Automated Design of Adaptive Controllers for Modular Robots using Reinforcement Learning

Paulina Varshavskaya, Leslie Pack Kaelbling and Daniela Rus
Computer Science and AI Laboratory
Massachusetts Institute of Technology
Cambridge MA, USA
{paulina|lpk|rus}@csail.mit.edu

Abstract

Designing distributed controllers for self-reconfiguring modular robots has been consistently challenging. We have developed a reinforcement learning approach which can be used both to automate controller design and to adapt robot behavior online. In this paper, we report on our study of reinforcement learning in the domain of self-reconfigurable modular robots: the underlying assumptions, the applicable algorithms, and the issues of partial observability, large search spaces and local optima. We propose and validate experimentally in simulation a number of techniques designed to address these and other scalability issues that arise in applying machine learning to distributed systems such as modular robots. We discuss ways to make learning faster, more robust and amenable to online application by giving scaffolding to the learning agents in the form of policy representation, structured experience and additional information.

With enough structure modular robots can run learning algorithms to both automate the generation of distributed controllers, and adapt to the changing environment and deliver on the self-organization promise with less interference from human designers, programmers and operators.

Keywords: Learning and Adaptive Systems, Cellular and Modular Robots, Animation and Simulation.

1 Introduction

Self-reconfigurable modular robots are made up of distinct physical modules, which have degrees of freedom to move with respect to each other and to connect to or disconnect from each other. Such a 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. In studying self-reconfigurable modular robots (SRMRs) we give up on easier centralized control in the hope of increased versatility and robustness. However, design of distributed controllers can be challenging. Most modular robotic systems run task-specific, hand-designed algorithms; for example, the rule-based systems for the robotic Molecule (Butler et al. 2004) took hours of designer time to synthesize. Some notable exceptions include systems where 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).

To improve SRMR usability, we use statistical machine learning techniques to automatically develop distributed controllers for this type of robots. Specifically, we work in the reinforcement learning (RL) framework (Sutton & Barto 1998) where agents observe and act in the environment and receive a signal which tells them how well they are doing. Unlike with evolutionary algorithms, re-

inforcement learning optimizes during the lifetime of the learning agent, that is, the robot, as it attempts to perform a task. This leads us to believe that the same paradigm can be used not only to automate controller design for SRMRs, but also to 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.

In our research we are ultimately pursuing both goals in applying RL methods to self-reconfigurable modular robots. This paper is the result of our study in applying a class of RL methods known as policy search to the problem of locomotion gaits in SRMRs. The focus here is on off-line controller design, but with a view to extend the same RL paradigm to on-line adaptation. We start by carefully delineating the fundamental assumptions underlying learning algorithms, as well as those required in distributed robotic systems such as SRMRs. We show in section 2 what issues we need to address in choosing methods from the RL arsenal, as a naïvely straightforward application of the most powerful algorithms reveals a fundamental conflict of assumptions. In section 3 we present the policy search algorithm, as well as a number of extensions. These are motivated by a desire for faster learning and reliably good behaviors. We then present experiments in section 4 showcasing the capabilities and limitations of policy search learning for SRMR systems and our extensions, which we discuss in section 5. Prior and related work can be found in section 6.

2 Conflicting assumptions

The most effective of the techniques collectively known as reinforcement learning (RL) all make a number of assumptions about the nature of the environment in which the learning takes place. 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. We will demonstrate below that in the case of modular robots, even their very simplified abstract kinematic models violate 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.

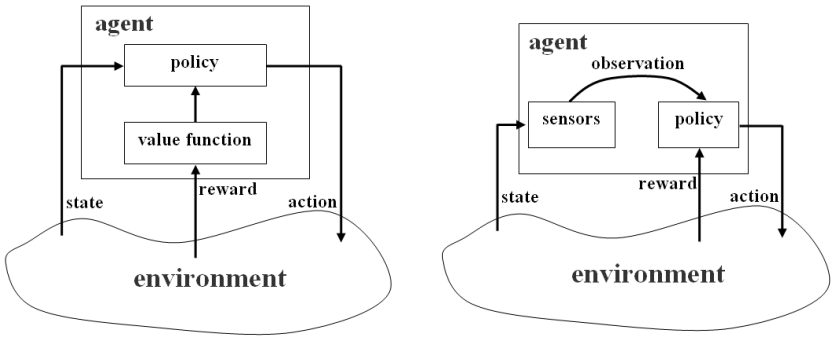
2.1 Assumptions of a Markovian world

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

2.1.1 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 1a. 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 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 such a model is either unknown initially or impossibly difficult to compute for all possible states. Optimization techniques can be used in those situations.

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 1a) 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 1998) for a good introduction to the problems addressed by reinforcement learning and the details of standard solutions.



(a) (b)

Figure 1: 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.

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.

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.

2.1.2 Factored 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 MDP, but execute factored actions $\mathbf{a_t} = \{a_t^1 ... a_t^n\}$, if there are n modules. The individual components of a factored action $\mathbf{a_t}$ 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 2.2, to learn and maintain a policy with respect to each state is in practice unrealistic, as well as prohibitively expensive in both experience and amount of computation.

2.1.3 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 1b 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 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. 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).

2.2 Assumptions of the kinematic model

(1) (2)

Figure 2: 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 our development and experiments on the sliding-cube model for lattice-based self-reconfigurable robots, previously used in the literature (Butler et al. 2004, Fitch & Butler 2006, Varshavskaya et al. 2004). 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 2: 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) such as the Molecule (Kotay & Rus 2005) or MTRAN-II (Kamimura et al. 2004), 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 robot. 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 unit amount of time to execute

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 entirely asynchronous, 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.

2.3 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 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 a single-agent MDP. In this paper, we focus on algorithms developed for partially observable processes.

2.4 Case study: locomotion by self-reconfiguration

We now examine the problem of synthesizing locomotion gaits for modular self-reconfiguring robots. The abstract kinematic model of a 2D lattice-based robot is shown in figure 3. 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, which corresponds to the vertical (x, y) plane, where we show compass directions for clarity of exposition later; 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[1]. The ground line underlying the lattice is not perceived by the modules (for example, by module in cell number 5). It is treated as free space for the purposes of observation; however the simulator will prevent the modules from moving through it or from disconnecting the robot completely from it.

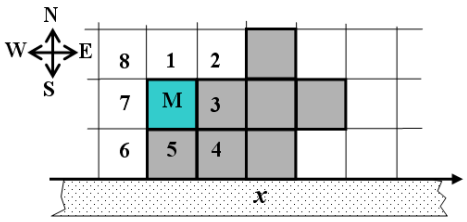


Figure 3: The setup for locomotion by self-reconfiguration on a lattice in 2D.

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[2]. The modules are learning a locomotion gait with the eastward direction of motion receiving positive rewards, and the westward receiving negative rewards.

Clearly, if the full global configuration of this modular robot were known to each module, then each of them

[1] When two neighboring cells at face positions that border the same corner are empty, M will assume that the corner position is also empty. For example, in figure 3, M cannot see but assumes that position 8 is empty. It will not matter either way, since M will not be able to move into the corner position without at least one relevant face neighbor (1 or 7) present.

[2] In that case the simulator will make any action that would disconnect the robot fail.

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 3 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 the next section were not developed exclusively for this scenario. They are general techniques applicable to multi-agent domains. However, the example problem above may be useful in understanding how they instantiate to the SRMR domain.

3 Independent POMDP learning

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

Section 3.1 describes the derivation of the basic GAPS algorithm.

3.1 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 (Peshkin 2001) 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 descent 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):

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

where $\Xi(h)$ is not known to the learning agent and does not depend on θ, and $\Psi(h,\theta)$ is known and differentiable given 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,

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

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

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
```

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. 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 section 3.4.

GAPS has another property interesting for our twin purposes of controller generation and run-time adaptation. A theorem (Peshkin 2001) 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 experience and the same rewards, the two instantiations of the algorithm will find the same solutions. The practical consequences of this result are examined in section 4.3.

In the remainder of this section, we describe extensions to the basic algorithm based on the following idea. We can improve the performance of GAPS and speed up the learning process by imparting more knowledge to the modules. This additional information comes in two types: on the one hand, we can guide the search by creating a more compact representation (section 3.2) or introducing constraints on exploration (section 3.3); on the other hand, we can initialize the search at a smarter starting point (section 3.4).

3.2 GAPS learning with feature spaces

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:

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

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
```

The accumulated traces λ_k are only slightly more computationally involved than the simple counts of the original GAPS algorithm. The updates are just as intuitive, however, as they still assign more credit to those features that differentiate more between actions normalized by their likelihood under the current policy.

The algorithm (Algorithm 2) is guaranteed to converge to a local maximum in policy value space, for a given feature and policy representation.

3.3 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.

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 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. $A_t^i = \{NOP\}$[3] 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[4]
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.

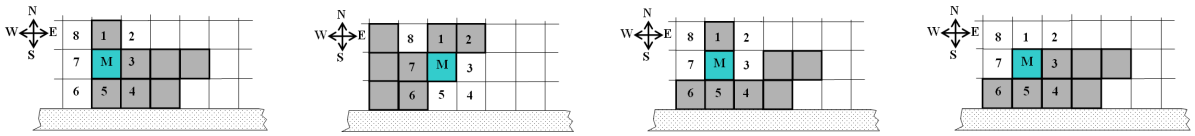
We predict that the added space structure and constraints that were introduced here will result in the modular robot finding good policies with less experience.

3.4 Smarter starting points

Stochastic gradient ascent can be sensitive to the starting point in the policy space from which search initiates. In the case of learning to locomote by self-reconfiguration, it is also easier for the modules to learn if exploration starts from a reasonable policy. Here, we discuss two potential strategies for seeding the algorithms with initial parameters that may lead to better policies.

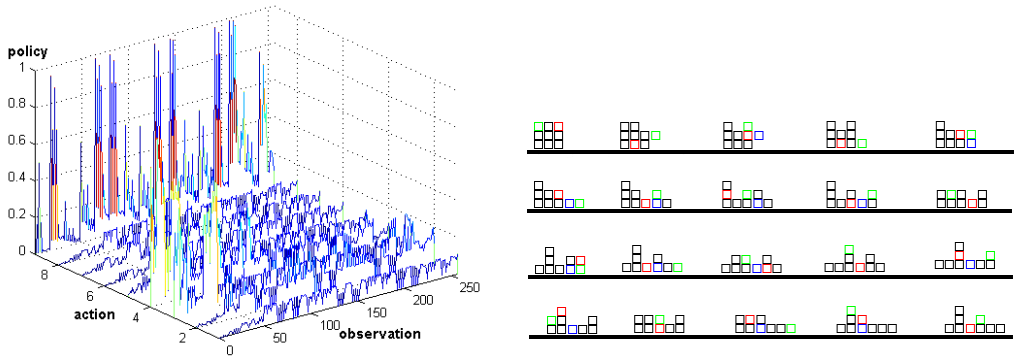
[3] NOP stands for 'no operation' and means the module's action is to stay in its current location and not attempt any motion.

[4] This is a very conservative rule designed to prevent robot disconnections.



(a) (b) (c) (d)

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



(a) (b)

Figure 5: (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.

3.4.1 Incremental GAPS learning

It is possible to leverage the modular nature of our problem 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. We notice that 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 nine 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. 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. The two initial modules initially run the GAPS algorithm. They may stop either after a pre-specified number of episodes, or after their policy parameters have converged to some values 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 or until convergence. 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.

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

4 Experiments

In this section we present results from a number of experiments designed to test how well the basic GAPS algorithm and various extensions perform on the locomotion task using the sliding-cube 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. Then we explore the extensions introduced in section 3 as they provide structure, constraints and initialization information to GAPS learners. We predicted that the extensions should speed up learning by requiring less experience and less exploration.

4.1 The experimental setup

We conduct experiments on a simulated two-dimensional modular robot, which is shown in figure 3. As previewed in section 2.4, 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, meaning that each module executes its action in turn, and no module can go twice before every other module has taken its turn. This assumption is unrealistic; we hope to relax it in future work. 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 5 shows how a policy (a) can be executed by modules to produce a locomotion gait (b) and therefore gain reward.

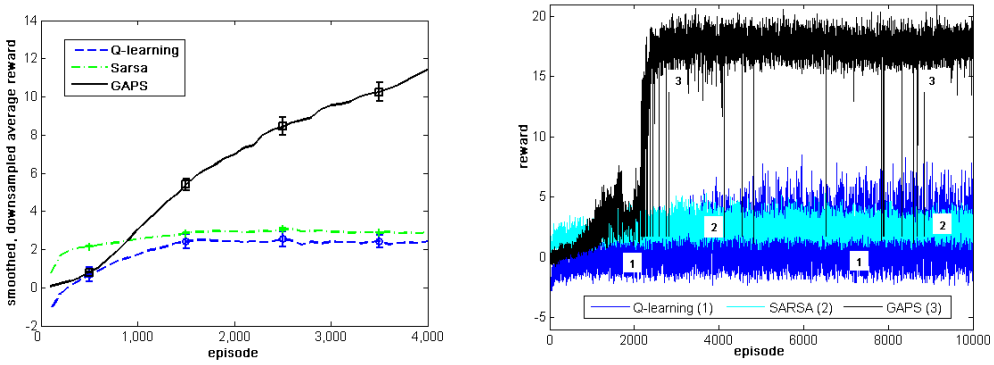
We run the experiments in two broad conditions. First, the goal of automatically generating controllers can in principle be achieved in simulation off-line, and then imparted to the robot for run-time execution. In that case, we can require all modules to share one set of policy parameters, that is, 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.

4.2 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 and get the same reward.

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 the centralized learner as the reward at the end of 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. 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. The curves are shown with standard error bars every 1,000 episodes.



(a) (b)

Figure 6: Performance of policies learned by 15 modules in the 2D locomotion task running Q-learning, Sarsa and GAPS: (a) smoothed, 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.

4.2.1 MDP-like learning vs. gradient ascent

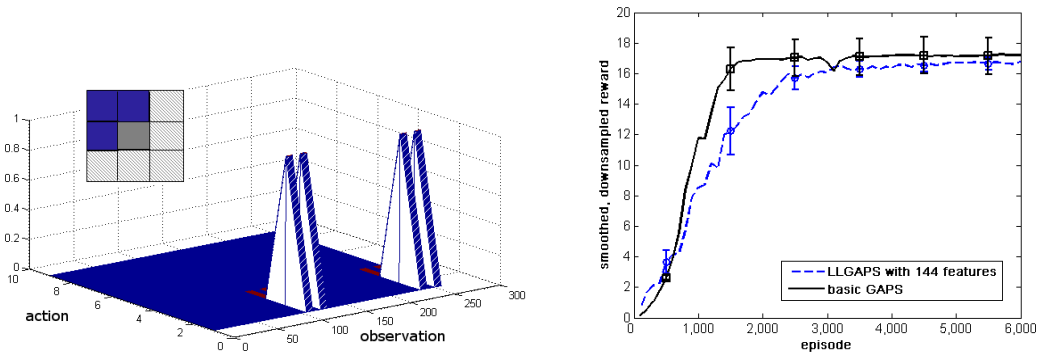
The first set of experiments is pitting the GAPS algorithm against two powerful algorithms which make the Markov assumption: Q-learning (Watkins & Dayan 1992) and Sarsa (Sutton 1995). Sarsa is an on-policy algorithm, and therefore may do better than Q-learning on our task. However, our prediction was 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[5]. Figure 6a shows the smoothed average learning curves, with standard error, 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. In test trials this discrepancy manifested itself as finding a good policy for moving eastward (one such policy is shown in figure 5) 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 6b we see the raw rewards collected at each episode in one typical trial run of all three learning algorithms.

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

4.2.2 Learned and 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 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 8b), achieves an average reward per episode of 5.8 ($\sigma = 0.9$), whereas its counterpart learned using GAPS (execution sequence shown in figure 8a) achieves an average reward of 16.5 ($\sigma = 1.2$). Table 1 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). Figure 8 shows how the learned and the hand-designed gaits differ. By learning with no constraints and a very simple objective function (maxi-



(a) (b)

Figure 7: 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, downsampled average rewards over 10 runs, with standard error: comparison between original GAPS and LLGAPS.

●○	→ NE	
●, ○●, ○● / ○, ○○, ○○	→ SE	● current actor
●, ●, ○● / ○○, ○○○, ○○○	→ E	○ neighbor module
○, ○○, ○○ / ○●, ○●, ○● / ○, ○, ○	→ S	

Table 1: Rules for eastward locomotion, distilled from the best policy learned by GAPS.

mize 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.

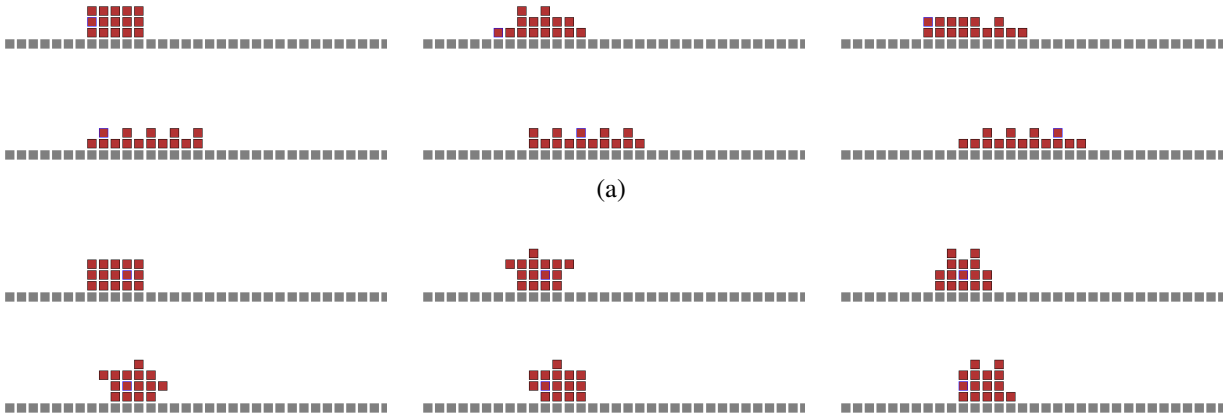
4.2.3 Learning in feature spaces

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 3.2. 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 (see for example figure 7a), for a total of 144 features. In all cases, modules were learning from the space of reactive policies only.

Figure 7b 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.

In general, we expect gradient ascent algorithms to be sensitive to the learning rate and temperature parameters. Our previously reported results (Varshavskaya et al. 2006)



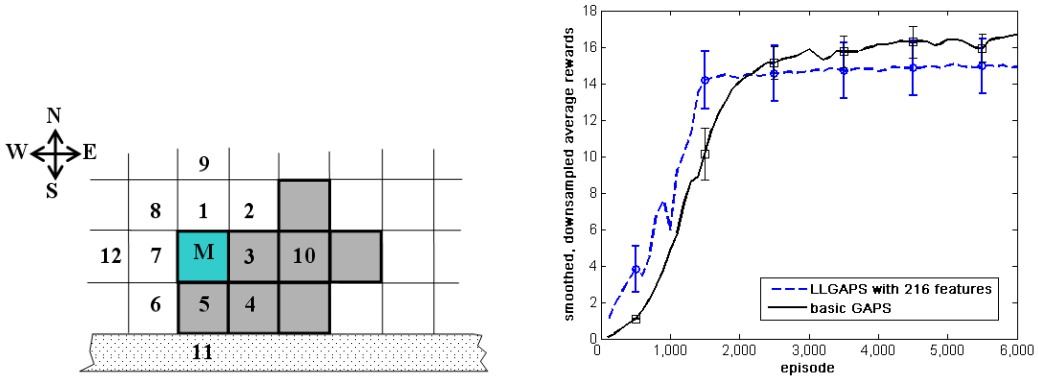
(a)

(b)

Figure 8: Screenshot sequence of 15 modules executing (a) the best policy found by GAPS, and (b) the hand-designed policy for eastward locomotion. Robots composed of a larger number of modules behave in a similar way, subject to a potentially longer transition phase from the compact initial configuration to the thread configuration for the learned policy.

indicated that for 6 modules learning to locomote, LLGAPS converged significantly faster than GAPS to the same good policies. This was due to a poor selection of parameters for the basic GAPS algorithm. With the current settings, we see in figure 7b that 6 modules learn equally well with equal amounts of experience, either with the basic GAPS or the feature-based algorithm. However, increasing the number of modules reveals that LLGAPS is more prone to finding unacceptable local minima, as explained in section 4.2.5. 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[6]. Thus the original Moore neighborhood plus four additional bits of observation are available to every module, as shown in figure 9a. 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, 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 9b, 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 LLGAPS does not perform as well as the basic algorithm after both have converged, and again is more likely to find locally optimal policies that are not locomotion gaits. We conclude that 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.

[6]Again, 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.



(a) (b)

Figure 9: (a) During experiments with an extended observation, modules had access to 12 bits of observation as shown here. (b) Smoothed, downsampled average rewards, with standard error, obtained by 6 modules over 10 trials: comparison between basic GAPS and LLGAPS.

4.2.4 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 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. In fact, figure 10 shows that there is a marked improvement, especially as the number of modules grows.

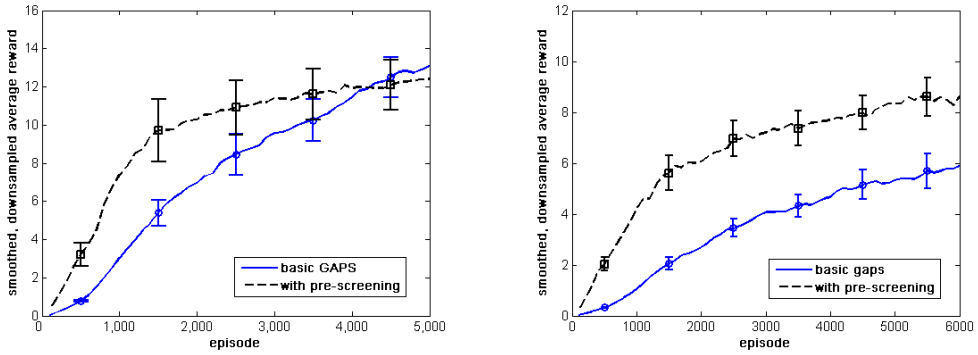
4.2.5 Scalability and local optima

Stochastic gradient ascent algorithms are guaranteed to converge to a local maximum in policy space. However, those maxima may represent globally suboptimal policies. Figure 11 shows execution snapshots of two 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 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 11: Two local optima resulting in modules stuck in a bad configuration from which they will not move.

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. The feature-based LLGAPS is even more prone to fall for local optima, as shown in table 2. This ef-



(a) (b)

Figure 10: Smoothed, downsampled average rewards, with standard error, over 10 trial runs of modules running the basic GAPS algorithm versus GAPS with pre-screening for actions resulting in legal moves: (a) 15 modules, and (b) 20 modules.

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

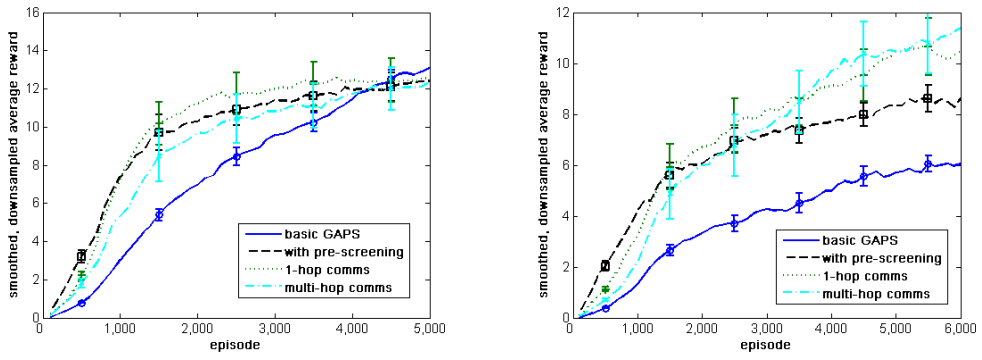
Table 2: In each of 10 learning trials, the learning was stopped after 10,000 episodes and the resulting policy was tested 10 times. The table shows the mean number of times, with standard error, during these 10 test runs, that modules became stuck in some configuration due to a locally optimal policy.

fect 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.

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.

4.2.6 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



(a) (b)

Figure 12: Smoothed, downsampled average rewards, with standard error, 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: (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).

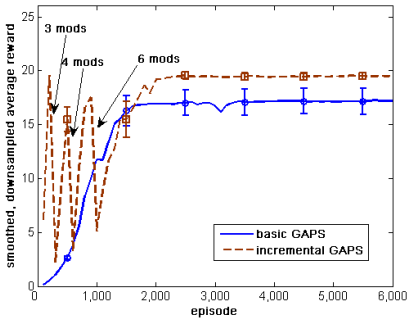
another module underneath. If a module's bit is set it is not allowed to move at this timestep. We thus create more time in which other modules may move into the empty space underneath to fill in the base. Note that although we have increased the number of possible observations by a factor of 2 by introducing 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). Experimentally (see table 2) we find that it is not enough to ask just one neighbor. While the addition of a single request halved the number of stuck configurations in a hundred test trials for 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 12a shows the learning curves for both sets of experiments with 15 modules practically undistinguishable.

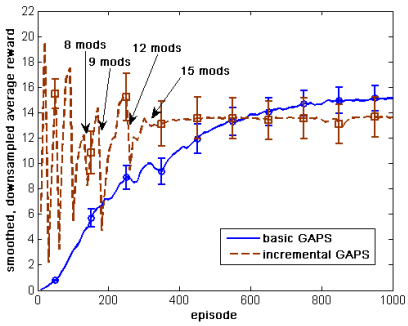
However, the average obtained rewards should be lower for algorithms that consistently produce more suboptimal policies. This distinction is more visible when the modular system is scaled up. When 20 modules run the four proposed versions of gradient ascent, there is a clear distinction in average obtained reward, with the highest value achieved by policies resulting from multi-hop communications (see figure 12b). 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.

4.2.7 Learning from better starting points

Figure 13 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. Nev-



(a)



(b)

Figure 13: 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.

ertheless, we see in table 3 that only 2.1 out of 10 test runs on average produce arm-like protrusions when the algorithm was seeded in the 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

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

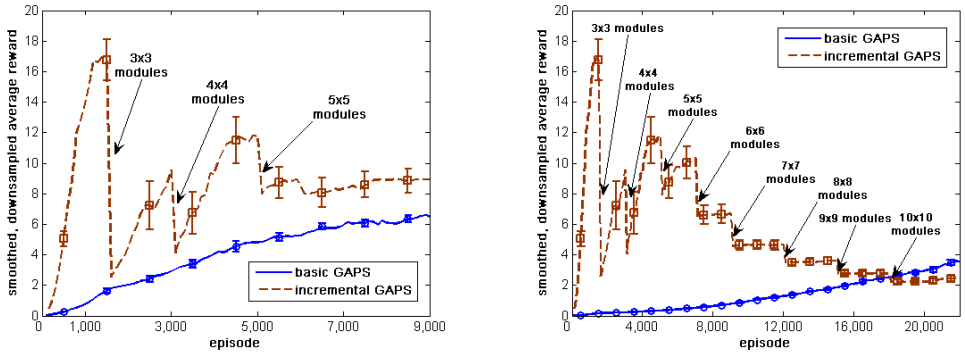
Table 3: Mean number, with standard error, of dysfunctional configurations, out of 10 test trials for 10 learned policies, after 15 and 20 modules learning with basic vs. incremental GAPS.

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 14 and table 4. 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.

	GAPS	IGAPS
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: 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.

Figure 15 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) (b)

Figure 14: 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).

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).

Statistical analysis reveals that mean rewards obtained by GAPS with or without partial policy knowledge differ significantly 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 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.

4.3 Learning from individual experience

In all of the experiments reported in section 4.2 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 rewards and 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 expect that it will take proportionately more time for individual modules to find good policies on their own. In addition,

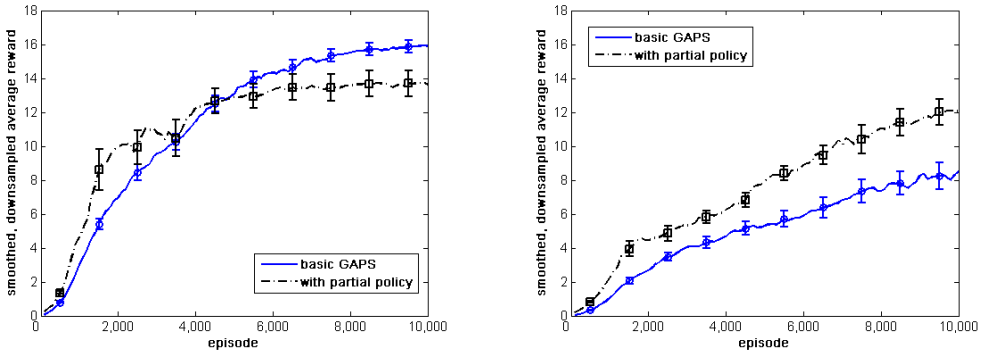


Figure 15: Smoothed, downsampled average rewards, with standard error, over 10 trials: effect of introducing a partially known policy for (a) 15 modules and (b) 20 modules.

we would like to limit each module i's knowledge of the reward function to its own displacement $R_i = \Delta x_i$ in T timesteps (in the case of centralized learning, each module would receive a reward that is the mean of individual displacements: $R = \frac{1}{N}\sum_{i=1}^{N} R_i$). This limits communication requirements among modules, but presents an additional difficulty. In the experiments that follow, each module is now optimizing its own displacement along the horizontal line, using its own experience only, and the equivalence theorem no longer holds.

We describe the results of experiments on modules learning individual policies without sharing observation, reward 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. We report smoothed (100-point moving window), downsampled average rewards for the sake of clarity.

We see in figure 16 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

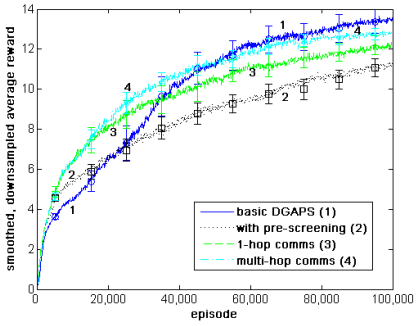
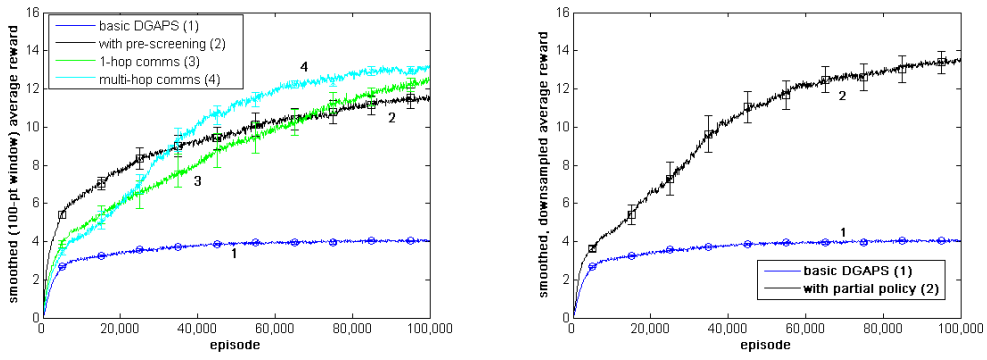


Figure 17: Smoothed downsampled averaged reward, with standard error, over 10 trials of 15 modules running algorithms seeded with a partial policy, with and without pre-screening for legal actions and extra communicated observations.

by initializing the optimization with a partially specified policy. The same two "rules" were used in these experiments as in section 4.2.7. Figure 17 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, and 3) GAPS with legal actions and an extra observation bit com-



(a) (b)

Figure 16: Smoothed, downsampled average rewards, with standard error, over 10 trials with 15 modules learning to locomote using only their own experience: (a) original distributed GAPS vs. GAPS with pre-screening for legal motions and 1-hop communications with neighbors below, and (b) original distributed GAPS vs. GAPS with pre-screening for legal motions and a partially specified policy as starting point.

municated by the one-hop and the multi-hop protocols. After a while, the curves in 1) and 3) 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.

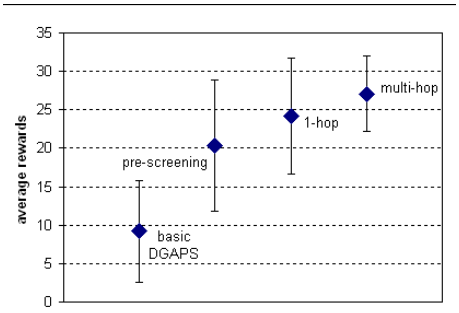


Figure 18: Average rewards achieved by 15 modules after 100,000 episodes of distributed GAPS learning seeded with partially known policy, with various degrees of constraints and information, with standard deviation bars.

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 18 shows that multi-hop communications are still necessary to reduce the chances of getting stuck in a bad local optimum. In addition, in the third column of table 5, which reports the incidence rates of unacceptable local optima in sets of experiments with 15 and 20 modules, we see that increasing the learning episode length to 100 time steps helps disambiguate between the performance of the algoirthm in different conditions. However, increasing the episode length during learning will never alone eliminate arm-like configurations, since they occur due to a short sequence of suboptimal local decisions, which can be made at any point during the episode, including early on.

5 Discussion

The experimental results of section 4 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

	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 restrictions/info	1.2±0.4	10±0.0	9.9±0.1
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 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.

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 kind 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 3.3 and 4.2.4) or any action other than NOP in special cases (section 4.2.6). On the other hand, we initialize the algorithm at a better starting point by an incremental addition of modules (sections 3.4.1 and 4.2.7) or by a partially pre-specified policy (section 4.2.7). We have also explored compact representations through feature spaces as a means of describing fewer policies and thereby reducing the search problem (sections 3.2 and 4.2.3). 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. We saw GAPS and LLGAPS both likely to converge to globally suboptimal policies based on this representation. Restricting exploration to legal motions only 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 communication 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. 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 4.3). 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 can-

not afford to run a physical system for 100,000 episodes of 50 actions per module each. 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. 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.

Another concern, when it comes to potential portability of our results and the learning approach to physical robots, is that in this paper, we have employed a very simple model of the robot and its environment: an abstract kinematic model in 2D. In the future, we hope to extend our experiments to more realistic simulators that take into account the third dimension and the robot's actuation capabilities. We are also investigating the extent to which our results might apply to non-synchronized systmes. All of the results reported here have assumed a turn-taking execution of actions within each timestep of the episode. If this assumption is relaxed, we expect the controllers to break down. However, preliminary results in a scenario where modules executed their actions in an entirely random order over the length of the episode (one module can go several times before another one gets a chance) show that the best learned gaits may be somewhat resilient in this respect: for 15 modules having learned locomotion gaits with basic centralized GAPS, testing with turn-taking execution produced an average of 0.1 ± 0.1 unacceptable non-gait configurations out of 10 trials, while testing with random-order execution produced only a slight increase to an average of 1.9 ± 0.3 out of 10 such configurations. Otherwise, the gaits look similar to those obtained with turn-taking execution, only making slower progress, and thus receiving less reward on average, due to the random ordering of actions. At the same time, when learned policies are likely to be stuck in unacceptable local minima, this tendency will increase with the introduction of random ordering: for 20 modules, where testing in the original, turn-taking condition produced an average of 5 ± 1.7 non-gait configurations out of 10 trials, testing with random ordering resulted in an average of 9.3 ± 0.1 out of 10. Extending our approach further to fully asynchronous execution is another promising direction for future research.

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. We also have not tested enough different smarter starting points to be able to say what kind of rule it is good to pre-specify.

As we explore collaboration between the human designer and the automated learning agent, our experiments have brought 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. Experiments have shown that a simple performance measure such as displacement during a time period cannot always disambiguate between good locomotion gaits and simply shifting the center of mass into an arm-like protrusion. We are currently working to address these issues.

In addition, we would like to extend our work using the idea of coordination between modules that goes beyond single-bit exchanges, inspired by recent work in the RL community, such as coordination graphs (Guestrin et al. 2002, Kok & Vlassis 2006).

6 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.

6.1 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 in this paper were previously formulated in less detail (Varshavskaya et al. 2004, Varshavskaya et al. 2006). All the experimental results are new and have not been previously reported.

6.2 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 (2006) 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.

6.3 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. In robotics, multi-agent reinforcement learning has been applied to teams of robots with Q-learning as the algorithm of choice (e.g., Mataric 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.

More recently there has been much theoretical and algorithmic work in multi-agent reinforcement learning, notably in cooperative reinforcement learning through distributed value functions (Schneider et al. 1999), coordination graphs (Guestrin et al. 2002, Kok & Vlassis 2006), and individual reward estimation from a common signal using Kalman filters (Chang et al. 2004). The Robocup competition (Kitano et al. 1997) has also driven research in collaborative multi-agent reinforcement 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.

6.4 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).

7 Conclusion

In this paper we have presented the framework of reinforcement learning as it can be applied in the domain of self-reconfigurable modular robots. We have discerned two purposes for statistical learning in our field: that of automating the writing of distributed controllers for the robots, and that of allowing the robots to adapt their controllers at run-time, if changes in the environment, task or robot's composition warrant such adaptation. We have examined the assumptions under which powerful learning techniques are guaranteed to work in the context of modular robots, and found at least one way of approaching the formal difficulties presented by our distributed domain through stochastic gradient ascent in policy space. We then developed a number of extensions to the basic GAPS algorithm aimed at constraining the problem and thus reducing the amount of learning experience required by the algorithms. We have finally presented, and discussed, empirical evidence for our claims, based on experiments in a simulated kinematic lattice-based robot.

Acknowledgements

The authors gratefully acknowledge the support of The Boeing Company.

References

Andre, D. & Russell, S. (2000), Programmable reinforcement learning agents, *in* T. K. Leen, T. G. Dietterich & V. Tresp, eds, 'Advances in Neural Information Processing Systems', MIT Press, Cambridge, MA.

Bagnell, J. A., Kakade, S., Ng, A. Y. & Schneider, J. (2004), Policy search by dynamic programming, *in* S. Thrun, L. K. Saul & B. Scholkopf, eds, 'Advances in Neural Information Processing Systems', Vol. 16, MIT Press, Cambridge, MA.

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.

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

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* S. Thrun, L. K. Saul & B. Scholkopf, eds, 'Advances in Neural Information Processing Systems', Vol. 16, MIT Press, Cambridge, MA.

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 the RSS Workshop on Self-Reconfigurable Modular Robotics', Philadelphia, PA.

Grudic, G., Kumar, V. & Ungar, L. H. (2003), Using policy gradient reinforcement learning on autonomous robot controllers, *in* 'International Conference on Intelligent Robots and Systems', Las Vegas NV.

Guestrin, C., Koller, D. & Parr, R. (2002), Multiagent planning with factored MDPs, *in* T. G. Dietterich, S. Becker & Z. Ghahramani, eds, 'Advances in Neural Information Processing Systems', Vol. 14, MIT Press, Cambridge, MA.

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* 'International Conference on Intelligent Robots and Systems', Sendai, Japan.

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. (2005), Efficient locomotion for a self-reconfiguring robot, *in* 'International Confernece on Robotics and Automation', Barcelona, Spain.

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.

Martin, M. (2004), The essential dynamics algorithm: Fast policy search in continuous worlds, Vision And Modeling Technical Report 582, MIT Media Laboratory, Cambridge, MA.

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

Mytilinaios, E., Marcus, D., Desnoyer, M. & Lipson, H. (2004), Designed and evolved blueprints for physical self-replicating machines, *in* 'Ninth International Conference on Artificial Life (ALIFE IX)', Boston MA.

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* 'International Symposium on Experimental Robotics (ISER)', Singapore.

Ng, A. Y. & Jordan, M. (2000), PEGASUS: A policy search method for large mdps and pomdps, *in* 'International Conference on Uncertainty in AI (UAI)', Stanford, CA.

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

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

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

Shoham, Y., Powers, R. & Grenager, T. (2003), Multi-agent reinforcement learning: a critical survey, Technical report, Stanford University, Palo Alto, CA.

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

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, Cambridge, MA, pp. 1038–1044.

Sutton, R. S. & Barto, A. G. (1998), *Reinforcement Learning*, a Bradford Book, the MIT Press, Cambridge MA.

Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. (2000), Policy gradient methods for reinforcement learning with function approximation, *in* T. K. Leen, T. G. Dietterich & V. Tresp, eds, 'Advances in Neural Information Processing Systems 12', MIT Press, Cambridge, MA.

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

Varshavskaya, P., Kaelbling, L. P. & Rus, D. (2004), Distributed learning for modular robots, *in* 'International Conference on Intelligent Robots and Systems', Sendai, Japan.

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.

Watkins, C. J. C. H. & Dayan, P. (1992), 'Q-learning', *Machine Learning* **8**(3-4), 279–292.

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, 'Proceedings of the Intl. Workshop on Distributed Autonomous Robotic Systems (DARS)', Vol. 5, Springer, pp. 455–464.

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