provided by the techniques presented here.

We now turn to the question of whether the problems and tasks faced by SRMRs are indeed amenable to optimization by reinforcement learning. This is not a trivial question since modular robots are distributed, locally controlled systems of agents that are fundamentally coupled to each other and the environment. The question is: are the assumptions made by reinforcement learning algorithms satisfied in our domain?

1.3 Conflicting assumptions

The most effective of the techniques collectively known as reinforcement learning all make a number of assumptions about the nature of the environment in which the learning takes place: specifically, that the world is stationary (not changing over time in important ways) and that it is fully observed by the learning agent. These assumptions lead to the possibility of powerful learning techniques and accompanying theorems that provide bounds and guarantees on the learning process and its results. Unfortunately, these assumptions are usually violated by any application domain involving a physical robot operating in the real world: the robot cannot know directly



(a) (b)

Figure 1-3: The reinforcement learning framework. (a) In a fully observable world, the agent can estimate a value function for each state and use it to select its actions. (b) In a partially observable world, the agent does not know which state it is in due to sensor limitations; instead of a value function, the agent updates its policy parameters directly.

the state of the world, but may observe a local, noisy measure on parts of it. We will demonstrate below that in the case of modular robots, even their very simplified abstract kinematic models violate the assumptions necessary for the powerful techniques to apply and the theorems to hold. As a result of this conflict, some creativity is essential in designing or applying learning algorithms to the SRMR domain.

1.3.1 Assumptions of a Markovian world

Before we examine closely the conflict of assumptions, let us briefly review the concept of reinforcement learning.

Reinforcement learning

Consider the class of problems in which an agent, such as a robot, has to achieve some task by undertaking a series of actions in the environment, as shown in figure 1-3a. The agent perceives the state of the environment, selects an action to perform from its repertoire and executes it, thereby affecting the environment which transitions to a new state. The agent also receives a scalar signal indicating its level of performance, called the reward signal. The problem for the agent is to find a good strategy (called a policy) for selecting actions given the states. The class of such problems can be described by stochastic processes called Markov decision processes (see below). The problem of finding an optimal policy can be solved in a number of ways. For instance, if a model of the environment is known to the agent, it can use dynamic programming algorithms to optimize its policy (Bertsekas 1995). However, in many cases the model is either unknown initially or impossibly difficult to compute. In these cases, the agent must act in the world to learn how it works.

Reinforcement learning is sometimes used in the literature to refer to the class of problems that we have just described. It is more appropriately used to name the set of statistical learning techniques employed to solve this class of problems in the cases when a model of the environment is not available to the agent. The agent may learn such a model, and then solve the underlying decision process directly. Or it may estimate a value function associated with every state it has visited (as is the case in figure 1-3a) from the reward signal it has received. Or it may update the policy directly using the reward signal. We refer the reader to a textbook (Sutton & Barto 1999) for a good introduction to the problems addressed by reinforcement learning and the details of standard solutions.

It is assumed that the way the world works does not change over time, so that the agent can actually expect to optimize its behavior with respect to the world. This is the assumption of a stationary environment. It is also assumed that the probability of the world entering the state s' at the next time step is determined solely by the current state s and the action chosen by the agent. This is the Markovian world assumption, which we formalize below. Finally, it is assumed that the agent knows all the information it needs about the current state of the world and its own actions in it. This is the full observability assumption.

Multi-agent MDPs

When more than one agent are learning to behave in the same world that all of them affect, this more complex interaction is usually described as a multi-agent MDP. Different formulations of processes involving multiple agents exist, depending on the assumptions we make about the information available to the learning agent or agents.

In the simplest case we imagine that learning agents have access to a kind of oracle that observes the full state of the process, but execute factored actions (i.e., that is called one action of the MDP is the result of coordinated mini-actions of all modules executed at the same time). The individual components of a factored action need to be coordinated among the agents, and information about the actions taken by all modules also must be available to every learning agent. This formulation satisfies all of the strong assumptions of a Markovian world, including full observability. However, as we argue later in section 1.3.2, to learn and maintain a policy with respect to each state is in practical SRMRs unrealistic, as well as prohibitively expensive in both experience and amount of computation.

Partially observable MDPs

Instead we can assume that each agent only observes a part of the state that is local to its operation. If the agents interact through the environment but do not otherwise communicate, then each agent is learning to behave in a partially observable MDP (POMDP), ignoring the distributed nature of the interaction, and applying its learning algorithm independently. Figure 1-3b shows the interaction between an agent and the environment when the latter is only partially observable through the agent's sensors. In the partially observable formulation, there is no oracle, and the

modules have access only to factored (local, limited, perhaps noisy) observations over the state. In this case, the assumption of full observability is no longer satisfied. In general, powerful learning techniques such as Q-learning (Watkins & Dayan 1992) are no longer guaranteed to converge to a solution, and optimal behavior is much harder to find.

Furthermore, the environment with which each agent interacts comprises all the other agents who are learning at the same time and thus changing their behavior. The world is non-stationary due to many agents learning at once, and thus cannot even be treated as a POMDP, although in principle agents could build a weak model of the competence of the rest of agents in the world (Chang et al. 2004).

1.3.2 Assumptions of the kinematic model

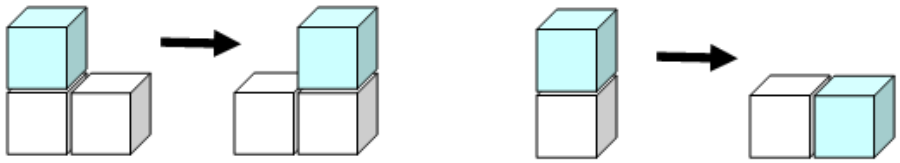


Figure 1-4: The sliding-cube kinematic model for lattice-based modular robots: (1) a sliding transition, and (2) a convex transition.

How does our kinematic model fit into these possible sets of assumptions? We base most of our development and experiments on the standard sliding-cube model for lattice-based self-reconfigurable robots (Butler et al. 2004, Fitch & Butler 2006, Varshavskaya et al. 2004, Varshavskaya et al. 2007). In the sliding-cube model, each module of the robot is represented as a cube (or a square in two dimensions), which can be connected to other module-cubes at either of its six (four in 2D) faces. The cube cannot move on its own; however, it can move, one cell of the lattice at a time, on a substrate of like modules in the following two ways, as shown in figure 1-4: 1) if the neighboring module M_1 that it is attached to has another neighbor M_2 in the direction of motion, the moving cube can slide to a new position on top of M_2, or 2) if there is no such neighbor M_2, the moving cube can make a convex transition to the same lattice cell in which M_2 would have been. Provided the relevant neighbors are present in the right positions, these motions can be performed relative to any of the six faces of the cubic module. The motions in this simplified kinematic model can actually be executed by physically implemented robots (Butler et al. 2004). Of those mentioned in section 1.2.1, the Molecule, MTRAN-II, and SuperBot can all form configurations required to perform the motions of the kinematic model, and therefore controllers developed in simulation for this model are potentially useful for these robots as well.

The assumptions made by the sliding-cube model are that the agents are individual modules of the model. In physical instantiations, more than one physical module may

coordinate to comprise one simulation "cube". The modules have limited resources, which affects any potential application of learning algorithms:

limited actuation: each module can execute one of a small set of discrete actions; each action takes a fixed amount of time to execute, potentially resulting in a unit displacement of the module in the lattice

limited power: actuation requires a significantly greater amount of power than computation or communication

limited computation and memory: on-board computation is usually limited to microcontrollers

clock: the system may be either synchronized to a common clock, which is a rather unrealistic assumption, or asynchronous[1], with every module running its code in its own time

Clearly, if an individual module knew the global configuration and position of the entire robot, as well as the action each module is about to take, then it would also know exactly the state (i.e., position and configuration) of the robot at the next time step, as we assume a deterministic kinematic model. Thus, the world can be Markovian. However, this global information is not available to individual modules due to limitations in computational power and communications bandwidth. They may communicate with their neighbors at each of their faces to find out the local configuration of their neighborhood region. Hence, there is only a partial observation of the world state in the learning agent, and our kinematic model assumptions are in conflict with those of powerful RL techniques.

1.3.3 Possibilities for conflict resolution

We have established that modular robots have intrinsic partial observability, since the decisions of one module can only be based on its own state and the observations it can gather from local sensors. Communications may be present between neighboring modules but there is generally no practical possibility for one module to know or infer the total system state at every timestep, except perhaps for very small-scale systems.

Therefore, for SRMR applications, we see two possibilities for resolving the fundamental conflict between locally observing, acting and learning modules and the Markov assumption. We can either resign ourselves to learning in a distributed POMDP, or we can attempt to orchestrate a coordination scheme between modules. In the first case, we lose convergence guarantees for powerful learning techniques such as Q-learning, and must resort to less appealing algorithms. In the second case, that of coordinated MDPs, we may be able to employ powerful solutions in practice (Guestrin et al. 2002, Kok & Vlassis 2006); however, the theoretical foundations of such application is not as sound as that of single-agent MDP. In this thesis, we focus on algorithms developed for partially observable processes.

[1]Later, in chapter 5 we introduce the notion of partially asynchronous execution, which assumes an upper bound on communication delays.

1.3.4 Case study: locomotion by self-reconfiguration

We now examine the problem of synthesizing locomotion gaits for lattice-based modular self-reconfiguring robots. The abstract kinematic model of a 2D lattice-based robot is shown in figure 1-5. The modules are constrained to be connected to each other in order to move with respect to each other; they are unable to move on their own. The robot is positioned on an imaginary 2D grid; and each module can observe at each of its 4 faces (positions 1, 3, 5, and 7 on the grid) whether or not there is another connected module on that neighboring cell. A module (call it M) can also ask those neighbors to confirm whether or not there are other connected modules at the corner positions (2, 4, 6, and 8 on the grid) with respect to M. These eight bits of observation comprise the local configuration that each module perceives.

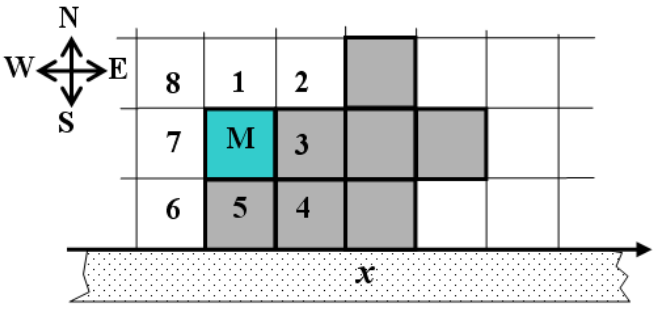


Figure 1-5: The setup for locomotion by self-reconfiguration on a lattice in 2D.

Thus, the module M in the figure knows that lattice cells number 3, 4, and 5 have other modules in them, but lattice cells 1, 2, and 6-8 are free space. M has a repertoire of 9 actions, one for moving into each adjacent lattice cell (face or corner), and one for staying in the current position. Given any particular local configuration of the neighborhood, only a subset of those actions can be executed, namely those that correspond to the sliding or convex motions about neighbor modules at any of the face sites. This knowledge may or may not be available to the modules. If it is not, then the module needs to learn which actions it will be able to execute by trying and failing.

The modules are learning a locomotion gait with the eastward direction of motion receiving positive rewards, and the westward receiving negative rewards. Specifically, a reward of $+1$ is given for each lattice-unit of displacement to the East, -1 for each lattice-unit of displacement to the West, and 0 for no displacement. M's goal in figure 1-5 is to move right. The objective function to be maximized through learning is the displacement of the whole robot's center of mass along the x axis, which is equivalent to the average displacement incurred by individual modules.

Clearly, if the full global configuration of this modular robot were known to each module, then each of them could learn to select the best action in any of them with an MDP-based algorithm. If there are only two or three modules, this may be done as there are only 4 (or 18 respectively) possible global configurations. However,

the 8 modules shown in figure 1-5 can be in a prohibitively large number of global configurations; and we are hoping to build modular systems with tens and hundreds of modules. Therefore it is more practical to resort to learning in a partially observable world, where each module only knows about its local neighborhood configuration.

The algorithms we present in this thesis were not developed exclusively for this scenario. They are general techniques applicable to multi-agent domains. However, the example problem above is useful in understanding how they instantiate to the SRMR domain. Throughout the thesis, it will be used in experiments to illustrate how information, search constraints and the degree of inter-module communication influence the speed and reliability of reinforcement learning in SRMRs.

1.4 Overview

This thesis provides an algorithmic study, evaluated empirically, of using distributed reinforcement learning by policy search with different information sources for learning distributed controllers in self-reconfiguring modular robots.

At the outset, we delineate the design parameters and aspects of both robot and task that contribute to three dimensions along which policy search algorithms may be described: the number of parameters to be learned (the number of knobs to tweak), the starting point of the search, and the amount and quality of experience that modules can gather during the learning process. Each contribution presented in this thesis relates in some way to manipulating at least one of these issues of influence in order to increase the speed and reliability of learning and the quality of its outcome.

For example, we present two novel representations for lattice-based SRMR control, both of which aim to compress the search space by reducing the number of parameters to learn. We quickly find, however, that compact representations require careful construction, shifting the design burden away from design of behavior, but ultimately mandating full human participation. On the other hand, certain constraints on exploration are very easy for any human designer to articulate, for example, that the robot should not attempt physically impossible actions. These constraints effectively reduce the number of parameters to be learned. Other simple suggestions that can be easily articulated by the human designer (for example: try moving up when you see neighbors on the right) can provide good starting points to policy search, and we also explore their effectiveness.

Further, we propose a novel variations on the theme of policy search which are specifically geared to learning of group behavior. For instance, we demonstrate how incremental policy search with each new learning process starting from the point at which a previous run with fewer modules had stopped, reduces the likelihood of being trapped in an undesirable local optimum, and results in better learned behaviors. This algorithm is thus a systematic way of generating good starting points for distributed group learning.

In addition, we develop a framework of incorporating agreement (also known as consensus) algorithms into policy search, and we present a new algorithm with which individual learners in distributed systems such as SRMRs can agree on both rewards

and experience gathered during the learning process — information which can be incorporated into learning updates, and will result in consistently better policies and markedly faster learning. This algorithmic innovation thus provides a way of increasing the amount and quality of individuals' experience during learning. Active agreement also has an effect on the kind of policies that are learned.

Finally, as we evaluate our claims and algorithms on a number of tasks requiring SRMR cooperation, we also inquire into the possibility of post-learning knowledge transfer between policies which were learned for different tasks from different reward functions. It turns out that local geometric transformations of learned policies can provide another systematic way of automatically generating good starting points for further learning.

1.5 Thesis contributions

This thesis contributes to the body of knowledge on modular robots in that it provides a framework and algorithms for automating controller generation in a distributed way in SRMRs. It contributes to multi-agent reinforcement learning by establishing a new application area, as well as proposing novel algorithmic variations. We see the following specific contributions:

- A framework for distributed automation of SRMR controller search through reinforcement learning with partial observability
- A systematic empirical study of search constraints and guidance through extra information and inter-agent communications
- An algorithm for incremental learning using the additive nature of SRMRs
- An algorithm for policy search with spatial features
- An algorithm for distributed reinforcement learning with inter-agent agreement on rewards and experience
- A novel minimal representation for learning in lattice-based SRMRs
- A study of behavioral knowledge transfer through spatial policy transformation
- Empirical evaluation on the following tasks:

  directed locomotion

  directed locomotion over obstacle fields

  tall structure building

  reaching to random goal positions

These contributions bring us forward towards delivering on the promise of adaptability and versatility of SRMRS — properties often used for motivating SRMR research, but conspicuously absent from most current systems. The results of this study are significant, because they present a framework in which future efforts in automating controller generation and online adaptation of group behavior can be grounded. In addition, this thesis establishes empirically where the difficulties in such cooperative group learning lie and how they can be mitigated through the exploitation of relevant domain properties.

1.6 Thesis outline

We are proposing the use of reinforcement learning for both off-line controller design, and as actual adaptive control[2] for self-reconfiguring modular robots. Having introduced the problem and detailed the fundamental underlying conflict of assumptions, we now prepare to resolve it in the following thesis chapters.

A related work review in both SRMRs and distributed reinforcement learning can be found in chapter 2. In chapter 3 we derive the basic stochastic gradient ascent algorithm in policy space and identify the key issues contributing to the speed and reliability of learning in SRMRs. We then address some of these issues by introducing a representational variation on the basic algorithm, and present initial experimental results from our case study (see section 1.3.4).

We explore the space of extra information and constraints that can be employed to guide the learning algorithms and address the identified key issues in chapter 4. Chapter 5 proposes agreement algorithms as a natural extension to gradient ascent algorithms for distributed systems. Experimental results to validate our claims can be found throughout chapters 3 to 5. In chapter 6 we perform further evaluation of the thesis findings with different objective functions. Finally, we discuss the significance of those results and limitations of this approach in chapter 7 and conclude with directions for future work.

[2] In the nontechnical sense of adaptable behavior.

Chapter 2

Related Work

We are building on a substantial body of research in both reinforcement learning and distributed robotic systems. Both these fields are well established and we cannot possibly do them justice in these pages, so we describe here only the most relevant prior and related work. Specifically we focus our discussion on relevant research in the field of self-reconfigurable modular robots, cooperative and coordinated distributed RL, and agreement algorithms in distributed learning.

2.1 Self-reconfiguring modular robots

This section gives a summary of the most important work in the sub-field of lattice-based SRMRs, as well as the research efforts to date in automating controller design and planning for such robots.

2.1.1 Hardware lattice-based systems

There are a great number of lattice-based self-reconfiguring modular robotics platforms, or those that are capable of emulating lattice-based self-reconfiguration. The field dates back to the Metamorphic (Pamecha et al. 1996) and Fracta (Murata et al. 1994) systems, which were capable of reconfiguring their shape on a plane. The first three-dimensional systems appeared shortly afterwards, e.g., 3D-Fracta (Murata et al. 1998), the Molecule (Kotay et al. 1998), and the TeleCube (Suh et al. 2002). All these systems have degrees of freedom to connect and disconnect modules to and from each other, and move each module on the 2D or 3D lattice on a substrate of other modules. More recent and more sophisticated robots include the Modular Transformers MTRAN-II (Kamimura et al. 2004) and the SuperBot (Shen et al. 2006), all of which are capable of functioning as both chain-type and lattice-based robots, as well as the exclusively non-cubic lattice-based ATRON (Østergaard et al. 2006).

In this thesis, we are using the "sliding cubes" abstraction developed by Butler et al. (2004), which represents the motions of which abstract modules on a cubic lattice are capable. As we have previously mentioned, the Molecule, MTRAN-II and MTRAN-III, and the SuperBot, are all capable of actuating these abstract motions.

More recent hardware systems have had a greater focus on stochastic self-assembly from modular parts, e.g., the 2D systems such as programmable parts (Bishop et al. 2005) and growing machines (Griffith et al. 2005), where the energy and mobility is provided to the modules through the flow generated by an air table; and the 3D systems such as those of (White et al. 2005), where modules are in passive motion inside a liquid whose flow also provides exogenous energy and mobility to the modular parts. Miche (Gilpin et al. 2007) is a deterministic self-disassembly system that is related to this work insofar as the parts used in Miche are not moving. Such work is outside the scope of this thesis.

2.1.2 Automated controller design

In the modular robotics community, the need to employ optimization techniques to fine-tune distributed controllers has been felt and addressed before. The Central Pattern Generator (CPG) based controller of MTRAN-II was created with parameters tuned by off-line genetic algorithms (Kamimura et al. 2004). Evolved sequences of actions have been used for machine self-replication (Mytilinaios et al. 2004, Zykov et al. 2005). Kubica & Rieffel (2002) reported a partially automated system where the human designer collaborates with a genetic algorithm to create code for the TeleCube module. The idea of automating the transition from a global plan to local rules that govern the behavior of individual agents in a distributed system has also been explored from the point of view of compilation in programmable self-assembly (Nagpal 2002).

Reinforcement learning can be employed as just another optimization technique off-line to automatically generate distributed controllers. However, one advantage of applying RL techniques to modular robots is that the same, or very similar, algorithms can also be run on-line, eventually on the robot, to update and adapt its behavior to a changing environment. Our goal is to develop an approach that will allow such flexibility of application. Some of the ideas reported here were previously formulated in less detail by Varshavskaya et al. (2004) and (2006, 2007).

2.1.3 Automated path planning

In the field of SRMRs, the problem of adapting to a changing environment and goal during the execution of a task has not received as much attention, with one notable recent exception. Fitch & Butler (2007) use an ingenious distributed path planning algorithm to automatically generate kinematic motion trajectories for the sliding cube robots to all move into a goal region. Their algorithm, which is based on a clever distributed representation of cost/value and standard dynamic programming, works in a changing environment with obstacles and a moving goal, so long as at least one module is always in the goal region. The plan is decided on and executed in a completely distributed fashion and in parallel, making it an efficient candidate for very large systems.

We view our approaches as complementary. The planning algorithm requires extensive thinking time in between each executed step of the reconfiguration. By

contrast, in our approach, once the policy is learned it takes no extra run-time planning to execute it. On the other hand, Fitch and Butler's algorithm works in an entirely asynchronous manner; and they have demonstrated path-planning on very large numbers of modules.

2.2 Distributed reinforcement learning

While the reinforcement learning paradigm has not been applied previously to self-reconfigurable modular robotics, there is a vast research field of distributed and multi-agent reinforcement learning (Stone & Veloso 2000, Shoham et al. 2003). Applications have included network routing and traffic coordination among others.

2.2.1 Multi-agent Q-learning

In robotics, multi-agent reinforcement learning has been applied to teams of robots with Q-learning as the algorithm of choice (e.g., Matarić 1997, Fernandez & Parker 2001). It was then also discovered that a lot of engineering is needed to use Q-learning in teams of robots, so that there are only a few possible states and a few behaviors to choose from. Q-learning has also been attempted on a non-reconfiguring modular robot (Yu et al. 2002) with similar issues and mixed success. The Robocup competition (Kitano et al. 1997) has also driven research in collaborative multi-agent reinforcement learning.

2.2.2 Hierarchical distributed learning

Hierarchies of machines (Andre & Russell 2000) have been used in multi-agent RL as a strategy for constraining policies to make Q-learning work faster. The related issue of hierarchically optimal Q-function decomposition has been addressed by Marthi et al. (2006).

2.2.3 Coordinated learning

More recently there has been much theoretical and algorithmic work in multi-agent reinforcement learning, notably in cooperative reinforcement learning. Some solutions deal with situations where agents receive individual rewards but work together to achieve a common goal. Distributed value functions introduced by Schneider et al. (1999) allowed agents to incorporate neighbors' value functions into Q-updates in MDP-style learning. Coordination graphs (Guestrin et al. 2002, Kok & Vlassis 2006) are a framework for learning in distributed MDPs where values are computed by algorithms such as variable elimination on a graph representing inter-agent connections. In mirror situations, where individual agents must learn separate value functions from a common reward signal, Kalman filters (Chang et al. 2004) have been used to estimate individuals' reward contributions.

2.2.4 Reinforcement learning by policy search

In partially observable environments, policy search has been extensively used (Peshkin 2001, Ng & Jordan 2000, Baxter & Bartlett 2001, Bagnell et al. 2004) whenever models of the environment are not available to the learning agent. In robotics, successful applications include humanoid robot motion (Schaal et al. 2003), autonomous helicopter flight (Ng et al. 2004), navigation (Grudic et al. 2003), and bipedal walking (Tedrake et al. 2005), as well as simulated mobile manipulation (Martin 2004). In some of those cases, a model identification step is required prior to reinforcement learning, for example from human control of the plant (Ng et al. 2004), or human motion capture (Schaal et al. 2003). In other cases, the system is brilliantly engineered to reduce the search task to estimating as few as a single parameter (Tedrake et al. 2005).

Function approximation, such as neural networks or coarse coding, has also been widely used to improve performance of reinforcement learning algorithms, for example in policy gradient algorithms (Sutton et al. 2000) or approximate policy iteration (Lagoudakis & Parr 2003).

2.3 Agreement algorithms

Agreement algorithms were first introduced in a general framework of distributed (synchronous and asynchronous) computation by Tsitsiklis (e.g., Tsitsiklis et al. 1986). The theory includes convergence proofs used in this dissertation, and results pertaining to asynchronous distributed gradient-following optimization. The framework, theory and results are available as part of a widely known textbook (Bertsekas & Tsitsiklis 1997). Agreement is closely related to swarming and flocking, which have long been discussed and modeled in the biological and physical scientific literature (e.g., Okubo 1986, Vicsek et al. 1995).

Current research in agreement, which is also more frequently known as consensus, includes decentralized control theory (e.g., Jadbabaie et al. 2003) and networks literature (e.g., Olfati-Saber & Murray 2004). As original models were developed to predict the behavior of biological groups (swarms, flocks, herds and schools), current biological and ecological studies of group behavior validate the earlier models (e.g., Buhl et al. 2006). A comprehensive review of such work is beyond the scope of this thesis.

In machine learning, agreement has been used in the consensus propagation algorithm (Moallemi & Van Roy 2006), which can be seen as a special case of belief propagation on a Gaussian Markov Random Field. Consensus propagation guarantees very fast convergence to the exact average of initial values on singly connected graphs (trees). The related belief consensus algorithm (Olfati-Saber et al. 2005) enables scalable distributed hypothesis testing.

In the reinforcement learning literature, agreement was also used in distributed optimization of sensor networks (Moallemi & Van Roy 2003). This research is closest to our approach, as it proposes a distributed optimization protocol using policy

gradient MDP learning together with pairwise agreement on rewards. By contrast, our approach is applicable in partially observable situations. In addition, we take advantage of the particular policy search algorithm to let learning agents agree not only on reward, but also on experience. The implications and results of this approach are discussed in chapter 5.

Chapter 3

Policy Search in Self-Reconfiguring Modular Robots

The first approach we take in dealing with distributed actions, local observations and partial observability is to describe the problem of locomotion by self-reconfiguration of a modular robot as a multi-agent Partially Observable Markov Decision Process (POMDP).

In this chapter, we first describe the derivation of the basic GAPS algorithm, and in section 3.3 the issues concerning its implementation in a distributed learning system. We then develop a variation using a feature-based representation and a log-linear policy in section 3.4. Finally, we present some experimental results (section 3.5) from our case study of locomotion by self-reconfiguration, which demonstrate that policy search works where Markov-assuming techniques do not.

3.1 Notation and assumptions

We now introduce some formal notation which will become useful for the derivation of learning algorithms. The interaction is formalized as a Markov decision process (MDP), a 4-tuple $\langle S, A, T, R \rangle$, where S is the set of possible world states, A the set of actions the agent can take, $T : S \times A \rightarrow P(S)$ the transition function defining the probability of being in state $s' \in S$ after executing action $a \in A$ while in state s, and $R : S \times A \rightarrow \mathbf{R}$ a reward function. The agent maintains a policy $\pi(s, a) = P(a|s)$ which describes a probability distribution over actions that it will take in any state. The agent does not know T. In the multi-agent case we imagine that learning agents have access to a kind of oracle that observes the full state of the MDP, but execute factored actions $\mathbf{a_t} = \{a_t^1 ... a_t^n\}$, if there are n modules. In a multi-agent partially observable MDP, there is no oracle, and the modules have access only to factored observations over the state $\mathbf{o_t} = Z(s_t) = \{o_t^1 ... o_t^n\}$, where $Z : S \rightarrow O$ is the unknown observation function which maps MDP states to factored observations and O is the set of possible observations.

We assume that each module only conditions its actions on the current observation; and that only one action is taken at any one time. An alternative approach could

be to learn controllers with internal state (Meuleau et al. 1999). However, internal state is unlikely to help in our case, as it would multiply drastically the number of parameters to be learned, which is already considerable.

To learn in this POMDP we use a policy search algorithm based on gradient ascent in policy space (GAPS), proposed by Peshkin (2001). This approach assumes a given parametric form of the policy $\pi_\theta(o,a) = P(a|o,\theta)$, where θ is the vector of policy parameters, and the policy is a differentiable function of the parameters. The learning proceeds by gradient ascent on the parameters θ to maximize expected long-term reward.

3.2 Gradient ascent in policy space

The basic GAPS algorithm does hill-climbing to maximize the value (that is, long-term expected reward) of the parameterized policy. The derivation starts with noting that the value of a policy π_θ is $V_\theta = E_\theta[R(h)] = \sum_{h\epsilon H} R(h)P(h|\theta)$, where θ is the parameter vector defining the policy and H is the set of all possible experience histories. If we could calculate the derivative of V_θ with respect to each parameter, it would be possible to do exact gradient ascent on the value by making updates $\Delta\theta_k = \alpha\frac{\partial}{\partial\theta_k}V_\theta$. However, we do not have a model of the world that would give us $P(h|\theta)$ and so we will use stochastic gradient ascent instead. Note that $R(h)$ is not a function of θ, and the probability of a particular history can be expressed as the product of two terms (under the Markov assumption and assuming learning is episodic with each episode comprised of T time steps):

$$\begin{aligned}
P(h|\theta) &= \\
&= P(s_0)\prod_{t=1}^{T} P(o_t|s_t)P(a_t|o_t,\theta)P(s_{t+1}|s_t,a_t) \\
&= \left[P(s_0)\prod_{t=1}^{T} P(o_t|s_t)P(s_{t+1}|s_t,a_t)\right]\left[\prod_{t+1}^{T} P(a_t|o_t,\theta)\right] \\
&= \Xi(h)\Psi(h,\theta),
\end{aligned}$$

where $\Xi(h)$ is not known to the learning agent and does not depend on θ, and $\Psi(h,\theta)$ is known and differentiable under the assumption of a differentiable policy representation. Therefore, $\frac{\partial}{\partial\theta_k}V_\theta = \sum_{h\epsilon H} R(h)\Xi(h)\frac{\partial}{\partial\theta_k}\Psi(h,\theta)$.

The differentiable part of the update gives:

$$\begin{aligned}\frac{\partial}{\partial\theta_k}\Psi(h,\theta &= \frac{\partial}{\partial\theta_k}\prod_{t=1}^{T}\pi_\theta(a_t,o_t) = \\ &= \sum_{t=1}^{T}\left[\frac{\partial}{\partial\theta_k}\pi_\theta(a_t,o_t)\prod_{\tau\neq t}\pi_\theta(a_\tau,o_\tau)\right] \\ &= \sum_{t=1}^{T}\left[\frac{\frac{\partial}{\partial\theta_k}\pi_\theta(a_t,o_t)}{\pi_\theta(a_t,o_t)}\prod_{t=1}^{T}\pi_\theta(a_t,o_t)\right] \\ &= \Psi(h,\theta)\sum_{t=1}^{T}\frac{\partial}{\partial\theta_k}\ln\pi_\theta(a_t,o_t).\end{aligned}$$

Therefore,

$$\tfrac{\partial}{\partial\theta_k}V_\theta = \textstyle\sum_{h\epsilon H} R(h)\left(P(h|\theta)\sum_{t=1}^{T}\frac{\partial}{\partial\theta_k}\ln\pi_\theta(a_t,o_t)\right).$$

The stochastic gradient ascent algorithm operates by collecting algorithmic traces at every time step of each learning episode. Each trace reflects the contribution of a single parameter to the estimated gradient. The makeup of the traces will depend on the policy representation.

The most obvious representation is a lookup table, where rows are possible local observations and columns are actions, with a parameter for each observation-action pair. The GAPS algorithm was originally derived (Peshkin 2001) for such a representation, and is reproduced here for completeness (Algorithm 1). The traces are obtained by counting occurrences of each observation-action pair, and taking the difference between that number and the expected number that comes from the current policy. The policy itself, i.e., the probability of taking an action a_t at time t given the observation o_t and current parameters θ is given by Boltzmann's law:

$$\pi_\theta(a_t,o_t) = P(a_t|o_t,\theta) = \frac{e^{\beta\theta(o_t,a_t)}}{\sum_{a\in A}e^{\beta\theta_(o_t,a)}},$$

where β is an inverse temperature parameter that controls the steepness of the curve and therefore the level of exploration.

This gradient ascent algorithm has some important properties. As with any stochastic hill-climbing method, it can only be relied upon to reach a local optimum in the represented space, and will only converge if the learning rate is reduced over time. We can attempt to mitigate this problem by running GAPS from many initialization points and choosing the best policy, or by initializing the parameters in a smarter way. The latter approach is explored in chapter 4.

3.3 Distributed GAPS

GAPS has another property interesting for our twin purposes of controller generation and run-time adaptation in SRMRs. In the case where each agent not only observes

Algorithm 1 GAPS (observation function o, M modules)

```
Initialize parameters θ ← small random numbers
for each episode do
  Calculate policy π(θ)
  Initialize observation counts N ← 0
  Initialize observation-action counts C ← 0
  for each timestep in episode do
    for each module m do
      observe o_m and increment N(o_m)
      choose a from π(o_m, θ) and increment C(o_m, a)
      execute a
    end for
  end for
  Get global reward R
  Update θ according to
  θ( o, a ) += α R( C(o, a) − π(o, a, θ) N(o) )
  Update π(θ) using Boltzmann's law
end for
```

and acts but also learns its own parameterized policy, the algorithm can be extended in the most obvious way to Distributed GAPS (DGAPS), as was also done by Peshkin (2001). A theorem in his thesis says that the centralized factored version of GAPS and the distributed multi-agent version will make the same updates given the same experience. That means that, given the same observation and action counts, and the same rewards, the two instantiations of the algorithm will find the same solutions.

In our domain of a distributed modular robot, DGAPS is naturally preferable. However, in a fully distributed scenario, agents do not in fact have access to the same experience and the same rewards as all others. Instead of requiring an identical reward signal for all agents, we take each module's displacement along the x axis to be its reward signal: $R_m = x_m$, since we assume that modules do not communicate. This means that the policy value landscape is now different for each agent. However, the agents are physically coupled with no disconnections allowed. If the true reward is $R = \frac{\sum_{m=1}^{N} x_m}{N}$, and each individual reward is $R_m = x_m$, then each R_m is a bounded estimate of R that's at most $N/2$ away from it (in the worst-case scenario where all modules are connected in one line). Furthermore, as each episode is initialized, modules are placed at random in the starting configuration of the robot. Therefore, the robot's center of mass and $R = E[x_m]$ is the expected value of any one module's position along the x axis. We can easily see that in the limit, as the number of turns per episode increases and as learning progresses, each x_m approaches the true reward.

This estimate is better the fewer modules we have and the larger R is. Therefore it makes sense to simplify the problem in the initial phase of learning, while the rewards are small, by starting with fewer modules, as we explore in chapter 4. It may also make sense to split up the experience into larger episodes, which would generate larger rewards all other things being equal, especially as the number of modules increases. Other practical implications of diverse experience and rewards among the learning

agents are examined in chapter 5.

3.4 GAPS learning with feature spaces

In some domains it is unfeasible to enumerate all possible observations and maybe even all possible actions. When we are confronted with a feature-based representation for a domain, a log-linear[1] version of the basic GAPS algorithm can be employed, which we derive below.

We propose to represent the policy compactly as a function of a number of features defined over the observation-action space of the learning module. Suppose the designer can identify some salient parts of the observation that are important to the task being learned. We can define a vector of feature functions over the observation-action space $\Phi(a,o) = [\phi_1(a,o)\phi_2(a,o)...\phi_n(a,o)]^T$. These feature functions are domain-dependent and can have a discrete or continuous response field. Then the policy encoding for the learning agent (the probability of executing an action a_t) is:

$$\pi_\theta(a_t, o_t) = P(a_t|o_t,\theta) = \frac{e^{\beta\Phi(a_t,o_t)\cdot\theta}}{\sum_{a\epsilon A} e^{\beta\Phi(a,o_t)\cdot\theta}},$$

where β is again an inverse temperature parameter. This definition of probability of selecting action a_t is the counterpart of Boltzmann's law in feature space. The derivation of the log-linear GAPS algorithm (LLGAPS) then proceeds as follows:

$$\begin{aligned}
&\frac{\partial}{\partial\theta_k} lnP(a_t|o_t,\theta) = \\
&\quad = \frac{\partial}{\partial\theta_k}\left(\beta\Phi(a_t,o_t)\cdot\theta - ln\left(\sum_{a\epsilon A} e^{\beta\Phi(a,o_t)\cdot\theta}\right)\right) \\
&\quad = \beta\,\phi_k(a_t,o_t) - \frac{\frac{\partial}{\partial\theta_k}\sum_a e^{\beta\Phi(a,o_t)\cdot\theta}}{\sum_a e^{\beta\Phi(a,o_t)\cdot\theta}} \\
&\quad = \beta\left(\phi_k(a_t,o_t) - \frac{\sum_a \phi_k(a,o_t)e^{\beta\Phi(a,o_t)\cdot\theta}}{\sum_a e^{\beta\Phi(a,o_t)\cdot\theta}}\right) \\
&\quad = \beta\left(\phi_k(a_t,o_t) - \sum_a \phi_k(a,o_t)\pi_\theta(a,o_t)\right) = \lambda_k(t).
\end{aligned}$$

The accumulated traces λ_k are only slightly more computationally involved than the simple counts of the original GAPS algorithm: there are $|A| + N$ more computations per timestep than in GAPS, where A is the set of actions, and N the number of features. The updates are just as intuitive, however, as they still assign more credit to those features that differentiate more between actions, normalized by the actions' likelihood under the current policy.

The algorithm (see Algorithm 2) is guaranteed to converge to a local maximum in policy value space, for a given feature and policy representation. However, the feature space may exclude good policies — using a feature-based representation shifts the

[1]The term log-linear refers to the log-linear combination of features in the policy representation.

Algorithm 2 LLGAPS (Observation function o, N feature functions ϕ)

```
Initialize parameters θ ← small random numbers
for each episode do
  Initialize traces Λ ← 0
  for each timestep t in episode do
    Observe current situation o_t and get features response for every action Φ(∗, o_t)
    Sample action a_t according to policy (Boltzmann's law)
    for k = 1 to N do
      Λ_k ← Λ_k + β (φ_k(a_t, o_t) − Σ_a φ_k(a, o_t) π_θ(a, o_t))
    end for
  end for
  θ ← θ + α R Λ
end for
```

burden of design from policies to features. Without an automatic way of providing good features, the process of controller generation cannot be called automatic either. Experimentally (section 3.5.4), we attempt to verify the robustness of LLGAPS to variations in feature spaces by providing the algorithm with a set of ad hoc features deemed salient for the lattice-based motion domain.

3.5 Experimental Results

In this section we present results from a number of experiments designed to test how well the basic GAPS algorithm and its feature-based extension perform on the locomotion task using the sliding cubes kinematic model in simulation. First, we establish empirically the difference between applying a technique derived for MDPs to a partially observable distributed POMDP by comparing how GAPS, Q-learning and Sarsa (Sutton 1995), which is an on-policy temporal difference, MDP-based method, fare on the task.

3.5.1 The experimental setup

We conduct experiments on a simulated two-dimensional modular robot, which is shown in figure 3-1a. The simulation is based solely on the abstract kinematic model of lattice reconfiguration, and we make no attempt to include physical realism, or indeed any physical laws[2]. However, the simulation enforces two simple rules: (1) the robot must stay fully connected at all times, and (2) the robot must stay connected to the "ground" at $y = 0$ by at least one module at all times. Actions whose result would violate one of these rules fail and are not executed. The guilty module simply forfeits its turn.

[2]Lattice-based SRMRs to date have seen only kinematic controllers. Dynamics enter into play when the robot configuration generates non-negligible lever forces on the module-to-module connectors. When learning is applied to physical robots, the dynamics of physical reality need to be taken into account. We discuss some implications of these concerns in section 7.3.

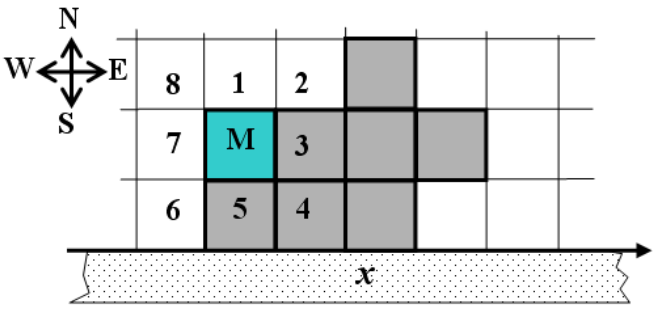


Figure 3-1: Experimental setup: simulated 2D modular robot on a lattice.

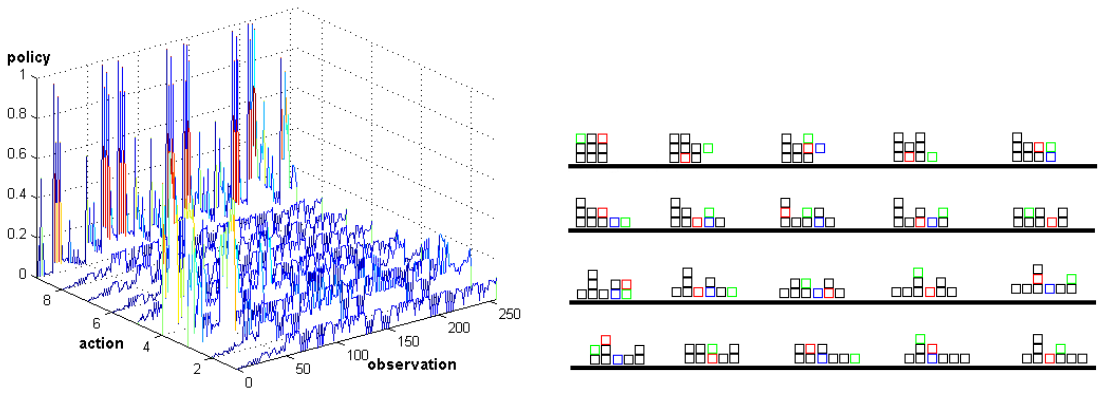
As previewed in section 1.3.4, each module can observe its immediate Moore neighborhood (eight immediate neighbor cells) to see if those cells are occupied by a neighboring module or empty. Each module can also execute one of nine actions, which are to move into one of the neighboring cells or a NOP. The simulator is synchronous, and modules execute their actions in turn, such that no module can go twice before every other module has taken its turn. This assumption is unrealistic; however, it helps us avoid learning to coordinate motion across the entire robot. There are examples in the literature (e.g., in Fitch & Butler 2007) of algorithms intended for use in lattice-based SRMRs that provide for turn-taking. The task is to learn to move in one direction (East) by self-reconfiguration. The reward function measures the progress the modules make in one episode along the x axis of the simulator. Figure 3-2 shows how a policy (a) can be executed by modules to produce a locomotion gait (b) and therefore gain reward.

We run the experiments in two broad conditions. First, the goal of automatically generating controllers can in principle be achieved in simulation off-line, and then imparted to the robot for run-time execution. In that case, we can require all modules to share one set of policy parameters, that is, to pool their local observations and actions for one learning "super-agent" to make one set of parameter updates, which then propagates to all the modules on the next episode. Second, we run the same learning algorithms on individual modules operating independently, without sharing their experience. We predict that more experience will result in faster learning curves and better learning results.

3.5.2 Learning by pooling experience

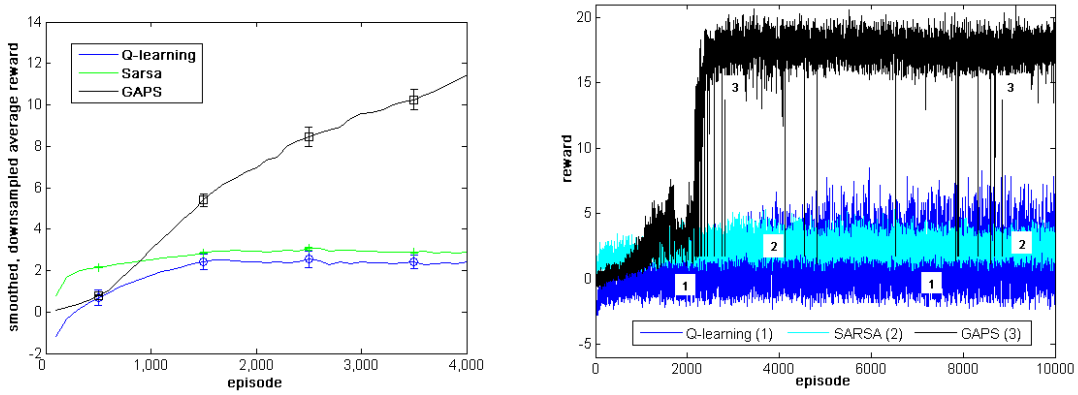
The focus of this section is on the first condition, where a centralized algorithm is run off-line with factored observations and actions. All modules execute the same policy.

In each case, the experiment consisted of 10 runs of the learning algorithm, starting from a randomized initial state. The experiments were set up as episodic learning with each episode terminating after 50 timesteps. The distance that the robot's center of mass moved during an episode was presented to all modules or, equivalently, the centralized learner as the reward at the end of the episode.



(a) (b)

Figure 3-2: (a) A locomotion gait policy (conditional probability distribution over actions given an observation). (b) First few configurations of 9 modules executing the policy during a test run: in blue, module with lowest ID number, in red, module which is about to execute an action, in green, module which has just finished its action.



(a) (b)

Figure 3-3: Performance of policies learned by 15 modules in the 2D locomotion task running Q-learning, Sarsa and GAPS: (a) smoothed (100-point moving window), downsampled average rewards per learning episode over 10 trials (Q-learning and Sarsa each), 50 trials (GAPS), with standard error (b) typical single trial learning curves.

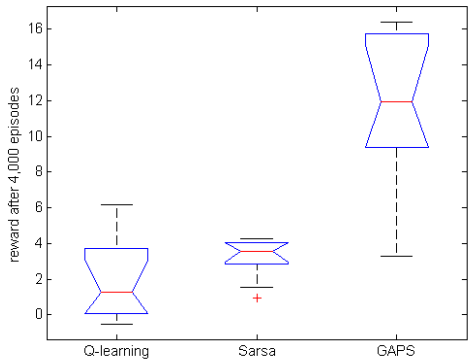


Figure 3-4: Reward distributions after 4,000 episodes of learning for Q-learning, Sarsa and GAPS (10 trials per algorithm). This box and whisker plot, generated using MATLAB®, shows the lower quartile, the median and the upper quartile values as box lines for each algorithm. The whiskers are vertical lines that show the extent of the rest of the data points. Outliers are data beyond the end of the whiskers. The notches on the box represent a robust estimate of the uncertainty about the medians at the 5% significance level.

Unless noted otherwise, in all conditions the learning rate started at $\alpha = 0.01$, decreased over the first $1,500$ episodes, and remained at its minimal value of 0.001 thereafter. The inverse temperature parameter started at $\beta = 1$, increased over the first $1,500$ episodes, and remained at its maximal value of 3 thereafter. This ensured more exploration and larger updates in the beginning of each learning trial. These parameters were selected by trial and error, as the ones consistently generating the best results. Whenever results are reported as smoothed, downsampled average reward curves, the smoothing was done with a 100-point moving window on the results of every trial, which were then downsampled for the sake of clarity, to exclude random variation and variability due to continued within-trial exploration.

3.5.3 MDP-like learning vs. gradient ascent

The first set of experiments is pitting GAPS against two powerful algorithms which make the Markov assumption: Q-learning (Watkins & Dayan 1992) and Sarsa (Sutton 1995). Sarsa is an on-policy algorithm, and therefore may do better than Q-learning in our partially observable world. However, our prediction was that both would fail to reliably converge to a policy in the locomotion task, whereas gradient ascent would succeed in finding a locally optimal policy[3].

Hypothesis: GAPS is capable of finding good locomotion gaits, but Q-learning and Sarsa are not.

[3]As we will see later, a locally optimal policy is not always a good locomotion gait.

Figure 3-3a shows the smoothed average learning curves for both algorithms. Fifteen modules were learning to locomote eastward in 10 separate trial runs (50 trials for GAPS). As predicted, gradient ascent receives considerably more reward than either Q-learning or Sarsa. The one-way ANOVA[4] reveals that the rewards obtained after 4,000 episodes of learning differed significantly as a function of the learning algorithm used ($F(2, 27) = 33.45, p < .01$). Figure 3.5.2 shows the mean rewards obtained by the three algorithms, with error bars and outliers. The post-hoc multiple comparisons test reports that the GAPS mean reward is significantly different from both Sarsa and Q-learning at 99% confidence level, but the latter are not significantly different from each other. In test trials this discrepancy manifested itself as finding a good policy for moving eastward (one such policy is shown in figure 3-2) for GAPS, and failing to find a reasonable policy for Q-learning and Sarsa: modules oscillated, moving up-and-down or left-and-right and the robot did not make progress. In figure 3-3b we see the raw rewards collected at each episode in one typical trial run of both learning algorithms.

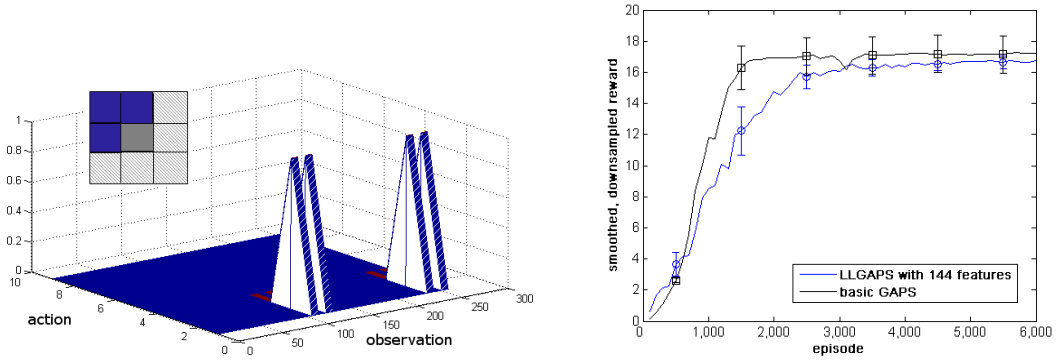
While these results demonstrate that modules running GAPS learn a good policy, they also show that it takes a long time for gradient ascent to find it. We next examine the extent to which we can reduce the number of search space dimensions, and therefore, the experience required by GAPS through employing the feature-based approach of section 3.4.

3.5.4 Learning in feature spaces

We compare experiments performed under two conditions. In the original condition, the GAPS algorithm was used with a lookup table representation with a single parameter $\theta(o, a)$ for each possible observation-action pair. For the task of locomotion by self-reconfiguration, this constituted a total of $2^8 \times 9 = 2304$ parameters to be estimated.

In the log-linear function approximation condition, the LLGAPS algorithm was run with a set of features determined by hand as the cross product of salient aspects of neighborhood observations and all possible actions. The salient aspects were full corners and straight lines, and empty corners and straight lines. The features were obtained by applying partial masks to the observed neighborhood, and selecting a particular action, with response fields as shown in figure 3-5. This gave a total of 144 features, the partial masks for which are fully detailed in figure 3-6. In all cases, modules were learning from the space of reactive policies only.

[4]The rewards are most probably not normally distributed: indeed, the distribution will be multimodal, with "good" and "bad" trials clustered together. It may therefore be more appropriate to use the nonparametric Kruskal-Wallis test, which does not make the normal distribution assumption. The Kruskal-Wallis test also reveals a significant difference in reward as a function of the learning algorithm ($\chi^2(2, 27) = 16.93, p < .01$). However, the KW-test does not transfer well to multi-way analysis (Toothaker & Chang 1980), which will be necessary in further experiments. We will therefore use ANOVA, because it is robust to some assumption violations, and because we are only interested in large effects. But see also Brunner & Puri (2001) for a framework for multi-way nonparametric testing.



(a) (b)

Figure 3-5: 6 modules learning with LLGAPS and 144 features. (a) One of the features used for function approximation in the LLGAPS experiments: this feature function returns 1 if there are neighbors in all three cells of the upper left corner and $a_t = 2$(NE). (b) Smoothed (100-point moving window), downsampled average rewards over 10 runs, with standard error: comparison between original GAPS and LLGAPS.

Hypothesis: LLGAPS's compact representation will lead to faster learning (better convergence times), as compared to standard GAPS.

Figure 3-5b presents the results of comparing the performance of LLGAPS with the original GAPS algorithm on the locomotion task for 6 modules. We see that both algorithms are comparable in both their speed of learning and the average quality of the resulting policies. If anything, around episode 1,500 basic GAPS has better performance. However, a two-way mixed-factor ANOVA (repeated measures every 1000 episodes, GAPS vs. LLGAPS) reveals no statistical significance. Our hypothesis was wrong, at least for this representation.

However, increasing the number of modules reveals that LLGAPS is more prone to finding unacceptable local minima, as explained in section 4.3.2. We also hypothesized that increasing the size of the observed neighborhood would favor the feature-based LLGAPS over the basic GAPS. As the size of the observation increases, the number of possible local configuration grows exponentially, whereas the features can be designed to grow linearly. We ran a round of experiments with an increased neighborhood size of 12 cells obtained as follows: the acting module observes its immediate face neighbors in positions 1,3,5, and 7, and requests from each of them a report on the three cells adjacent to their own faces[5]. Thus the original Moore neighborhood plus four additional bits of observation are available to every module, as shown in figure 3-7a. This setup results in $2^{12} \times 9 = 36,869$ parameters to estimate for GAPS. For LLGAPS we incorporated the extra information thus obtained into an additional 72 features as follows: one partial neighborhood mask per extra neighbor present,

[5]If no neighbor is present at a face, and so no information is available about a corner neighbor, it is assumed to be an empty cell.

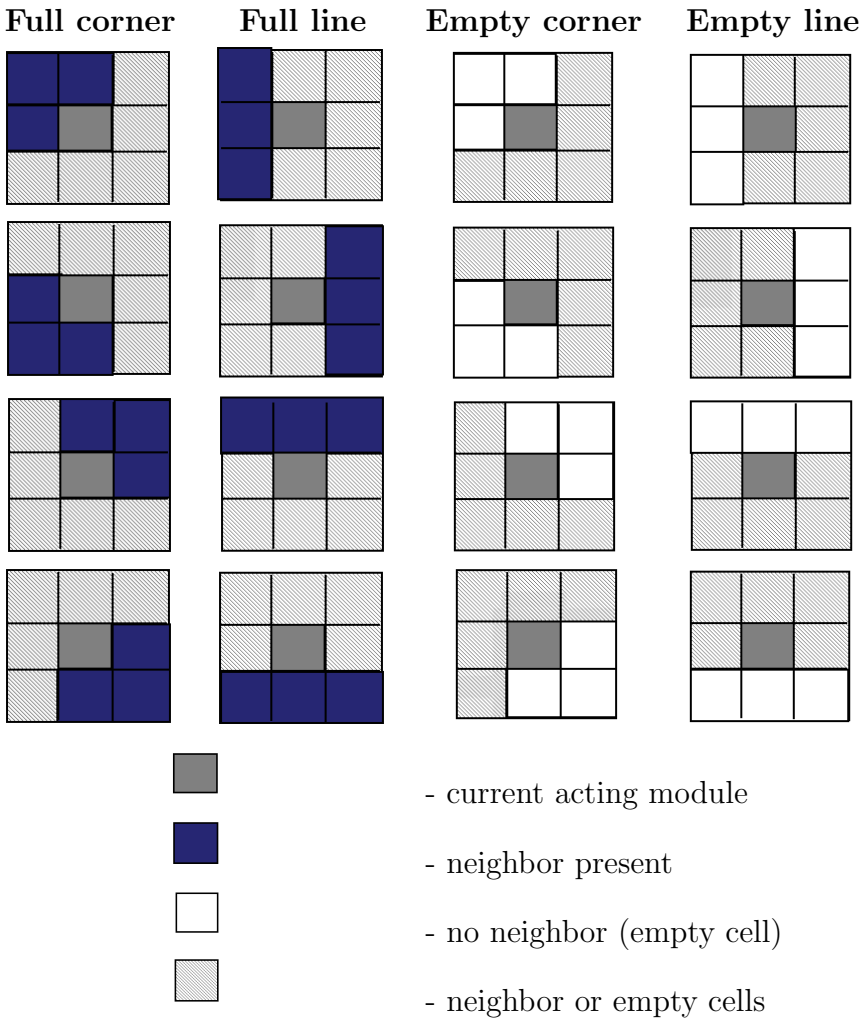


Figure 3-6: All 16 observation masks for feature generation: each mask, combined with one of 9 possible actions, generates one of 144 features describing the space of policies.

	15 modules	**20 modules**
GAPS	0.1±0.1	5±1.7
LLGAPS with 144 features	6.6±1.3	9.7±0.2

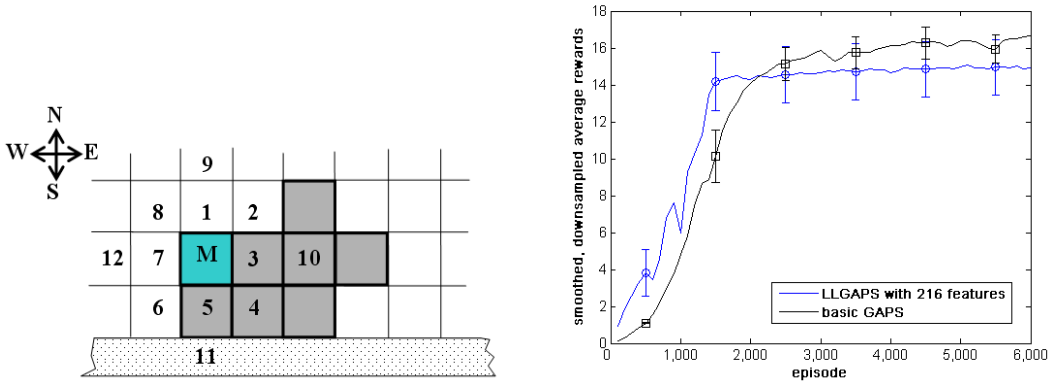
Table 3.1: Mean number of configurations that do not correspond to locomotion gaits, out of 10 test trials, for 10 policies learned by GAPS vs. LLGAPS with 144 features, after 6,000 episodes of learning, with standard error. We will use the mean number of non-gait configurations throughout this thesis as one of the performance measures for various algorithms, representations and constraints. The standard error $\hat{\sigma}/\sqrt{(N)}$ (N is the number of sampled policies, here equal to 10) included in this type of table both gives an estimate of how spread-out the values from different policies are, and is a measure of quality of the mean value as a statistic.

and one per extra neighbor absent, where each of those 8 partial masks generates 9 features, one per possible action. We found that increasing the observation size indeed slows the basic GAPS algorithm down (figure 3-7b, where LLGAPS ran with a much lower learning rate to avoid too-large update steps: $\alpha = 0.005$ decreasing to $\alpha = 1e^{-5}$ over the first 1,500 episodes and remaining at the minimal value thereafter). During the first few hundred episodes, LLGAPS does better than GAPS. However, we have also found that there is no significant difference in performance after both have found their local optima (mixed-factor ANOVA with repeated measures every 1000 episodes, starting at episode 500 reveals no significance for GAPS vs. LLGAPS but a significant difference at 99% confidence level for the interaction between the two factors of episodes and algorithm). Table 3.1 shows that the feature-based LLGAPS is even more prone to fall for local optima.

Two possible explanations would account for this phenomenon. On the one hand, feature spaces need to be carefully designed to avoid these pitfalls, which shifts the human designer's burden from developing distributed control algorithms to developing sets of features. The latter task may not be any less challenging. On the other hand, even well-designed feature spaces may eliminate redundancy in the policy space, and therefore make it harder for local search to reach an acceptable optimum. This effect is the result of compressing the policy representation into fewer parameters. When an action is executed that leads to worse performance, the features that contributed to the selection of this action will all get updated. That is, the update step results a simultaneous change in different places in the observation-action space, which can more easily create a new policy with even worse performance. Thus, the parameters before the update would be locally optimal.

3.5.5 Learning from individual experience

When one agent learns from the combined experience of all modules' observations, actions and rewards, it is not surprising that the learning algorithm converges faster



(a) (b)

Figure 3-7: (a) During experiments with an extended observation, modules had access to 12 bits of observation as shown here. (b) Smoothed (100 point moving window), downsampled average rewards obtained by 6 modules over 10 trials, with standard error: comparison between basic GAPS and LLGAPS.

than in the distributed case, as the learner has more data on each episode. However, with the size of the robot (i.e., the number of modules acting and learning together) grows the discrepancy between the cumulative experience of all agents, and the individual experience of one agent. When many modules are densely packed into an initial shape at the start of every episode, we can expect the ones "stuck" in the middle of the shape to not gather any experience at all, and to receive zero reward, for at least as long as it takes the modules on the perimeter to learn to move out of the way. While a randomized initial position is supposed to mitigate this problem, as the number of modules grows, so grows the probability that any one module will rarely or never be on the perimeter at the outset of an episode. This is a direct consequence of our decision to start locomotion from a tightly packed rectangular configuration with a height-to-length ratio closest to 1. That decision was motivated by the self-unpacking and deploying scenario often mentioned in SRMR research.

Hypothesis: Distributed GAPS will learn much slower than centralized factored GAPS, and may not learn at all.

As demonstrated in figure 3-8, indeed in practice, as few as 15 modules running the fully distributed version of the GAPS algorithm do not find any good policies at all, even after 100,000 episodes of learning. This limitation will be addressed in chapters 4 and 5.

3.5.6 Comparison to hand-designed controllers

The structure of the reward signal used during the learning phase determines what the learned policies will do to achieve maximum reward. In the case of eastward locomotion the reward does not depend on the shape of the modular robot, only on how

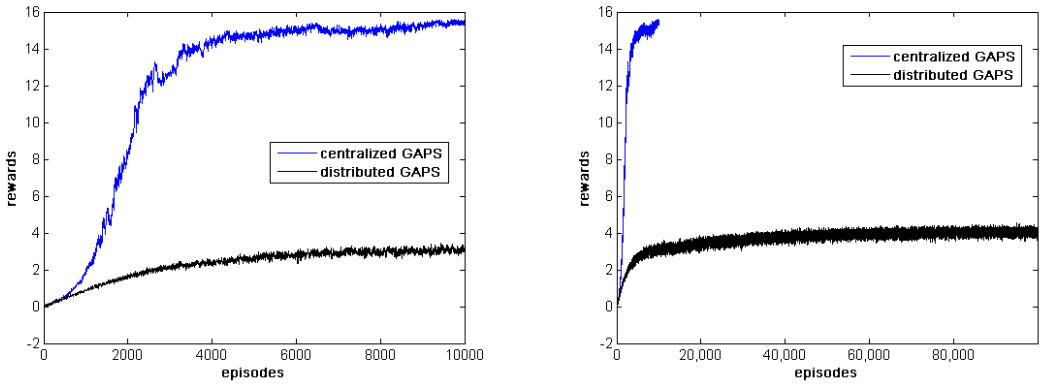


Figure 3-8: Comparison of learning performance between the centralized, factored GAPS and the distributed GAPS (DGAPS) implementations: smoothed (10-point window) average rewards obtained during 10 learning trials with 15 modules. (a) first 10,000 episodes, and (b) to 100,000 episodes of learning.

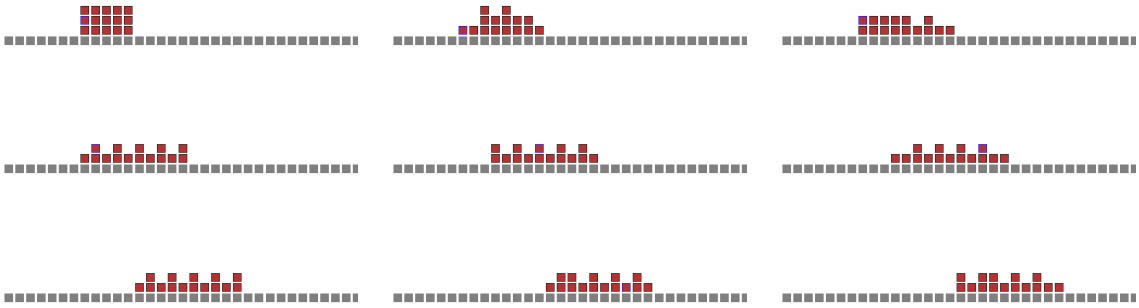


Figure 3-9: Screenshot sequence of 15 modules executing the best policy found by GAPS for eastward locomotion.

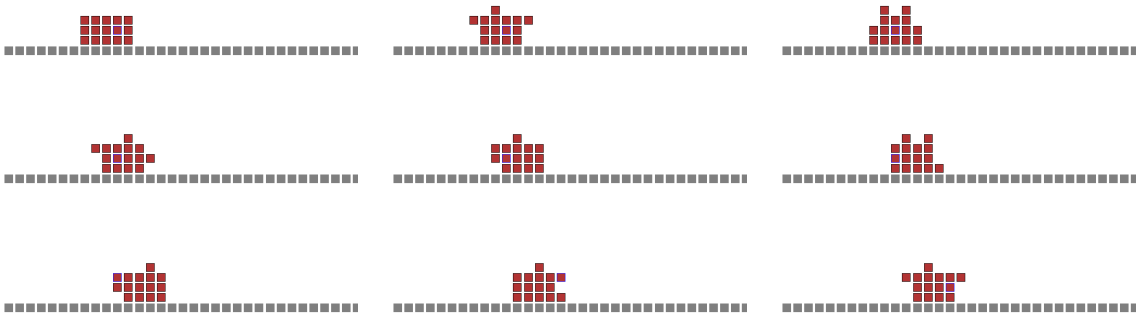
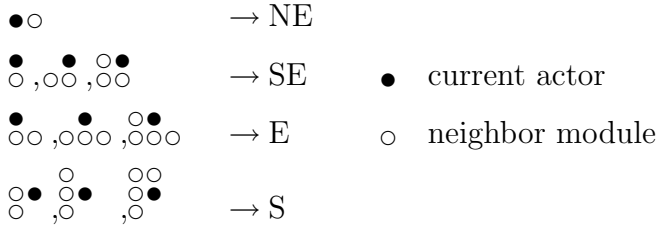


Figure 3-10: Screenshot sequence of 15 modules executing the hand-designed policy for eastward locomotion.

Table 3.2: Rules for eastward locomotion, distilled from the best policy learned by GAPS.



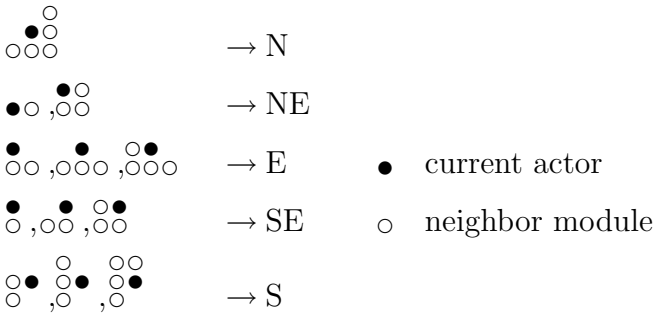
far east is has gone at the end of an episode. On the other hand, the hand-designed policies of Butler et al. (2001) were specifically developed to maintain the convex, roughly square or cubic overall shape of the robot, and to avoid any holes within that shape. It turns out that one can go farther faster if this constraint is relaxed. The shape-maintaining hand-designed policy, executed by 15 modules for 50 time steps (as shown in figure 3.5.6), achieves an average reward per episode of 5.8 ($\sigma = 0.9$), whereas its counterpart learned using GAPS (execution sequence shown in figure 3.5.6) achieves an average reward of 16.5 ($\sigma = 1.2$). Table 3.2 graphically represents the rules distilled from the best policy learned by GAPS. The robot executing this policy unfolds itself into a two-layer thread, then uses a thread-gait to move East. While very good at maximizing this particular reward signal, these policies no longer have the "nice" properties of the hand-designed policies of Butler et al. (2001). By learning with no constraints and a very simple objective function (maximize horizontal displacement), we forgo any maintenance of shape, or indeed any guarantees that the learning algorithm will converge on a policy that is a locomotion gait. The best we can say is that, given the learning setup, it will converge on a policy that locally maximizes the simple reward.

3.5.7 Remarks on policy correctness and scalability

Some learned policies are indeed correct, if less than perfect, locomotion gaits. Their correctness or acceptability is determined by an analysis of learned parameters. Consider the stochastic policy to which GAPS converged in table 3.3. The table only shows those observation-action pairs where $\theta(o, a) >> \theta(o, a')$ for all a' and where executing a results in motion. This policy is a local optimum in policy space — a small change in any $\theta(o, a)$ will lead to less reward. It was found by the learner on a less than ideal run and it is different from the global optimum in table 3.2. We argue that this policy will still correctly perform the task of eastward locomotion with high probability as the $\theta(o, a)$ gets larger for the actions a shown.

Note first of all that if the rules in 3.3 were deterministic, then we could make an argument akin to the one in Butler et al. (2001) where correctness is proven for a hand-designed policy. Intuitively, if we start with a rectangular array of modules and assume that each module gets a chance to execute an action during a turn, then some rule can always be applied, and none of the rules move any modules west, so that eastward locomotion will always result. This crucially depends on our

Table 3.3: Learned rules for eastward locomotion: a locally optimal gait.



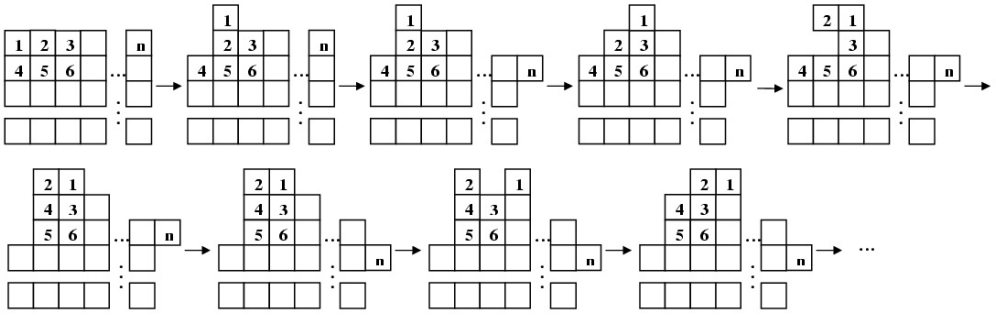


Figure 3-11: Locomotion of nm modules following rules in table 3.3, treated as deterministic. After the last state shown, module **n** executes action S, modules **1** then **2** execute action E, then module **4** can execute action NE, and so forth. The length of the sequence is determined by the dimensions n and m of the original rectangular shape.

assumption of synchronous turn-taking. Figure 3-11 shows the first several actions of the only possible cyclic sequence for nm modules following these rules if treated as deterministic. The particular assignment of module IDs in the figure is chosen for clarity and is not essential to the argument.

However, the rules are stochastic. The probability of the robot's center of mass moving eastward over the course of an episode is equal to the probability, where there are T turns in an episode, that during t out of T turns the center of mass moved eastward and $t > T - t$. As $\theta(o, a) \rightarrow \infty$ so $\pi(o, a, \theta) \rightarrow 1$ for the correct action a, so t will also grow and we will expect correct actions. And when the correct actions are executed, the center of mass is always expected to move eastwards during a turn. Naturally, in practice once the algorithm has converged, we can extract deterministic rules from the table of learned parameters by selecting the highest parameter value per state. However, it has also been shown (Littman 1994) that for some POMDPs the best stochastic state-free policy is better than the best deterministic one. It may be that randomness could help in our case also.

While some locally optimal policies will indeed generate locomotion gaits like the one just described, others will instead form protrusions in the direction of higher reward without moving the whole robot. Figure 3-12 shows some configurations

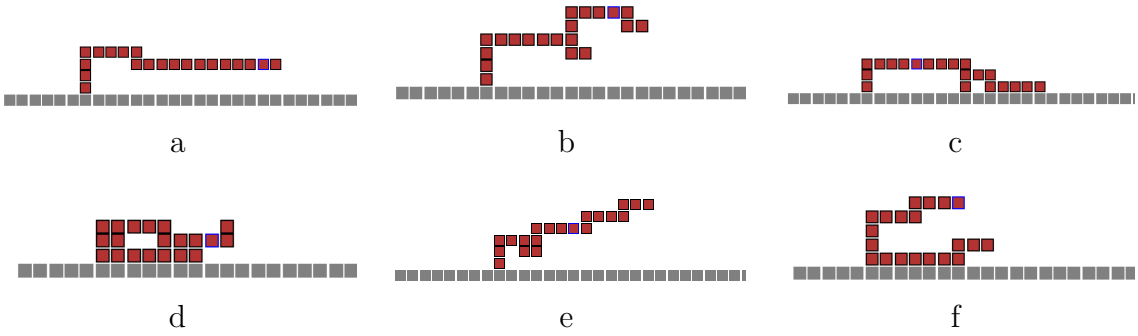
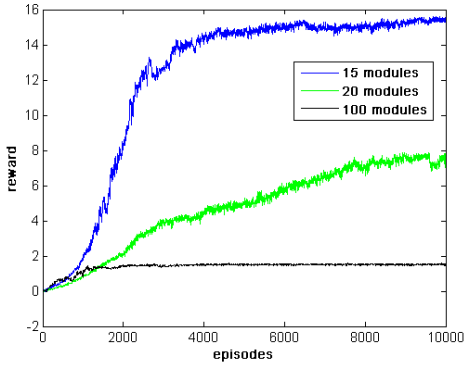


Figure 3-12: Some locally optimal policies result in robot configurations that are not locomotion gaits.

resulting from such policies — the prevalence of these will grow with the robot size. In the next chapter, we examine in detail several strategies for constraining the policy search and providing more information to the learning agents in an effort to avoid local optima that do not correspond to acceptable locomotion gaits.



(a)

no. modules	no. bad configs
15	0.1±0.1
20	5±1.7
100	10±0.0

(b)

Figure 3-13: Scalability issues with GAPS: 15, 20 and 100 modules. (a) Smoothed (10-point window) average reward over 10 learning trials. (b) Mean number of stuck configurations, out of 10 test trials for each of 10 learned policies after 10,000 episodes of learning, with standard error.

Local optima, especially those that do not correspond to acceptable locomotion gaits, become more of a problem for larger robots consisting of more modules. Figure 3-13a shows that 20 modules running the same GAPS algorithm do not perform as well as 15 modules. If we increase the size of the robot to 100 modules, no significant increase in reward over time occurs. The table in figure 3-13b demonstrates that the robot gets stuck in locally optimal but unacceptable configurations more often as the number of modules grows.

no. modules	no. bad configs	no. bad policy
15	0.1±0.1	2±1.3
20	0.9±0.8	3±1.5

Table 3.4: Results of introducing a stability requirement into the simulator: mean number of non-gaits, out of 10 test runs, for 10 learned policies, with standard error ("bad policy" refers to test in which the modules stopped moving after a while despite being in an acceptable configuration; this effect is probably due to the reward structure in these experiments: since unstable configurations are severely penalized, it is possible that the modules prefer, faced with a local neighborhood that previously led to an unstable configuration, to not do anything.)

(a) (b)

Figure 3-14: Results of introducing a stability constraint into the simulator: two non-gait configurations.

At this point we might note that some of the undesirable configurations shown in figure 3-12 would not be possible on any physical robot configuration, since they are not stable. Although we explicitly state our intent to work only with abstract rules of motion, at this point we might consider introducing another rule enforced by the simulator: (3) the robot's center of mass must be within its footprint. In a series of experiments, we let the modules learn in this new environment, where any episode where rule (3) is broken is immediately terminated with a reward of -10 to all modules (centralized factored learning).

Hypothesis: Penalizing unstable configurations will reduce the number of dysfunctional (stuck) locally optimal configurations in GAPS.

The results of the experiments (table 3.4) show that non-gait local optima still exist, even though unstable long protrusions are no longer possible. Some such configurations can be seen in figure 3-14a and 3-14b. For the rest of this thesis, we return to the original two rules of the simulator.

3.6 Key issues in gradient ascent for SRMRs

Whenever stochastic gradient ascent algorithms are used, three key issues affect both the speed of learning and the resulting concept or policy on which the algorithm converges. These issues are:

number of parameters The more parameters we have to estimate, the more data or experience is required (sample complexity), which results in slower learning for a fixed amount of experience per history (episode of learning).

amount of experience Conversely, the more data or experience is available to the agents per learning episode, the faster learning will progress.

starting point Stochastic gradient ascent will only converge to a local optimum in policy space. Therefore the parameter settings at which the search begins matter for both the speed of convergence and the goodness of the resulting policy.

In lattice-based self-reconfiguring modular robots, these three variables are affected by a number of robot and problem parameters. Of the three, the starting point is the most straightforward: the closer initial policy parameters lie to a good locally optimal policy, the faster the robot will learn to behave well.

3.6.1 Number of parameters to learn

The more parameters need to be set during learning, the more knobs the agent needs to tweak, the more experience is required and the slower the learning process becomes. In lattice-based SRMRs, the following problem setup and robot variables affect the number of policy parameters that need to be learned.

Policy representation

Clearly, the number of possible observations and executable actions determines the number of parameters in a full tabular representation. In a feature-based representation, this number can be reduced. However, we have seen in this chapter that the recasting of the policy space into a number of log-linearly compositional features is not a straightforward task. Simple reduction in the number of parameters does not guarantee a more successful learning approach. Instead, good policies may be excluded from the reduced space representation altogether. There may be more local optima that do not correspond to acceptable policies at all, or less optima overall, which would result in a harder search problem.

The number of parameters to set can also be reduced by designing suitable preprocessing steps on the observation space of each agent.

Robot size

In modular robots, especially self-reconfiguring modular robots, the robot size influences the effective number of parameters to be learned. We can see this relationship clearly in our case study of locomotion by self-reconfiguration on a 2D lattice-based robot. The full tabular representation of a policy in this case involves 2^8 possible local observations and 9 possible actions, for a total of $256 \times 9 = 2,304$ parameters. However, if the robot is composed of only two modules, learning to leap-frog over

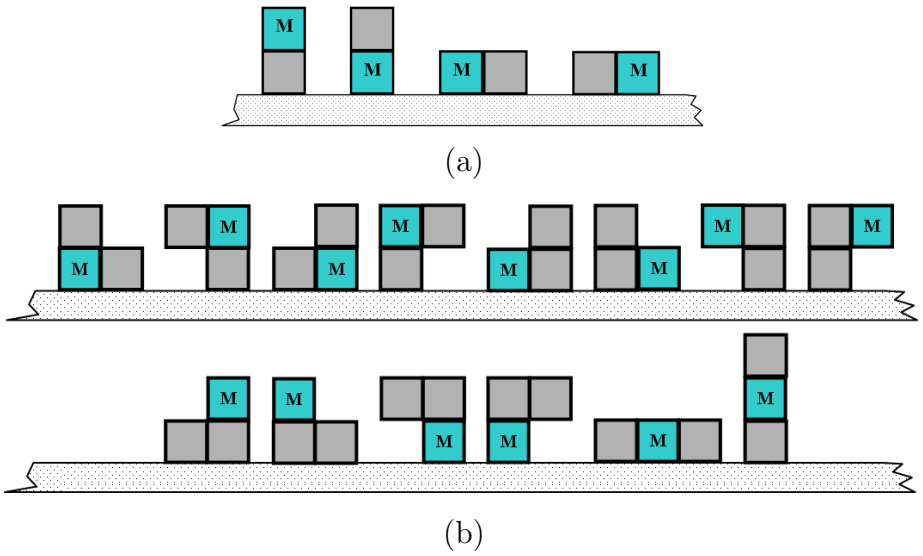


Figure 3-15: All possible local observations for (a) two modules, and (b) three modules, other than those in (a).

each other in order to move East, then effectively each agent only sees 4 possible observations, as shown in figure 3-15a. If there are only three modules, there are only 18 possible local observations. Therefore the number of parameters that the agent has to set will be limited to $4 \times 9 = 36$ or $18 \times 9 = 162$ respectively. This effective reduction will result in faster learning for smaller robots composed of fewer modules.

Search constraints

Another way of reducing the effective number of parameters to learn is to impose constraints on the search by stochastic gradient ascent. This can be achieved, for example, by containing exploration to only a small subset of actions.

3.6.2 Quality of experience

The amount and quality of available experience will influence both the speed and reliability of learning by stochastic gradient ascent. Specifically, experience during the learning phase should be representative of the expected situations for the rest of the robot lifetime, in order for the learned policy to generalize properly.

Episode length and robot size

For learning by GAPS, there is a straightforward correlation between the number of timesteps in an episode and the quality of the updates in that episode. Longer episodes result in larger counts of observations and actions, and therefore in better estimates of the "surprise factor" between the expected and actual actions performed.

In SRMRs, robot size also contributes to the relationship between episode length and quality of learning updates. In particular, for our case study task of locomotion by self-reconfiguration, there is a clear dependency between the number of modules in a robot and the capability of the simple reward function to discriminate between good and bad policies. To take an example, there is a clear difference between 15 modules executing a good locomotion gait for 50 timesteps and the same number of modules forming an arm-like protrusion, such as the one shown in figure 3-12a. This difference is both visible to us and to the learning agent through the reward signal (center of mass displacement along the x axis), which will be much greater in the case of a locomotion gait. However, if 100 modules followed both policies, after 50 timesteps, due to the nature of the threadlike locomotion gaits, the robot would not be able to differentiate between what we perceive as a good locomotion policy, or the start of an arm-like protrusion, based on reward alone. Therefore, as robot size increases, we must either provide more sophisticated reward functions, or increase episode length to increase the amount of experience per episode for the learning modules.

Shared experience

We have seen that centralized factored GAPS can learn good policies while fully distributed GAPS cannot. This is due to the amount and quality of experience the modules have access to during learning. In the centralized version, each module makes the same updates as all others as they share all experience: the sum total of all encountered observations and all executed actions play into each update. In the distributed version, each module is on its own for collecting experience and only has access to its own observation and action counts. This results in extremely poor to nonexistent exploration for most modules — especially those initially placed at the center of the robot. If it were possible for modules to share their experience in a fully distributed implementation of GAPS, we could expect successful learning due to the increase in both amount and quality of experience available to each learner.

3.7 Summary

We have formulated the locomotion problem for a SRMR as a multi-agent POMDP and applied gradient-ascent search in policy value space to solve it. Our results suggest that automating controller design by learning is a promising approach. We should, however, bear in mind the potential drawbacks of direct policy search as the learning technique of choice.

As with all hill-climbing methods, there is a guarantee of GAPS converging to a local optimum in policy value space, given infinite data, but no proof of convergence to the global optimum is possible. A local optimum is the best solution we can find to a POMDP problem. Unfortunately, not all local optima correspond to reasonable locomotion gaits.

In addition, we have seen that GAPS takes on average a rather long time (measured in thousands of episodes) to learn. We have identified three key issues that con-

tribute to the speed and quality of learning in stochastic gradient ascent algorithms such as GAPS, and we have established which robot parameters can contribute to the make-up of these three variables. In this chapter, we have already attempted, unsuccessfully, to address one of the issues — the number of policy parameters to learn — by introducing feature spaces. In the next two chapters, we explore the influence of robot size, search constraints, episode length, information sharing, and smarter policy representations on the speed and reliability of learning in SRMRs. The goal to keep in sight as we report the results of those experiments, is to find the right mix of automation and easily available constraints and information that will help guide automated search for the good distributed controllers.

Chapter 4

How Constraints and Information Affect Learning

We have established that stochastic gradient ascent in policy space works in principle for the task of locomotion by self-reconfiguration. In particular, if modules can somehow pool their experience together and average their rewards, provided that there are not too many of them, the learning algorithm will converge to a good policy. However, we have also seen that even in this centralized factored case, increasing the size of the robot uncovers the algorithm's susceptibility to local minima which do not correspond to acceptable policies. In general, local search will be plagued by these unless we can provide either a good starting point, or guidance in the form of constraints on the search space.

Modular robot designers are usually well placed to provide either a starting point or search constraints, as we can expect them to have some idea about what a reasonable policy, or at least parts of it, should look like. In this chapter, we examine how specifying such information or constraints affects learning by policy search.

4.1 Additional exploration constraints

In an effort to reduce the search space for gradient-based algorithms, we are looking for ways to give the learning modules some information that is easy for the designer to specify yet will be very helpful in narrowing the search. An obvious choice is to let the modules pre-select actions that can actually be executed in any one of the local configurations.

During the initial hundreds of episodes where the algorithm explores the policy space, the modules will attempt to execute undesirable or impossible actions which could lead to damage on a physical robot. Naturally, one may not have the luxury of thousands of trial runs on a physical robot anyway.

Each module will know which subset of actions it can safely execute given any local observation, and how these actions will affect its position; yet it will not know what new local configuration to expect when the associated motion is executed. Restricting search to legal actions is useful because it (1) effectively reduces the number

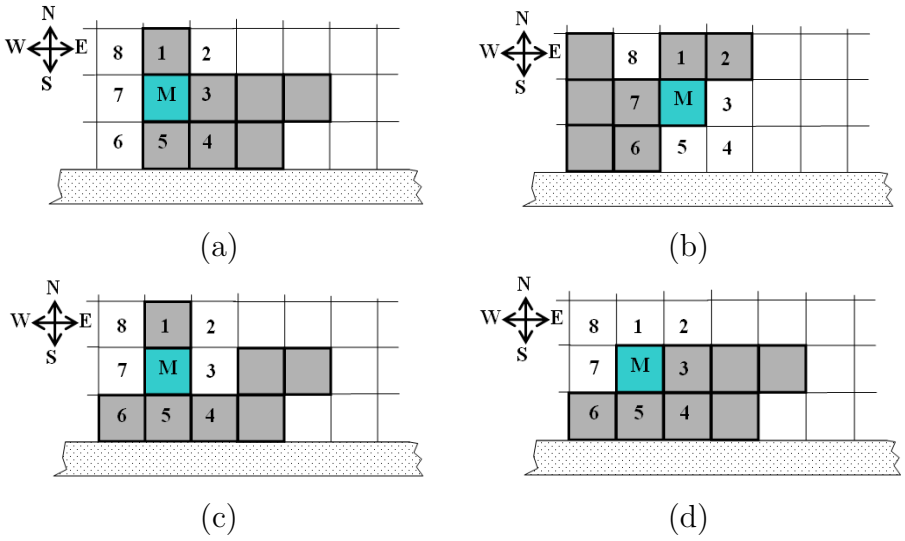


Figure 4-1: Determining if actions are legal for the purpose of constraining exploration: (a) $A = \{NOP\}$, (b) $A = \{NOP\}$, (c) $A = \{NOP\}$ (d) $A = \{2(NE), 7(W)\}$.

of parameters that need to be learned and (2) causes the initial exploration phases to be more efficient because the robot will not waste its time trying out impossible actions. The second effect is probably more important than the first.

The following rules were used to pre-select the subset A_t^i of actions possible for module i at time t, given the local configuration as the immediate Moore neighborhood (see also figure 4-1):

1. $A_t^i = \{NOP\}$[1] if three or more neighbors are present at the face sites
2. $A_t^i = \{NOP\}$ if two neighbors are present at opposite face sites
3. $A_t^i = \{NOP\}$ if module i is the only "cornerstone" between two neighbors at adjacent face sites[2]
4. $A_t^i = \{$legal actions based on local neighbor configuration and the sliding cube model$\}$
5. $A_t^i = A_t^i -$ any action that would lead into an already occupied cell

These rules are applied in the above sequence and incorporated into the GAPS algorithm by setting the corresponding $\theta(o_t, a_t)$ to a large negative value, thereby making it extremely unlikely that actions not in A_t^i would be randomly selected by the policy. Those parameters are not updated, thereby constraining the search at every time step.

[1] NOP stands for 'no operation' and means the module's action is to stay in its current location and not attempt any motion.

[2] This is a very conservative rule, which prevents disconnection of the robot locally.

We predict that the added space structure and constraints that were introduced here will result in the modular robot finding good policies with less experience.

4.2 Smarter starting points

Stochastic gradient ascent can be sensitive to the starting point in the policy space from which search initiates. Here, we discuss two potential strategies for seeding the algorithms with initial parameters that may lead to better policies.

4.2.1 Incremental GAPS learning

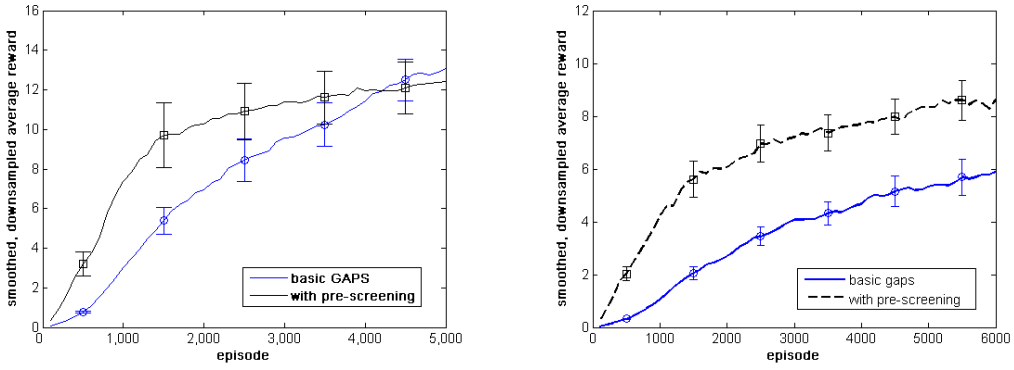
It is possible to leverage the modular nature of our platform in order to improve the convergence rate of the gradient ascent algorithm and reduce the amount of experience required by seeding the learning in an incremental way. As we have seen in chapter 3, the learning problem is easier when the robot has fewer modules than the size of its neighborhood, since it means that each module will see and have to learn a policy for a smaller number of observations.

If we start with only two modules, and add more incrementally, we effectively reduce the problem search space. With only two modules, given the physical coupling between them, there are only four observations to explore. Adding one other module means adding another 14 possible observations and so forth. The problem becomes more manageable. Therefore, we have proposed the Incremental GAPS (IGAPS) algorithm.

IGAPS works by initializing the parameters of N modules' policy with those resulting from $N-1$ modules having run GAPS for a number of episodes but not until convergence to allow for more exploration in the next (N'th) stage. Suppose a number of robotic agents need to learn a task that can also be done in a similar way with fewer robots. We start with the smallest possible number of robots; in our case of self-reconfiguring modular robots we start with just two modules learning to locomote, which run the GAPS algorithm. After the pre-specified number of episodes, their parameter values will be θ_{c2}. Then a new module is introduced, and the resulting three agents learn again, starting from θ_{c2} and using GAPS for a number of episodes. Then a fourth module is introduced, and the process is repeated until the required number of modules has been reached, and all have run the learning algorithm to convergence of policy parameters.

4.2.2 Partially known policies

In some cases the designer may be able to inform the learning algorithm by starting the search at a "good" point in parameter space. Incremental GAPS automatically finds such good starting points for each consecutive number of modules. A partially known policy can be represented easily in parameter space, in the case of representation by lookup table, by setting the relevant parameters to considerably higher values.



(a) (b)

Figure 4-2: Smoothed (100-point moving window), downsampled average rewards comparing the basic GAPS algorithm versus GAPS with pre-screening for actions resulting in legal moves, with standard error: (a) 15 modules (50 trials in basic condition, 30 trials with pre-screening), (b) 20 modules (20 trials in each condition).

Starting from a partially known policy may be especially important for problems where experience is scarce.

4.3 Experiments with pooled experience

As in section 3.5.2, the experimental results presented here were obtained during 10 independent learning trials, assuming all modules learn and execute the same policy from the same reward signal, which is the displacement of the robot's center of mass along the x axis during the episode.

Unless noted otherwise, in all conditions the learning rate started at $\alpha = 0.01$, decreased over the first $1,500$ episodes, and remained at its minimal value of 0.001 thereafter. The inverse temperature parameter started at $\beta = 1$, increased over the first $1,500$ episodes, and remained at its maximal value of 3 thereafter. This ensured more exploration and larger updates in the beginning of each learning trial. These parameters were selected by trial and error, as the ones consistently generating the best results. For clarity of comparison between different experimental conditions, we smoothed the rewards obtained during each learning trial with a 100-point moving average window, then downsampled each run, in order to smooth out variability due to ongoing exploration as the modules learn. We report smoothed, downsampled average reward curves with standard error bars ($\hat{\sigma}/\sqrt{N}$, N is the number of trials in the group) every 1,000 episodes.

4.3.1 Pre-screening for legal motions

We then tested the effect of introducing specific constraints into the learning problem. We constrain the search by giving the robot partial a priori knowledge of the effect

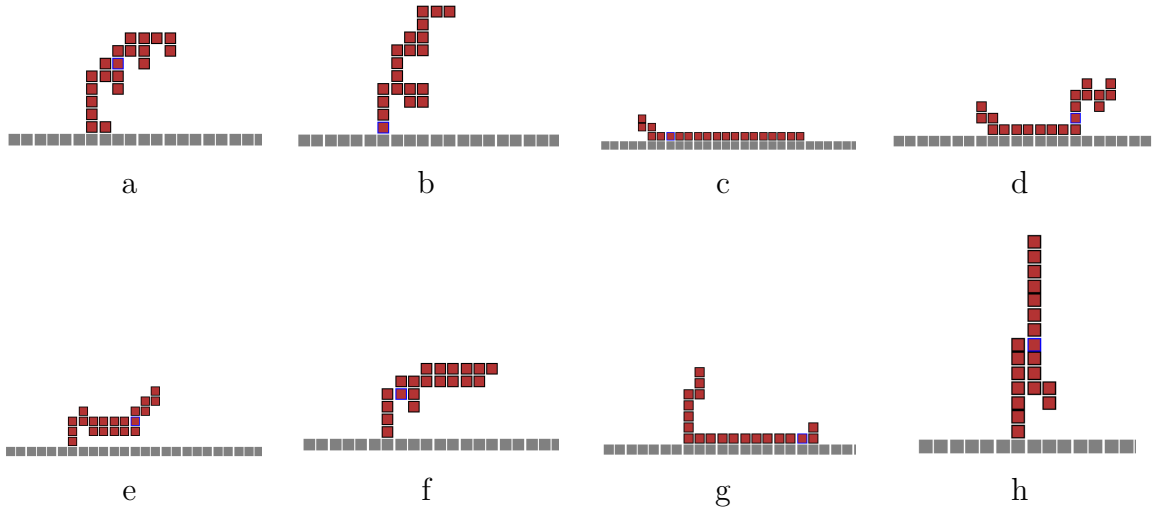


Figure 4-3: Some local optima which are not locomotion gaits, found by GAPS with pre-screening for legal actions and (a)-(f) 1-hop communications, (g)-(h) multi-hop communications.

of its actions. The module will be able to tell whether any given action will generate a legal motion onto a new lattice cell, or not. However, the module still will not know which state (full configuration of the robot) this motion will result in. Nor will it be able to tell what it will observe locally at the next timestep. The amount of knowledge given is really minimal. Nevertheless, with these constraints the robot no longer needs to learn not to try and move into lattice cells that are already occupied, or those that are not immediately attached to its neighbors. Therefore we predicted that less experience would be required for our learning algorithms.

Hypothesis: Constraining exploration to legal actions only will result in faster learning (shorter convergence times).

In fact, figure 4-2 shows that there is a marked improvement, especially as the number of modules grows.

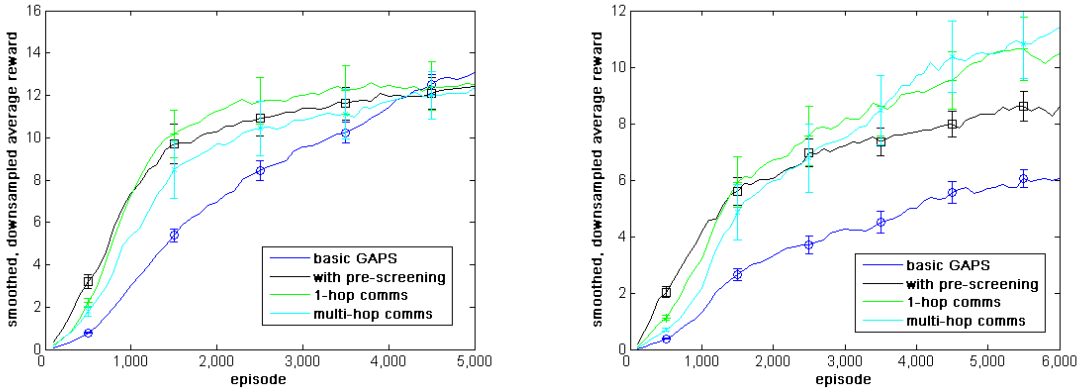
4.3.2 Scalability and local optima

Stochastic gradient ascent algorithms are guaranteed to converge to a local maximum in policy space. However, those maxima may represent globally suboptimal policies. Figure 3-12 shows execution snapshots of some locally optimal policies, where the robot is effectively stuck in a configuration from which it will not be able to move. Instead of consistently sending modules up and down the bulk of the robot in a thread-like gait that we have found to be globally optimal given our simple reward function, the robot develops arm-like protrusions in the direction of larger rewards. Unfortunately, for any module in that configuration, the only locally good actions, if

	15 modules	**20 modules**
GAPS	0.1±0.1	5±1.7
GAPS with pre-screened legal actions		
no extra comms	0.3±0.3	4.2±1.5
1-hop	0.3±0.2	2.5±1.3
multi-hop	0.3±0.2	0.2±0.1

Table 4.1: In each of 10 learning trials, the learning was stopped after 10,000 episodes and the resulting policy was tested 10 times. The table shows the mean number of times, with standard error, during these 10 test runs, that modules became stuck in some configuration due to a locally optimal policy.

any, will drive them further along the protrusion, and no module will attempt to go down and back as the rewards in that case will immediately be negative.



(a) (b)

Figure 4-4: Smoothed (100-point window), downsampled average rewards comparing the basic GAPS algorithm to GAPS with pre-screening for legal actions and an extra bit of communicated observation through a 1-hop or a multi-hop protocol, with standard error: (a) for 15 modules (50 basic GAPS trials, 30 GAPS with pre-screening trials, 20 trials each communications), (b) for 20 modules (20 trials each basic GAPS and pre-screening, 10 trials each communications).

Local optima are always present in policy space, and empirically it seems that GAPS, with or without pre-screening for legal motions, is more likely to converge to one of them with increasing number of modules comprising the robot.

Having encountered this problem in scaling up the modular system, we introduce new information to the policy search algorithms to reduce the chance of convergence to a local maximum.

4.3.3 Extra communicated observations

In our locomotion task, locally optimal but globally suboptimal policies do not allow modules to wait for their neighbors to form a solid supporting base underneath them. Instead, they push ahead forming long protrusions. We could reduce the likelihood of such formations by introducing new constraints, artificially requiring modules to stop moving and wait in certain configurations. For that purpose, we expand the local observation space of each module by one bit, which is communicated by the module's South/downstairs neighbor, if any. The bit is set if the South neighbor is not supported by the ground or another module underneath. If a module's bit is set it is not allowed to move at this timestep. We thus create more time in which other modules may move into the empty space underneath to fill in the base. Note that although we have increased the number of possible observations by a factor of 2 by introducing the extra bit, we are at the same time restricting the set of legal actions in half of those configurations to $\{NOP\}$. Therefore, we do not expect GAPS to require any more experience to learn policies in this case than before.

We investigate two communication algorithms for the setting of the extra bit. If the currently acting module is M_1 and it has a South neighbor M_2, then either 1) M_1 asks M_2 if it is supported; if not, M_2 sends the `set-bit` message and M_1's bit is set (this is the one-hop scheme), or 2) M_1 generates a support request that propagates South until either the ground is reached, or one of the modules replies with the `set-bit` message and all of their bits are set (this is the multi-hop scheme).

Hypothesis: Introducing an additional bit of observation communicated by the South neighbor will reduce the number of non-gait local optima found by GAPS.

Experimentally (see table 4.1) we find that it is not enough to ask just one neighbor. While the addition of a single request on average halved the number of stuck configurations per 10 test trials for policies learned by 20 modules, the chain-of-support multi-hop scheme generated almost no such configurations. On the other hand, the addition of communicated information and waiting constraints does not seem to affect the amount of experience necessary for learning. Figure 4-4a shows the learning curves for both sets of experiments with 15 modules practically undistinguishable.

However, the average obtained rewards should be lower for algorithms that consistently produce more suboptimal policies. This distinction is more visible when the modular system is scaled up. When 20 modules run the four proposed versions of gradient ascent, there is a clear distinction in average obtained reward, with the highest value achieved by policies resulting from multi-hop communications (see figure 4-4b). The discrepancy reflects the greatly reduced number of trial runs in which modules get stuck, while the best found policies remain the same in all conditions.

A note on directional information

Table 4.1 shows that even with the extra communicated information in the multi-hop scenario, it is still possible for modules to get stuck in non-gait configurations. Might we be able to reduce the number of such occurrences to zero by imparting even more

information to the modules? In particular, if the modules knew the required direction of motion, would the number of "stuck" configurations be zero?

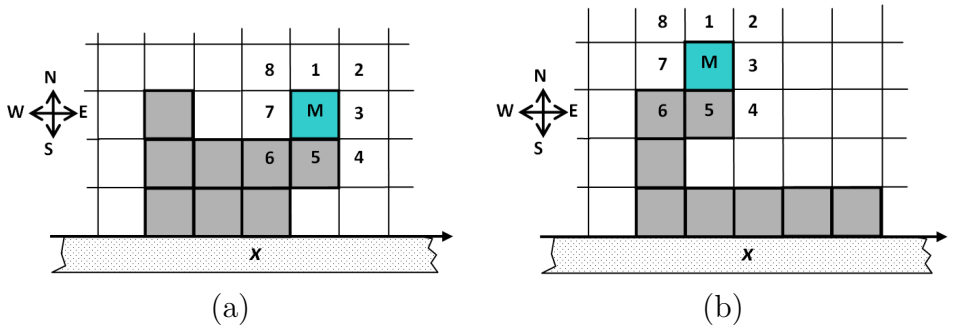


Figure 4-5: Two global configurations that are aliased assuming only local, partial observations, even in the presence of directional information: (a) module M is really at the front of the robot and should not move, (b) module M should be able to move despite having no "front" neighbors.

In anticipation of chapter 6, where we introduce a sensory model allowing modules to perceive local gradients, let us assume that each module knows the general direction (eastward) of desired locomotion. It can still only use this new directional information in a local way. For example, we can constrain the module to only attempt motion if its column is supported by the ground, as discussed earlier, *and* if it perceives itself locally to be on the "front" of the robot with respect to the direction of desired motion. Even then, the module can not differentiate locally between the two situations depicted in figure 4-5, and therefore, the number of stuck configurations may not be assumed to go to zero even with directional information, so long as the condition of local, partial observation is preserved. Other communication schemes to differentiate between such situations may be devised, but they are outside the scope of this thesis.

Empirically, we ran 10 trials of GAPS with pre-screening for legal actions and multi-hop communications with additional directional information available to 15 and 20 modules. Each module decided whether or not to remain stationary according to the following rule: if its column is not supported by the ground, as returned by the chain of communications, and if it sees no neighbors on the right (that is, the front), then the modules chooses the NOP action. Otherwise, an action is picked according to the policy and legal pre-screening. Each of the resulting 10 policies for both robot sizes where then tested 10 times. On average, 15 modules got stuck 0.2 ± 0.2 times out of 10 test runs, and 20 modules got stuck 0.3 ± 0.2 times in a non-gait configuration.

The stochastic nature of the policies themselves and of the learning process makes it impossible for us to devise strategies where zero failures is guaranteed, short of writing down a complete policy at the start. The best we can do is minimize failure by reasonable means. For the rest of this chapter, and until chapter 6, we will not consider any directional information to be available to the modules.

# modules	height=3	height=4	
	baseline	baseline	multi-hop
15	0.1±0.1	0.2±0.1	-
16	0.1±0.1	5.4±1.3	2.1±1.0
17	0.1±0.1	3.9±1.5	2.5±1.3
18	1.0±1.0	1.5±1.1	0.5±0.3
19	0.2±0.1	3.3±1.3	1.1±1.0
20	0.0±0.0	5.0±1.7	0.2±0.1

Table 4.2: Mean number of non-gait stuck configurations per 10 test trials, with standard error, of 10 policies learned by modules running centralized GAPS with (multi-hop) or without (baseline) pre-screening for legal actions and multi-hop communications.

4.3.4 Effect of initial condition on learning results

We have established that the larger 20-module robot does not in general learn as fast or as well as the smaller 15-module robot. However, it is important to understand that there is no simple relationship between the number of robot modules and the quality of the learned policies. We do, however, see the following effects:

Importance of initial condition The same number of modules started in different configurations will likely lead to different learning times and resulting policies.

Small size effect When the robot is composed of up to 10 modules, in reasonable initial configurations, the learning problem is relatively easy due to a limited number of potential global configurations.

Large size effect Conversely, when the robot is composed of upwards of 100 modules, the sheer scale of the problem results in a policy space landscape with drastically more local optima which do not correspond to locomotion gaits. In this case, even with the best initial conditions, modules will have trouble learning in the basic setup. However, biasing the search and scaling up incrementally, as we explore in this chapter, will help.

Therefore, when comparing different algorithms and the effect of extra constraints and information, throughout this thesis, we have selected two examples to demonstrate these effects: 1) 15 modules in a lower aspect ratio initial configuration (height/width = 3/5), which is helpful for horizontal locomotion, and 2) 20 modules in a higher aspect ratio initial condition (height/width = 4/5), which creates more local optima related problems for the learning process. The initial configurations were chosen to most closely mimic a tight packing for transportation in rectangular "boxes".

In table 4.2, we report one of the performance measures used throughout this thesis (the mean number of non-gait configurations per 10 test runs of a policy) for all robot sizes between 15 and 20 modules, in order to demonstrate the effect of the initial condition. In all cases, at the start of every episode the robot's shape was a

	25 modules	90 modules	100 modules
height=3	1.1±1.0	9±1.0	9.4±0.4

Table 4.3: Larger robots get stuck with locally optimal non-gait policies despite a lower aspect ratio of the initial configuration.

rectangle or a rectangle with an appendage, depending on the number of modules and the required height. For example, 16 modules with a height of 4 were initially in a 4×4 square configuration, but with a height of 3, they were placed in a $5 \times 3 + 1$, the last module standing on the ground line attached to the rest of the robot on its side. We see that all robot sizes do almost equally well when they have a low aspect ratio (height=3). When the initial aspect ratio is increased (height=4), there is a much higher variance in the quality of policies, but in most cases the same number of modules learn far worse than before. We can also see that introducing legality constraints and multi-hop communications in the latter case of the taller initial configuration helps reduce the number of non-gait configurations across the board.

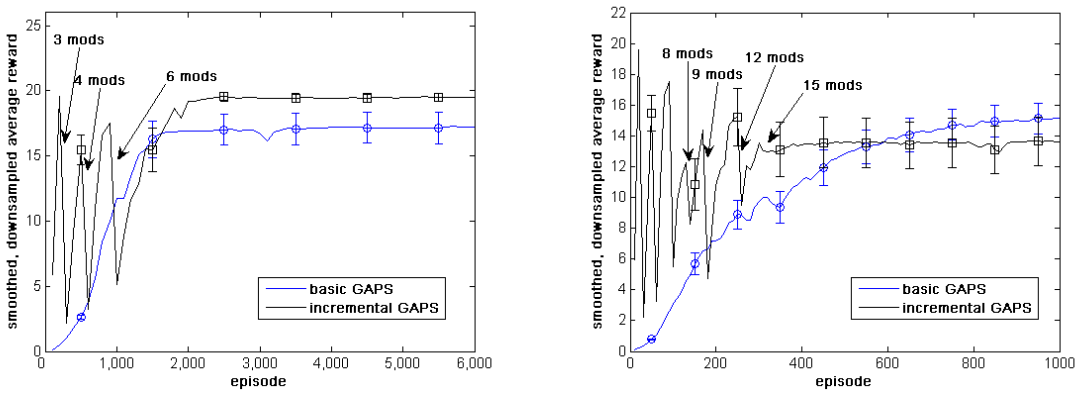
While the initial condition is important to policy search, it does not account for all effects associated with robot size: the number of modules also matters, especially when it increases further. Table 4.3 shows that much larger robots still find more non-gait policies.

Clearly, both initial conditions and robot size influence the learning process. In the rest of this thesis, with the exception of experiments in incremental learning in section 4.3.5, we will continue to compare the performance of algorithms, representations and search constraints for two points: 15 and 20 modules. This allows us to test our approach on robots with different aspect ratios. The choice is reasonable, as part of the motivation for modular robots is to see them reconfigure into useful shapes after being tightly packed for transportation. Nevertheless we should bear in mind that initial conditions, not only in terms of the point in policy space where search starts (we will turn to that next), but also in terms of the robot configuration, will matter for learning performance.

4.3.5 Learning from better starting points

Hypothesis: Incremental GAPS will result in both faster learning, and fewer undesirable local optima, as compared with standard GAPS.

Figure 4-6 shows how incremental GAPS performs on the locomotion task. Its effect is most powerful when there are only a few modules learning to behave at the same time; when the number of modules increases beyond the size of the local observation space, the effect becomes negligible. Statistical analysis fails to reveal any significant difference between the mean rewards obtained by basic GAPS vs. IGAPS for 6, 15, or 20 modules. Nevertheless, we see in figure 4-7b that only 2.1 out of 10 test runs on average produce arm-like protrusions when the algorithm was seeded in the



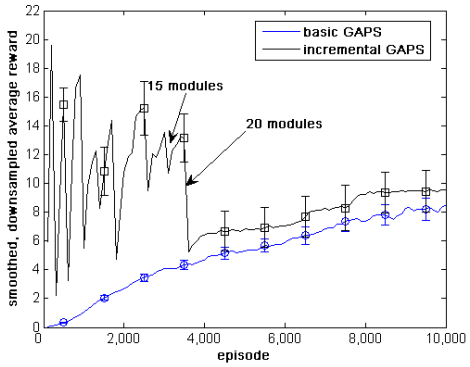
(a) (b)

Figure 4-6: Smoothed, downsampled average rewards, with standard error, over 10 trials: comparison between basic GAPS and the incremental extension (a) for 6 modules and (b) for 15 modules.

incremental way, against 5 out of 10 for basic GAPS. The lack of statistical significance on the mean rewards may be due to the short time scale of the experiments: in 50 timesteps, 20 modules may achieve very similar rewards whether they follow a locomotion gait or build an arm in the direction of larger rewards. A drawback of the incremental learning approach is that it will only work to our advantage on tasks where the optimal policy does not change as the number of modules increases. Otherwise, IGAPS may well lead us more quickly to a globally suboptimal local maximum.

We have taken the incremental idea a little further in a series of experiments where robot size was increased in larger increments. Starting with 4 modules in a 2×2 configuration, we have added enough modules at a time to increase the square length by one, eventually reaching a 10×10 initial configuration. The results can be seen in figure 4.3.5 and table 4.3.5. The number of locally optimal non-gait configurations was considerable for 100 modules, even in the IGAPS condition, but it was halved with respect to the baseline GAPS.

Figure 4-9 shows the results of another experiment in better starting points. Usually the starting parameters for all our algorithms are initialized to either small random numbers or all zeros. However, sometimes we have a good idea of what certain parts of our distributed controller should look like. It may therefore make sense to seed the learning algorithm with a good starting point by imparting to it our incomplete knowledge. We have made a preliminary investigation of this idea by partially specifying two good policy "rules" before learning started in GAPS. Essentially we initialized the policy parameters to a very strong preference for the 'up' (North) action when the module sees neighbors only on its right (East), and a correspondingly strong preference for the 'down' (South) action when the module sees neighbors only on its left (West).



(a)

	15 modules	20 modules
GAPS	0.1±0.1	5±1.7
IGAPS	0.3±0.3	2.1±1.3

(b)

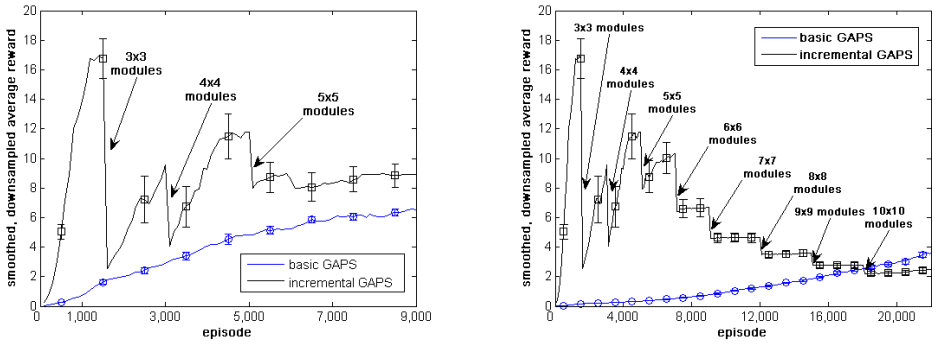
Figure 4-7: Incremental GAPS with 20 modules: (a) Smoothed (100-point window), downsampled average rewards obtained by basic vs. incremental GAPS while learning locomotion (b) The table shows the mean number, with standard error, of dysfunctional configurations, out of 10 test trials for 10 learned policies, after 15 and 20 modules learning with basic vs. incremental GAPS; IGAPS helps as the number of modules grows.

Hypothesis: Seeding GAPS with a partially known policy will result in faster learning and fewer local optima.

A two-way mixed-factor ANOVA on episode number (within-condition variation: tested after 1,500 episodes, 3,000 episodes, 6,000 episodes and 10,000 episodes) and algorithm starting point (between-condition variation) reveals that mean rewards obtained by GAPS with or without partial policy knowledge differ significantly ($F = 19.640, p < .01$) for 20 modules, but not for 15. Again, we observe an effect when the number of modules grows. Note that for 15 modules, the average reward is (not statistically significantly) lower for those modules initialized with a partially known policy. By biasing their behavior upfront towards vertical motion of the modules, we have guided the search away from the best-performing thread-like gait, which

non-gait locally optimal configurations		
	gaps	**incremental**
4x4 modules	5.4±1.3	1.3±0.9
5x5 modules	10±0.0	1.7±1.2
10x10 modules	10±0.0	4.3±1.6

Table 4.4: Mean number, with standard error, of locally optimal configurations which do not correspond to acceptable locomotion gaits out of 10 test trial for 10 learned policies: basic GAPS algorithm vs. incremental GAPS with increments by square side length. All learning trials had episodes of length T=50, but test trials had episodes of length T=250, in order to give the larger robots time to unfold.



(a) (b)

Figure 4-8: Incremental learning by square side length: smoothed, downsampled average rewards for (a) 25 and (b) 100 modules learning eastward locomotion with basic vs. incremental GAPS. Rewards obtained in 50 timesteps by basic GAPS are greater than those of incremental GAPS for 100 modules; this highlights the effect of shorter episode lengths on learning, as policies learned by incremental GAPS are much more likely to be locomotion gaits (see text and table 4.3.5).

relies more heavily on horizontal motion. The effect will not necessarily generalize to any designer-determined starting point. In fact, we expect there to be a correlation between how many "rules" are pre-specified before learning and how fast a good policy is found.

We expect extra constraints and information to be even more useful when experience is scarce, as is the case when modules learn independently in a distributed fashion without tieing their policy parameters to each other.

4.4 Experiments with individual experience

In all of the experiments reported in section 3.5.2 the parameters of the modules' policies were tied; the modules were learning together as one, sharing their experience and their behavior. However, one of our stated goals for applying reinforcement learning to self-reconfigurable modular robots is to allow modules to adapt their controllers at run-time, which necessitates a fully distributed approach to learning, and individual learning agents inside every module. An important aspect of GAPS-style algorithms is that they can be run in such an individual fashion by agents in a multi-agent POMDP and they will make the same gradient ascent updates provided that they receive the same experience. Tieing policy parameters together essentially increased the amount of experience for the collective learning agent. Where a single module makes T observations and executes T actions during an episode, the collective learning agent receives nT observations and nT actions during the same period of time, where n is the number of modules in the robot. If we run n independent learners using the same GAPS algorithm on individual modules, we can therefore

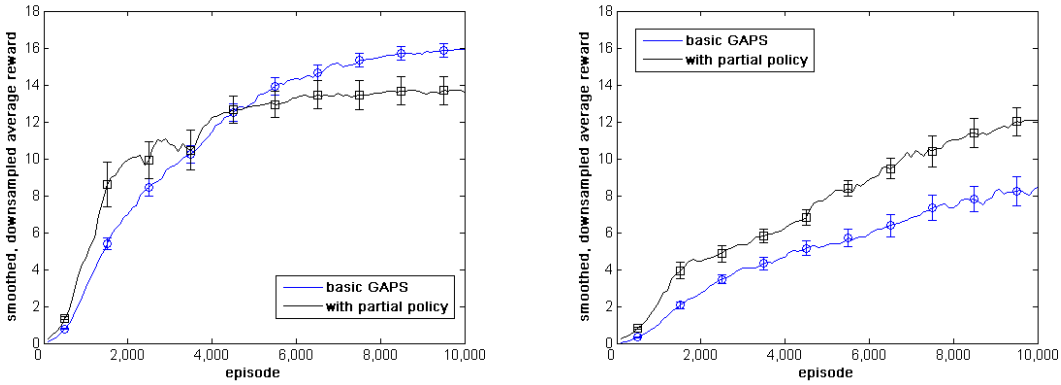


Figure 4-9: Smoothed average rewards over 10 trials: effect of introducing a partially known policy for (a) 15 modules and (b) 20 modules.

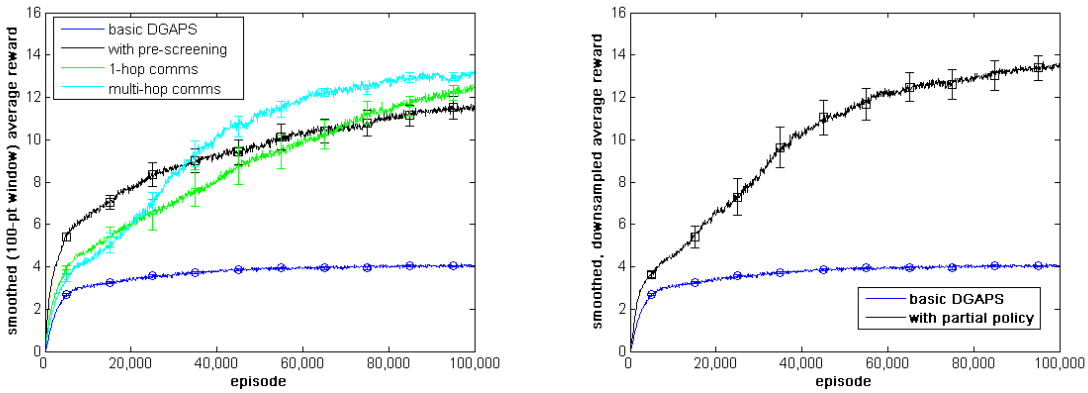
expect that it will take proportionately more time for individual modules to find good policies on their own.

Here we describe the results of experiments on modules learning individual policies without sharing observation or parameter information. In all cases, the learning rate started at $\alpha = 0.01$, decreased uniformly over 7,000 episodes until 0.001 and remained at 0.001 thereafter. The inverse temperature started at $\beta = 1$, increased uniformly over 7,000 episodes until 3 and remained at 3 thereafter. The performance curves in this sections were smoothed with a 100-point moving average and downsampled for clarity.

Hypothesis: Legality constraints and and extra observation bit communicated by neighbors will have a more dramatic effect in distributed GAPS.

We see in figure 4-10 that the basic unconstrained version of the algorithm is struggling in the distributed setting. Despite a random permutation of module positions in the start state before each learning episode, there is much less experience available to individual agents. Therefore we observe that legal actions constraints and extra communicated observations help dramatically. Surprisingly, without any of these extensions, GAPS is also greatly helped by initializing the optimization with a partially specified policy. The same two "rules" were used in these experiments as in section 4.3.5. A two-way 4×2 ANOVA on the mean rewards after 100,000 episodes was performed with the following independent variables: information and constraints group (one of: none, legal pre-screening, 1-hop communications and multi-hop communications), and partially known policy (yes or no). The results reveal a statistically significant difference in means for both main factors, and for their simple interaction, at the 99% confidence level.[3]

[3]These results do not reflect any longitudinal data or effects.



(a) (b)

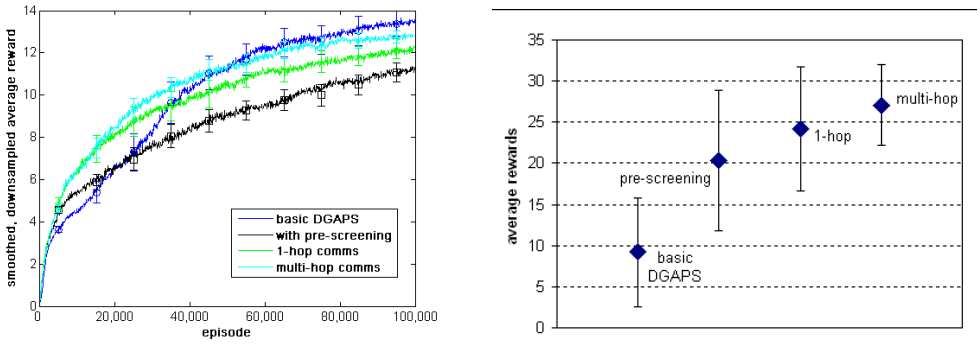
Figure 4-10: Smoothed, downsampled average rewards over 10 trials with 15 modules learning to locomote using only their own experience, with standard error: (a) basic distributed GAPS vs. GAPS with pre-screening for legal motions vs. GAPS with pre-screening and 1-hop (green) or multi-hop (cyan) communications with neighbors below, and (b) basic distributed GAPS vs. GAPS with all actions and a partially specified policy as starting point.

Hypothesis: Seeding policy search with a partially known policy will help dramatically in distributed GAPS.

Figure 4-11a compares the average rewards during the course of learning in three conditions of GAPS, where all three were initialized with the same partial policy: 1) basic GAPS with no extra constraints, 2) GAPS with pre-screening for legal actions only, 3) GAPS with legal actions and an extra observation bit communicated by the 1-hop protocol, and 4) by the multi-hop protocol. The curves in 1) and 4) are almost indistinguishable, whereas 2) is consistently lower. However, this very comparable performance can be due to the fact that there is not enough time in each episode (50 steps) to disambiguate between an acceptable locomotion gait and locally optimal protrusion-making policies.

Hypothesis: Test runs with longer episodes will disambiguate between performance of policies learned with different sources of information. In particular, distributed GAPS with multi-hop communications will find many fewer bad local optima.

To test that hypothesis, in all 10 trials we stopped learning at 100,000 episodes and tested each resulting policy during 10 runs with longer episodes (150 timesteps). Figure 4-11b and table 4.5 show that multi-hop communications are still necessary to reduce the chances of getting stuck in a bad local optimum.



(a) (b)

Figure 4-11: (a) Smoothed, downsampled averaged reward over 10 trials of 15 modules running algorithms seeded with a partial policy, with and without pre-screening for legal actions and extra communicated observations, with standard error, (b) Average rewards obtained by 15 modules with episodes of length $T = 150$ after 100,000 episodes of distributed GAPS learning seeded with partially known policy, with various degrees of constraints and information, with standard deviation.

4.5 Discussion

The experimental results of section 4.3 are evidence that reinforcement learning can be used fruitfully in the domain of self-reconfigurable modular robots. Most of our results concern the improvement in SRMR usability through automated development of distributed controllers for such robots; and to that effect we have demonstrated that, provided with a good policy representation and enough constraints on the search space, gradient ascent algorithms converge to good policies given enough time and experience. We have also shown a number of ways to structure and constrain the learning space such that less time and experience is required and local optima become less likely. Imparting domain knowledge or partial policy knowledge requires more involvement from the human designer, and our desire to reduce the search space in this way is driven by the idea of finding a good balance between human designer skills and the automatic optimization. If the right balance is achieved, the humans can seed the learning algorithms with the kinds of insight that is easy for us; and the robot can then learn to improve on its own.

We have explored two ways of imparting knowledge to the learning system. On the one hand, we constrain exploration by effectively disallowing those actions that would result in a failure of motion (sections 4.1 and 4.3.1) or any action other than NOP in special cases (section 4.3.3). On the other hand, we initialize the algorithm at a better starting point by an incremental addition of modules (sections 4.2.1 and 4.3.5) or by a partially pre-specified policy (section 4.3.5). These experiments suggest that a good representation is very important for learning locomotion gaits in SRMRs. Local optima abound in the policy space when observations consists of the 8 bits of the immediate Moore neighborhood. Restricting exploration to legal motions only

	15 modules	**20 modules**	
	T=50	T=50	T=100
Distributed GAPS	10±0.0	10±0.0	
DGAPS with pre-screened legal actions			
1-hop comms	1.3±0.2	10±0.0	9.6±0.4
multi-hop comms	0.9±0.3	9.6±0.4	3.7±1.1
DGAPS with partially known policy			
no extra restrictions or info	1.2±0.4	10±0.0	9.9±0.1
pre-screened legal actions	3.8±0.6	8.8±0.6	9.5±0.4
+ 1-hop	0.8±0.2	9.9±0.1	7.6±0.7
+ multi-hop	0.3±0.3	4.7±1.0	1.3±0.3

Table 4.5: In each of the 10 learning trials, the learning was stopped after 100,000 episodes and the resulting policy was tested 10 times. The table shows the mean number of times, with standard error, during these 10 test runs for each policy, that modules became stuck in some configuration due to a locally optimal policy.

did not prevent this problem. It seems that more information than what is available locally is needed for learning good locomotion gaits.

Therefore, as a means to mitigate the prevalence of local optima, we have introduced a very limited communications protocol between neighboring modules. This increased the observation space and potentially the number of parameters to estimate. However, more search constraints in the form of restricting any motion if lack of support is communicated to the module, allowed modules to avoid the pitfalls of local optima substantially more often.

We have also found that local optima were more problematic the more modules were acting and learning at once; they were also preferentially found by robots which started from taller initial configurations (higher aspect ratio). The basic formulation of GAPS with no extra restrictions only moved into a configuration from which the modules would not move once in 100 test trials when there were 15 modules learning. When there were 20 modules learning, the number increased to almost half of all test trials. It is important to use resources and build supporting infrastructure into the learning algorithm in a way commensurate with the scale of the problem; more scaffolding is needed for harder problems involving a larger number of modules.

Restricting the search space and seeding the algorithms with good starting points is even more important when modules learn in a completely distributed fashion from their experience and their local rewards alone (section 4.4). We have shown that in the distributed case introducing constraints, communicated information, and partially known policies all contribute to successful learning of a locomotion gait, where the basic distributed GAPS algorithm has not found a good policy in 100,000 episodes. This is not very surprising, considering the amount of exploration each individual module needs to do in the unrestricted basic GAPS case. However, given enough constraints and enough experience, we have shown that it is possible to use the same

RL algorithms on individual modules. It is clear that more work is necessary before GAPS is ported to physical robots: for instance, we cannot afford to run a physical system for 100,000 episodes of 50 actions per module each.

In the next chapter, we extend our work using the idea of coordination and communication between modules that goes beyond single-bit exchanges.

Chapter 5

Agreement in Distributed Reinforcement Learning

In a cooperative distributed reinforcement learning situation, such as in a self-reconfiguring modular robot learning to move, an individual learning agent only has access to what it perceives locally, including a local measure of reward. This observation goes against a common assumption in the RL literature (e.g., Peshkin et al. 2000, Chang et al. 2004) that the reward is a global scalar available unaltered to all agents[1]. Consequently, where most research is concerned with how to extract a measure of a single agent's performance from the global reward signal[2], this is not a problem for locally sensing SRMR modules. A different problem, however, presents itself: the local reward is not necessarily a good measure of the agent's performance. For instance, it is possible that the agent's policy is optimal, but the bad choices made by others prevent it from ever finding itself in a configuration from which it can move.

Especially in the early stages of learning, the agents' policies will be near-random to ensure proper exploration. However, most modules will be stuck in the middle of the robot and will not have a chance to explore any actions other than NOP until the modules on the perimeter of the robot have learned to move out of the way.

The twin problems of limited local experience and locally observed but not necessarily telling reward signals are addressed in this chapter through the application of agreement algorithms (Bertsekas & Tsitsiklis 1997). We demonstrate below that agreement can be used to exchange both local reward and local experience among modules, and will result in a fully distributed GAPS implementation which learns good policies just as fast as the centralized, factored implementation.

[1]This assumption is not made in research on general-payoff fully observable stochastic Markov games (e.g., Hu & Wellman 1998).

[2]David Wolpert famously compares identical-payoff multi-agent reinforcement learning to trying to decide what to do based on the value of the Gross Domestic Product (e.g., Wolpert et al. 1999).

5.1 Agreement Algorithms

Agreement (also known as consensus) algorithms are pervasive in distributed systems research. First introduced by Tsitsiklis et al. (1986), they have been employed in many fields including control theory, sensor networks, biological modeling, and economics. They are applicable in synchronous and partially asynchronous settings, whenever a group of independently operating agents can iteratively update variables in their storage with a view to converge to a common value for each variable. Flocking is a common example of an agreement algorithm applied to a mobile system of robots or animals: each agent maintains a velocity vector and keeps averaging it with its neighbors' velocities, until eventually, all agents move together in the same direction.

5.1.1 Basic Agreement Algorithm

First, we quickly review the basic agreement algorithm, simplifying slightly from the textbook (chapter 7 of Bertsekas & Tsitsiklis 1997). Suppose there are N modules and therefore N processors, where each processor i is updating a variable x_i. For ease of exposition we assume that x_i is a scalar, but the results will hold for vectors as well. Suppose further that each i sends to a subset of others its current value of x_i at certain times. Each module updates its value according to a convex combination of its own and other modules' values:

$$x_i(t+1) = \sum_{j=1}^{N} a_{ij} x_j(\tau_j^i(t)), \qquad \text{if } t \in T^i,$$
$$x_i(t+1) = x_i(t), \qquad \text{otherwise,}$$

where a_{ij} are nonnegative coefficients which sum to 1, T^i is the set of times at which processor i updates its value, and $\tau_j^i(t)$ determines the amount of time by which the value x_j is outdated. If the graph representing the communication network among the processors is connected, and if there is a finite upper bound on communication delays between processors, then the values x_i will exponentially converge to a common intermediate value x such that $x_{min}(0) \leq x \leq x_{max}(0)$ (part of Proposition 3.1 in Bertsekas & Tsitsiklis (1997)). Exactly what value x the processors will agree on in the limit will depend on the particular situation.

5.1.2 Agreeing on an Exact Average of Initial Values

It is possible to make sure the modules agree on the exact average of their initial values. This special case is accomplished by making a certain kind of updates synchronous and pairwise disjoint updates.

Synchronicity means that updates are made at once across the board, such that all communicated values are out of date by the same amount of time. In addition, modules communicate and update their values in disjoint pairs of neighbors, equally and symmetrically:

$$x_i(t+1) = x_j(t+1) = \frac{1}{2}\left(x_i(t) + x_j(t)\right).$$

In this version of agreement, the sum $x_1 + x_2 + ... + x_N$ is conserved at every time step, and therefore all x_i will converge to the exact arithmetic mean of the initial values $x_i(0)$. Actually, equal symmetrical averaging between all members of each fully connected clique of the graph representing neighboring processor communications will also be sum-invariant, such that all nodes agree on the exact average of initial values. This fact will be relevant to us later in this chapter, where we use agreement together with policy search in distributed learning of locomotion gaits for SRMRs.

In partially asynchronous situations, or if we would like to speed up agreement by using larger or overlapping sets of neighbors, we cannot in general say what value the processors will agree upon. In practice, pairwise disjoint updates lead to prohibitively long agreement times, while using all the nearest neighbors in the exchange works well.

5.1.3 Distributed Optimization with Agreement Algorithms

Agreement algorithms can also be used for gradient-style updates performed by several processors simultaneously in a distributed way (Tsitsiklis et al. 1986), as follows:

$$x_i(t+1) = \sum_{j=1}^{N} a_{ij} x_j(\tau_j^i(t)) + \gamma s_i(t),$$

where γ is a small decreasing positive step size, and $s_i(t)$ is in the possibly noisy direction of the gradient of some continuously differentiable nonnegative cost function. It is assumed that noise is uncorrelated in time and independent for different i's.

However, GAPS is a *stochastic* gradient ascent algorithm, which means that the updates performed climb the estimated gradient of the policy value function $\nabla\hat{V}$, which may, depending on the quality of the agents' experience, be very different from ∇V, as shown in figure 5-1. A requirement of the partially asynchronous gradient-like update algorithm described in this section is that the update direction $s_i(t)$ be in the same quadrant as ∇V, but even this weak assumption is not necessarily satisfied by stochastic gradient ascent methods. In addition, the "noise", described by the $w_i(t)$ term above, is correlated in a SRMR, as modules are physically attached to each other. In the rest of this section, we nevertheless propose a class of methods for using the agreement algorithm with GAPS, and demonstrate their suitability for the problem of learning locomotion by self-reconfiguration.

5.2 Agreeing on Common Rewards and Experience

When individual modules learn in a completely distributed fashion, often most modules will make zero updates. However, if modules share their local rewards, all contributing to a common result, the agreed-upon R will only be equal to zero if no modules have moved at all during the current episode, and otherwise all modules will make some learning update. We expect that fact to speed up the learning process.

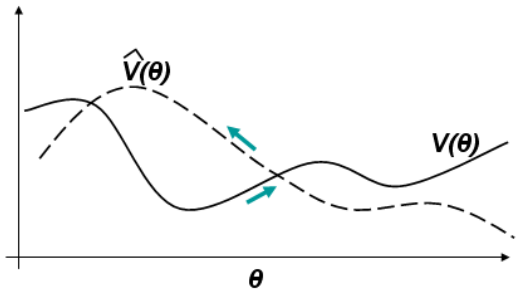


Figure 5-1: Stochastic gradient ascent updates are not necessarily in the direction of the true gradient.

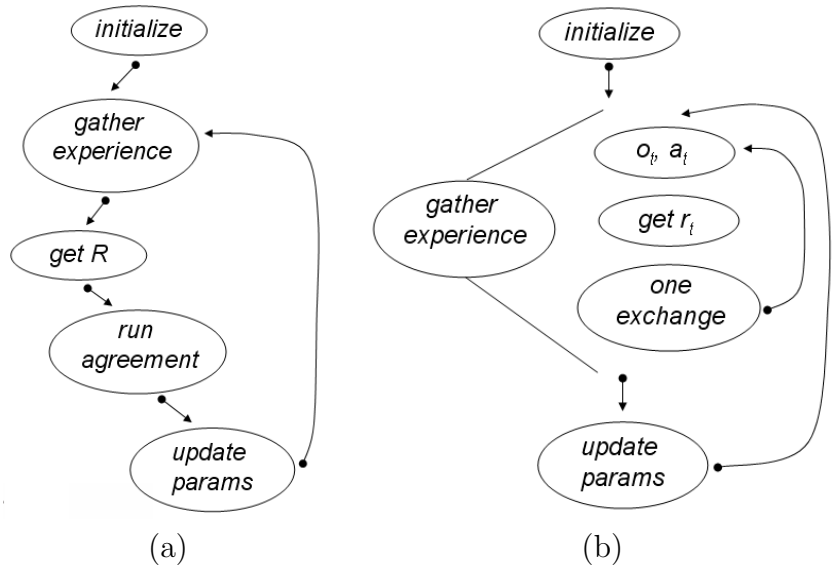


Figure 5-2: Two implementations of agreement exchanges within the GAPS learning algorithm: (a) at the end of each episode, and (b) at each time step.

In addition, we expect the agreed-upon R to be a more accurate measure of how well the whole robot is doing, thereby also increasing the likelihood of better updates by all modules. This measure will be exactly right if the modules use synchronous equal symmetrical pairwise disjoint updates, although this is harder to achieve and slower in practice.

A theorem in Peshkin (2001) states that a distributed version of GAPS, where each agent maintains and updates its own policy parameters, will make exactly the same updates as the centralized factored version of GAPS, provided the same experience and the same rewards. As we have seen, this proviso does not hold in general for the task of locomotion by self-reconfiguration. However, neighboring modules can exchange local values of reward to agree on a common value. What happens if modules also exchange local experience?

In the centralized GAPS algorithm, the parameter updates are computed with

Algorithm 3 Asynchronous ID-based collection of experience. This algorithm can be run at the end of each episode before the learning update rule.

Require: unique id
  send $message = \langle id, R, C_o, C_{oa} \rangle$ to all neighbors
  loop
    if timeout **then**
      proceed to **update**
    end if
    whenever message $m = \langle id^m, R^m, C_o^m, C_{oa}^m \rangle$ arrives
    if id^m not seen before **then**
      store reward $R_s(id^m) \leftarrow R^m$
      add experience $C_o += C_o^m$, $C_{oa} += C_{oa}^m$
      resend m to all neighbors
    end if
  end loop
  update:
  average rewards $R \leftarrow mean(R_s)$
  GAPS update rule

the following rule:

$$\Delta\theta_{oa} = \alpha R \left(C_{oa} - C_o \pi_\theta(o, a) \right),$$

where the counters of experienced observations C_o and chosen actions per observation C_{oa} represent the sum of the modules' individual counters. If there is a way for the modules to obtain these sums through local exchanges, then the assumptions of the theorem are satisfied and we can expect individual modules to make the same updates as the global learner in the centralized case, and therefore to converge to a local optimum.

There are at least two possible ways of obtaining these sums. The first algorithm assumes a unique ID for every module in the system—a reasonable assumption if we consider physically instantiated robotic modules. At the end of each episode, each module transmits a message to its face neighbors, which consists of its ID and the values of all nonzero counters. Upon receiving such a message in turn, the module checks that the ID has not been seen before. If the ID is new, the receiving module will add in the values from the message to its current sums. If not, the message is ignored. The only problem with this ID-based algorithm, shown also as Algorithm 3, is termination. We cannot assume that modules know how many of them comprise the system, and therefore must guess a timeout value beyond which a module will no longer wait for messages with new IDs and will proceed to update its parameters.

An alternative method is again based on the agreement algorithm. Recall that using synchronous, equal and symmetric pairwise disjoint updates guarantees convergence to a common x which is the exact average of initial values of x_i. If at the end of an episode, each module runs such an agreement algorithm on each of the counters that it maintains, then each counter C^i will converge to $\overline{C} = \frac{1}{N} \sum_{i=1}^{N} C^i$. If

Algorithm 4 Asynchronous average-based collection of experience. This algorithm may be run at the end of each episode, or the procedure labeled **average** may be used at each time step while gathering experience, as shown also in figure 5-2b.

send message $message = \langle R, C_o, C_{oa} \rangle$ to a random neighbor
repeat
  if timeout **then**
    proceed to **update**
  end if
  whenever $m = \langle R^m, C_o^m, C_{oa}^m \rangle$ arrives
  average:
  reward $R \leftarrow \frac{1}{2}(R + R^m)$
  experience $C_o \leftarrow \frac{1}{2}(C_o + C_o^m)$, $C_{oa} \leftarrow \frac{1}{2}(C_{oa} + C_{oa}^m)$
  send to a random neighbor new message $m' = \langle R, C_o, C_{oa} \rangle$
until converged
update:
GAPS update rule

this quantity is then substituted into the GAPS update rule, the updates become:

$$\begin{aligned}\Delta\theta_{oa} &= \alpha R \left(\frac{1}{N} \sum_{i=1}^{N} C_{oa}^i - \pi_\theta(o,a) \frac{1}{N} \sum_{i=1}^{N} C_o^i \right) \\ &= \frac{1}{N} \alpha R \left(C_{oa} - C_o \pi_\theta(o,a) \right),\end{aligned}$$

which is equal to the centralized GAPS updates scaled by a constant $\frac{1}{N}$. Note that synchronicity is also required at the moment of the updates, i.e., all modules must make updates simultaneously, in order to preserve the stationarity of the underlying process. Therefore, modules learning with distributed GAPS using an agreement algorithm with synchronous pairwise disjoint updates to come to a consensus on both the value of the reward, and the average of experience counters, will make stochastic gradient ascent updates, and therefore will converge to a local optimum in policy space.

The algorithm (Algorithm 4) requires the scaling of the learning rate by a constant proportional to the number of modules. Since stochastic gradient ascent is in general sensitive to the learning rate, this pitfall cannot be avoided. In practice, however, scaling the learning rate up by an approximate factor loosely dependent (to the order of magnitude) on the number of modules, works just as well.

Another practical issue is the communications bandwidth required for the implementation of agreement on experience. While many state of the art microcontrollers have large capacity for speedy communications (e.g., the new Atmel AT32 architecture includes built-in full speed 12 Mbps USB), depending on the policy representation, significant downsizing of required communications can be achieved. The most obvious compression technique is to not transmit any zeros. If exchanges happen at the end of the episode, instead of sharing the full tabular representation of the counts, we notice

Algorithm 5 Synchronous average-based collection of experience. This algorithm may be run at the end of each episode, or the procedure labeled **average** may be used at each time step while gathering experience, as shown also in figure 5-2b.

send to all n neighbors $message = \langle R, C_o, C_{oa} \rangle$

for all time steps **do**

average:

receive from all n neighbors message $m = \langle R^m, C_o^m, C_{oa}^m \rangle$

average reward $R \leftarrow \frac{1}{n+1}\left(R + \sum_{m=1}^{n} R^m\right)$

average experience $C_o \leftarrow \frac{1}{n+1}\left(C_o + \sum_{m=1}^{n} C_o^m\right)$, $C_{oa} \leftarrow \frac{1}{n+1}\left(C_{oa} + \sum_{m=1}^{n} C_{oa}^m\right)$

send to all n neighbors $m' = \langle R, C_o, C_{oa} \rangle$

end for

update:

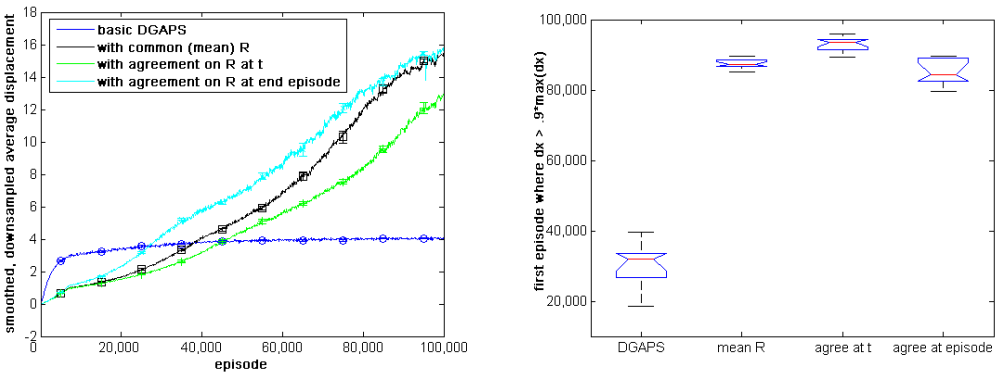
GAPS update rule

that often experience histories $h =< o^1, a^1, o^2, a^2, ..., o^T, a^T >$ are shorter — in fact, as long as $2T$, where T is the length of an episode. In addition, these can be sorted by observation with the actions combined: $h_{sorted} =< o_1 : a_1 a_2 a_1 a_4 a_4, o_2 : a_2, o_3 : a_3 a_3, ... >$, and finally, if the number of repetitions in each string of actions warrants it, each observation's substring of actions can be re-sorted according to action, and a lossless compression scheme can be used, such as the run-length encoding, giving $h_{rle} =< o_1 : 2a_1 a_2 2a_4, o_2 : a_2, o_3 : 2a_3, ... >$. Note that this issue unveils the trade-off in experience gathering between longer episodes and sharing experience through neighbor communication.

How can a module incorporate this message into computing the averages of shared experience? Upon receiving a string h_{rle}^m coming from module m, align own and m's contributions by observation, creating a new entry for any o_i that was not found in own h_{rle}. Then for each o_i, combine action substrings, re-sort according to action and re-compress the action substrings such that there is only one number n_j preceding any action a_j. At this point, all of these numbers n are halved, and now represent the pairwise average of the current module's and m's combined experience gathered during this episode. Observation counts (averages) are trivially the sum of action counts (averages) written in the observation's substring.

5.3 Experimental Results

We ran the synchronized algorithm described as Algorithm 5 on the case study task of locomotion by self-reconfiguration in 2D. The purpose of the experiments was to discover how agreement on common rewards and/or experience affects the speed and reliability of learning with distributed GAPS. As in earlier distributed experiments in chapter 4, the learning rate α here was initially set at the maximum value of 0.01 and steadily decreased over the first 7,000 episodes to 0.001, and remained at that value thereafter. The temperature parameter β was initially set to a minimum value of 1 and increased steadily over the first 7,000 episodes to 3, and remained at that value



(a) (b)

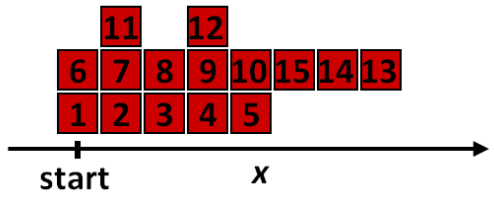
Figure 5-3: Smoothed (100-point window), downsampled average center of mass displacement, with standard error, over 10 runs with 15 modules learning to locomote eastward, as compared to the baseline distributed GAPS and standard GAPS with given global mean reward. (a) The modules run the agreement algorithm on rewards to convergence at the end of episodes. (b) Distribution of first occurrences of center of mass displacement greater than 90% of maximal displacement for each group: baseline DGAPS, DGAPS with given mean as common reward, and DGAPS with agreement in two conditions – two exchanges per timestep and to convergence at the end of episode.

thereafter. This encouraged more exploration during the initial episodes of learning.

Figure 5-3a demonstrates the results of experiments with the learning modules agreeing exclusively on the rewards generated during each episode. The two conditions, as shown in figure 5-2, diverged on the time at which exchanges of reward took place. The practical difference between the end-of-episode versus the during-the-episode conditions is that in the former, averaging steps can be taken until convergence[3] on a stable value of R, whereas in the latter, all averaging stops with the end of the episode, which will likely arrive before actual agreement occurs.

Figure 5-3a shows that despite this early stopping point, not-quite-agreeing on a common reward is significantly better than using each module's individual estimate. As predicted, actually converging on a common value for the reward results, on average, in very good policies after 100,000 episodes of learning in a fully distributed manner. By comparison, as we have previously seen, distributed GAPS with individual rewards only does not progress much beyond an essentially random policy. In order to measure convergence times (that is, the speed of learning) for the different algorithms, we use a single-number measure obtained as follows: for each learning run, average robot center of mass displacement dx was smoothed with 100-point moving window and downsampled to filter out variability due to within-run exploration and random variation. We then calculate a measure of speed of learning (akin to convergence time) for each condition by comparing the average first occurrence of dx

[3]As recorded by each individual module separately.



reward mean: 2.4
agreement gives: 0.63

(a)

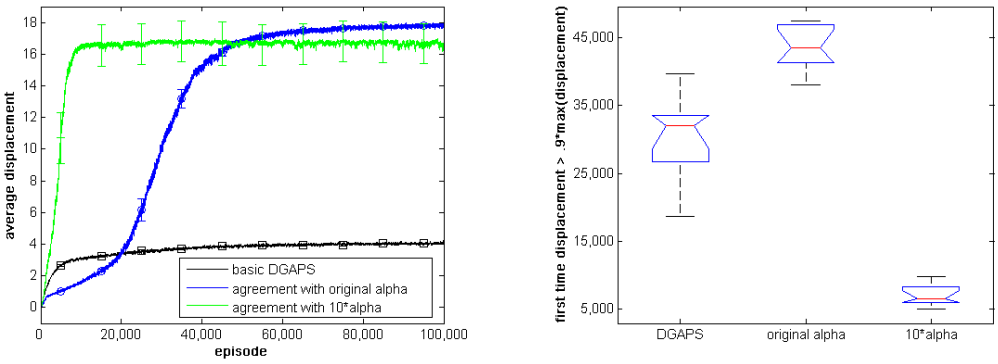
	no. stuck
DGAPS	10±0.0
DGAPS with common R	0.4±0.2
agreement at each t	0.5±0.2
agreement at end episode	0.2±0.1

(b)

Figure 5-4: Effect of active agreement on reward during learning: (a) The modules with greatest rewards in such configurations have least neighbors (module 15 has 1 neighbor and a reward of 5) and influence the agreed-upon value the least. The modules with most neighbors (modules 1-10) do not have a chance to get any reward, and influence the agreed-upon value the most. (b) Mean number of stuck configurations, with standard error, out of 10 test runs each of 10 learned policies, after 15 modules learned to locomote eastward for 100,000 episodes.

greater than 90% of its maximum value for each condition, as shown in figure 5-3b. The Kruskal-Wallis test with four groups (baseline original DGAPS with individual rewards, DGAPS with common reward $R = \frac{1}{N}\sum_{i=1}^{N} R_i$ available to individual modules, agreement at each timestep, and agreement at the end of each episode) reveals statistically significant differences in convergence times ($\chi^2(3, 36) = 32.56, p < 10^{-6}$) at the 99% confidence level. Post-hoc analysis shows no significant difference between DGAPS with common rewards and agreement run at the end of each episode. Agreement (two exchanges at every timestep) run during locomotion results in a significantly later learning convergence measure for that condition – this is not surprising, as exchanges are cut off at the end of episode when the modules still have not necessarily reached agreement.

While after 100,000 episodes of learning, the policies learned with agreement at the end of each episode and those learned with given common mean R receive the same amount of reward, according to figure 5-3a, agreement seems to help more earlier in the learning process. This could be a benefit of averaging among all immediate neighbors, as opposed to pairwise exchanges. Consider 15 modules which started forming a protrusion early on during the learning process, such that at the end of some episode, they are in the configuration depicted in figure 5-4a. The average robot displacement here is $R_{mean} = 2.4$. However, if instead of being given that value, modules actively run the synchronous agreement at the end of this episode, they will eventually arrive at the value of $R_{agree} = 0.63$. This discrepancy is due to the fact that most of the modules, and notably those that have on average more neighbors, have received zero individual rewards. Those that have, because they started forming an arm, have less neighbors and therefore less influence on R_{agree}, which results in smaller updates for their policy parameters, and ultimately with less learning "pull"



(a) (b)

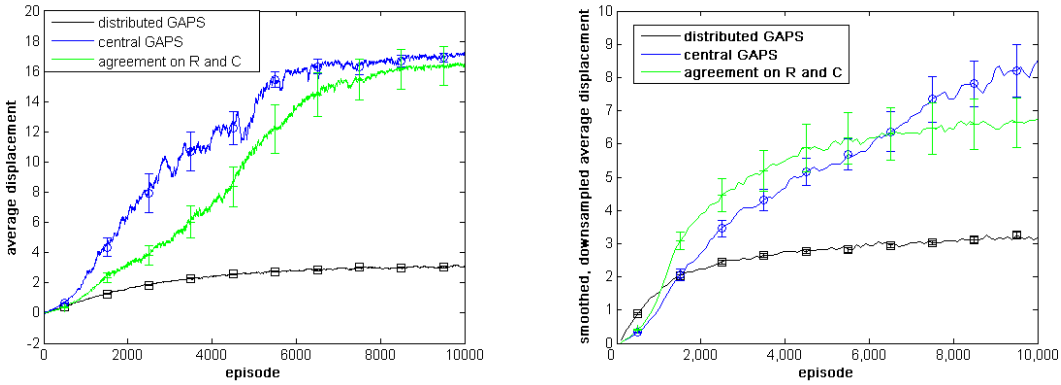
Figure 5-5: Smoothed (100-point window), downsampled average center of mass displacement, with standard error, over 10 runs with 15 modules learning to locomote eastward, as compared to the baseline distributed GAPS. (a) The modules run the agreement algorithm on rewards and experience counters at the end of episodes with original vs. scaled learning rates. (b) Distribution of first occurrences of center of mass displacement greater than 90% of maximal displacement for each group.

exerted by the arm.

Our hypothesis was therefore that, during test runs of policies learned by DGAPS with common reward versus DGAPS with agreement algorithms, we will see significantly more dysfunctional stuck configurations in the first condition. The table in figure 5-4b details the number of times, out of 10 test trials, that policies learned by different algorithms were stuck in such configurations. Our findings do not support our hypothesis: both types of agreement, as well as the baseline algorithm with common reward, generate similar numbers of non-gait policies.

For the set of experiments involving agreement on experience as well as rewards, the learning rate α was scaled by a factor of 10 to approximately account for the averaging of observation and observation-action pair counts[4]. Figure 5-5a demonstrates that including exchanges of experience counts results in dramatic improvement in how early good policies are found. There is also a significant difference in learning speed between learning with the original learning rate or scaling it to account for averaging of experience (Kruskal-Wallis test on the first occurrence of center of mass displacement that is greater than 90% of maximal displacement for each group reports 99% significance: $\chi^2(2, 27) = 25.31, p < .10^{-5}$). Finally, we see in figure 5-6a that distributed GAPS with agreement on both reward and experience learns comparatively just as fast and just as well as the centralized version of GAPS for 15 modules. The three-way Kruskal-Wallis test reports 99% significance level ($\chi^2(2, 27) = 16.95, p = .0002$); however testing the centralized versus agreement groups only shows no significance. However, when we increase the robot size to 20 modules (and episode length to 100 timesteps) in figure 5-6b, all three conditions yield different reward curves, with

[4]This scaling does not in itself account for the difference in experimental results. Scaling up α in the baseline distributed GAPS algorithm does not help in any way.



(q) (b)

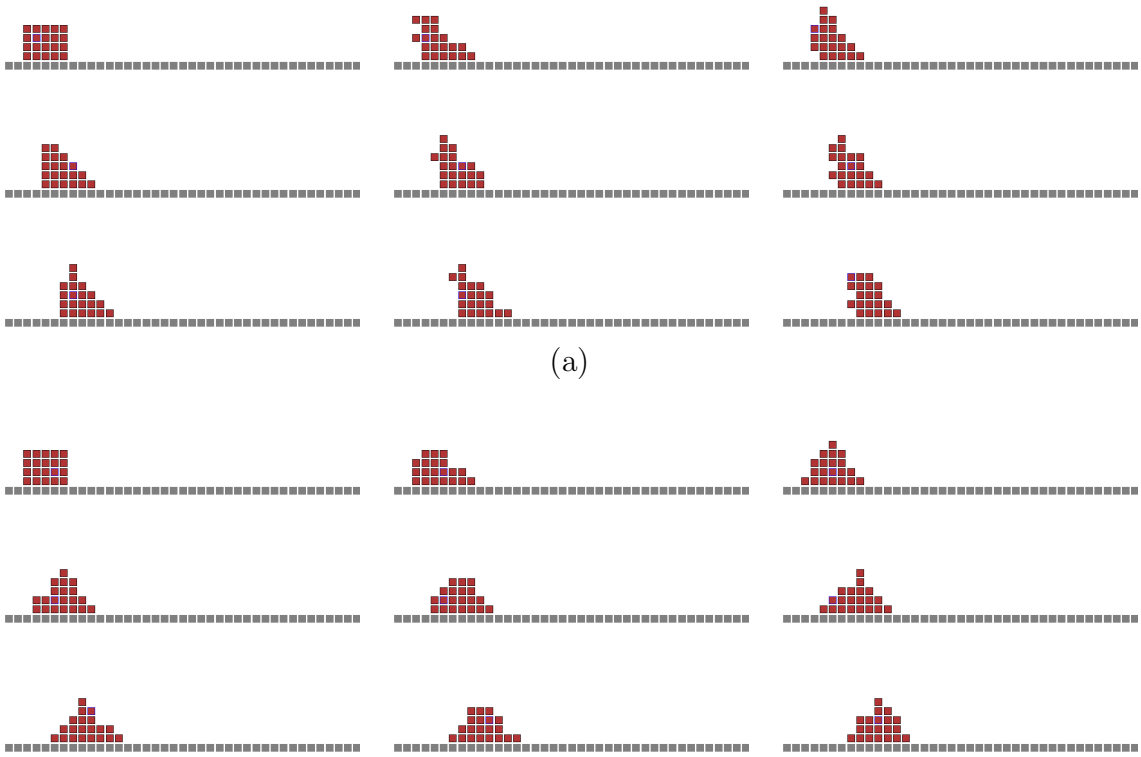
Figure 5-6: Comparison between centralized GAPS, distributed GAPS and distributed GAPS with agreement on experience: (a) smoothed (100-point window) average center of mass displacement over 10 trials with 15 modules learning to locomote eastward (T=50), (b) same for 20 modules with episode length T=100: learning was done starting with $\alpha = 0.2$ and decreasing it uniformly over the first 1,500 episodes to 0.02.

	15 mods, T=50	**20 mods, T=100**
distributed GAPS	10±0.0	10±0.0
centralized GAPS	0.1±0.1	5±1.7
agreement on R and C	1.1 ±1.0	6±1.6

Table 5.1: Mean number of non-gait local optima, with standard error, out of 10 test trials for 10 learned policies. The starting learning rate was $\alpha = 0.1$ for 15 modules and $\alpha = 0.2$ for 20 modules.

agreement-based distributed learning getting significantly less reward than centralized GAPS.

Where does this discrepancy come from? In table 5.1, we see that learning in a fully distributed way with agreement on reward and experience makes the modules slightly more likely to find non-gait local optima than the centralized version. However, a closer look at the policies generated by centralized learning vs. agreement reveals another difference: gaits found by agreement are more compact than those found by central GAPS. Figure 5-7 demonstrates in successive frames two such compact gaits (recall figure 3.5.6 for comparison). Out of all the acceptable gait policies found by GAPS with agreement, none of them unfolded into a two-layer thread-like gait, but all maintained a more blobby shape of the robot, preserving something of its initial configuration and aspect ratio. Such gaits necessarily result in lower rewards; however, this may be a desirable property of agreement-based learning algorithms.



(a)

(b)

Figure 5-7: Screenshots taken every 5 timesteps of 20 modules executing two compact gait policies found by learning with agreement on reward and experience.

5.4 Discussion

We have demonstrated that with agreement-style exchanges of rewards and experience among the learning modules, we can expect to see performance in distributed reinforcement learning to increase by a factor of ten, and approach that of a centralized system for smaller numbers of modules and lower aspect ratios, such as we have with 15 modules. This result suggests that the algorithms and extensions we presented in chapters 3 and 4 may be applicable to fully distributed systems. With higher numbers of modules (and higher aspect ratios of the initial configuration), there is a significant difference between rewards obtained by centralized learning and distributed learning with agreement, which is primarily due to the kinds of policies that are favored by the different learning algorithms.

While centralized GAPS maximizes reward by unfolding the robot's modules into a two-layer thread, distributed GAPS with agreement finds gaits that preserve the blobby shape of the initial robot configuration. This difference is due to the way modules calculate their rewards by repeated averaging with their neighbors. As we explained in section 5.3, modules will agree on a common reward value that is not the mean of their initial estimates, if the updates are not pairwise. This generates an effect where modules with many neighbors have more of an influence on the resulting agreed-upon reward value than those with fewer neighbors. It therefore makes sense that locomotion gaits found through this process tend to keep modules within closer proximity to each other. There is, after all, a difference between trying to figure out what to do based on the national GDP, and trying to figure out what to do based on how you and your neighbors are faring.

We have developed a class of algorithms incorporating stochastic gradient ascent in policy space (GAPS) and agreement algorithms, creating a framework for fully distributed implementations of this POMDP learning technique.

This work is very closely related to the distributed optimization algorithm described by Moallemi & Van Roy (2003) for networks, where the global objective function is the average of locally measurable signals. They proposed a distributed policy gradient algorithm where local rewards were pairwise averaged in an asynchronous but disjoint (in time and processor set) way. They prove the resulting local gradient estimates converge in the limit to the global gradient estimate. Crucially, they do not make any assumptions about the policies the processors are learning, beyond continuity and continuous differentiability. In contrast, here we are interested in a special case where ideally all modules' policies are identical. Therefore, we are able to take advantage of neighbors' experience as well as rewards.

Practical issues in sharing experience include the amount of required communications bandwidth and message complexity. We have proposed a simple way of limiting the size of each message by eliminating zeros, sorting experience by observation and action, and using run-length encoding as representation of counts (and ultimately, averages). We must remember, however, that in most cases, communication is much cheaper and easier than actuation, both in resource (e.g., power) consumption, and in terms of the learning process. Modules would benefit greatly from sharing experience with higher communication cost, and therefore learning from others' mistakes, rather

than continuously executing suboptimal actions that in the physical world may well result in danger to the robot or to something or someone in the environment.

If further reduction in communications complexity is required, in recent work Moallemi & Van Roy (2006) proposed another distributed protocol called consensus propagation, where neighbors pass two messages: their current estimate of the mean, and their current estimate of a quantity related to the cardinality of the mean estimate, i.e. a measure of how many other processors so far have contributed to the construction of this estimate of the mean. The protocol converges extremely fast to the exact average of initial values in the case of singly-connected communication graphs (trees). It may benefit GAPS-style distributed learning to first construct a spanning tree of the robot, then run consensus propagation on experience. However, we do not want to give up the benefits of simple neighborhood averaging of rewards in what concerns avoidance of undesirable local optima.

We may also wish in the future to develop specific caching and communication protocols for passing just the required amount of information, and therefore requiring less bandwidth, given the policy representation and current experience.

Chapter 6

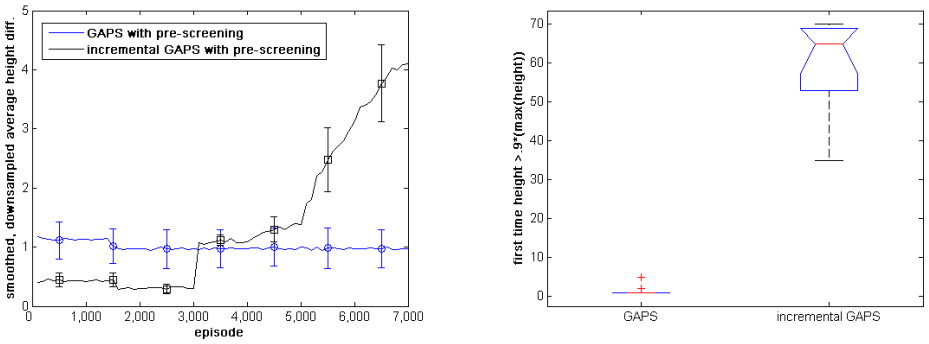
Synthesis and Evaluation: Learning Other Behaviors

So far we had focused on a systematic study of learning policies for a particular task: simple two-dimensional locomotion. However, the approach generalizes to other objective functions.

In this chapter, we combine the lessons learned so far about, on the one hand, the importance of compact policy representation that 1) still includes our ideal policy as an optimum, but 2) creates a relatively smooth policy value landscape, and, on the other hand, that of good starting points, obtained through incremental learning or partially known policies. Here, we take these ideas further in order to evaluate our approach. Evaluation is based on different tasks requiring different objective functions, namely:

- tall structure (tower) building
- locomotion over obstacle fields
- reaching to an arbitrary goal position in robot workspace

Reducing observation and parameter space may lead to good policies disappearing from the set of representable behaviors. In general, it will be impossible to achieve some tasks using limited local observation. For example, it is impossible to learn a policy that reaches to a goal position in a random direction from the center of the robot, using only local neighborhoods, as there is no way locally of either observing the goal or otherwise differentiating between desired directions of motion. Therefore, we introduce a simple model of minimal sensing at each module and describe two ways of using the sensory information: by incorporating it into a larger observation space for GAPS, or by using it in a framework of geometric policy transformations, for a kind of transfer of learned behavior between different spatially defined tasks.



(a) (b)

Figure 6-1: 25 modules learning to build tall towers (reaching upward) with basic GAPS versus incremental GAPS (in both cases with pre-screened legal actions): (a) Smoothed (100-point window), downsampled average height difference from initial configuration, with standard error (b) Distribution of first occurrence of 90% of maximal height achieved during learning, for each group separately.

6.1 Learning different tasks with GAPS

We start by validating policy search with the standard representation of chapters 3 through 5, on two tasks that are more complex than simple locomotion: building tall towers, and locomotion over fields of obstacles.

6.1.1 Building towers

In this set of experiments, the objective was to build a configuration that is as tall as possible. The rewards were accumulated during each episode as follows: $r^i_{t+1} = 5 \times (y^i_{t+1} - y^i_t)$, with the global reward calculated as $R = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r^i_t$. This objective function is maximized when globally modules have created the most height difference since the initial configuration. In addition, it can be collected in a distributed way.

In order to penalize unstable formations, in these experiments we used the same additional stability constraints as earlier in section 3.5.7, namely, if the (global) center of mass of the robot is moved outside its x-axis footprint, the modules fall, and the global reward becomes $R = -10$. Individual modules may also fall with very low probability which is proportional to the module's y position relative to the rest of the robot and inversely proportional to the number of its local neighbors. This also results in a reward of -10, but this time for the individual module only.

Ten learning trials with 25 modules each were run in two different conditions: either the modules used the basic GAPS algorithm or incremental GAPS with increments taken in the length of the original square configuration: from a length of 2 (4 modules) to the final length of 5 (25 modules). In both cases, modules pre-screened for legal actions only. These experiments validate the incremental approach on a

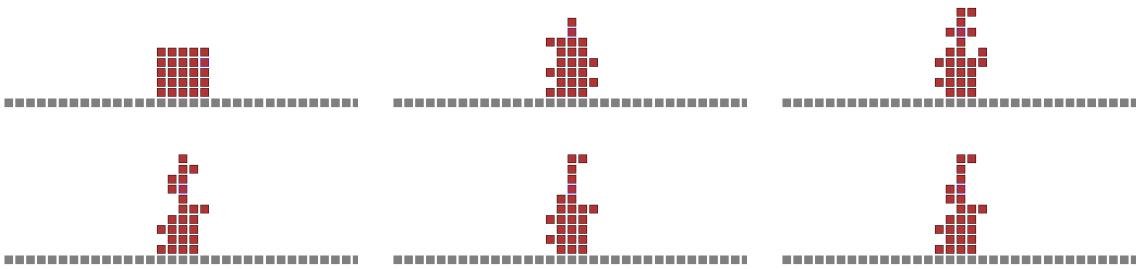


Figure 6-2: A sequence of 25 modules, initially in a square configuration, executing a learned policy that builds a tall tower (frames taken every 5 steps).

non-locomotion task. We can see in figure 6-1a that on average, modules learning with basic GAPS do not increase the height of the robot over time, whereas modules taking the incremental learning approach create an average height difference after 7,000 episodes.

Figure 6-1c shows the distribution of the first occurrence of a height difference greater than 90% of the maximal height difference achieved in each group. We take this as a crude measure of converge speed. By this measure basic GAPS converges right away, while incremental GAPS needs on average over 6,000 episodes (a Kruskal-Wallis non-parametric ANOVA shows a difference at 99% significance level: $\chi^2(1, 18) = 15.26, p < .0001$). However, by looking also at figure 6-1a, we notice that the maximal values are very different. Incremental GAPS actually learns a tower-building policy, whereas basic GAPS does not learn anything.

Figure 6-2 demonstrates an execution sequence of a policy learned by 25 modules through the incremental process, using pre-screening for legal actions. Not all policies produce tower-like structures. Often the robot reaches up from both sides at once, standing tall but with double vertical protrusions. This is due to two facts. The first is that we do not specifically reward tower-building, but simple height difference. Therefore, policies can appear successful to modules when they achieve any positive reward. This problem is identical to the horizontal protrusions appearing as local maxima to modules learning to locomote in chapter 4. However, the other problem is that the eight immediately neighboring cells do not constitute a sufficient representation for disambiguating between the robot's sides in a task that is essentially symmetric with respect to the vertical. Therefore modules can either learn to only go up, which produces double towers, or they can learn to occasionally come down on the other side, which produces something like a locomotion gait instead. A larger observed neighborhood could be sufficient to disambiguate in this case, which again highlights the trade-off between being able to represent the desired policy and compactness of representation.

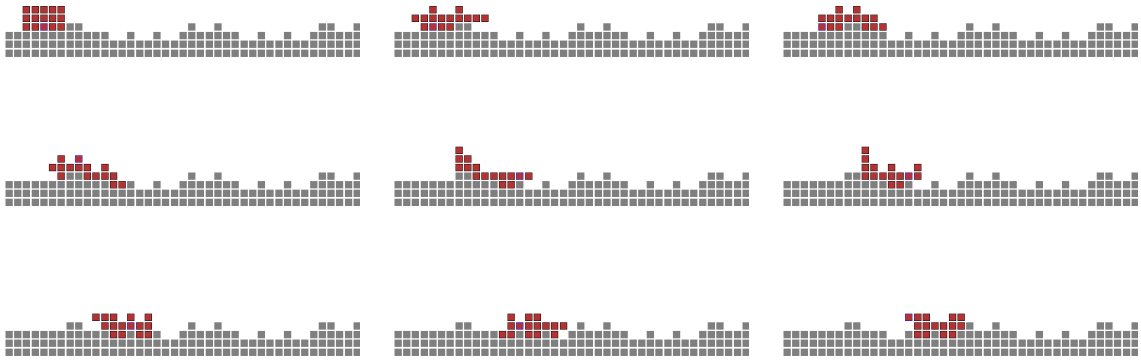


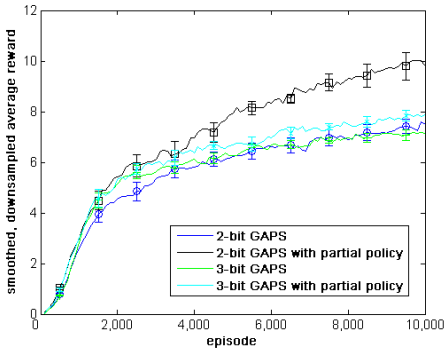
Figure 6-3: First 80 time-steps of executing a policy for locomotion over obstacles, captured in frames every 10 steps (15 modules). The policy was learned with standard-representation GAPS seeded with a partial policy.

6.1.2 Locomotion over obstacles

Standard GAPS, which defines observations straightforwardly over the 8 neighboring cells, has 256 possible observations when there are no obstacles to be observed. However, when we introduce obstacles, there are now 3 possible states of each local cell, and therefore $3^8 = 6,561$ possible observations, or $3^8 \times 9 = 59,049$ parameters to learn. We expect basic GAPS to do poorly on a task with such a large parameter space to search. We could naturally pretend that observations should not be affected by obstacles — in other words, each module can only sense whether or not there is another module in a neighboring cell, and conflates obstacles and empty spaces.

We perform a set of experiments with the larger and smaller representation GAPS, trying to learn a policy for "rough terrain" locomotion over obstacles. Figure 6-3 shows a sequence of a test run where 15 modules execute a policy for locomotion over obstacles learned with standard-representation GAPS. Due to the limited local observability and conservative rules to prevent disconnections, it is essential that the robot learns to conform its shape to the shape of the terrain as show in the figure. Otherwise, in a local configuration where the acting module m_i is standing on the ground (or an obstacle) but does not observe any neighbors also connected to the ground, it will not move, unless another module can move into a position to unblock m_i.

Experiments were performed in two conditions: with or without seeding the search with a partially known policy. When specified, the partially known policy was in all cases equivalent to the one in earlier experiments (section 4.3.5): we specify that when there are neighbors on its right, the module should go up, and when there are neighbors on its left, the module should go down. Figure 6-4a shows the smoothed (100-point moving window), downsampled average rewards obtained by GAPS with a larger (3^8 observations) and smaller (2^8 observations) representations, with and without partial policy.



(a)

	2-bit	**3-bit**
basic GAPS	10±0.0	10±0.0
partial policy	4.0±1.0	5.1±0.6

(b)

Figure 6-4: Locomotion over obstacles with or without initializing GAPS to a partially known policy, for both 2-bit and 3-bit observations, learned by 15 modules. (a) Smoothed (100-point window), downsampled average rewards, with standard error. (b) Mean number of non-gait stuck configurations, with standard error, out of 10 test runs for each of 10 learned policies.

The table shown in figure 6-4b represents the average, for a sample of 10 learned policies, number of times modules were stuck in a dysfunctional configuration due to a local optimum that does not correspond to a locomotion gait. Locomotion over obstacles is a very hard task for learning, and the results reflect that, with even 15 modules not being able to learn any good policies without us biasing the search with a partially known policy. In addition, we observe that increasing the number of parameters to learn by using a more general 3-bit representation leads to a significant drop in average rewards obtained by policies after 10,000 episodes of learning: average reward curves for biased and unbiased GAPS with 3-bit observation are practically indistinguishable. While the mean number of arm-like locally optimal configurations reported in the table for the biased condition and 3-bit observation is significantly lower than for unbiased GAPS, this result fails to take into account that the "good" policies in that case still perform poorly. The more blind policies represented with only 2 bits per cell of observations fared much better due to the drastically smaller dimensionality of the policy parameter space in which GAPS has to search. Larger, more general observation spaces require longer learning.

6.2 Introducing sensory information

In previous chapters, we had also assumed that there is no other sensory information available to the modules apart from the existence of neighbors in the immediate Moore neighborhood (eight adjacent cells). As we had pointed out before, this limits the class of representable policies and excludes some useful behaviors. Here, we introduce a minimal sensing model.

6.2.1 Minimal sensing: spatial gradients

We assume that each module is capable of making measurements of some underlying spatially distributed function f such that it knows, at any time, in which direction lies the largest gradient of f. We will assume the sensing resolution to be either one in 4 (N, E, S or W) or one in 8 (N, NE, E, SE, S, SW, W or NW) possible directions. This sensory information adds a new dimension to the observation space of the module. There are now 2^8 possible neighborhood observations $\times 5$ possible gradient observations[1] $\times 9$ possible actions $= 11,520$ parameters, in the case of the lower resolution, and respectively $2^8 \times 9 \times 9 = 20,736$ parameters in the case of the higher resolution. This is a substantial increase. The obvious advantage is to broaden the representation to apply to a wider range of problems and tasks. However, if we severely downsize the observation space and thus limit the number of parameters to learn, it should speed up the learning process significantly.

6.2.2 A compact representation: minimal learning program

Throughout this thesis we have established the dimensions and parameters which influence the speed and reliability of reinforcement learning by policy search in the domain of lattice-based self-reconfigurable modular robots. In particular, we have found that the greatest influence is exerted by the number of parameters which represent the class of policies to be searched. The minimal learning approach is to reduce the number of policy parameters to the minimum necessary to represent a reasonable policy.

To achieve this goal, we incorporate a pre-processing step and a post-processing step into the learning algorithm. Each module still perceives its immediate Moore neighborhood. However, it now computes a discrete local measure of the weighted center of mass of its neighborhood, as depicted in figure 6-5. At the end of this pre-processing step, there are only 9 possible observations for each module, depending on which cell the neighborhood center of mass belongs to.

In addition, instead of actions corresponding to direct movements into adjacent cells, we define the following set of actions. There are 9 actions, corresponding as before to the set of 8 discretized directions plus a NOP. However, in the minimalist learning framework, each action corresponds to a choice of general heading. The particular motion that will be executed by a module, given the local configuration and the chosen action, is determined by the post-processing step shown in figure 6-5. The module first determines the set of legal motions given the local configuration. From that set, given the chosen action (heading), it determines the closest motion to this heading up to a threshold. If no such motions exist, the module remains stationary. The threshold is set by the designer and in our experiments is equal to 1.1, which allows motions up to 2 cells different from the chosen heading.

Differentiating between actions (headings) and motions (cell movements) allows the modules to essentially learn a discrete trajectory as a function of a simple measure

[1] Four directions of largest gradient plus the case of a local optimum (all gradients equal to a threshold).

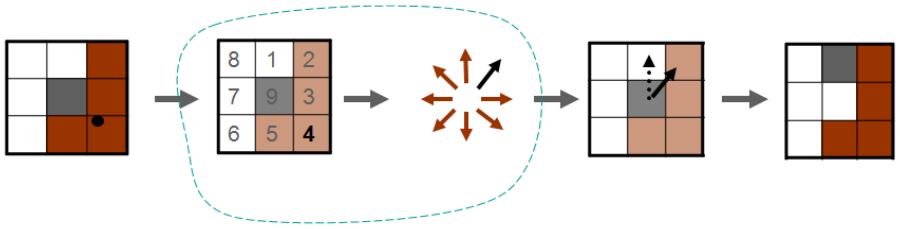


Figure 6-5: Compressed learning representation and algorithm: pre-processing (a) compute local neighborhood's weighted center of mass, biased towards the periphery, (b) discretize into one of 9 possible observations; GAPS (c) from observation, choose action according to the policy: each action represents one of 8 possible headings or a NOP; post-processing (d) select closest legal movement up to threshold give local configuration of neighbors, (e) execute movement. Only the steps inside the dashed line are part of the actual learning algorithm.

of locally observed neighbors.

6.2.3 Experiments: reaching for a random target

The modules learned to reach out to a target positioned randomly in the robot's workspace (reachable $x, y-$space). The target (x_0, y_0) was also the (sole) maximum of f, which was a distance metric. The targets were placed randomly in the half-sphere defined by the $y = 0$ line and the circle with its center at the middle of the robot and its radius equal to $(N + M)/2$, where N is the number of modules and M the robot's size along the y dimension.

Fifteen modules learned to reach to a random goal target using GAPS with the minimal representation and minimal gradient sensing framework with the lower sensory resolution (4 possible directions or local optimum) of section 6.2.1. Figure 6-6 shows an execution sequence leading to the reaching of one of the modules to the target: the sequence was obtained by executing a policy learned after 10,000 episodes. Ten learning trials were performed, and each policy was then tested for 10 test trials. The average number of times a module reached the goal in 50 timesteps during the test trials for each policy was 7.1 out of 10 (standard deviation: 1.9).

We record two failure modes that contribute to this error rate. On the one hand, local observability and stochastic policy means a module may sometimes move into a position that completes a box-type structure with no modules inside (as previously shown in figure 3-12b). When that happens, the conservative local rules we use to prevent robot disconnection will also prevent any corner modules of such a structure from moving, and the robot will be stuck. Each learned policy led the robot into this box-type stuck configuration a minimum of 0 and a maximum of 3 out of the 10 test runs (mean: 1.2, standard deviation: 1). We can eliminate this source of error by allowing a communication protocol to establish an alternative connection beyond the local observation window (Fitch & Butler (2007) have one such protocol). This

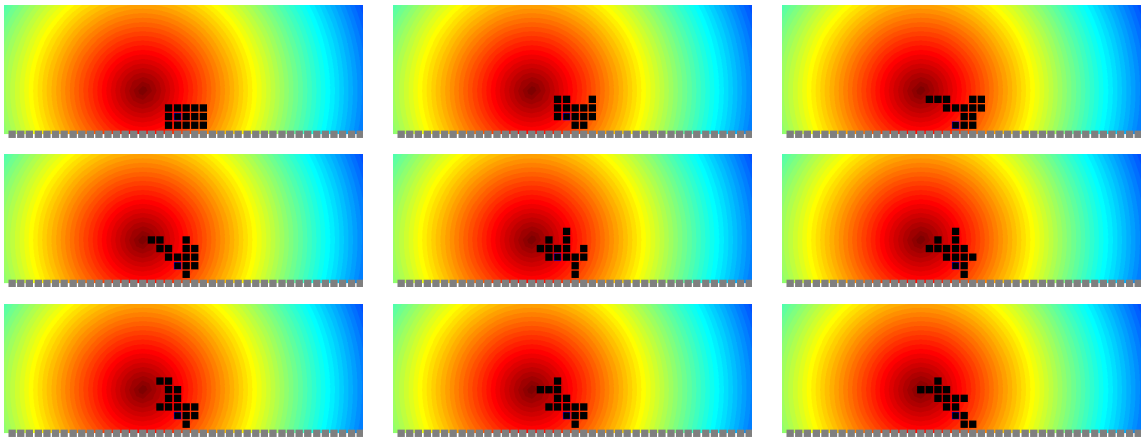


Figure 6-6: Reaching to a random goal position using sensory gradient information during learning and test phases. Policy execution captured every 5 frames.

leaves an average of 1.7 proper failures per 10 test trials (standard deviation: 1.6). Three out of 10 policies displayed no failures, and another two failed only once.

6.3 Post learning knowledge transfer

If a function over the (x, y) space has a relationship to the reward signal such that sensing the largest function gradient gives the modules information about which way the greatest reward will lie, then there should be a way of using that information to reuse knowledge obtained while learning one behavior, to another one under certain conditions. For example, if modules have learned to reach any goal randomly positioned in the whole (x, y) space, that policy would subsume any locomotion policies. Since distributed learning is hard, can modules learn one policy that would apply to different tasks, given the gradient information? We are also interested in the situations where learning a simpler policy would give modules an advantage when presented with a more difficult task, in the spirit of good starting points for policy search that we are exploring in this thesis.

In the next sections we study possible knowledge transfer from policies learned with underlying gradient information in two ways: using gradients as extra observation, or using gradient information in order to perform geometric transformations of the old policy for the new task.

6.3.1 Using gradient information as observation

Suppose modules have learned a policy with the observation space defined over the local neighborhood and direction of greatest gradient of some sensed function f defined over the (x, y) space. Suppose further that the reward signal R was proportional to the gradient of f. The modules are now presented with a new spatially defined task, where the reward signal R' is proportional to the gradient of a new spatially defined

function f'. The intuition is that if f' is sufficiently similar to f, and if all possible gradient observations have been explored during the learning of the old policy, then the modules will get good performance on the new task with the old policy. The condition of sufficient similarity in this case is that f' be a linear transformation of f. This is subject to symmetry breaking experimental constraints such as requiring that the modules stay connected to the ground line at all times.

The idea that having seen all gradient observations and learned how to behave given them and the local configuration should be of benefit in previously unseen scenarios can be taken further. In the next section, we experiment with a limited set of gradient directions during the learning phase, then test using random directions.

6.3.2 Experiments with gradient sensing

In this set of experiments, 16 modules, originally in a 4x4 square configuration, learned using GAPS with minimal representation and minimal sensing in the following setup. At each episode, the direction of reward was chosen randomly between eastward, westward and upward. Therefore, the modules learned a policy for three out of the four possible local gradients. We tested the policies obtained in 10 such learning trials, after 10,000 episodes each, and separated them into two groups based on performance (5 best and 5 worst). We then test the policies again, this time on the task of reaching to a random goal position within the robot's reachable workspace. Our prediction is that this kind of policy transfer from locomotion and reaching up to reaching to a random goal should be possible, and therefore that the quality of policy learned on this small subset of possible directions will determine whether the modules succeed on the second test task.

In these experiments, the 5 best policies succeeded in reaching the random goal on average in 7.8 out of 10 test trials (standard deviation: 1.6) and the 5 worst policies succeeded on average in 5 out of 10 trials (standard deviation: 1.9). The random policy never succeeded. The difference in success rate between the two groups is statistically significant ($F(1,8)=6.32$, $p=.0361$) at the 95% confidence level. As we expected, learning with gradients enables modules to transfer some behavioral knowledge to a different task.

6.3.3 Using gradient information for policy transformations

It is interesting to note that in lattice-based SRMRs, the observation and action sets are spatially defined. Whether an observation consists of the full local neighborhood configuration surrounding module i or a post-processed number representing the discretized relative position of the neighbors' local center of mass, it comes from the spatial properties of the neighborhood. The set of actions consists of motions in space (or directions of motion in space), which are defined by the way they affect said properties. In the cases where the reward function R is also spatially defined, local geometric transformations exist that can enable agents to transfer their behavioral knowledge encoded in the policy to different reward functions.

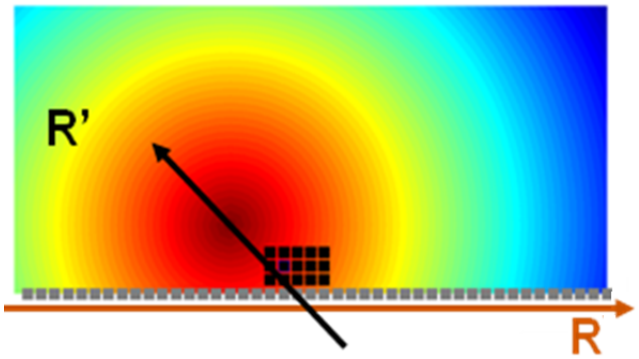


Figure 6-7: When there is a known policy that optimizes reward R, modules should be able to use local geometric transformations of the observation and policy to achieve better than random performance in terms of reward R' (sensed as direction of greatest gradient). The ground line breaks the rotational transformations when $\angle RR' > \pi/2$ such that a flip by the vertical becomes necessary. The implementation is discretized with $\pi/4$ intervals, which corresponds to 8 cells.

Suppose $R = \nabla f(x, y)$ in two-dimensional space and a good policy exists, parameterized by $\theta(o, a)$, where o is the result of local spacial filtering. Note that, in the minimal learning representation, both o and a are discretized angles from the vertical. If we now transform R, say, by rotating $f(x, y)$ with respect to the module by an angle ω, then rotating also the observation and action such that $o' = o - \omega$ and $a' = a - \omega$ now describes the new situation; and taking $\pi'_\theta(o', a') = \pi_\theta(o, a)$ should give the action that should be executed in the new situation. Figure 6-7 shows how rotating the reward coordinate frame works. The idea is intuitive in the minimal representation, since actually executed motions are divorced from the actions represented by the policy, the latter being simply desirable motion directions. However, the transformation also works in the standard GAPS representation. Figure 6-8 shows how to transform the neighborhood observation and find the equivalent actions in a standard representation.

The rotation example, while intuitive, breaks down because the existence of the ground line, to which the modules are tied, breaks the rotational symmetry. However, it should be possible to use symmetry about the vertical axis instead: in the experiments in section 6.3.4 we transform the observations and actions first by rotating by up to $\pi/2$, then by flipping about the vertical axis if necessary.

While some transfer of knowledge will be possible, there is no reason to believe that transformation alone will give a resulting policy that will be optimal for the new reward. Instead, like incremental learning and partially specified policies, the use of transformations can be another way of biasing search.

Algorithm 6 GAPS with transformations at every timestep

```
Initialize parameters θ according to experimental condition
for each episode do
  Calculate policy π(θ)
  Initialize observation counts N ← 0
  Initialize observation-action counts C ← 0
  for each timestep in episode do
    for each module m do
      sense direction of greatest gradient ω
      observe o
      if ω <= π/2 then
        rotate o by −ω → o′
      else
        flip o by vertical and rotate by π/2 − ω → o′
      end if
      increment N(o′)
      choose a from π(o′, θ)
      if ω <= π/2 then
        rotate a by ω → a′
      else
        flip a by vertical and rotate by ω − π/2 → a′
      end ifand increment C(o′, a′)
      execute a′
    end for
  end for
  Get global reward R
  Update θ according to
  θ( o′, a′ ) += α R( C(o′, a′) − π(o′, a′, θ) N(o′) )
  Update π(θ) using Boltzmann's law
end for
```

6.3.4 Experiments with policy transformation

We performed a set of experiments where 15 modules learned the eastward locomotion task with the minimal representation. After 10,000 episodes, the learning was stopped, and the resulting policies were first tested on the learning task and then transformed as described in section 6.3.3. Figure 6-9 shows equivalent situations before and after the transformation. The policies were then tested with the task of westward locomotion. Out of 8 policies that corresponded to acceptable eastward locomotion gaits after 10,000 episodes of learning, 7 performed equally well in 10 test trials each in westward locomotion when flipped. The remaining two policies, which made armlike protrusions, did not show locomotion gaits when flipped either.

In another set of experiments, 15 and 20 modules learned with a standard GAPS representation using 2^8 possible observations, and the modules performed transformations of the local observation and corresponding policy at every timestep. Here, we use the higher resolution sensing model with 8 possible gradient directions (plus the

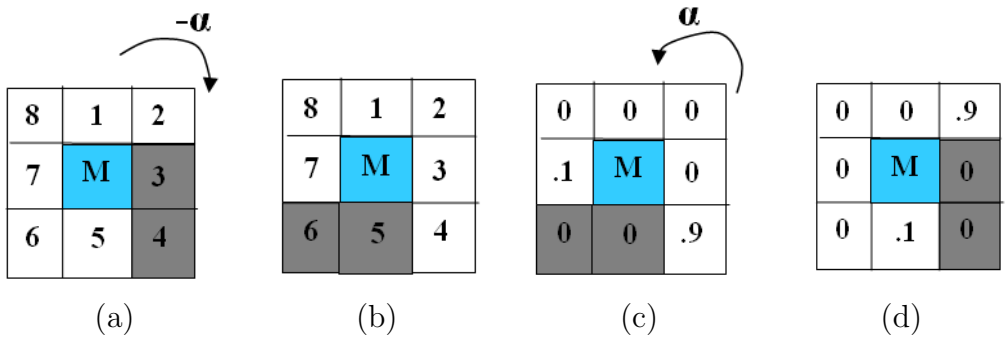


Figure 6-8: Policy transformation with standard GAPS representation when the reward direction is rotated by $\pi/2$ from the horizontal (i.e., the modules now need to go North, but they only know how to go East): (a) find local neighborhood and rotate it by $-\pi/2$ (b)-(c) the resulting observation generates a policy (d) rotate policy back by $\pi/2$ to get equivalent policy for moving North.

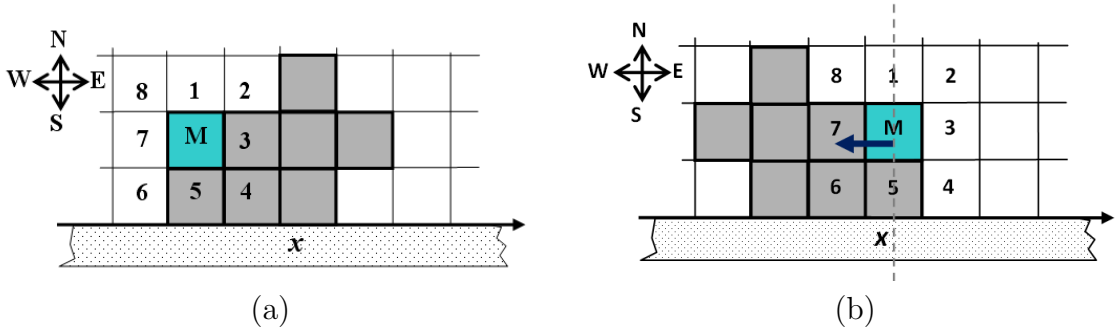


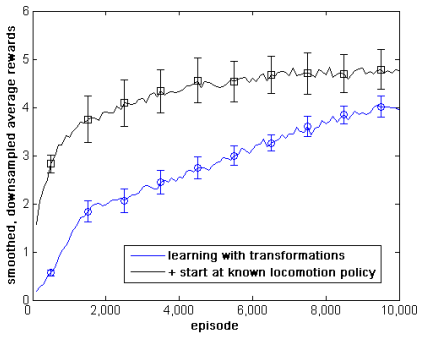
Figure 6-9: Equivalent observation and corresponding policy for module M (a) before and (b) after the flip.

case of a local optimum). Thus, modules maintain a single policy with the standard number of parameters – the sensory information is used in transformations alone, and the modules are learning a "universal" policy that should apply in all directions (see Algorithm 6). The experiments were then run in three conditions with ten trials in each condition:

1. Modules learned to reach to a random target from scratch, with parameters initialized to small random numbers (baseline).

2. Modules learned eastward locomotion, but were tested on the task of reaching to a random target (post-learning knowledge transfer).

3. Modules learned to reach to a random target with parameters initialized to an existing policy for eastward locomotion, obtained from condition 2 (biasing search with a good starting point).



(a)

mean success ± std. err.	15 mods	20 mods
learn with transformations	8.1 ± .4	5.1 ± .6
learn eastward locomotion, test with transformations	6.4 ± .8	5.0 ± .8
use policy for eastward locomotion as starting point	7.6 ± .4	8.6 ± .5
random policy	3	0

(b)

Figure 6-10: (a) Average over 10 learning trials of smoothed (100-point window), downsampled reward over time (learning episodes) for 20 modules running centralized learning with episode length of T=50, learning to reach to an arbitrary goal position with geometric transformations of observations, actions, and rewards, starting with no information (blue) vs. starting with existing locomotion policy (black). Standard error bars at 95% confidence level. (b) Table of average number of times the goal was reached out of 10 test trials, with standard error: using locomotion policy as a good starting point for learning with transformations works best with a larger number of modules.

Table 6-10b summarizes the success rates of the policies learned under these three conditions on the task of reaching to a randomly placed target within the robot's reachable space, with the success rate of the random policy shown for comparison. We can see that all conditions do significantly better than the random policy, for both 15 and 20 modules. In addition, a nonparametric Kruskal-Wallis ANOVA shows no significant difference between the success rate of the three conditions for 15 modules, but for 20 modules, there is a significant difference ($\chi^2(2, 27) = 13.54, p = .0011$) at the 99% confidence level. A post-hoc multiple comparison shows that the policies from condition 3 are significantly more successful than all others. As the problem becomes harder with a larger number of modules, so biasing the search with a good starting point generated by geometrically transforming the policy helps learning more.

Fig. 6-10a shows the smoothed, downsampled average rewards (with standard error) obtained over time by 20 modules learning with transformations in conditions 1 and 3. We can see clearly that biasing GAPS with a good starting point obtained through transforming a good policy for a different but related task, results in faster learning, and in more reward overall.

6.4 Discussion

We have demonstrated the applicability of GAPS to tasks beyond simple locomotion, and presented results in learning to build tall structures (reach upward), move over rough terrain and reach to a random goal position. As we move from simple locomo-

tion to broader classes of behavior for lattice-based SRMRs, we quickly outgrow the standard model for each module's local observations. Therefore, in this chapter we have introduced a model of local gradient sensing that allows expansion of the robots' behavioral repertoire, as amenable to learning. We have additionally introduced a novel, compressed policy representation for lattice-based SRMRs that reduces the number of parameters to learn.

With the evaluation of our results on different objective functions, we have also presented a study of the possibility of behavioral knowledge transfer between different tasks, assuming sensing capabilities of spatially distributed gradients, and rewards defined in terms of those gradients. We find that it is possible to transfer a policy learned on one task to another, either by folding gradient information into the observation space, or by applying local geometric transformations of the policy (or, equivalently, observation and action) based on the sensed gradient information. The relevance of the resulting policies to new tasks comes from the spatial properties of lattice-based modular robots, which require observations, actions and rewards to be local functions of the lattice space.

Finally, we demonstrate empirically that geometric policy transformations can provide a systematic way to automatically find good starting points for further policy search.

Chapter 7

Concluding Remarks

7.1 Summary

In this thesis, we have described the problem of automating distributed controller generation for self-reconfiguring modular robots (SRMRs) through reinforcement learning. In addition to being an intuitive approach for describing the problem, distributed reinforcement learning also provides a double-pronged mechanism for both control generation and online adaptation.

We have demonstrated that the problem is technically difficult due to the distributed but coupled nature of the modular robots, where each module (and therefore each agent) has only access to local observations and a local estimate of reward. We then proposed a class of algorithms taking Gradient Ascent in Policy Space (GAPS) as a baseline starting point. We have shown that GAPS-style algorithms are capable of learning policies in situations where the more powerful Q-learning and Sarsa are not, as they make the strong assumption of full observability of the MDP. We identified three key issues in using policy gradient algorithms for learning, and specified which parameters of the SRMR problem contribute to these issues. We then addressed scalability issues presented by learning in SRMRs, including the proliferation of local optima unacceptable for desired behaviors, and suggested ways to mitigate various aspects of this problem through search constraints, good starting points, and appropriate policy representations. In this context, we discussed both centralized and fully distributed learning. In the latter case, we proposed a class of agreement-based GAPS-style algorithms for sharing both local rewards and local experience, which we have shown speed up learning by at least a factor of 10. Along the way we provided experimental evidence for our algorithms' performance on the problem of locomotion by self-reconfiguration on a two-dimensional lattice, and further evaluated our results on problems with different objective functions. Finally, we proposed a geometric transformation framework for sharing behavioral knowledge between different tasks and as another systematic way to automatically find good starting points for distributed search.

7.2 Conclusions

We can draw several conclusions from our study.

1. The problem of partially observable distributed reinforcement learning (as in SRMRs) is extremely difficult, but it is possible to use policy search to learn reasonable behaviors.

2. Robot and problem design decisions affect the speed and reliability of learning by gradient ascent, in particular:

   the number of parameters to learn is affected by the policy representation, the number of modules comprising the robot, and search constraints, if any;

   the quality of experience is affected by the length of each learning episode, the robot size, and the sharing of experience, if any;

   the starting point is affected by any initial information available to modules, and by the additive nature of modular robots, which enables learning by incremental addition of modules for certain tasks;

   the initial condition in addition to the starting point in policy space, the initial configuration of the robot affects the availability of exploratory actions;

   the quality of the reward estimate is affected by the sharing of rewards through agreement, if any.

3. It is possible to create good design strategies for mitigating scalability and local optima issues by making appropriate design decisions, such as pre-screening for legal actions — which effectively reduces the number of parameters to be learned — or using smart and appropriate policy representations — which nonetheless require careful human participation and trade-offs between representational power (the ideal policy must be included), number of policy parameters, and the quality of policy value landscape that these parameters create (it may have many local optima but lack the redundancy of the full representation).

4. Unlike centralized but factored GAPS, fully distributed gradient ascent with partial observability and local rewards is impossible without improvement in either (a) amount and quality of experience, or (b) quality of local estimates of reward. Improvement in (a) can be achieved through good design strategies mentioned above, and either (a) or (b) or both benefit tremendously from agreement-style sharing among local neighbors.

5. Agreement on both reward and experience results in learning performance that is empirically almost equivalent to the centralized but factored scenario for smaller numbers of modules.

6. The difference in reward functions between centralized learning and distributed learning with agreement generates a difference in learned policies: for larger numbers and initial configurations of higher aspect ratios, agreement by neighborhood averaging of local estimates of rewards and experience favors the finding of more compact gaits, and is unlikely to find the same two-layer thread-like gaits that prevail in the solutions found by centralized learning.

7. Some post-learning knowledge transfer is possible between different spatially defined tasks aimed at lattice-based SRMRs. It is achievable through simple gradient sensing, which is then used either as an additional observation component, or in a framework of local geometric policy transformations. Either way, the resulting policies may be perfectly adequate for the new task, or at the least can become good starting points for further search.

While this thesis is mainly concerned with learning locomotion by self-reconfiguration on a two-dimensional square lattice, the results obtained herein are in no way restricted to this scenario. In chapter 6 we already generalized our findings to different objective functions in the same domain. This generalization required only minimal additional assumptions, which would ensure that good policies would be within the search space. For example, extending to locomotion over obstacles involved introducing a third possible observation value per neighborhood cell, which increased the number of policy parameters. Another example is the objective of reaching to an arbitrary goal in the workspace, which necessitated the additional assumption of a simple sensory model. In both cases, however, the structure of the algorithm did not change, and the lessons we have learned about scaffolding learning by constraining and biasing the search remain valid.

In addition, we believe that the same arguments and ideas will be at play whenever cooperative distributed controller optimization is required in networked or physically connected systems where each agent suffers from partial observability and noisy local estimates of rewards. The set of these situations could include coordination of networked teams of robots, robotic swarm optimization, and learning of other cooperative group behavior. For example, locomotion gaits learned by modules in this thesis could potentially be useful in formation control of a team of wheeled mobile robots, whose connections to each other are those of remaining within communication range, rather than being physically attached. Similarly, in manufacturing and self-assembly scenarios, particularly whenever the environment can be partitioned into a geometric lattice, it should be possible to employ the same learning strategies described in this thesis, provided enough relevant information and search constraints are available. The result would be more automation in coordinated and cooperative control in these domains with less involvement from human operators.

7.3 Limitations and future work

At present, we have developed an additional automation layer for the design of distributed controllers. The policy is learned offline by GAPS; it can then be transferred

to the robots. We have not addressed online adaptation in this thesis. In the future we would like to see the robots, seeded with a good policy learned offline, to run a much faster distributed adaptation algorithm to make practical run-time on-the-robot adjustments to changing environments and goals. We are currently exploring new directions within the same concept of restricting the search in an intelligent way without requiring too much involved analysis on the part of the human designer. It is worth mentioning anyway that the computations required by the GAPS algorithm are simple enough to run on a microcontroller with limited computing power and memory.

A more extensive empirical analysis of the space of possible representations and search restrictions is needed for a definitive strategy in making policy search work fast and reliably in SRMRs. In particular, we would like to be able to describe the properties of good feature spaces for our domain, and ways to construct them. A special study of the influence of policy parameter redundancy would be equally useful and interesting: for example, we would like to know in general when restricting exploration to only legal actions might be harmful instead of helpful because it removes a source of smoothness and redundancy in the landscape.

We have studied how search constraints and information affect the learning of locomotion gaits in SRMRs in this thesis. The general conclusion is that more information is always better, especially when combined with restrictions on how it may be used, which essentially guides and constrains search. This opens up the theoretical question on the amount and kinds of information needed for particular systems and tasks, which could provide a fruitful direction of future research.

As we explore collaboration between the human designer and the automated learning agent, our experiments bring forward some issues that such interactions could raise. As we have observed, the partial information coming from the human designer can potentially lead the search away from the global optimum. The misleading can take the form of a bad feature representation, or an overconstrained search, or a partial policy that favors suboptimal actions.

Another issue is that of providing a reward signal to the learning agents that actually provides good estimates of the underlying objective function. Experiments have shown that a simple performance measure such as displacement during a time period cannot always disambiguate between good behavior (i.e., locomotion gait) and an unacceptable local optimum (e.g., a long arm-like protrusion in the right direction). In this thesis, we have restricted experiments to objective functions that depend on a spatially described gradient. Future work could include more sophisticated reward signals.

We have also seen throughout that the learning success rate (as measured, for example, by the percentage of policies tested which exhibited correct behavior) depends not only on robot size, but also on the initial condition, specifically, the configuration that the robot assumes at the start of every episode. In this thesis, we have limited our study to a few such initial configurations and robot sizes in order to compare the different approaches to learning. In the future, we would like to describe the space of initial conditions, and research their influence on learning performance more broadly.

Finally, the motivation of this work has been to address distributed RL problems of

a cooperative nature; and as a special case of those problems, some of our approaches (e.g., sharing experience) is applicable to agents essentially learning identical policies. While this is a limitation of the present work, we believe that most algorithmic and representational findings we have discussed should also be valid in less identical and less cooperative settings. In particular, sharing experience but not local rewards might lead to interesting developments in games with non-identical payoffs. These developments are also potential subjects for future work.

In addition to the above, we would like to see the ideas described in this thesis applied to concrete models of existing modular robots, as well as to other distributed domains, such as chain and hybrid modular robots, ad hoc mobile networks, and robotic swarms.

Bibliography

Andre, D. & Russell, S. (2000), Programmable reinforcement learning agents, *in* 'Neural Information Processing Systems', Vancouver, Canada.

Bagnell, J. A., Kakade, S., Ng, A. Y. & Schneider, J. (2004), Policy search by dynamic programming, *in* 'Advances in Neural Information Processing Systems 16'.

Baxter, J. & Bartlett, P. L. (2001), 'Infinite-horizon gradient-based policy search', *Journal of Artificial Intelligence Research* **15**, 319–350.

Bertsekas, D. P. (1995), *Dynamic programming and optimal control*, Athena Scientific, Belmont MA.

Bertsekas, D. P. & Tsitsiklis, J. N. (1997), *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific.

Bishop, J., Burden, S., Klavins, E., Kreisberg, R., Malone, W., Napp, N. & Nguyen, T. (2005), Self-organizing programmable parts, *in* 'International Conference on Intelligent Robots and Systems, IEEE/RSJ Robotics and Automation Society'.

Brunner, E. & Puri, M. L. (2001), 'Nonparametric methods in factorial designs', *Statistical papers* **42**(1), 1–52.

Buhl, J., Sumpter, D. J., Couzin, I., Hale, J., Despland, E., Miller, E. & Simpson, S. J. (2006), 'From disorder to order in marching locusts', *Science* **312**, 1402–1406.

Butler, Z., Kotay, K., Rus, D. & Tomita, K. (2001), Cellular automata for decentralized control of self-reconfigurable robots, *in* 'Proceedings of the International Conference on Robots and Automation'.

Butler, Z., Kotay, K., Rus, D. & Tomita, K. (2004), 'Generic distributed control for locomotion with self-reconfiguring robots', *International Journal of Robotics Research* **23**(9), 919–938.

Chang, Y.-H., Ho, T. & Kaelbling, L. P. (2004), All learning is local: Multi-agent learning in global reward games, *in* 'Neural Information Processing Systems'.

Everist, J., Mogharei, K., Suri, H., Ranasinghe, N., Khoshnevis, B., Will, P. & Shen, W.-M. (2004), A system for in-space assembly, *in* 'Proc. Int. Conference on Robots and Systems (IROS)'.

Fernandez, F. & Parker, L. E. (2001), 'Learning in large cooperative multi-robot domains', *International Journal of Robotics and Automation: Special issue on Computational Intelligence Techniques in Cooperative Robots* **16**(4), 217–226.

Fitch, R. & Butler, Z. (2006), A million-module march, *in* 'Digital Proceedings of RSS Workshop on Self-Reconfigurable Modular Robotics', Philadelphia, PA.

Fitch, R. & Butler, Z. (2007), 'Million-module march: Scalable locomotion for large self-reconfiguring robots', submitted to the IJRR Special Issue on Self-Reconfigurable Modular Robots.

Gilpin, K., Kotay, K. & Rus, D. (2007), Miche: Self-assembly by self-disassembly, *in* 'Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems', San Diego, CA.

Griffith, S., Goldwater, D. & Jacobson, J. M. (2005), 'Robotics: Self-replication from random parts', *Nature* **437**, 636.

Grudic, G. Z., Kumar, V. & Ungar, L. (2003), Using policy reinforcemnet learning on autonomous robot controllers, *in* 'Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems', Las Vegas, Nevada.

Guestrin, C., Koller, D. & Parr, R. (2002), Multiagent planning with factored mdps, *in* 'Advances in Neural Information Processing Systems (NIPS)', Vol. 14, The MIT Press.

Hu, J. & Wellman, M. P. (1998), Multiagent reinforcement learning: Theoretical framework and an algorithm, *in* J. Shavlik, ed., 'International Conference on Machine Learning', Morgan Kaufmann, Madison, Wisconsin, pp. 242–250.

Jadbabaie, A., Lin, J. & Morse, A. S. (2003), 'Coordination of groups of mobile autonomous agents using nearest neighbor rules', *IEEE Transactions on Automatic Control* **48**(6).

Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K. & Kokaji, S. (2004), Distributed adaptive locomotion by a modular robotic system, M-TRAN II – from local adaptation to global coordinated motion using cpg controllers, *in* 'Proc. of Int. Conference on Robots and Systems (IROS)'.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. & Osawa, E. (1997), RoboCup: The robot world cup initiative, *in* W. L. Johnson & B. Hayes-Roth, eds, 'Proceedings of the First International Conference on Autonomous Agents (Agents'97)', ACM Press, New York, pp. 340–347.

Kok, J. R. & Vlassis, N. (2006), 'Collaborative multiagent reinforcement learning by payoff propagation', *Journal of Machine Learning Research* **7**, 1789–1828.

Kotay, K. & Rus, D. (2004), Generic distributed assembly and repair algorithms for self-reconfiguring robots, *in* 'IEEE Intl. Conf. on Intelligent Robots and Systems', Sendai, Japan.

Kotay, K. & Rus, D. (2005), Efficient locomotion for a self-reconfiguring robot, *in* 'Proc. of Int. Conference on Robotics and Automation (ICRA)'.

Kotay, K., Rus, D., Vona, M. & McGray, C. (1998), The self-reconfiguring robotic molecule, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation', Leuven, Belgium, pp. 424–431.

Kubica, J. & Rieffel, E. (2002), Collaborating with a genetic programming system to generate modular robotic code, *in* W. et al., ed., 'GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference', Morgan Kaufmann Publishers, New York, pp. 804–811.

Lagoudakis, M. G. & Parr, R. (2003), 'Least-squares policy iteration', *Journal of Machine Learning Research* **4**, 1107–1149.

Littman, M. L. (1994), Memoryless policies: Theoretical limitations and practical results, *in* 'From Animals to Animats 3: Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior (SAB)', Cambridge, MA.

Marthi, B., Russell, S. & Andre, D. (2006), A compact, hierarchically optimal q-function decomposition, *in* 'Procedings of the Intl. Conf. on Uncertainty in AI', Cambridge, MA.

Martin, M. (2004), The essential dynamics algorithm: Fast policy search in continuous worlds, *in* 'MIT Media Lab Tech Report'.

Matarić, M. J. (1997), 'Reinforcement learning in the multi-robot domain', *Autonomous Robots* **4**(1), 73–83.

Meuleau, N., Peshkin, L., Kim, K.-E. & Kaelbling, L. P. (1999), Learning finite-state controllers for partially observable environments, *in* 'Proc. of 15th Conf. on Uncertainty in Artificial Intelligence (UAI)'.

Moallemi, C. C. & Van Roy, B. (2003), Distributed optimization in adaptive networks, *in* 'Proceedings of Intl. Conference on Neural Information Processing Systems'.

Moallemi, C. C. & Van Roy, B. (2006), 'Consensus propagation', *IEEE Transactions on Information Theory* **52**(11).

Murata, S., Kurokawa, H. & Kokaji, S. (1994), Self-assembling machine, *in* 'Proceedings of IEEE Int. Conf. on Robotics and Automation (ICRA)', San Diego, California, p. 441448.

Murata, S., Kurokawa, H., Yochida, E., Tomita, K. & Kokaji, S. (1998), A 3d self-reconfigurable structure, *in* 'Proceedings of the 1998 IEEE International Conference on Robotics and Automation', pp. 432–439.

Murata, S., Yoshida, E., Kurokawa, H., Tomita, K. & Kokaji, S. (2001), 'Self-repairing mechanical systems', *Autonomous Robots* **10**, 7–21.

Mytilinaios, E., Marcus, D., Desnoyer, M. & Lipson, H. (2004), Design and evolved blueprints for physical self-replicating machines, *in* 'Proc. of 9th Int. Conference on Artificial Life (ALIFE IX)'.

Nagpal, R. (2002), Programmable self-assembly using biologically-inspired multiagent control, *in* 'Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)', Bologna, Italy.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E. & Liang, E. (2004), Autonomous inverted helicopter flight via reinforcement learning, *in* 'ISER'.

Ng, A. Y. & Jordan, M. (2000), Pegasus: A policy search method for large mdps and pomdps, *in* 'Proc. of Int. Conference on Uncertainty in AI (UAI)'.

Okubo, A. (1986), 'Dynamical aspects of animal grouping: Swarms, schools, flocks and herds', *Advances in Biophysics* **22**, 1–94.

Olfati-Saber, R., Franco, E., Frazzoli, E. & Shamma, J. S. (2005), Belief consensus and distributed hypothesis testing in sensor networks, *in* 'Proceedings of the Workshop on Network Embedded Sensing and Control', Notre Dame University, South Bend, IN.

Olfati-Saber, R. & Murray, R. M. (2004), 'Consensus problems in networks of agents with switching topology and time-delays', *IEEE Trans. on Automatic Control* **49**(9), 1520–1533.

Østergaard, E. H., Kassow, K., Beck, R. & Lund, H. H. (2006), 'Design of the ATRON lattice-based self-reconfigurable robot', *Autonomous Robots* **21**(2), 165–183.

Pamecha, A., Chiang, C.-J., Stein, D. & Chirikjian, G. (1996), Design and implementation of metamorphic robots, *in* 'Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference'.

Peshkin, L. (2001), Reinforcement Learning by Policy Search, PhD thesis, Brown University.

Peshkin, L., Kim, K., Meuleau, N. & Kaelbling, L. P. (2000), Learning to cooperate via policy search, *in* 'Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)', p. 489.

Schaal, S., Peters, J., Nakanishi, J. & Ijspeert, A. (2003), Learning movement primitives, *in* 'Int. Symposium on Robotics Research (ISRR)'.

Schneider, J., Wong, W.-K., Moore, A. & Riedmiller, M. (1999), Distributed value functions, *in* 'Proceedings of the International Conference on Machine Leanring'.

Shen, W.-M., Krivokon, M., Chiu, H., Everist, J., Rubenstein, M. & Venkatesh, J. (2006), 'Multimode locomotion via SuperBot reconfigurable robots', *Autonomous Robots* **20**(2), 165177.

Shoham, Y., Powers, R. & Grenager, T. (2003), 'Multi-agent reinforcement learning: a critical survey'.

Stone, P. & Veloso, M. M. (2000), 'Multiagent systems: A survey from a machine learning perspective', *Autonomous Robots* **8**(3), 345–383.

Suh, J. W., Homans, B. & Yim, M. (2002), Telecubes: Mechanical design of a module for self-reconfiguration, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation', Washington, DC.

Sutton, R. S. (1995), Generatlization in reinforcement learning: Successful examples using sparse coarse coding, *in* D. S. Touretzky, M. C. Mozer & M. E. Hasselmo, eds, 'Advances in Neural Information Processing Systems', MIT Press, pp. 1038–1044.

Sutton, R. S. & Barto, A. G. (1999), *Reinforcement Learning: An Introduction*, MIT Press.

Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. (2000), Policy gradient methods for reinforcement learning with function approximation, *in* 'Advances in Neural Information Processing Systems 12'.

Tedrake, R., Zhang, T. W. & Seung, H. S. (2005), Learning to walk in 20 minutes, *in* 'Proc. 14th Yale Workshop on Adaptive and Learning Systems', Yale University, New Haven, CT.

Toothaker, L. E. & Chang, H. S. (1980), 'On "the analysis of ranked data derived from completely randomized factorial designs"', *Journal of Educational Statistics* **5**(2), 169–176.

Tsitsiklis, J. N., Bertsekas, D. P. & Athans, M. (1986), 'Distributed asynchronous deterministic and stochastic gradient optimization algorithms', *IEEE Transactions on Automatic Control* **AC-31**(9), 803–812.

Varshavskaya, P., Kaelbling, L. P. & Rus, D. (2004), Distributed learning for modular robots, *in* 'Proc. Int. Conference on Robots and Systems (IROS)'.

Varshavskaya, P., Kaelbling, L. P. & Rus, D. (2006), On scalability issues in reinforcement learning for self-reconfiguring modular robots, *in* 'Digital Proceedings of RSS Workshop on Self-Reconfigurable Modular Robotics', Philadelphia, PA.

Varshavskaya, P., Kaelbling, L. & Rus, D. (2007), 'Automated design of adaptive controllers for modular robots using reinforcement learning', *accepted for publication in International Journal of Robotics Research, Special Issue on Self-Reconfigurable Modular Robots* .

Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I. & Schochet, O. (1995), 'Novel type of phase transition in a system of self-driven particles', *Physical Review Letters* **75**(6), 1226–1229.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Q-learning', *Machine Learning* **8**, 279–292.

White, P., Zykov, V., Bongard, J. & Lipson, H. (2005), Three-dimensional stochastic reconfiguration of modular robots, *in* 'Proceedings of Robotics: Science and Systems', Cambridge, MA.

Wolpert, D., Wheeler, K. & Tumer, K. (1999), Collective intelligence for control of distributed dynamical systems, Technical Report NASA-ARC-IC-99-44.

Yim, M., Duff, D. G. & Roufas, K. D. (2000), Polybot: a modular reconfigurable robot, *in* 'Proceedings of the IEEE International Conference on Robotics and Automation'.

Yu, W., Takuya, I., Iijima, D., Yokoi, H. & Kakazu, Y. (2002), Using interaction-based learning to construct an adaptive and fault-tolerant multi-link floating robot, *in* H. Asama, T. Arai, T. Fukuda & T. Hasegawa, eds, 'Distributed Autonomous Robotic Systems', Vol. 5, Springer, pp. 455–464.

Zykov, V., Mytilinaios, E., Adams, B. & Lipson, H. (2005), 'Self-reproducing machines', *Nature* **435**(7038), 163–164.