

ARTIFICIAL GOVERNMENT: META-LEARNING FOR
MULTI-AGENT COORDINATION AND CONTROL

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Abraham Bagherjeiran

May 2005

ARTIFICIAL GOVERNMENT: META-LEARNING FOR
MULTI-AGENT COORDINATION AND CONTROL

Abraham Bagherjeiran

APPROVED:

Ricardo Vilalta
Dept. of Computer Science

Christoph F. Eick
Dept. of Computer Science

Venkat Subramaniam
Industry

Dean, College of Natural Sciences and Mathematics

ARTIFICIAL GOVERNMENT: META-LEARNING FOR
MULTI-AGENT COORDINATION AND CONTROL

An Abstract of a Thesis
Presented to
the Faculty of the Department of Computer Science
University of Houston

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Abraham Bagherjeiran
May 2005

Abstract

Artificial Government uses meta-learning and unsupervised learning to coordinate agents in heterogeneous multi-agent systems. It acts as a mediator between two classes of agent: producers and consumers. Producers produce solutions to problems in some environment. Consumers can reuse these solutions for their own purposes. Artificial Government proposes a new class of agent: the mediator. It uses feedback from the consumers to direct the producers toward solutions that will better satisfy the consumers. This thesis describes the components of Artificial Government and an experiment that demonstrates its effectiveness in a multi-agent reinforcement learning domain. The experiment shows that the producers can more effectively satisfy the consumers with the presence of the mediator than without it.

Contents

1	Introduction	1
1.1	Machine Learning	1
1.1.1	Learning Tasks	3
1.1.2	Learning Algorithms	4
1.1.2.1	Supervised Learning	5
1.1.2.2	Unsupervised Learning	5
1.1.2.3	Reinforcement	7
1.1.3	Meta-Learning	8
1.2	Relation to Artificial Government	9
1.3	Preview	10
2	Related Work	11
2.1	Introduction	11
2.2	Blackboard Systems	13
2.3	Reinforcement Learning	14
2.3.1	Flat Reinforcement Learning	15
2.3.2	Hierarchical Reinforcement Learning	18
2.3.3	Multi-Agent Reinforcement Learning	20
2.4	Learning Classifier Systems	22
2.4.1	Overview	22
2.4.2	Extensions	23
2.5	Meta-Learning	25
2.6	Multi-Agent Coordination and Control	26
2.6.1	Combining Models	26
2.6.2	Interaction Mechanisms	27
2.6.3	Behavior Specifications	27
2.6.4	Meta-Knowledge	28

3	Artificial Government	30
3.1	Introduction	30
	3.1.1 Motivation	30
	3.1.2 Agents	32
3.2	Components	34
	3.2.1 Model-Policy Space	35
	3.2.2 Producers	37
	3.2.3 Consumers	39
	3.2.4 Mediator	41
	3.2.4.1 Coordination	41
	3.2.4.2 Direct Control	43
3.3	Adaptive Clustering	46
3.4	Relation to Other Work	50
4	Experiments	53
4.1	Description	53
4.2	Producer	53
	4.2.1 Pole-Balancing Environment	53
	4.2.2 Initial Settings	54
	4.2.3 MP Space	56
4.3	Consumer	58
4.4	Mediator	58
4.5	Results	59
5	Conclusion	62
5.1	Limitations	63
5.2	Future Work	64
	5.2.1 Producers	64
	5.2.2 Consumers	65
	5.2.3 Mediator	65
	5.2.4 Applications	67
	5.2.4.1 Quality of Service Management	67
	5.2.4.2 Distributed Image Retrieval	68
	5.2.4.3 Commercial Applications	68
	Bibliography	70

Chapter 1

Introduction

This chapter presents a brief and informal introduction to those aspects of machine learning relevant to the reader's understanding of Artificial Government. It also provides a high-level overview of Artificial Government as an introduction to Chapter 3. The primary target of this chapter is the reader who is unfamiliar with the topics of this thesis.

1.1 Machine Learning

Much of artificial intelligence in general and machine learning in particular assume the existence of a learning agent in an environment. A learning agent is usually a program in a computer, but even people encounter problems common to a learning agent.

A learning agent exists within an environment and tries to manipulate it to reach a goal. For the purposes of this thesis, the definition of an agent follows that of the

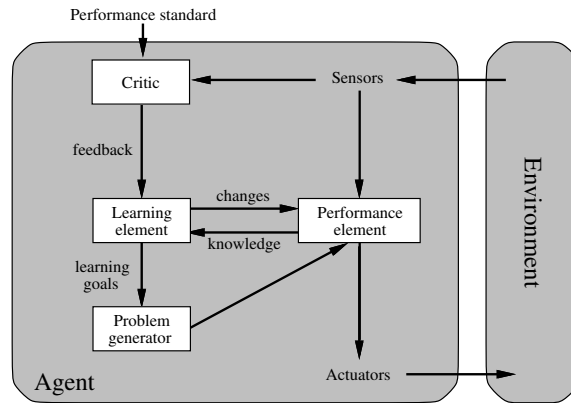


Figure 1.1: Learning agent model from *Artificial Intelligence: A Modern Approach* [?].

learning agent from *Artificial Intelligence: A Modern Approach* [?] as shown in Figure 1.1. A learning agent interacts with an environment. It perceives the environment through sensors and performs actions in the environment through actuators. The agent critiques its actions on the environment through a performance element that influences a learning element. The learning element helps the agent achieve its goals as it decides the actions it will take in the future. The agent's performance element responds to the sensors from the environment, alters the agent's knowledge, and responds to both internal and external goals placed upon the agent.

During its interactions with the environment, the learning agent can benefit from two activities: thinking and learning. Thinking or reasoning is a process that draws conclusions based on facts. Agents that can think are very popular in the field of artificial intelligence. Learning is a different process than thinking in that the learning agent must learn the facts before it can think about them. Algorithms that learn facts rather than perform sophisticated reasoning on them are the subject of

machine learning.

1.1.1 Learning Tasks

A machine learning algorithm typically consists of three components: the task, feedback, and learning algorithm. The task is the set of all the inputs that the algorithm will receive and output that the algorithm will generate. Feedback tells the algorithm when it should or should not have generated the output for the corresponding input. The learning algorithm uses the feedback to generate a hypothesis that transforms input into output for the task.

Two of the most challenging aspects of machine learning are bias and generalization. A learning algorithm generates a hypothesis of the concept it is trying to learn, but finding the right hypothesis can be difficult. Bias allows the algorithm to dismiss certain hypotheses before any input actually arrives. Usually bias is implicit in the design of the algorithm. It represents the algorithm designer's belief that certain hypotheses are better than others. Generalization is the ability to make correct decisions given input that the algorithm has not previously encountered. Together, these concepts allow the agent to effectively learn concepts that describe information it has yet to encounter.

In the example of a learning agent, bias and generalization influence how the agent learns. Often, the designer of the agent will give it some prior knowledge of the environment. One of the most popular methods of introducing bias into a learning system is to create features. Features are the properties of the environment that the agent can observe. In an environment in which an agent controls a moving

object, for example, the features may include position, velocity and acceleration. Finding the right features is key to a successful learning agent, but generalizing to new feature values is also important. For a learning algorithm to generalize, it must remember what is most useful about the information it has previously seen. Many learning algorithms form an abstract model of feature values so that when new features arrive, the agent can partially match them to previously seen information and take appropriate action. One of the agent's primary tasks is to find good models of its tasks that generalize to new information.

1.1.2 Learning Algorithms

One can categorize machine learning algorithms based on the amount of feedback they receive. Supervised learning algorithms receive the right answer. Reinforcement learning algorithms receive a reward value that hints at the right answer. Unsupervised learning algorithms receive no feedback; thus, they are totally dependent on their bias for learning.

A further categorization is the time period in which learning occurs. Batch or offline learning occurs when the algorithm receives all its input and any feedback before any real testing can occur. Online learning occurs when the algorithm can learn as input arrives and when testing occurs at the same time as learning. Continual learning algorithms are online learning algorithms that never stop learning.

1.1.2.1 Supervised Learning

Supervised learning makes the strongest assumption on the feedback. In supervised learning, the learning algorithm always has access to the right answer. The algorithm receives the action it should have performed in the environment. With supervision, one can evaluate the performance of the algorithm based on the difference between its answers and the correct answers—its error. A supervised learning algorithm finds models that minimize error with respect to unseen information.

The majority of supervised learning algorithms are batch algorithms. A subfield of machine learning called pattern classification specifically deals with problems in which the learning algorithm must differentiate between several classes of incoming data. It is difficult to envision the learning agent in batch supervised learning, but one can consider the agent being trained offline for online learning and acting.

1.1.2.2 Unsupervised Learning

Unsupervised learning relies only on the bias within the learning algorithm to learn from data; no external feedback is present. The input to the algorithm consists of a set of objects, and learning occurs in batches. Each object is a vector of values each of which is called a feature. The output can take a variety of forms such as a label indicating group membership, a set of labels, or model of the relationships among the attributes and objects.

Unsupervised learning algorithms optimize some explicit or implicit criteria. Examples include distance between representatives of groups of objects or connectedness between groups of objects. Because the algorithm is unsupervised, it simulates

supervised feedback based on patterns that the designer believes exist in the data.

Examples of unsupervised learning algorithms include clustering and rule mining algorithms. Clustering algorithms look for specific shapes in the data or data that fits pre-determined models. They produce labels to indicate where in the supposed model individual objects fit. Partitional clustering algorithms divide the objects into distinct groups and label objects according to which group they belong. These algorithms normally rely on criteria about the spatial orientation of the objects like distance to representatives. Agglomerative hierarchical clustering algorithms successively merge groups of points and return the path of groups to which objects belong. As an example of rule mining algorithms, association rule mining looks for frequent coincidences among features in the objects of data.

Applications of unsupervised learning also apply it in online learning environments. In these online contexts, a learning agent attempts to form its own models from data as they arrive. As new data arrive, the agent can change its model. Without feedback, the agent only has its ability to make sense of the data. An example of online unsupervised learning in human society is science. Scientists form theories by making observations. If new data arrive, scientists may change their theories to better explain the data. Most of modern science is empirical in that the models only change their predictions given new empirical observations.

The main limitation of unsupervised learning algorithms is that they are unsupervised. This means that they must rely on the designers' assumptions about the structure of the objects in the data. Often, however, users of unsupervised learning algorithms have some idea of the structure they expect to exist in the data, but they

cannot explicitly convey this information to the algorithm to make it a supervised learning algorithm. Solving this class of problem would provide the unsupervised learning algorithms more feedback but not enough to make it a supervised learning algorithm.

1.1.2.3 Reinforcement

Reinforcement learning algorithms require more feedback than unsupervised learning but less than supervised learning algorithms. Specifically, they operate in online contexts and require a reward signal such that the learning goal is to learn a hypothesis whose use brings the algorithm the maximum cumulative reward.

The typical reinforcement learning tasks differ from those of both supervised and unsupervised learning with the introduction of time. In other machine learning tasks, the input is often a set of objects each of which is a vector of feature values. In most variations of the reinforcement learning problem, the input is a temporally ordered sequence of states. Each state is a vector of feature values. The output is a value that is interpreted as an action for the state. The learning algorithm maps states to actions subject to the reward value.

The feedback assumes that the learning agent is embedded in an environment. The environment emits sensor signals as states and receives control input in the form of actions. Based on the state and action, the environment transitions to a new state and the learning mechanism receives a reward. Examples of reinforcement learning algorithms are further detailed in Chapter 2. Reinforcement learning agents learn policies of actions in addition to models.

1.1.3 Meta-Learning

In supervised, reinforcement, and unsupervised learning algorithms, the learning algorithm makes the assumption that it is possible to learn the concept. Specifically, the perfect hypothesis or model is one that the algorithm is able to find. This is not always the case, however. The learning agent may have bad features or a learning algorithm that is incapable of representing the perfect hypothesis. One solution is for the agent to learn from multiple tasks. Although the tasks may differ from each other, the meta-learning assumption is that they are similar enough for the agent to apply its experience from one task to another. When a learning agent has to solve related tasks, it can learn to solve new tasks using knowledge of its previously solved tasks. It may also learn that the features in one problem are useful for learning in another problem.

Meta-learning or bias learning is concerned with the problem of learning to learn. The learning algorithm learns the best set of possible hypotheses from several tasks. In the traditional or single-task learning algorithm, the algorithm starts with the same initial knowledge for every task; thus, it forgets what it learned from the last task. In meta-learning, the algorithm starts a new task with the knowledge it learned from a previous task. Although this does not completely solve the problem of poor possible hypotheses, searching through possible hypothesis families allows the learning mechanism can enhance its ability to learn in new training tasks.

The learning agent can generalize over tasks in addition to data in a particular task. If multiple agents can share their experiences in their problems, the agents that have solved similar problems in the past can more easily solve new problems

presented to the group. The learning agent learns to reuse existing models and policies in new, different problems.

1.2 Relation to Artificial Government

In short, Artificial Government (AG) applies unsupervised learning to perform meta-learning on reinforcement learning agents. Although the description of Artificial Government appears in Chapter 3, this section demonstrates how these different aspects of machine learning come together in Artificial Government.

In Artificial Government, learning agents have problems to solve. Along the way, some agents will produce solutions to smaller problems and others will try to reuse these solutions when solving more complex problems. Since all of the agents have different problems to solve, some of the solutions to smaller problems will not always work in the more complex problems. AG tries to influence producers to yield solutions that are closer to what the consumers want.

In Artificial Government, unsupervised learning algorithms group solutions to problems. The groups have two different interpretations. From the perspective of the producer, the groups are similar solutions to similar problems. The producer's position in the group shows how similar it is to others. From the perspective of the consumer, the groups are potential solutions to its own problem. The consumer looks for good solutions in the groups but would like to avoid having to search through all of them. AG must balance these different interpretations of groups to make the producer groups more closely coincide with the consumer groups. When this occurs, producers produce exactly those solutions that the consumer will be most interested

in using.

Reinforcement learning only has a small role to play in Artificial Government. Producers are assumed to be reinforcement learning agents that are embedded in an environment. Their solutions are the models and policies that maximize the cumulative reward value in their environment. In the context of AG, the problem that a reinforcement learning producer tries to solve depends on its reward function; its solution to the problem is the policy.

Artificial Government influences the reinforcement learning algorithms with meta-learning. The method is to manipulate the reward function. Through careful manipulation, AG can steer producers toward more useful solutions to problems. Following the government's advice, the producers learn to learn in such a way as to better satisfy the consumers. AG makes it more rewarding for the producers to solve parts of the consumer's problem.

1.3 Preview

This thesis describes preliminary work on the components and operation of Artificial Government. Chapter 2 describes in more detail related work from several areas of machine learning. Chapter 3 details the components and foundational models for Artificial Government. Chapter 4 presents experimental results that illustrate how Artificial Government can influence reinforcement learning agents toward new goals. Finally, Chapter 5 relates the conclusions from these experiments and describes ideas for further work.

Chapter 2

Related Work

2.1 Introduction

This chapter describes relevant work that is similar either in implementation or in spirit to Artificial Government. The organization is loosely chronological as it moves from the blackboard systems of the 70s to recent advances hierarchical reinforcement learning and multi-agent coordination architectures. Blackboard systems consolidate information from various sources onto a global knowledge base called the blackboard. Learning classifier systems learn rules that specify what to do in a situation in the environment and typically use genetic algorithms. Reinforcement learning algorithms differ from the more familiar supervised learning algorithms in that the trainer provides only a reward function for actions in the environment rather than the correct answer. Multi-agent coordination architectures draw on results from different fields of research to define the means by which autonomous agents should interact to solve

their own and any higher-level goals. The discussion in this chapter of these diverse but related fields is necessarily brief to emphasize the most important aspects. Although Artificial Government is designed for multi-agent systems, the discussion pays particular attention to the details of simple single-agent learning algorithms both for sake of comprehension and as future bases for Artificial Government agents.

A multi-agent system consists of several agents [22]. Heterogeneous systems place no constraints on the internal structure of the agents; they assume that agents are learning agents. Homogeneous systems assume that agents have the same internal structure and only differ in their environments or goals. Usually, multi-agent systems are within the same environment, but each agent may have its own goals and different components.

In a multi-agent system, two important aspects of relationships between the agents exist. Coordination describes how agents should divide the work among themselves and how they should interact to achieve their own and any higher-level goals placed upon them. Control is more specific than coordination in that it describes where, when, and how agents should place their effort onto a problem. The control problem in a system of agents that can independently learn and operate is a difficult one that should take into account both the goals that the control system must achieve and those of the individual agent.

One of the most interesting solutions to the control problem is the use of meta-knowledge. Meta-knowledge is knowledge about knowledge. In a multi-agent system, meta-knowledge involves knowledge about an agent's behavior within the system and

between the agent and its goals. The knowledge arises from outside the agent's internal structure so that an external observer can observe or derive the meta-knowledge with limited information about the internal structure of the agents.

The related work in this chapter describes several competing approaches to the problem of the learning agent and coordination of multi-agent systems. The discussion is biased toward solutions for coordination of multi-agent systems that use meta-knowledge or models of agent behavior that allow an agent to solve its goals and higher-level goals.

2.2 Blackboard Systems

Blackboard systems [24, 18] are problem solving systems reminiscent of expert systems. They use a distributed architecture to solve knowledge-intensive problems. The blackboard is the primary data structure that stores partial solutions to the problem, called entries, in different levels of abstraction. Entries describe the solutions to the problem or the process of solving the problem. Entries arrive on the blackboard from knowledge sources (KSs) that are event-driven user-implemented procedures. KSs read the entries on the blackboard, perform processing according to the domain-specific knowledge, and alter or create new entries on the blackboard. Blackboard systems place no restrictions on the internal structure of the KSs but require them to use a consistent entry format. The scheduler uses the current state of the blackboard and the conditions that specify when a KS should operate to activate KSs. The scheduler solves the problem by deciding which KSs to activate at any stage in the solution of the problem.

Blackboard systems as practical problem solving systems were popular in the late 1980s and some applications continue to appear in the literature. HEARSAY versions I, II, and III were blackboard applications to speech understanding [24, 18]. They organized the blackboard into layers for different aspects of speech such as words, phrases, syllables, and signal components. Knowledge sources used statistical and signal processing knowledge to process the audio signals. The BB-1 blackboard system [30] planned errand sequences in an artificial town scenario. Many blackboard systems incorporated the previously described components into their systems. One of the more interesting successors is the CASSANDRA architecture that distributed blackboards according to the data layers [18]. The description of CASSANDRA also included a good review of blackboard architecture and served as the primary reference for the preceding paragraph. More recent work in blackboard systems applies them to industrial production scheduling [46], patient monitoring [42], and creating belief networks for military command and control [51]. As a latest development, Craig complements his explanations of blackboard systems by updating the terminology. He equates KSs to independent agents that use a global database (blackboard) for all public communication although they may communicate among themselves [19].

2.3 Reinforcement Learning

Reinforcement learning algorithms view the world as having distinct, discernible states and actions that can occur when the environment is in a state. An agent within this environment receives a reward for its actions according to an externally defined function. The agent's task is to maximize this reward function; thus, it tries

to find the action that will lead the agent to the best reward for every possible state in the environment. It calls this mapping its policy, and the agent attempts to find the best policy. This section describes three different architectures that perform reinforcement learning in probabilistic domains.

2.3.1 Flat Reinforcement Learning

The earliest reinforcement learning (RL) algorithms were flat—they learned to execute action to maximize a reward function defined over the possible states and actions in the environment. Flat RL algorithms interpret the environment as a probabilistic model of states influenced by actions. Although learning this model is important, the most common flat RL algorithms avoid this step in favor of learning optimal actions given a state.

The general model of the environment in which agents find themselves is one of states, actions and rewards. Most flat RL agents specify this model to conform with Markov Decision Processes (MDPs). An MDP is a probabilistic model in which only the previous environmental state and action influence the current state [33]. In a non-deterministic MDP, actions in a given state result in successor states according to a probability distribution over the possible set of states [38].

Definition Let S be a set of perceivable states in the environment, A be a set of actions that an agent can execute in an environment, δ is a function mapping states and actions to a probability distribution on states $\delta : S \times A \rightarrow \Pi(S)$, and r be a function mapping states and actions to a real number $r : S \times A \rightarrow \mathbb{R}$. The 4-tuple $\langle S, A, \delta, r \rangle$ is a non-deterministic Markov Decision Process if and only

if for all time t , $s \in S$, and $a \in A$: $P(s_{t+1} | s_1, \dots, s_t, a_t) = P(s_{t+1} | s_t, a_t) = \delta(s_t, a_t)$

Agents have the ability to sense the current state of the environment as it occurs. When both the reward function and the transition function δ do not change over time, the MDP is called stationary or static. Most RL learning algorithms assume a static model.

Given the Markov Decision Process model of the environment, an agent seeks a policy to maximize the cumulative reward in the limit with respect to time. A policy $\pi : S \rightarrow A$ [38] maps the current state to the agent's expectation of the action with the highest reward. Intuitively, the agent would like to be able to know which is the best action to take in every conceivable situation in which it finds itself. This policy provides the agent with the highest possible reward for all time; it maximizes the cumulative reward equation [38] :

$$V^\pi(s) \equiv E \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

where s is the current state and r_t is the reward at t time steps in the future. The scalar constant γ is the temporal discount factor that describes the amount to which the agent discounts future rewards. The expectation is over the set of all possible states at each time step. Since the agent does not have the information necessary to exactly solve this equation prior to operation, it may make mistakes and must approximate the solution from past information. The function Q [38] allows the agent to approximate the cumulative reward given its current state and the action

it chooses to execute at the current time.

$$Q(s, a) \equiv E[r(s, a)] + \gamma \sum_{s' \in S} P(s' | s, a) \max_{a'} Q(s', a')$$

In the Q function, s is the current state, a is the current action that the agent chooses at this time when in state s , the expectation computes the expected reward for this action, s' is one of the possible future states, and $Q(s', a')$ is the agents' value of these possible outcomes. This equation defines the value of a state and action as the expected reward and the discounted future value of all successor states assuming that the agents acts optimally in the future. Optimal actions result from the agent's current Q values of states and actions. To learn this optimal policy, the Q -learning algorithm applies variations on the following update rule [33] at each time step:

$$Q'(s, a) := Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

where s and a are the current state and action, r is the last reward that the agent received, s' is a potential successor state, and γ and α are discount constants that specify the amount that future values and rewards count toward learning. This update rule eventually converges to the optimum policy assuming in the limit of infinite state accesses.

In practice, the rule does not require an infinite amount of time to converge, but it does require many iterations. It is mostly useful in practice because it does not require a well-defined model. Although the derivation of the Q learning rule and its corresponding algorithm assume the environment follows an MDP, the algorithm can work in environments that do not satisfy this assumption. Model-based RL algorithms can outperform model-free algorithms [2, 33], but often no model is available.

Also in practice, an agent often confronts the exploration-exploitation dilemma in which it is not always beneficial to take the expected best action. Sometimes, a random action can prove to be more profitable because of the agent's incomplete knowledge or the dynamics of the environment. Although this is indeed a difficult problem, it is not really relevant in the discussion of the algorithms.

Extensions to flat RL focus on improvements to the convergence of the Q learning rule and its variants. One of the most promising techniques is to use clustering. Clustering the state space with vector quantization to minimize quantization error showed improvements in the robotic soccer domain [25]. Another approach is to cluster Q values, reward values, or both to create an index over previous training. The idea is that when the environment changes, an RL agent can lookup old policies based on performance during learning [16]. The approach utilizes a distance metric on Q values and another one on reward values. Each distance metric induces a clustering on the values that the agent estimates during learning. An agglomerative clustering method creates an index of increasingly similar tasks that the agent encounters. The results of the experiment show an improvement in the learning time. Other improvements to flat RL led to the development of hierarchical reinforcement learning.

2.3.2 Hierarchical Reinforcement Learning

Although the Q learning algorithm works well on simple problems, problems can be so complex that an enumeration of the possible states and actions is too large and inefficient to store in a table. Hierarchical reinforcement learning (HRL) algorithms

combat the problem of complexity through creative decomposition of the model and policy space. The assumption is that the reward function requires the agent to perform specific behaviors at different times, and the behaviors do not change with repetition. The agent’s learning problem is simpler in these hierarchical reward environments than in the equivalent flat environment. This is because behaviors describe presumably disjoint subsets of the state space of the environment. Since behaviors transition to other behaviors, the modeled state space is significantly smaller than the equivalent flat space.

The two most popular hierarchical reinforcement learning algorithms, MAXQ [21] and HAM [43] are similar in their decomposition of the environment but differ in the names of terms. This description combines the two methods to provide a general overview to how HRL can occur. The user divides the state and action space into partitions which are typically meaningful to the user. An example is a sequence of navigation actions. The partitions contain states and actions subject to flat reinforcement learning, and the user can form hierarchies of these partitions. Both the algorithms [21, 43] apply a variant of the Q learning rule to aggregated states. The state at the top level of a hierarchy is the activation of one of the lower levels, and the state of the lowest level is real environmental state. The Q value at each level derives from the reward of activation of each lower level, and at the lowest level it is the reward for executing actions in the environment. As the HRL agent interacts with its environment, the action sequences that are most rewarding have highest value.

Since its inception, HRL has become popular and inspired many extensions. The

FYNESSE architecture [47] incorporates prior knowledge into an RL algorithm. It utilizes predefined fuzzy control rules and learns to control the activation of the rules for the given environmental state. Although it is not hierarchical in structure, it similarly decomposes the learning problem. Other work in HRL uses partially observable Markov decision process (POMDP) environment models and their hierarchical counterparts as HRL algorithms to combat some of the difficulties in the original algorithms. Some recent work uses data-mining techniques to learn the structure of a hierarchical POMDP. In the intelligent home domain, Youngblood [63] mines a sequence of state transitions to form frequent transitions for a hierarchical hidden Markov model (HMM) without actions. The action policy arises from a reward structure on an automatically generated POMDP from the corresponding HMM. Bakker and Schmidhuber [4] employ clustering on the observation to discover the hierarchy in reinforcement learning. Clustering observations to form aggregated states is the first step. These aggregated states form subgoals in the hierarchy. The last step is to associate learned policies from action to the subgoals they best serve. The association allows the algorithm to discover its own goal hierarchy and the policies to achieve it.

2.3.3 Multi-Agent Reinforcement Learning

Hierarchical reinforcement learning algorithms focus on which pieces of a global state to activate at any given time. Implicit in the approach are the assumptions of the continued presence of the pieces and the complete access to their internal structure. Within a single agent, these assumptions are completely justified. In a multi-agent

environment, however, agents may use different internal structures or must operate without the direct intervention of higher levels of the hierarchy. Multi-agent reinforcement learning (MRL) algorithms attempt to learn and coordinate the actions of distinct agents. Since most of the difficulty in MRL lies in the coordination of multiple agents, most of the solutions attempt to find ways to manage the individual agent's solutions in the presence of multiple agents. Most of the MRL algorithms assume that the agents create policies for Markov decision processes, so they attempt to partition the state space of the environment across the agents.

Recent work shows that one can use multiple agents to learn policies for large flat state spaces more efficiently than with flat RL algorithms. The decentralized Markov decision process [9, 10] MRL paradigm assumes that agents within the same environment can form mostly independent models and control policies. When transitions between states cross partitions, the models are no longer independent. A decentralized RL algorithm can operate under both these conditions provided that transitions are of a particular format. Agents periodically interface with a base or higher-level agent and send their models to this agent. The higher-level agent then attempts to partially learn optimal policies for several agents and adapt the individual agent's reward function to reflect the actions of the other agents. Under this manipulation of rewards, agents can operate apart from other agents and the higher-level agent and still attain global goals. The results show that these algorithms converge to the optimal joint policy than an equivalent flat learning algorithm across the entire state space. Another approach [26] to learning policies in large spaces is to use agents to learn pieces of the optimal policy and then combine them into larger solutions

without too much effort. It learns multi-agent MDPs where each agent has the same model but different reward functions. The joint reward function is the sum of the individual reward function. Because of this knowledge, each agent can learn joint agents without explicit access to the other agents.

Other approaches to MRL rely more on the interactions between agents rather than their internal structure. Sun uses [50] a heuristic related to learning error to determine when an agent cannot successfully learn a subspace of the policy space. When the error is small enough, his algorithm partitions the space by splitting the domain along an attribute values and creates new agents. The coordination mechanism must track the partition and error of each agent within its control. It does not allow for merging partitions or editing them after creation. Guestrin [28] combines the decentralized MDP and partitioning approach with predetermined coordination graphs over the agents. Agents can then determine which agents affect their reward and combine learning in those agents' spaces rather than over the joint reward space.

2.4 Learning Classifier Systems

2.4.1 Overview

Learning classifier systems (LCS) evolve intelligent agents as collections of rules that transform environmental signals into environmental actions. This description of LCS follows a recent, clear description of the XCS [15], a popular classifier system. The agent exists in an environment that emits sensor values at each instant in time. The

sensors are typically binary-valued but recent work extends this to other representations [60]. Environmental signals may depend on previous values to indicate a multi-step problem or are independent in the case of a single-step problem. After processing the sensor values, the agent can take one of a fixed set of actions in the environment and will receive a reward for its effort. Actions depend on classifiers that match part of the sensor values in the environment and recommend an action. Classifiers contain condition, action, and prediction values. The condition values are sensor values with don't care symbols that match any value. The action is one of the fixed set of action values that the agent can take in the environment. The prediction estimates the value of this classifier with respect to the reward it will receive. The agent creates the set of all classifiers that match the environment's sensor reading. Each of the classifiers in this match set recommend an action to take. The action with the largest reward value will be the action the agent takes, and the classifiers that recommended this action will form the action set. After the agent takes the action, the reward will affect only the classifiers in the action set. At predetermined times during execution, the agent will apply a genetic algorithm to the classifiers to form new classifiers based on the principle of survival of the fittest. In addition to the genetic algorithm, the agent will manage the classifiers based on their accuracy in predicting rewards using an algorithm that is similar to the Q -learning algorithm.

2.4.2 Extensions

The more than twenty-five years of research in classifier systems has seen many incarnations of LCS and extensions to the original ideas. Originally, LCS were new

applications for genetic algorithms [11]. Holland's version of LCS consists of production rules that manipulate an internal message list and only sometimes resulted in actions. The ZCS classifier system uses links among rules to form rule clusters or corporations to increase learning [54]. The XCS system that is the basis of the previous description of classifier systems shifted the learning mechanism from total reliance on genetic algorithms to a combination of genetics and reinforcement learning [6, 15, 14, 59]. XCS revitalized the field and sparked other improvements. The Anticipatory Classifier System (ACS) [12, 13, 35, 49] brought a more explicit modeling concept to LCS as its classifiers match sensor values from the environment and also predicted the next sensor values. ACS also further reduced the influence of genetic algorithms on the learning mechanism. With ACS as the latest version, LCS's have gone from genetic algorithm applications to learning systems with only a small reliance on genetic algorithms.

Research in learning classifier systems (LCS) brings together ideas from the blackboard systems, reinforcement learning, and multi-agent systems fields although researchers only recently established these connections. Early on, authors acknowledged the connection between LCS and blackboard systems [11]. Classifiers act as knowledge sources and the message list from the environment acts as a blackboard. Unlike blackboard systems, researchers made the connection between reinforcement learning and classifier systems only later [34]. Without the use of genetic algorithms, a simple classifier system learns the same concepts as the table-based Q -learning algorithm [23]. The LCS idea has, however, inspired reinforcement learning architectures such as the economics-based multi-agent coordination systems HAYEK [7].

Other applications to multi-agent systems focus on learning rules across multiple agents and combining disparate knowledge by genetic algorithms [52].

2.5 Meta-Learning

Meta-learning research focuses on how a learning algorithm can use information it learns in a single training task on new, unseen tasks. The goal of much of the meta-learning or bias-learning research is to find algorithms that lead to better hypotheses across all possible learning tasks. Practical research focuses on improving existing supervised and reinforcement learning algorithms with learned knowledge and user-provided feedback.

In supervised learning algorithms, successive training on new tasks can allow an algorithm to learn from data that is not present in the current training set. The idea is to transfer knowledge that the algorithm acquires from old learning tasks to new ones to reduce expected error of the learned hypothesis across all possible learning tasks and points in the tasks [8]. Algorithms that perform feature selection for neural networks [8] and use distance function manipulation for instance-based classifiers [53] have shown promise.

Meta-learning also applies for reinforcement learning. Most techniques utilize external feedback throughout the lifetime of the algorithm to improve its performance [37].

In general, the most interesting approaches to meta-learning try to capture the degree to which tasks are related and exploit that in future learning. Artificial Government exploits the relatedness between tasks and their solutions from different

perspectives to meet external objectives.

2.6 Multi-Agent Coordination and Control

In a multi-agent system (MAS), agents pursue their own goals. During their pursuits, agents will often encounter and interact with other agents. Solutions to the coordination and control problem in MASs start from simple task division to learning agent behaviors and exploiting the knowledge for control. Artificial Government occupies the latter half of this spectrum.

2.6.1 Combining Models

Combining agents' models to form aggregate models facilitates global problem solving and improves learning time for portions of the model. This is easy to do when agents share the environment or the modeling language. With Bayesian networks, an aggregate model consists of the average of the individual models for all variables. Aggregated models assist in multi-agent coordination by essentially reading all the agent's models [5]. Other work with Markov models, as previously described [28], assumes that the agents share a model but have different reward functions. An agent can then learn its own policy and make some assumptions about the other agents. Combining models improves solutions to large problems, but does not provide a good way to control agents.

2.6.2 Interaction Mechanisms

Coordination by interaction mechanisms provide explicit algorithms and requirements that agents should follow when they interact with other agents. A useful interaction mechanism, particularly in human society, is the economic market. One of the earliest agent coordination mechanisms in the Artificial Intelligence domain was the WALRAS algorithm [57, 58]. The algorithm itself simply solved the market equilibrium problem—compute the price of a good that equates supply and demand. The authors of this algorithm applied it to scheduling where agents bid on schedules based on their estimation of the schedule’s value. The negotiation process resulted in a schedule that satisfied the most agents. Further applications of market theory to agents yield a resource allocation algorithm for power plants [62] and multi-robot environments [20]. Unfortunately, these authors noted that the market itself is no substitute for the individual agent’s ability to reason about its situation.

2.6.3 Behavior Specifications

Much of the work in MAS coordination and control revolves around explicit specifications of behavior that agents should follow. Formal languages allow agents or higher-level controllers to reason about agent actions and facilitate negotiations. Informal languages provide restricted reasoning but allow for control and coordination without the constrained expressiveness of formal languages.

Formal languages are popular in the MAS literature. The Z language [22] is particularly popular as a formal specification language. A particular example of its use

is in the description of normal agent behavior [61]. A user describes the expected behavior of an agent, and the consequences when an agent is in violation of its mandate. An extension to the idea of normative behavior involves the detection and correction of discrepancies between normal and current behavior [31]. A reasoning engine can alter the agents when it detects an error in the current behavioral specification.

Although formal languages allow for reasoning, informal languages are easier to write and can be more expressive. The MONAD architecture [56] defines a programming language for behavior and coordination of agents. Implementation of the program requires a hierarchical structure to coordinate the execution state across the agents. The architecture distributes work across agents and provides coordination among them. A related architecture distributes work according to predefined roles [41]. Agents take on a role to complete a task. Within this role, they learn to perform the task that the role defines. The coordination mechanism iteratively updates the best role allocation to the agents as they interact with the environment. The intuitive idea is that roles better apply to some agents that exist within a particular environment over others.

2.6.4 Meta-Knowledge

Formal specifications place a significant burden on the user to specify the behaviors. Coordination architectures that use meta-knowledge extract their knowledge from agents as they operate within the environment and leverage this knowledge for control or coordination. This area of multi-agent coordination is most relevant to Artificial Government. Jung uses a Markov team decision process to model agents in a

distributed constraint satisfaction problem [32]. The agents do not learn the models; they are useful in helping the agent decide which coordination strategy to choose for the problem. An earlier but similar work [36] controls processes with knowledge in the form of control laws to which the controller compares system variables. Based on knowledge of the control laws and higher-order derivatives of the system variables, the controller uses reasoning to find the best control strategy. Control strategies require planning in the space of possible strategies. Hierarchical planning algorithms benefit from task decomposition, but can sometimes require rediscovery of old plans when the hierarchy resembles a tree rather than a graph. Cox [17] describes a synergy algorithm to extract common portions of a plan to facilitate their reuse. With the same goal of more efficient global work, the INFFRA social reasoning architecture [44, 45] observes agent behavior, derives frames [27] to describe the behavior, and uses the frames to encourage further similar behavior among the agents. Frames attempt to capture similarity between agents' operation, purpose, relationships, and history. At a higher level in the hierarchy, the INFFRA architecture observes the frames of the agent and determines whether the agent continue to operate within the frame and whether the current frame is appropriate for the agent's goals and actions. The architecture alters the current frame to better suit it to its vision of the agent with respect to the agent's private goals. Although similar in intention, other work describes role-learning agents that autonomously subdivide a problem by learning to follow organizational rules within a given domain [40].

Chapter 3

Artificial Government

3.1 Introduction

Artificial Government (AG) helps problem-solving agents learn to reuse partial solutions to problems. Producer agents create solutions to problems in the form of models of their environment and policies for action given the models. Consumer agents build on the solutions of the producers to solve different problems. In between these two classes of agent, a mediator defines groups of producers' solutions and applies meta-learning to guide the producers toward solutions that better satisfy the consumers.

3.1.1 Motivation

In a system of life-long learning agents, each agent will solve different problems. During their lifetimes, two distinct agent groups will emerge. Of these groups, as can often be found in human societies, some agents can provide what other agents

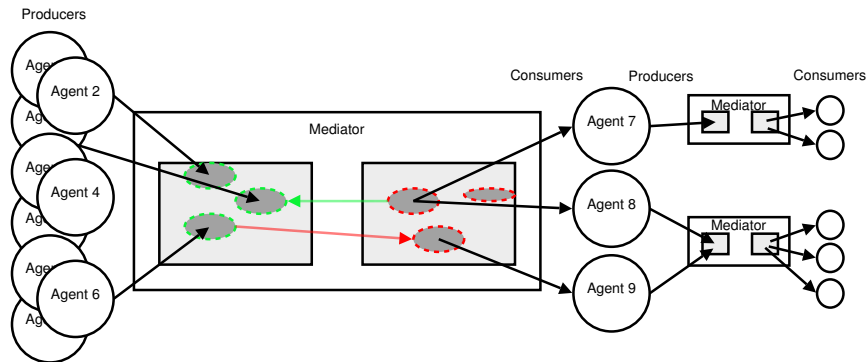


Figure 3.1: Illustrates the idea behind Artificial Government.

want to use. Humans learned to exchange solutions by creating economies wherein producers supply a particular good and consumer trade to acquire the good. Artificial Government allows agents to accomplish the same goal: exchanging solutions to problems for mutual benefit.

In AG, producer agents directly try to solve the problem. Consumer agents use the solutions to this problem as a partial solution to a different problem. For the consumers to efficiently solve their presumably more complex problem, they need to reuse rather than rediscover previous work on similar problems. Although this need is obvious, most of the literature on this topic overlook the important processes of locating and evaluating agents' previous work. Based on the assumption that similar problems have similar solutions, Artificial Government is a meta-learning architecture that learns how to help heterogeneous intelligent agents reuse partial solutions.

Figure 3.1 illustrates the functionality of the components of AG. In this Artificial Government, the agents collaborate on three different problems. The mediator

maintains two distinct perspectives on the given problem. On the left, producer agents generate different solutions to the problem. On the right, the consumers want to take the solution and apply it to a different problem for which they are the producers. They provide feedback to indicate which of the solutions would be useful to the other problem. The mediator forms groups of the both the producers' solutions and consumers' preferences for solutions. These two solution groupings are two different perspectives on the same problem. From the producers' perspective, the groups deal with how to solve this problem, but from the consumers' perspective, they deal with how to reuse the solution. Each consumer can become a producer for another problem with a different mediator. These consumers will take their solution, modify it and provide it to other consumers. The mediator learns to control the producers' groups to make them match the consumers' groups. Making producers more responsive to consumers is the goal of Artificial Government.

3.1.2 Agents

Artificial Government coordinates and controls heterogeneous multi-agent systems by extracting knowledge about the agents' model of their environment and goals. AG coordinates the agents through the creation of problem groups and prototypes from the information that the producer agents provide.

Artificial Government makes a few assumptions about the agents and the environments in which they operate:

- Agents are autonomous.
- Agents are heterogeneous.

- Agents share a modeling language.
- Similar problems have similar solutions.

The first assumption implies that agents can function without participating in AG, but they may find participation useful. The influence and benefit of AG arises when goals apply to the entire multi-agent system and not to individual agents. The second assumption means that AG places no restrictions on the internal algorithms or capabilities of the agents. Third, AG places some restrictions on the agents' model of their environment because it primarily uses these models for coordination and control. Agents can derive the models by any algorithm, but the model must be interchangeable with other models. Finally, the intuitive belief that similar problems have similar solutions underlies much of work in evolutionary computing and the use of approximations to estimate solutions to problems. To determine whether or not these assumptions are useful in practice is one of the primary goals of this thesis.

Artificial Government considers agents to take on one of three roles: producers, consumers, and mediators. Producers are problem-solving agents that learn models of an environment and policies for action given the models. Consumers reuse the models and policies in a different environment. They provide feedback to indicate how well the solution of a producer satisfies them. Mediators use the feedback from the consumers to direct the producers toward solutions that will better satisfy the consumers.

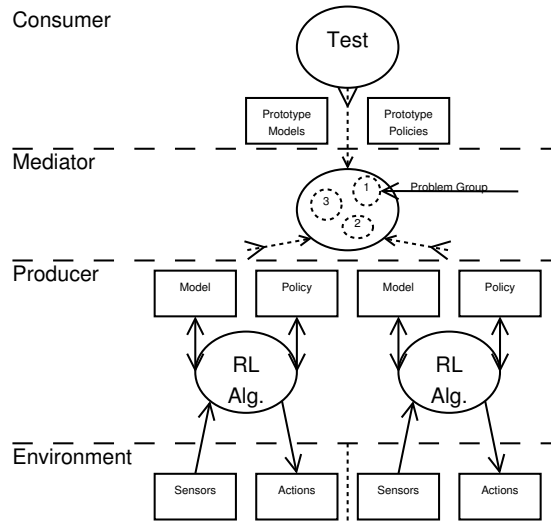


Figure 3.2: Artificial Government abstraction hierarchy.

3.2 Components

Artificial Government has a hierarchical architecture with four layers of abstraction as shown in Figure 3.2. At the bottom layer, the environment emits signals, receives control signals, and provides feedback. Producers at the next highest layer directly interact with the environment. They form models from the signals and feedback they receive. Given the models, they generate policies and use those policies to control the environment. The mediator at the next highest layer directly interacts with both consumers and producers. It receives the current models and policies from the producers and groups them together into problem groups. With knowledge of the group and their place in it, producers can actively try to achieve a better place within the group. These problem groups are sets of points in a model-policy (MP) space that are mutually desirable for most of the consumers and producers, and the

mediator uses both the problem groups and prototypes to coordinate agents' searches and to control individual agents. Control occurs when the government manipulates the reward that it provides the agents to lead them toward different MP values. Producers directly interact with the environment; mediators directly interact with producers. Consumers can indirectly interact with the environment by providing feedback to the mediator. Since AG has access to several agents, it can potentially learn to operate in a much larger environment and thus achieve a form of hierarchical learning.

3.2.1 Model-Policy Space

As producers pursue their goals they face a two-part learning problem. First, they form a model of the environment that will allow them to predict its response to their actions or inactions. The model alone is insufficient because it does not tell the producer how to act, so it also develops a policy of behavior based on the model, environmental feedback, and goals. Therefore, a producer is a life-long learner whose lifetime consists of finding models and policies that meet its continually changing goals.

In the multi-agent system of AG, producers share an environment but have different goals. Their models and policies will differ but may be similar. AG makes explicit the relationships between the models and policies of the producers it controls. To achieve multi-agent coordination, Artificial Government operates within a set of conceptual spaces. The first of these is the Model-Policy (MP) space which relates environmental models to their corresponding policies.

Definition Let m be a model of an environment E and p be a policy within that environment. The set of tuples $\{m, p\}$ defines a *model-policy (MP) space in environment E* if and only if p specifies a policy for the model m within an environment E .

Producers operate within an MP space. An environment induces a MP space in that producers can only form a model and policy of the environment in which they exist. At any given time, then, a producer occupies a single point within this space. In the rest of this discussion, the environment is assumed to be fixed for all producers and consumers. The discussion will refer to the MP space of this environment as *the MP space*. To understand how an agent can navigate this space, an additional axis is necessary.

Definition Let r be a reward value and $\{m, p\}$ be an MP point. The set of tuples $\{m, p, r\}$ defines a *model-policy-reward (MPR) space* if and only if r is the infinite discounted horizon reward that an agent can attain by following policy p under model m in an environment.

A producer's trajectory through the MP space should be toward a point of maximal cumulative reward in the infinite horizon. Because producers may have different goals, they work in different MPR spaces. Just as an environment induces an MP space, a goal or reward function in addition to an environment induces an MPR space. Neither the MP nor the MPR space examines the effect of time. In the environment, the producers can neither exactly predict the reward nor the environment's state because it is not deterministic. To an observer outside both the environment and

time, however, the environment proceeds through only one sequence of states. The MPR space is simply an enumeration of the maximal reward at the resolution of infinite time. To take individual moments of time into account, yet another axis is necessary.

Definition Let t be an instant in time and $\{m, p, r\}$ be an MPR point. The set of 4-tuples $\{m, p, r, t\}$ defines a *model-policy-reward-time (MPRT)* space if and only if r is the reward from the environment at time t that an agent receives when it follows policy p with model m in an environment.

Along the time axis, the MPRT space describes the agent’s process of learning to model and control the environment. With perfect knowledge, the trajectory follows the path of maximum reward at any point in time. The MPR space is the projection of the MPRT space onto a space without the temporal component, and the MP space is the projection of the MPR space without the reward. Similarly, the P space is the space of all policies that an agent encounters when it assumes that the model of the environment is constant. It is in this P space that most other RL techniques operate; AG controls agents in the MPRT space.

3.2.2 Producers

Producers are problem-solving agents that operate within an environment that presents them with a goal to achieve or a problem to solve. They achieve their goals by forming a model of the environment and a policy of action given the model. Producers then communicate both the model and policy to the AG so that it can track and

control their progress. Within AG, producers are problem generators; they seek both to solve their problem and to make this problem suitable for the consumers.

Although AG makes no assumptions on the internal construction of the producer, this discussion assumes that producers are reinforcement learning agents. A reinforcement learning agent exists in an environment

$$E = \{\mathcal{S}, \mathcal{A}\}$$

where \mathcal{S} is the set of all possible states that an agent can perceive about the environment and \mathcal{A} is the set of actions that the agent can perform in the environment. The set

$$S_P = (s_1, \dots, s_n)$$

is the state sequence that the producer will observe. In an interaction with the environment, the agent receives a state from the sequence and takes an action given the state. In model-based reinforcement learning, the agent learns a model of the environment

$$M(s, a, s') \rightarrow [0, 1]$$

where s and s' are the present and future states, a is an actions, and the value of the function is the transition probability. In addition to the model, the agent also learns a policy of actions

$$P : \mathcal{S} \rightarrow \mathcal{A}$$

which is simply a mapping from the state space \mathcal{S} to the action space \mathcal{A} . The policy depends on the model and the reward function R :

$$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$$

that returns a real-valued number given the policy. The input to the agent is a pair

$$\{S_P, R\}$$

where S_P is the state sequence and R is the reward function.

At any given time within this reinforcement learning context, the producer has learned one model and policy. Throughout its lifetime, the producer will search through several models and policies to find those that have the best cumulative reward value.

One can view the search process through the MP space as sampling points from a probability distribution over the MP space. This distribution is the probability that the producer can produce the point in MP space. The shape of the distribution $P_P(MP)$ is largely dependent on the producer's task. Since each producer has its own reward function, it will have a unique probability distribution. Artificial Government assumes that these MP-space probability distributions will tend to overlap for many producers.

3.2.3 Consumers

Like producers, consumers are problem-solving agents, but consumers have a different problem to solve. The solution to the producer's problem is only a part of the solution to the consumer's problem. Trying to solve a larger problem, the consumer should concentrate on the new parts of the larger problem without focusing on the parts that the producers have already solved. Partial solutions to the producer's problem can help the consumer solve its problem. Given a partial solution, the consumer

can easily tell whether the producer's solution helps. Consumers apply a test to a producer's solution and return the degree to which the solution helps.

In this discussion, the consumer has restricted set of k possible problems to solve. The consumer desires to solve k problems without performing k times the work of one producer. Without doing the work of k producers, a consumer can easily tell if a potential solution fits by performing a test

$$C(\cdot) \in [0, 1]$$

where the argument is a producer's model-policy point. A value of 1 indicates that the consumer is completely satisfied with the policy, and a value of 0 indicates that the consumer is not at all satisfied. A value between 0 and 1 indicates the consumer's degree of satisfaction with the point. The function differs from a producer's reward function because it operates on the entire policy rather than individual states.

Also like the producers, the consumers sample from a probability distribution. The distribution

$$P_C(MP) = P(C(MP) = 1)$$

is the probability that a consumer is maximally satisfied with a point in the MP space. The shape of this distribution depends on what the consumer wants to do with the point. Like the producer, each consumer will have a different distribution, and multiple consumers will tend to overlap.

3.2.4 Mediator

The mediator performs coordination and control on the producers to better satisfy the consumers. In coordination, it defines problem groups in the MP space that tell producers where they are in relation to the other producers. Prototypes within the problem groups allow consumers to easily find producers. In control, consumers provide feedback about their satisfaction with the producers' solutions. The mediator uses this feedback to guide producers toward new points within the MP space that it expects will better satisfy the consumers.

3.2.4.1 Coordination

Given the probabilistic description of the producers and consumers behavior in the MP space, the goal of the mediator is now to minimize the difference between these distributions. When the producer and consumer distributions overlap, the points most likely to be produced are precisely those points most likely to satisfy the consumers. The current version of the mediator simplifies the problem by making some assumptions on the two sets of distributions. First, the producers sample from a single distribution $P_P(MP)$, and the consumers sample from another distribution $P_C(MP)$. Second, these distributions are mixtures of normal distributions with a fixed number, k models. With this simplified model, the mediator can reuse unsupervised learning algorithms like the k -means clustering algorithm to achieve its goal.

The mediator defines problem groups among the producers and consumers. Problem groups partition the MP space, and a prototype of the group is a point within

a partition. The mediator creates these groups to summarize producers' reports of their positions within the MP space. Creation of these partitions requires a distance or similarity measure defined over the MP space.

Definition Let d be a function such that $d : MP \times MP \rightarrow \mathbb{R}$. The function d is a *model-policy distance function* if and only if for all MP points $\{M_1, P_1\}$ and $\{M_2, P_2\}$ $d(\{M_1, P_1\}, \{M_2, P_2\}) \geq 0$, is zero when the objects are highly similar and becomes larger the more they differ [29].

The distance function allows the specification of problem groups as mutually exclusive and exhaustive clusters in the MP space. A problem group has a representative called the prototype that is, for example, the centroid of the points in the group. This reliance on spatial characterizations of problems facilitates human understanding of the agents' search processes within the MP space and the government's influence of the MPR space. The following definition of the problem group uses this distance function.

Definition Let d be a model-policy distance function and O be the set of all MP points. The set $\mathbf{G} = \{G_1, \dots, G_k\}$ is a set of problem groups with prototype \bar{g}_i if and only if :

1. $o \in G_i \rightarrow \bar{g}_i = \arg \min_{\bar{g}_j} d(o, \bar{g}_j)$
2. For all groups G_i and G_j where $i \neq j$, $G_i \cap G_j = \emptyset$
3. $\bigcup_{i=1}^k G_i = O$

Each cluster has a prototype such that all the points in the group are closest to the prototype of the group. An MP point can be a member of a group because of the first

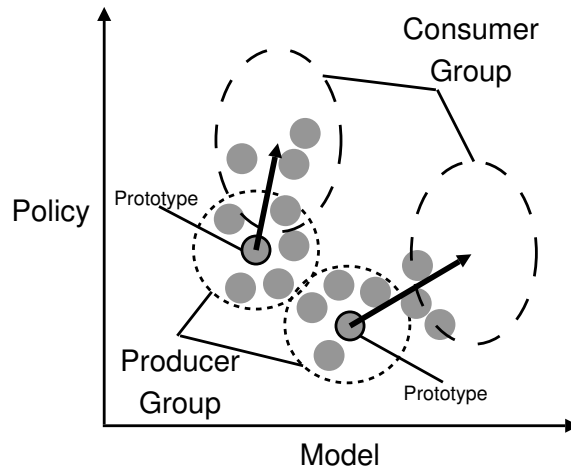


Figure 3.3: Artificial Government discovers groups of similar problems and solutions among problem-solving agents. Using consumer feedback on the prototypes, it reshapes the groups to make the producer groups better match the consumer groups.

condition of the definition. By convention, a producer is a member of the problem when its MP point is a member of that group.

The mediator maintains a set of problem groups over the MP space. From the producers' perspective, prototypes are goals to be achieved. From the consumers' perspective, they are potential solutions to another problem. An agent can view a prototype and use the distance function to estimate the amount of effort necessary to reach it. The new prototype may or may not lead to more reward for a particular producer, but it facilitates collaboration among them.

3.2.4.2 Direct Control

With the introduction of problem groups, the goal of the mediator is now to minimize the distance between the problems groups of the producers and consumers. As Figure

3.3 illustrates, the two sets of groups are very close when the producers' solutions best satisfy the consumers. To achieve this goal, the mediator performs two kinds of learning as it follows these five steps:

1. After the producers tried to solve their problems for a fixed period of time, they will send the mediator their current MP points. The mediator clusters the points to find the problem groups and prototypes.
2. Consumers perform a test on the solutions of the producers whose MP points are closest to the prototype. During the test, the consumer has access to the learned model and policy, but the producer does not do any learning. After the test, the consumer returns an indication of its satisfaction with the solution.
3. The mediator records the feedback for the prototypes. It estimates the reward that all the consumers give to the MP point

$$\hat{C}(MP) = \frac{1}{N} \sum_i^N \sum_j^n C_i(MP_j) K(MP - MP_j)$$

where N is the number of consumers, K is a Gaussian kernel function ($0 \leq K(\cdot) \leq 1$), $C_i(MP_j)$ is the evaluation of consumer C_i on point MP_j , and n is the number of MP points evaluated by consumers. Function \hat{C} estimates the number of consumers that are satisfied with any point in the MP space.

4. The mediator uses the new function estimate \hat{C} to alter the distance function weights. It employs a recent technique to learn distance function weights that can optimize a clustering given an arbitrary objective function (See Section

3.3). The objective function

$$F(G) = \sum_{\bar{g} \in G} \hat{C}(\bar{g})$$

computes the reward for the entire set of problem groups G as the sum of the estimated consumer satisfaction for each of the prototypes \bar{g} of the group.

5. The mediator sends the function estimate \hat{C} to the producers. Each producer uses this function to create a new reward function

$$R' = \alpha R + (1 - \alpha)\hat{C}(MP')$$

where R is the producer's original reward function, MP' is the producer's current MP point after its latest single step in the environment, and $\alpha \in [0, 1]$ is a predetermined weighting factor that indicates the degree to which the producer relies on the mediator's reward function.

In summary, the mediator performs two kinds of learning. First, it tries to make the producers' problem groups better match those that the consumer wants by learning a distance function from the consumers' feedback. In this kind of learning, the points do not change; only the group memberships and prototypes change. In the second kind of learning, the mediator creates a new reward function that guides the producers toward better MP points. In this kind of learning, the prototypes and group memberships do not change; the MP points change.

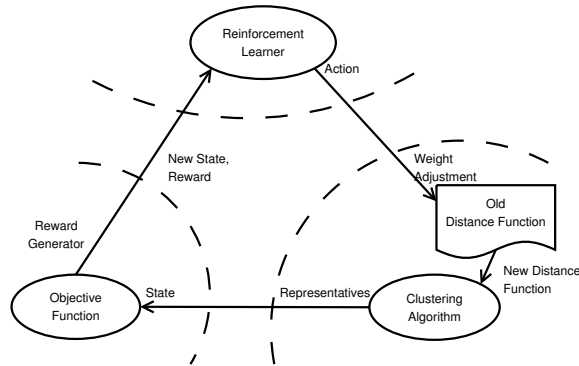


Figure 3.4: The adaptive clustering environment.

3.3 Adaptive Clustering

In Artificial Government, the mediator modifies the prototypes of the problem groups to better satisfy the consumers. With the simplifications to the MP space and the probability distributions thereof, the mediator applies a supervised clustering algorithm called adaptive clustering [3] to find the best prototypes. Adaptive clustering uses a reinforcement learning search algorithm to learn a distance function that maximizes the reward values of successive data clusterings. It applies when external feedback exists for a clustering task. It supports the reuse of clusterings by memorizing what worked well in a previous context. It explores multiple paths in a reinforcement learning environment when the goal is to find better cluster representatives based on arbitrary environmental feedback.

Adaptive clustering views clustering as a reinforcement learning problem with states, actions, and rewards in a possibly non-deterministic environment. The algorithm employs a reinforcement learning controller for a clustering algorithm whose environment is as shown in Figure 3.4. The reinforcement learning algorithm adjusts

the parameters of the clustering algorithm to produce better clusters. The result of each iteration of the clustering algorithm is a set of mutually exclusive and exhaustive clusters each of which has a representative object: e.g. the centroid. The clustering environment forms the state with these representatives. After observing the current state, the algorithm applies an action to adjust the weights of the distance metric. Given the state-action pair, the controller receives a reward. By learning weight-changing actions for the given set of representatives, the controller can retain its clustering knowledge for new instances of the same data distribution; this is particularly useful for online clustering.

The state space contains the current cluster representatives and distance function weights. Clustering algorithms usually operate in real-valued domains, but RL algorithms require distinct states. Since the use of discrete state information is more common in the literature, the cluster representatives first undergo a discretization step that divides the range of the variable into discrete intervals. Since adaptive clustering utilizes an existing partition-based clustering algorithm that relies on normalized data in the interval $[0, 1]$, the preprocessing step can effectively assume that each of the coordinates lie between 0 and 1. Given k clusters each of which is a point in a d -dimensional space with m possible values for each attribute, the number of states is $m^{d(k+1)}$; the additional 1 is for the weight vector. The length of the state vector \mathbf{s} is $d(k+1)$ and has the following form:

$$\begin{aligned} \mathbf{s} &= [\bar{c}_1, \dots, \bar{c}_k; \mathbf{w}] \\ &= [\bar{c}_{1_1}, \dots, \bar{c}_{1_d}; \dots; \bar{c}_{k_1}, \dots, \bar{c}_{k_d}; w_1, \dots, w_d] \end{aligned}$$

where $x_{i,j}$ is the j th coordinate value of the i th centroid and the optional weights.

Given a state, the learner can take one of several actions. The action either increases or decreases a single attribute weight. Given the old weight vector \mathbf{w} of length d , the action uses the following formula to generate the new weight vector \mathbf{w}' :

$$w'_i = \begin{cases} w_i \pm \Delta w_i & i = i^* \\ w_i & i \neq i^* \end{cases}$$

$$w'_i = \frac{w'_i}{\sum_l w'_l}$$

where i^* is the target attribute weight from the action, the constant $\Delta \in [.25, .5]$ is the randomly chosen percent change of the target weight, and the last equation is the renormalization of the weights. Given d attributes, the learner chooses $2d$ actions to increase or decrease an attribute indicated by either addition or subtraction of Δw_i .

Generically, the reward function must combine the influence of multiple objectives into a single scalar value. The current implementation returns the average of several component rewards, and leaves for future work the discovery of more sophisticated reward functions.

An important issue complicates the transformation of a clustering task to reinforcement learning. Reinforcement learning algorithms, in general, assume that the environment satisfies the Markov [33] assumption that the current state only depends on the immediately preceding state. Clustering, however, is an iterative process in which the representatives change almost deterministically with each iteration. At the level of the adaptive clustering algorithm, states would appear to change non-deterministically in response to different weight actions. From this level, the change in the state is simply assumed to be probabilistic—though possibly not Markovian.

In a traditional reinforcement learning environment, agents are only allowed to perform actions in the current state; that is, it is not possible for an agent to jump from one state to another state that has been visited in the past. In adaptive clustering, however, it is possible to search multiple paths in parallel and to support more sophisticated forms of exploration that perform actions on previously visited states. Therefore, our adaptive learning architecture, employs a search algorithm that operates on the top of the reinforcement controller whose main goal is to select states for further expansion.

Reinforcement learning, in general, assumes that the environment is outside the control of the agent; thus, algorithms like Q-learning use reward information to more intelligently choose which actions to perform on the current state. Our approach, extends this framework to support intelligent state selection functions in addition to selecting actions intelligently. This allows us to utilize parallel reinforcement learning agents that share value information which enables us to evaluate the reward value of states more quickly.

In our current implementation, each of the parallel reinforcement learning agents applies the Q-learning algorithm to maintaining a single table of state-action values across all the agents and maintains an open list $L = \{S_1, \dots, S_{|L|}\}$ of search states each of which consists of:

$$S_i = [\mathbf{s}, a, v]$$

where \mathbf{s} is the state vector from the environment, a is the action to execute, and v is the Q-value of the state-action pair according to value iteration. The search algorithm reads the Q-value of each state-action pair in the search state and keeps

the top $|L|$ most valuable states for execution. The existing prioritized sweeping Q-learning algorithm performs value iteration after execution of all actions in the open list. This search algorithm, then, is an example of a local beam search [?] in which the algorithm learns to determine which search states are best to expand.

The performance of the adaptive clustering algorithm largely depends on the performance of the underlying clustering algorithm. The reinforcement learning algorithm is fairly uncomplicated since states are stored in a lookup table and learning simply alters the states' values. Each action alters an array of weights. To generate the new state, however, the clustering algorithm must use the new weights to perform several iterations and generate new representatives. The reward function also uses some information about the data and can potentially be as expensive as the iteration of the clustering algorithm. In total, each step of the adaptive clustering algorithm performs a constant number of scans of the dataset. This makes the algorithm itself computationally expensive though still linear in the number of objects.

Although this algorithm was originally designed for Artificial Government, it has been successfully applied to other machine learning contexts. Returning to Artificial Government, adaptive clustering uses the coordinates of the MP prototypes as states in its reinforcement learning search. The result is a vector of weights such that with these weights, the consumer's feedback should be very high.

3.4 Relation to Other Work

AG builds upon and extends previous work on the problem of partial solution reuse. Blackboard systems [18] use a global knowledge base to store partial solutions.

Agents can utilize knowledge from the blackboard, but the blackboard itself simply stores the solutions and does not learn. Hierarchical Reinforcement Learning [21, 43, 4] algorithms decompose a problem into partial solutions and then learn when to use them, but the user must provide both the decomposition and the partial solutions. Recent multi-agent reinforcement learning algorithms [26, 10] implement a global state space in a distributed shared memory system but require uniform state space structure and other restrictions on the state space not typically available in heterogeneous systems. Market-oriented algorithms for multi-agent coordination [58, 62] use a market through which autonomous agents can exchange partial solutions, but markets rely on the one-good-one-price property which is inapplicable for solutions that are similar but not the same.

One can apply AG to a variety of fields that can benefit from the intelligent ordering of solution components. In the top-down development of software, one often must refactor a component when the specification or capabilities of a component that depends on it changes. AG learns to perform this process in simple reinforcement domains. In hierarchical reinforcement learning, the hierarchical structure of the environment is assumed [21, 43] as is the precise structure of the lower levels of the hierarchy. AG assumes the size and number of components in the levels but learns the internal structure of the components and the links among them so that individual components can operate apart from the hierarchy. In multi-agent systems, AG allows a large group of agents with limited communication ability to coordinate themselves through the exchange of partial solutions to problems and rewards. As a very simple prototype in the next chapter, a consumer seeks three slightly different solutions to

a similar problem, and AG learns to make the producers find those solutions.

Chapter 4

Experiments

4.1 Description

This chapter describes an experiment on the pole-balancing problem which has some precedence in the reinforcement learning [48, 1] and classifier systems [55] fields. In the experiment, consumers search for solutions to the problem that fit into 3 distinct groups. The experiment compares the consumers' cumulative satisfaction with and without the mediator's influence.

4.2 Producer

4.2.1 Pole-Balancing Environment

In the pole-balancing environment, an agent must keep a pole balanced at a target agent by sliding a cart along a track. This simple environment is popular because it is easy to program and difficult to control. In the experiment, each producer controls

a single cart, but their environments have different initial conditions. Environmental conditions include gravity, target angle, and magnitude of the force. Producers use sensors to detect the state of the environment; the sensors measure the velocity of the cart and pole and the angle of the pole. The reward function is

$$R(\theta) = K(\theta - \theta^*)$$

where K is a (Gaussian) kernel function, θ is the current angle of the pole from the vertical, and θ^* is a target angle. The learning algorithm is Prioritized Sweeping [39]. Given the current state of the system, the next state arises from the following equations [1] with extensions to incorporate friction:

$$\begin{array}{l} \ddot{\Theta} = \frac{g \sin \Theta - C \cos \Theta - \left(\frac{\mu_c \dot{\Theta}}{M}\right)}{M} \\ \ddot{x} = C - \frac{m_p L \ddot{\Theta} \cos \Theta}{M} \\ C = \frac{F + m_p L \dot{\Theta}^2 \sin \Theta - \mu_c \text{Sign}(\dot{x})}{M} \\ M = m_c + m_p \end{array} \quad \left| \begin{array}{l} \dot{x} = x + \tau \dot{x} \\ \dot{x} = \dot{x} + \tau \ddot{x} \\ \dot{\Theta} = \Theta + \tau \dot{\Theta} \\ \dot{\Theta} = \dot{\Theta} + \tau \ddot{\Theta} \end{array} \right.$$

where x is the horizontal position of the cart, \dot{x} is the velocity of the cart, \ddot{x} is the acceleration of the cart, Θ is the angle of the pole, $\dot{\Theta}$ is the angular velocity of the pole, and τ is a sampling constant to simulate the dynamics for a small period of time. In this experiment, the sampling constant is also the time during which the producer takes an action. The values for the constants are in Table 4.1. An example of the environment of the agent is Figure 4.1.

4.2.2 Initial Settings

Each producer is in an environment with different initial conditions. The parameter in bold in Table 4.1 and the target angle are uniformly distributed across the following

Theta: 9 X -0.206

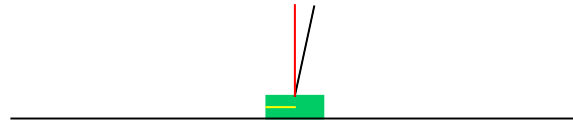


Figure 4.1: Graphical user interface for monitoring the progress of a single agent. The vertical line shows the target angle, and the horizontal line shows the applied force along the track

Constant	Name	Value
m_c	Mass of Cart	1.0 kg
μ_c	Friction of Cart	0.0005
m_p	Mass of Pole	0.1 kg
μ_p	Friction of Pole	0.005
τ	Sampling Constant	0.033 s
$ F $	Magnitude of Force	10 N
g	Acceleration due to Gravity	$9.81 \frac{m}{s^2}$
L	Length of Pole	0.5 m

Table 4.1: Values for simulation constants unique to an environment. Those constants in **bold** are variable environmental parameters.

ranges:

Parameter	Range
$ F $	$[12, 18]N$
θ^*	$[-11, 11]^\circ$

After initialization of the environment, Artificial Government runs for 5000 interactions. In each interaction, producers run 10 trials of 100 interactions with the environment. Each trial resets the pole-balancing environment and ends after 100 steps or the system moves outside the positional or angular bounds. A step is an interaction with the environment in which the producer obtains a new state from the environment and applies an action.

4.2.3 MP Space

The environment is the set of states and actions that the producer will encounter at any moment in time. In this experiment the states and actions come from the pole-balancing environment. At each instant, the environment sends the producer a vector

$$s = [x, \dot{x}, \theta, \dot{\theta}]$$

$$s \in S$$

where the components of the vector are discretized versions of the variables from the dynamics equations. Once the producer receives the state, it takes an action

$$A \in \{-F, F\}$$

where F is the randomly set magnitude of force. The producer's environment is

$$E = \{S, A\}$$

Given the randomly distributed force magnitude and target angle, the producer will encounter different sequences of states. The form of its model and policy, however, only depend on the environment. The model is a Markov Decision Process for the states and actions. The Prioritized Sweep Algorithm learns Q-values for state-action pairs given the model and reward function. The MP point of the producer at any point in time

$$MP = [s_1, a_1, \dots, s_5, a_5]$$

is a vector of the five most valuable state-action pairs according to the Q-values it learns. Although the model-policy point is not the entire model and policy of the producer, one assumes that any significant difference between two models and policies will occur in the most valuable state-action pairs.

The distance function between two model-policy points is defined over the discretized sensor values

$$\begin{aligned} d(MP_1, MP_2) &= \mathbf{w}|MP_1 - MP_2| \\ &= \mathbf{w}([x_{11}, x_{11}, \theta_{11}, \dot{\theta}_{11}, \dots, x_{15}, x_{15}, \theta_{15}, \dot{\theta}_{15}] - \\ &\quad [x_{21}, x_{21}, \theta_{21}, \dot{\theta}_{21}, \dots, x_{25}, x_{25}, \theta_{25}, \dot{\theta}_{25}]) \\ [x_1, x_2] - [y_1, y_2] &= [|x_1 - y_1|, |x_2 - y_2|] \end{aligned}$$

where $MP_{1,2}$ are the producers' MP points and \mathbf{w} is a weight vector whose components are positive and sum to 1. The dimensionality in this experiment is 25 since

each of the 5 MP points has 4 sensor values and 1 action. The mediator will use and adapt this distance function.

4.3 Consumer

The one and only consumer agent for this experiment is an abstraction of how several consumers might interact with the mediator. The consumer seeks three pole balancing solutions for three different target angles from the vertical: $\{-6^\circ, 0^\circ, +6^\circ\}$. During its test, it performs one trial run of pole balancing without any learning on each of the $k = 3$ prototype producers. The reward function returns the consumer's satisfaction with the solutions of the producers

$$\begin{aligned} C(MP_1, MP_2, MP_3) &= C(\mathbf{MP}) \\ &= \frac{x}{3} \end{aligned}$$

where x is the number of prototypes that meet the target angle for 10% of the interactions with the environment during the test. At time t , the cumulative level of satisfaction of a consumer is:

$$C(t) = \sum_{i=0}^t C(\mathbf{MP}_i)$$

where \mathbf{MP}_i is the set of prototypes at time i .

4.4 Mediator

The mediator closely follows the steps listed in Section 3.2.4.2. The only parameters are the number of clusters and the number of producers. The experiments choose

$k = 3$ clusters because the consumers, by assumption, only want 3 different solutions. This experiment, therefore, avoids the complication of deciding the right number of clusters. The number of producers is set to be 25.

Based on the description of the algorithm in Section 3.2.4.2, the producers and consumers can be physically separated from the mediator. The only explicit contact is necessary when updating the MP points and receiving the new reward functions. Otherwise, the producers and consumers can operate apart from each other. This suggests that a parallel implementation can quickly run the experiment. This experiment uses the MPI parallel programming interface to run each of the producers and the mediator in different processes. Since each of the producers performs the same number of iterations in the same environment, each takes about the same amount of time before it needs to communicate with the mediator. Using the all-to-one group communication methods, the mediator can efficiently synchronize information to all producers with one transmission. Although this particular implementation is not useful for heterogeneous agents, it does make the experiments run fast for the current project.

4.5 Results

The experiment compares the consumer's cumulative satisfaction versus the number of interactions between the producers and mediators for Artificial Government with and without the mediator's learning. Without learning, the set of prototypes is defined by clustering the points once and the consumers evaluate the same prototypes throughout the experiment. The mediator does not adapt the prototypes

or provide any new reward function to the producers. With learning, the mediator provides a new reward function after each interaction and learns weights that make the prototypes better fit the consumer's expectation.

The graph in Figure 4.2 shows the results for the experiment. The top curve shows the cumulative satisfaction with meta-learning. The bottom curve shows the cumulative satisfaction without meta-learning.

The bottom curve shows that at least for the first few iterations, neither methods satisfies the consumers. This is probably because of exploration. The reinforcement learning agents have exponentially decreasing exploration probability that initially starts high. Since high exploration leads to very random policies, the prototypes will not be of much use. In the bottom curve, the consumers only had access to the producers closest to the prototypes. Since the prototypes did not change, the satisfaction never improved.

The top curve shows that the producers learned to solve the right problems right. In both experiments, the producers learned to control the same environments. This suggests that the solutions that the consumers wanted were probably in the set of producers. With the adaptive clustering, the consumers could find the solutions they wanted. Without it, they could only use the first set of solutions at the beginning. The sudden jumps in the plot probably result from the producers better approximating the consumers' solutions. As the producers continue to learn in their environment, the consumers continue to evaluate them. The sharp increase in satisfaction toward the end of the experiment would be explained by consumers finding better solutions both by adaptive clustering and producer control. In summary, the results show that

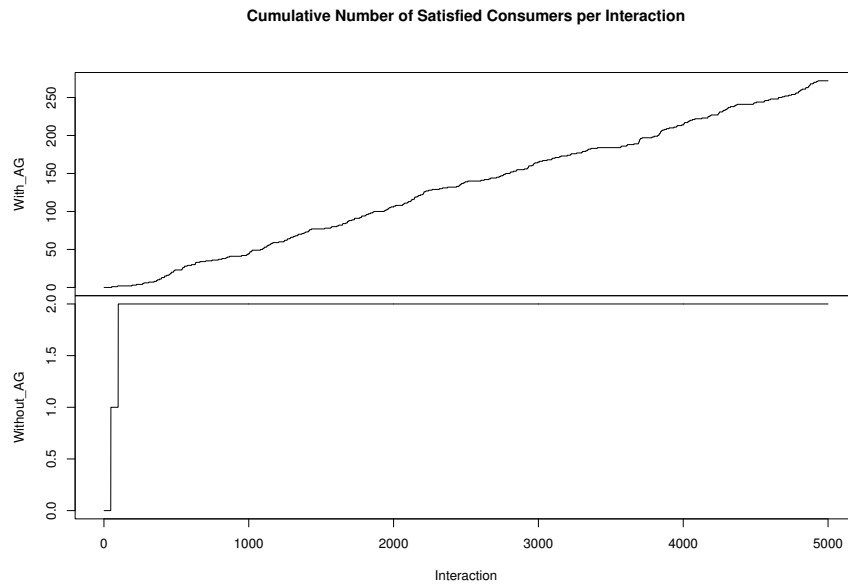


Figure 4.2: Cumulative number of interactions in which the consumer was completely satisfied with the solutions versus the number of interactions with AG.

meta-learning improves the cumulative satisfaction of the consumers.

Chapter 5

Conclusion

Artificial Government provides a set of roles for agents in a multi-agent system. Producers generate solutions to problems in a given environment. Consumers reuse these solutions in another environment and provide feedback based on how well the solutions perform in their environment. The mediator uses reinforcement learning to make the solutions that the producers produce better for the consumers. Following these roles, agents can work toward mutual benefit.

The functionality of the mediator is the main focus of this thesis. It concentrates on grouping the solutions (policies) of the producers and their contexts (models). Within the groups, representative points in this model-policy space serve as summaries for the consumers. Using a new clustering algorithm for supervised summary generation, the mediator learns to make groups whose prototypes are more satisfying to the consumers. In addition to summary generation, the mediator manipulates the reward functions of the producers to guide them toward solutions that are potentially more satisfying to the consumers.

In its current implementation, Artificial Government allows the mediator to control reinforcement learning agents in a set of related artificial control problems. The experiment measured the satisfaction of the consumers with and without any learning from the mediator. The results show that when the mediator can adapt the clusters and the producers, it can better satisfy the consumers.

5.1 Limitations

The focus of this thesis is to establish the different components and perform an initial study to determine whether the idea that similar problems have similar solutions is useful in distributed control. Not surprisingly, the approach has some limitations. The most significant limitation is the reliance on clustering.

Currently, the clustering method is biased toward a fixed number of clusters of a particular shape. This thesis offers no evidence to suggest that these assumptions on the distribution of the MP space are justified in the current problem. The results suggest that the assumptions may be valid, but more work is necessary to determine the best grouping algorithm to use. Since the consumer has a very restricted form, the reliance on a fixed number of clusters makes sense. In general, however, grouping must also occur at the consumer level to determine the best number of groups. Further work should look toward grouping algorithms that do not rely on clustering methods.

The current work relies on clustering because of the assumption that clusters in the MP space are useful to the producers and consumers. In this work, producers and consumers both make use of problem groups and MP prototypes. The experiments

demonstrate the utility of clustering in Artificial Government. What has yet to be demonstrated, however, is evidence to suggest that some other method of allowing the producers and consumers to reason about their positions in the MP space would be better. Future work should address this.

5.2 Future Work

The future work of Artificial Government will focus separately on producers, consumers, mediators, and new applications. Despite the limitations, the basic idea that producers should respond to the feedback of consumers is a widely recognized aspect of the real world. Eventually, research in multi-agent systems will focus on more interesting methods of achieving this.

5.2.1 Producers

The fundamental problem that the producers face is the incentive consumption problem. This problem arises when producers must meet additional objectives. In the current version of Artificial Government, the producer uses a reward function that is a weighted average of its initial reward function and that produced by the mediator. The question for future work concerns the motivation of the producer. What reward function will best direct the producer toward new solutions, and at what cost? The producer should have some utility-based motivation for interacting with the other agents in the Artificial Government. The introduction of utility also introduces the optimality of the solution with respect to decision theory. Specifically, does the mediator give the producer sufficient information to allow it to make optimal decisions

with respect to its own utility function? Another question for the producer is how to describe the MP space? Thus far in the work, the MP space is defined by the user, but this *ad hoc* description is not scalable to producers that the mediator has never previously encountered.

5.2.2 Consumers

The consumers face the opposite incentive production problem. The consumers need to be able to provide feedback so that producers can produce better solutions. What is the best feedback to provide? Consumers can also compete with each other for the pool of solutions. Competition introduces the notion of prices and preference. The presence of multiple consumers again introduces the need for grouping algorithms at the consumer level. Multiple consumers also bring to light issues about the size of the evaluation. To address these issues, future work, should investigate the use of sampling from the probability spaces of consumers and producers to allow for more robust testing methods. Future work should look into prices, negotiation, and grouping algorithms so that a large number of more sophisticated consumers can find what they want.

5.2.3 Mediator

A mediator tries to make the producers' solutions better for the consumers. Obvious directions for this research are better learning algorithms and better representations. The current work draws upon literature in the supervised summary generation domain that summarizes data so that users of the data can more readily find what

they want. Future work may look toward text-based information retrieval and search methods. Better representations should take into account the structure of the model-policy pairs for those that cannot be expressed as coordinates. Future work should also consider more than one mediator where each mediator has a specific portion of a set of problems to mediate. This brings up the issue of ontologies, relations among problems, and the potential for MP points in one MP space to be partially present in another MP space.

As future work should consider the inclusion of prices, the mediator may seek assistance from the economic literature. Economists have long studied the problem of exchange of mutually beneficial solutions. The mediator, however, introduces a new aspect to the exchange problem. The mediator must establish an exchange between similar solutions without full knowledge of all preferences. For example, a consumer tests an MP point x and returns an evaluation to the mediator, but the mediator must infer from the consumer its preferences on points y in the MP space such that $y \neq x$. The mediator must infer how a consumer will evaluate points it has not previously evaluated. To complicate matters, consumers may change their minds and may even stop looking for a solution if it finds one that fits its problem. Future work should address these issues that the fields of economics and artificial intelligence largely ignore.

5.2.4 Applications

Another important area for research is applications. For Artificial Government to be useful for an application, it must already have the concept of producers, consumers, and MP points. Some initial applications include Quality of Service (QoS) in networks, image retrieval, and e-commerce.

5.2.4.1 Quality of Service Management

In Quality of Service (QoS) management in computer networks, the goal is to provide better than Internet's best-effort service to users of a network. The dominant strategies in this area allow for the ends of the communication link to agree on some level of service as part of the initiation. Artificial Government is useful in this domain because it can learn to identify which flows are the most important without additional communication among the ends of the links. Considering the ends of the links as producers and consumers, the application of Artificial Government would view communication flows as points in the MP space. Producers could be servers whose traffic has a distinct time-series profile. Consumers would then be clients to the servers and provide them with some feedback regarding the quality of the communication experience. This feedback could relate to delays, jitter, or some other network-specific information. The mediator forms groups of similar flows and routers along the way can optimize the traffic according to the flow it follows. In this application, the mediator would identify flows for existing Differentiated Services (DiffServ) traffic allocation algorithms.

5.2.4.2 Distributed Image Retrieval

In the image retrieval domain, the goal is to provide users with the most similar image to a query image. Often, however, the objective function used for similarity is not the same as the user's subjective evaluation of the returned images. The users are consumers that provide an image and then return an evaluation of how well the returned images match the query. The image retrieval algorithm is the producer that learns a set of weights to better differentiate between images according to the consumer's responses. In traditional image retrieval, the image distance function is learned for groups of images, and only one group is assumed to exist. In the Artificial Government version of image retrieval, producers learn to specialize their distance functions given feedback from consumers. The consumers will only query an image retrieval that is close to the consumer's problem group. Because part of the mediator is a matchmaker, it can control the producers by deciding which images the producer should attempt to match. With this version of the mediator, the producer does not have to do any more work than traditional image retrieval requires. They achieve specialization by only accepting image queries that the mediator matches against the consumer; thus, producers do not need to expend any additional effort to achieve better retrieval results for consumers.

5.2.4.3 Commercial Applications

E-commerce applications would focus on providing consumers the products they want. The goal of the mediator would then be to provide information to producers for new products or modifications to better serve consumers. Although this is the

traditional motivation behind data mining, the explicit consideration of both what the producers produce and what the consumers want to consume is new. As producers would produce goods and services, they would use the MP space information that the mediator provides to plan for future production. Consumers would use the MP space information to evaluate potential goods based on the producer's prototypes. The most challenging aspect of this application would be defining the MP space and an appropriate distance function. The use of different products and many possible combinations would also necessitate multiple mediators that collaborate on multiple environments rather than assume a single environment.

Bibliography

- [1] C. W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, 1987.
- [2] C. G. Atkeson and J. C. Santamaría. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*, 1997.
- [3] A. Bagherjeiran, R. Vilalta, and C. F. Eick. Artificial government: Meta-learning for multi-agent coordination and control. In *Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005. Submitted for publication on March 22, 2005. Attached.
- [4] B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, and B. Krose, editors, *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8*, 2004.
- [5] B. Banerjee, S. Debnath, and S. Sen. Combining multiple perspectives. In *17th International Conf. on Machine Learning*, pages 33–40. Morgan Kaufmann, San Francisco, CA, 2000.
- [6] A. Barry. A hierarchical XCS for long path environments. In *Conference on Genetic and Evolutionary Computation Conference*, 2001.
- [7] E. Baum and I. Durdanovic. Evolution of cooperative problem-solving in an artificial economy. *Neural Computation*, 12(12):2743–2775, 2001.
- [8] J. Baxter. Theoretical models of learning to learn. In *Learning to Learn*. Kluwer Academic Publishers, 1998.
- [9] R. Becker, S. Zilberstein, and V. Lesser. Decentralized markov decision processes with event-driven interactions. In *AAMAS '04: Proceedings of the Third*

- International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 302–309, Washington, DC, USA, 2004. IEEE Computer Society.
- [10] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-independent decentralized markov decision processes. In *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 41–48, New York, NY, USA, 2003. ACM Press.
 - [11] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.
 - [12] M. V. Butz, D. E. Goldberg, and W. Stolzmann. The anticipatory classifier system and genetic generalization. Technical Report 2000032, University of Illinois at Urbana-Champaign, 2000.
 - [13] M. V. Butz and J. Hoffmann. Anticipations control behavior: animal behavior in an anticipatory learning classifier system. *Adapt. Behav.*, 10(2):75–96, 2002.
 - [14] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Theory of generalization and learning in XCS. Technical Report 2002011, University of Illinois at Urbana-Champaign, 2002.
 - [15] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. Technical Report 2000017, University of Illinois at Urbana-Champaign, 2000.
 - [16] J. L. Carroll, T. Peterson, and K. Seppi. Reinforcement learning task clustering. In *ICMLA*, 2003.
 - [17] J. S. Cox and E. H. Durfee. Discovering and exploiting synergy between hierarchical planning agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 281–288. ACM Press, 2003.
 - [18] I. D. Craig. *The Cassandra Architecture: Distributed Control in a Blackboard System*. Ellis Horwood Limited, 1989.
 - [19] I. D. Craig. From blackboards to agents. In *Online Proceedings of the VIM Project Spring Workshop on Collaboration Between Human and Artificial Societies*, 1997.
 - [20] M. B. Dias and A. Stentz. A market approach to multirobot coordination. Technical Report CMU-RI-TR-01-26, Carnegie Mellon University, 2001.

- [21] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [22] M. d’Inverno and M. Luck. *Understanding Agent Systems*. Springer, 2ed edition, 2004.
- [23] M. Dorigo and H. Bersini. A comparison of Q-learning and classifier systems. In *Proceedings of From Animals to Animats, Third International Conference on Simulation of Adaptive Behavior*, 1994.
- [24] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-II speech-understanding system:integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–53, 1980.
- [25] F. Fernández and D. Borrajo. Vector quantization applied to reinforcement learning. In *Proceedings of the Fifth Workshop on RoboCup*, pages 97–102, 1999.
- [26] M. Ghavamzadeh and S. Mahadevan. A multiagent reinforcement learning algorithm by dynamically merging markov decision processes. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 245–246, 2002.
- [27] E. Goffman. *Frame Analysis*. Harvard University Press, 1974.
- [28] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2002.
- [29] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.
- [30] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.
- [31] B. Horling, B. Benyo, and V. Lesser. Using self-diagnosis to adapt organizational structures. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 529–536. ACM Press, 2001.
- [32] H. Jung and M. Tambe. Performance models for large scale multiagent systems: using distributed POMDP building blocks. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 297–304. ACM Press, 2003.

- [33] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
- [34] T. Kovacs. Two views of classifier systems. In *Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*. Springer-Verlag, 2002.
- [35] P. L. Lanzi. Estimating classifier generalization and action’s effect: A minimalist approach. In *Genetic and Evolutionary Computation Conference*, pages 1894–1904, 2003.
- [36] D. Luzeaux. Learning knowledge-based systems and control of complex systems. In *15th IMACS World Congress*, August 1997.
- [37] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. In *Learning to Learn*. Kluwer Academic Publishers, 1998.
- [38] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [39] A. W. Moore and C. G. Atkeson. Prioritized sweeping–reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.
- [40] M. V. Nagendraprasad. *Learning Situation-Specific Control in Multi-Agent Systems*. PhD thesis, University of Massachusetts Amherst, May 1997.
- [41] R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 552–559, 2003.
- [42] J. Palma, R. Marín, M. Balsa, S. Barro, and P. Félix. A control model for distributed blackboard architecture based on task structures. In *International Symposium on Engineering of Intelligent Systems EIS’98*, pages 1198–1205, 1998.
- [43] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1998.
- [44] M. Rovatsos. Interaction frames for artificial agents. Technical Report FKI-244-01, Munic Technical University, 2001.
- [45] M. Rovatsos and G. WeiB. Achieving multiagent organisation by organising agent experience. In *Notes of 10th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, 2001.

- [46] N. M. Sadeh, D. W. Hildum, T. J. Laliberty, J. McA’Nulty, D. Kjenstad, and A. Tseng. A blackboard architecture for integrating process planning and production scheduling. *Concurrent Engineering: Research and Applications*, 2(6), 1998.
- [47] R. Schoknecht, M. Spott, and M. Riedmiller. FYNESSE: A new architecture for sequential decision problems. In *Computational Intelligence in Industriellen Einsatz*, number 1526 in VDI-Berichte, pages 109–118, May 2000.
- [48] A. Standfuss and R. Eckmiller. To swing up an inverted pendulum using stochastic real-valued reinforcement learning. In *4th International Conference on Artificial Neural Networks (ICANN ’94)*, pages 655–658, 1994.
- [49] W. Stolzmann. An introduction to anticipatory classifier systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, number 1813 in Lecture Notes in Artificial Intelligence, pages 175–194. Springer, 2000.
- [50] R. Sun and T. Peterson. Automatic partitioning for multi-agent reinforcement learning. In J. A. Meyer, A. Berthoz, D. Floreano, H. L. Roitblat, and S. W. Wilson, editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior (SAB2000)*, 2000.
- [51] C. Sutton, B. Burns, C. Morrison, and P. B. Cohen. Guided incremental construction of belief networks. In *5th International Symposium on Intelligent Data Analysis*. Springer-Verlag, 2003.
- [52] K. Takadama, T. Terano, and K. Shimohara. Learning classifier systems meet multiagent environments. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems*, pages 192–210, 2001.
- [53] S. Thrun and J. O’Sullivan. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to Learn*. Kluwer Academic Publishers, 1998.
- [54] A. Tomlinson and L. Bull. A zeroth level corporate classifier system. Technical Report UWELCSG99-001, UWE Learning Classifier Systems Group, University of the West of England, 1999.
- [55] K. Twardowski. Credit assignment for pole balancing with learning classifier systems. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 238–246, 1993.

- [56] T. Vu, J. Go, G. Kaminka, M. Veloso, and B. Browning. Monad: a flexible architecture for multi-agent control. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 449–456, New York, NY, USA, 2003. ACM Press.
- [57] M. P. Wellman. A market-oriented programming environment and its application to distributed multi-commodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.
- [58] M. P. Wellman. Market-oriented programming: Some early lessons. In S. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, River Edge, New Jersey, 1996.
- [59] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2), 1995.
- [60] S. W. Wilson. Get real! XCS with continuous-valued inputs. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, pages 209–219, 2000.
- [61] F. L. y López and M. Luck. A model of normative multi-agent systems and dynamic relationships. In G. Lindemann, D. Moldt, and M. Paolucci, editors, *Regulated Agent-Based Social Systems*, Lecture Notes in Artificial Intelligence 2934, pages 259–280. Springer, 2004.
- [62] F. Ygge. *Market-Oriented Programming and its Application to Power Management*. PhD thesis, Lund University, 1998.
- [63] G. M. Youngblood, E. O. Heierman, D. J. Cook, and L. B. Holder. Automated hierarchical POMDP construction through data-mining techniques in the intelligent environment domain. Technical Report CSE-2004-17, University of Texas at Arlington, 2004.