The Dissertation Committee for Adrian Kujaneck Agogino
certifies that this is the approved version of the following dissertation:

Design and Control of Large Collections of Learning Agents

Committee:

Joydeep Ghosh, Supervisor

Ross Baldick

Craig Chase

Gustavo de Veciana

Risto Miikkulainen

Kagan Tumer

Design and Control of Large Collections of Learning Agents

by

Adrian Kujaneck Agogino, B.S., M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2003

Dedicated to my parents and family.

Acknowledgments

This thesis represents the joint action of a complex multi-player game, with a wide diversity of players spanning many different states and nations. Foremost I would like to thank my advisor Joydeep Ghosh and my research mentor Kagan Tumer for their tremendous support of my work. I am grateful for Joydeep in directing me through the multitudes of fuzzy paths in machine learning. While giving me a great latitude in my research, he was able to guide me around time-consuming pitfalls, even when the utility signals I gave him were noisy and infrequent. I thank Kagan for his great insights on my thesis topic and for his boundless enthusiasm for my work. I also must give recognition to one of the most important players in this game, David Wolpert, who originated the framework for which my thesis is based upon. I also would like to thank Risto Miikkulainen for his guidance and encouragement related to my work in neural networks. I also thank my other committee members Ross Baldick, Craig Chase and Gustavo de Veciana for taking their time to review my work and generating important feedback and utility values.

I thank all the members of the LANS lab for their discussions and support. In particular to Kurt Bollacker for introducing me to machine learning and showing me how to navigate the intricacies of graduate school, and to Shailesh Kumar for his support and superb technical advice. I thank lab members Kui-yu Chang and Gunjan Gupta for mixing research with their long discussions on Mars, Linux, Hawaii, graphics and foreign cultures. I would also like to thank Alexander Strehl, Ken Stanley and Riccardo Signorelli for their friendship and advice in academics and outside of school.

Most importantly I thank my mother Alice, father Dan, step father Dale and sister Arianne for their love, sacrifice and support of education. Finally I need to thank the estimated fifty seven million agents involved in my work who loyally did their best to follow my utilities.

Design and Control of Large Collections of Learning Agents

Publication No. ____________

Adrian Kujaneck Agogino, Ph.D.
The University of Texas at Austin, 2003

Supervisor: Joydeep Ghosh

The intelligent control of multiple autonomous agents is an important yet difficult task. Previous methods used to address this problem have proved to be either too brittle, too hard to use, or not scalable to large systems. The Collective Intelligence project at NASA/Ames provides an elegant, machine-learning approach to address these problems. This approach mathematically defines some essential properties that a reward system should have to promote coordinated behavior among reinforcement learners. This thesis will focus on creating additional key properties and algorithms within the mathematics of the Framework of Collectives. The additions will allow agents to learn quickly in more complex systems. Also they will let agents learn with less knowledge of their environment. These additions will allow the framework to be applied more easily, to a much larger domain of multi-agent problems.

Table of Contents

List of Figures

Chapter 1

Introduction

A large number of problems are best solved with coordinated distributed computation. Some of these problems include controlling large numbers of robots, coordinating parallel processing and organizing numerous disparate consumer devices, such as PDAs, cell phones and personal navigation systems. In addition logically separate entities, such as “intelligent” agents and characters in entertainment computing, often involve coordinated distributed computation. As the cost of processors is reduced concurrent with the rise in communications, the size and importance of these domains will dramatically increase.

This dissertation will refer to any computational entity as a node of computation or an agent. This notion of an agent is designed to be very broad and flexible. This dissertation will be concerned with multi-agent problems that have the following characteristics

1. There is a well specified global objective.
2. Agents need to coordinate to achieve this objective.
3. Agents are adaptive.

4. Agents do not need centralized control.

5. There are many agents.

The insistence on a well specified objective is to differentiate from many multi-agent domains that are plagued with blurry objectives such as "work well together," or act "believably." These objectives are often found in entertainment computing literature [10, 42]). Instead the focus will be on problems where the objective can be mathematically defined in the form of a function. The second charactersitic is needed to avoid distributed problems where each agent is independent as far as the global objective is concerned. These problems reduce to finding competent single agent learners. The third charactersitic is highly desirable in any system and is especially important in large systems since failure in one part of the system can lead to global catastrophic failure when adaptation is not possible. The fourth charactersitic is important since heavily centralized control defeats the benefit of distributed low cost processors. Finally the last charactersitic forces a solution that cannot be easily hand crafted for a small set of specific interactions.

Unfortunately most existing systems fail at these problems in numerous ways. They often need centralized control and even small problems can cause catastrophic failure. In addition existing systems tend not to scale well to large numbers of agents. Interestingly economic principles often address these issues, as economies are built upon large groups of adaptive, autonomous agents (humans). In contrast to most agent models, the COIN ("COllective INtelligence") framework designed at NASA/Ames builds large, efficient collective systems by extending these economic principles, using concepts such as learnability to help form off equilibrium incentives [71]. COIN treats a distributed processing problem like an economy, where each node of computation is a self-interested learner, which maximizes a private utility. If the private utilities are set up correctly, then the self-interested learners will act to optimize the world utility for the entire system. This framework includes several mathematical models for collective learning and provides an excellent starting point for building successful multi-agent systems.

This dissertation will focus on finding solutions to the collective learning problem within the COIN framework. The first two chapters summarize research related to collectives and will provide an overview of the COIN framework. Chapter 4 then introduces time-extended problems in collectives including the Time Extended Bar Problem. Chapter 5 presents a series of solutions addressing the difficulties coming from limited communication. This chapter shows a method by which the noise-cancelation effect of an important utility can be preserved under communication restraints, allowing COIN theory to be applicable to a much larger domain of problems. Chapters 6 and 7 show how the COIN framework can be used to address the temporal credit assignment problem present in single agent reinforcement learning algorithms and evolutionary algorithms respectively. The later chapter then shows how COIN theory can be used to leverage some of the benefits of reinforcement learning in an evolutionary algorithm. Chapters 8 and 9 show how multi-agent systems can learn in time extended tasks, and unify credit assignment problem arising from multi-agent systems and the credit assignment problem coming from multi-time-step problems. Chapter 8 uses a discrete problem, while chapter 9 uses a continuous problem. In addition chapter 9 introduces visualization techniques to improve understanding of inherently complex multi-agent systems. Finally chapters 10 and 11 apply the COIN framework to data-mining problems.

Chapter 2

Background and Related Work

2.1 Rule Based Systems

Many early multi-agent systems consisted of monolithic programs that specified agent behavior for a specific problem. These systems were unprincipled and the ability of the agents to interact depended on the programmer's intuition. Some more general principles of multi-agent interaction came out of entertainment computing. The entertainment computing community introduced notions of collective scripts and agent role playing [33]. These systems though generally had fuzzy objectives, and the community was satisfied with agents that looked like they were acting collectively. When a script failed there was an emphasis on recovering in a way that looked believable. While these systems worked well in some domains, they would not be suitable to problems with harder objectives. There is also some multi-agent work that has come out of software engineering [30, 40], but these systems only provide a loose framework and their applicability or performance is not readily apparent.

More recently there have been several attempts at a more principled approach to rule-based collective learning. The CONSA system [68] provides a somewhat general framework for interaction between agents. This system specifies roles and assignments where agents can take on different roles, depending on their place in a role graph. There are also various team operators that direct the proper agents to take joint actions. To avoid some the brittleness of previous systems, they implement a conflict resolution paradigm, where agents can argue about conflicting goals. This architecture has been applied to various military applications such as helicopter combat as well as some civilian uses, such as personal assistants [50]. While it has achieved a number of successes, it also has retained many of the problems of rule-based approaches. The process of forming the rule hierarchy is highly labor intensive and only a small portion of the universal rule sets can be used. The system also still has robustness problems where it can fail catastrophically even in certain simple situations.

2.2 Economics

Macroeconomic theory is relevant to multi-agent learning since it addresses systems that contain huge numbers of adaptive self-interested agents. Often economists provide methods to reach a desirable equilibrium such as Walras's method for achieving clear markets through a process called tatonnement [73]. Another important equilibrium point is the "Nash equilibrium" where every agent has achieved its best possible strategy given the strategy of other agents [45]. Under certain conditions Nash showed that this equilibrium will always be reached. These economic principles are valuable since they can often be moved to more general problems. While useful, these algorithms tend to assume perfect knowledge in addition to centralized control, although some improvements have been made [21].

Some of these economic principles have been directly applied to multi-agent problems. For the domain of coordinating multiple rovers, a technique called “TraderBot” was created where agents could autonomously trade goals [83]. In this problem a collection of rovers tried to maximize the amount of information retrieved by arriving at goal sites, while minimizing the total distance traveled. Initially a central authority would assign the rovers goals using a fixed (possibly random) algorithm. The rovers would then “own” these goals, but at every time step they could sell them in an auction. When the reserve price of the auction is equal to the rover’s expected utility of reaching the goal, another agent will always win the auction if it can reach the goal more efficiently. The auction pushes the system to a local optimum in the assignment of rovers to goals. However the system may be far away form a global optimum, especially since the rovers do not learn so they cannot correct inaccurate goal-utility predictions. In addition the system has a centralized node, representing a single point of failure, and while the rovers can continue to operate independently, they have little coordination ability without the central node. In a related architecture, Sandholm and Lesser created an “automated negotiation system” where self-interested agents could autonomously negotiate contracts with each-other to complete tasks [53]. In this system various contract operators were introduced to setup a framework for efficient negotiation.

2.3 Artificial Life

Artificial life is concerned with making learning algorithms that are inspired by principles found in living organisms. These biologically inspired learners often exhibit very desirable properties, such as robustness and the ability to adapt. One important field of artificial life is evolution, which uses the concept of evolutionary adaptation found in nature. Since evolutionary systems typically involve evolving large populations of algorithms, they can be effective in some multi-agent domains. A multi-agent system can allow for a large genetic population leading to faster learning and superior performance.

In [4] it was shown that evolution of neural networks could be successfully applied to multi-agent predator/prey domains. In this problem a group of prey had the task of traveling from a start position to a goal as fast as possible, while avoiding a predator (Figure 2.1). Evolving online, the prey was able to develop complex behavior and evolved quickly when there was a large population of prey. This problem however did not need cooperation between prey. Also the evolution typically caused the population to converge to a single behavior, making it inappropriate for many multi-agent domains.

Another common field in artificial life is simulated ant colonies [18, 52, 72]. These colonies are inspired by the organizational ability of ants, and are often used in path finding algorithms. Ant colonies have been successfully applied to network routing [18], where artificial "ant" were sent as packets through the network, recorded timing information and updating routing tables. The ant-based routers were shown to have higher bandwidth and lower latency

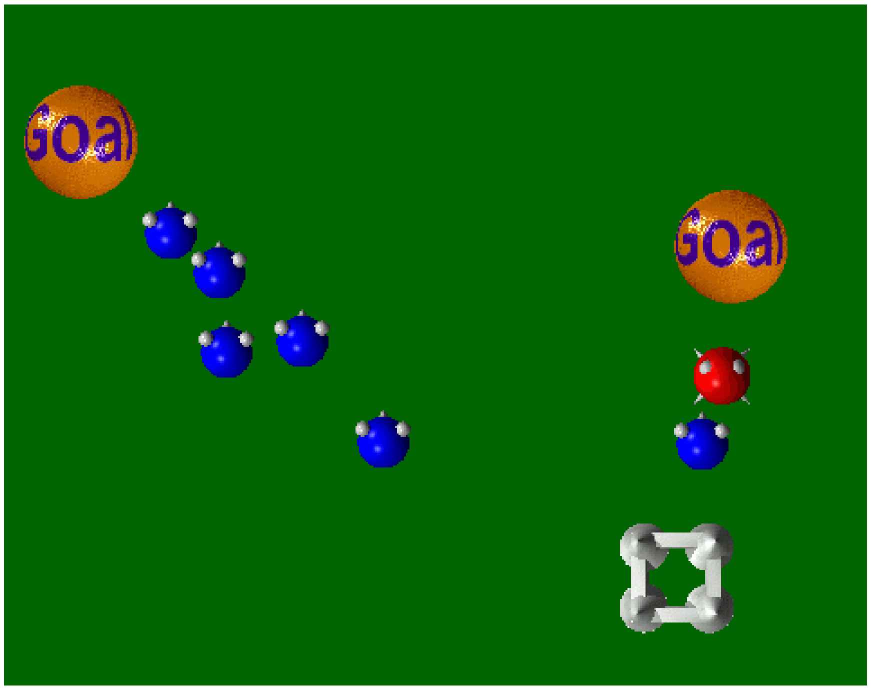


Figure 2.1: “Prey” evolves online to avoid predator.

than the traditional network routing systems. While ant colonies have had many successes, so far their applicability has been somewhat limited.

2.4 Reinforcement Learning

A reinforcement learner (RL) learns a solution to a Markov decision process by using rewards received as a result of actions [43, 63, 65]. Formally, given a state s, an RL takes an action a, receives a reward r and transitions to the next state s'. In deterministic environments the values r and s' are functions of s. In non-deterministic environments they are sampled from probability distributions conditioned on s. Often reinforcement learning algorithms have to be modified in non-deterministic environments to avoid cycles. However note the the RL does not need to know these distributions. They are implicitly determined from the environmental input received during exploration. Since they do not need to know these distributions ahead of time, reinforcement learning is a form of model free learning. The goal of the RL to take the right actions to maximize some function of the rewards it receives.

Some reinforcement learners do not work for full Markov decision processes and instead try to maximize the immediate reward received from an action, given the state. This is often done by keeping a table of action/state pairs that contain information about the previous rewards. The learner then chooses an action from the table based on the either a softmax or an ϵ-greedy approach. With the softmax approach, the action is usually chosen with the Boltzmann distribution, where the probability of choosing action a in state s is:

$$\frac{e^{Q(s,a)/T}}{\sum_{a' \in A} e^{Q(s,a')/T}} \tag{2.1}$$

where $Q(s,a)$ is the table value for action a in state s, and A is the set of all possible action from state s. With the ϵ-greedy approach the action associated with the highest table value is taken with probability $1-\epsilon$, and a random action is taken with probability ϵ. Learners that update table values based solely on the immediate reward will be referred throughout this dissertation as a “simple” learner.

Another popular class of reinforcement learners is one that maximizes a discounted sum of rewards received by the learner. Often these learners are used when there is a series of rewards received over time, such as a robot receiving feedback as it wanders through its surroundings. The discounted sum can either be over the life of the learner $\sum_{t=0}^{\infty} \gamma^t r_t$ or over an episode of length L for episodic tasks $\sum_{t=0}^{L} \gamma^t r_t$. A Q-learner is the most popular of this type of learner [74]. Q-learners work by computing a Q value for each state/action pair. This value is computing by combining an immediate reward, with an estimate of future rewards derived from the Q values of other states. For deterministic environments the Q value for state s and action a can be updated by the following formula:

$$Q(s, a) = r + \max_{a'} Q(s', a') \tag{2.2}$$

where r is the reward after taking action a in state s, and s' is the next state entered. Q-learners can often learn quickly, since even though they optimize a long term goal, they receive an immediate reward for every action. Note that this update formula is policy independent in that it is not concerned with the actions the agent actually takes in the future.

A learner very similar to a Q-learner is a Sarsa learner. The update rule for Sarsa is:

$$Q(s, a) = r + Q(s', a') \tag{2.3}$$

where a' is the action that the agent takes in the following state s'. This differs from Q-learning in that the action, a', used for the table lookup is the actual action taken instead of the best action possible from this state. Learners like Sarsa are called "on-policy" learners since a' is determined by the policy, instead of learners like Q-learners which are called “off-policy” learners. More convergence proofs have been done for off-policy learners, but in generally it is not clear which type is better. This paper uses both types of learners, with results showing that there is not a significant difference between the two in collective domains.

When the state space is continuous, Q-tables cannot be directly used for creating a policy for deciding the best action for a given state. Instead function approximators, such as neural networks, are often used to replace the Q-tables [75]. Typically there will be a function approximator for each action. The value computed from $r + \max_{a'} Q(s', a')$ will then be used as the target value to train the function approximator for state s. There are also many heuristics for “tiling” the state space, so that different regions of the space use different function approximators [64]. However many of these methods are ad-hoc and there are very few convergence guaranties when function approximations need to be used. In addition, in many domains, Q-learning works very poorly in conjunction with function approximators, since the “max” operator often increases the error present in the Q-value approximations [66].

While RLs perform well on many single agent problems, they typically have difficulties on multi-agent ones. If a single RL is used then the state spaces of all the agents have to be combined. This state space generally grows combinatorially causing the learner to learn too slowly. If each agent has its own RL then they may not coordinate their actions well. Some work has been done to reduce the state space to allow multi-robot learning [41], but many the coorperation issues are still unresolved. Leveraging game theory and reinforcement learning, Hu and Wellman provided an algorithm [35] where through Q-learning a pair of agents could reach a Nash equilibrium in general sum games, even when the agents did not know the reward function or state transition probabilities. However this algorithm scaled exponentially with the number of agents. More appropriate for a larger number of agents, Mataric has shown [41] that groups of foraging robots could be made to cooperate by constructing a set of utilities appropriate for the domain. Even though they could be effective, hand tailored private utilities often had to be laboriously modeled and were not "adaptive" to changing environments. In addition, the ability of a designer to hand tailor a utility may not scale well with the number of agents.

2.5 Team Games

One way for reinforcement learners to work together is to employ a learning paradigm known as a "team game." In a team game, each reinforcement learner receives as a reward value, the overall performance of the entire system. This way, when the learners optimize their reward values they are optimizing the global objective.

A well known application of team games is the Crites and Barto elevator problem [23]. In this problem they tried to optimize the average squared waiting time of elevator passengers in a four elevator system. This problem was challenging since the state space was very large. Given the position, direction, speed and buttons pushed on each elevator the total state space had 10^{22} states. To reduce the state space they gave each elevator its own reinforcement learner, which received the global objective of the average squared waiting time as its reward. After 60,000 hours of simulation time their solution gave superior results to the state of the art elevator control algorithm at the time. While team games are effective when there are a very small number of agents, they do not scale well to large numbers of agents. This problem occurs since in large systems an agent has trouble discerning its contribution to the team.

Chapter 3

Collective Intelligence Framework

COIN ("COllective INtelligence") is a framework for controlling collections of agents [70, 71, 79–81]. This framework is designed for situations where there is no centralized control and where there is a clear global objective function that needs to be optimized. It also shows its greatest benefits when used with a large collection of agents.

The COIN framework assumes that each agent has its own reinforcement learner that is able to learn to take actions that will result in high rewards. The task of COIN then is to create a system of rewards such that when each agent receives high rewards, the global objective function will be close to maximized. Also COIN needs to create a reward system that is simple enough that a reinforcement learner can learn it in a reasonable amount of time. In some ways COIN theory is related to the fundamental principles of economics where high economic output depends on individuals in large populations greedily optimizing their own needs. The COIN designer is analogous to a government, which tries to create a sufficiently complex rule set so that individuals do not profit from hurting society, but not a set so complicated that no one can follow it. The task of COIN designers is often much easier than the government though, since they have absolute control of the learning systems, and of the reinforcements that are used. However in other ways COIN theory differs greatly from traditional economic theory in that it provides for off-equilibrium incentives and is concerned with the learnability of problems, which will be discussed later in this chapter.

3.1 Formal COIN Model and Notation

COIN assumes that all the information about the world is stored in a Euclidean vector z, known as the worldline. This information is comprehensive and includes all of the environment, the agents' outward behavior as well as the agents' internal parameters and learning systems. Any non-deterministic behavior can be modeled as a random number generator that is in a non-observable portion of z. The state of the world at a given time t is z_t and is assumed to progress at discrete time intervals.

An agent η is any computational entity that has an effect on a portion of the world line z_η. The parts of the worldline that are not affected by the actions of η will be referred to as $z_{\hat{\eta}}$. In this dissertation a union of worldline elements will be expressed as a comma delineated list enclosed by parenthesis. For example the worldline z could be expressed as $(z_{\hat{\eta}}, z_\eta)$, a change in the worldline caused by η could be expressed as $(z_{\hat{\eta}}, z'_\eta)$ or a change in the worldline caused by fixing everything that η can influence to a constant C_η could be expressed as $(z_{\hat{\eta}}, C_\eta)$.Note that this is a very general notion of an agent and may include mathematical constructs that are traditionally not thought of as agents.

The goal of COIN theory is to maximize an arbitrary function of the world line referred to as the global utility $G(z)$. Example of global utilities include such things as the GDP for an economic system, or total bandwidth in a network. While the overall objective is to maximize $G(z)$, each node will try to maximize its own private utility function $g_\eta(z)$. The goal of COIN is to find ways of creating private utilities so that each agent is capable of maximizing their own $g_\eta(z)$, and through maximizing $g_\eta(z)$, $G(z)$ is also maximized.

3.2 Principles of Private Utilities

Private utilities must have certain properties so that a collective system will achieve a state of high global utility. Creating a utility with these properties is non-trivial and many obvious solutions fail. Consider for example a global utility which is simply a sum of private utilities $G(z) = \sum g_\eta(z)$. Even though this system has the property that if all the agents have a high private utility, then there may be a high global utility, this system will fail to achieve a high global utility, even if the agents are perfectly competent. An example of this is the classic "Tragedy of Commons" scenario, where a large group of herders are raising sheep on a grassy island. Let the global utility be the number of sheep on the island and the private utility of a herder be the number of sheep that the herder has on the island. Here the global utility is the sum of private utilities. In this scenario, an individual herder will be encouraged to keep bringing sheep to the island, since this action will help his own utility. However, due to environmental constraints if all the herders try to maximize their utility, all the grass on the island will be eaten, all the sheep will die, and the global utility will collapse.

COIN theory introduces a class of utilities, known as factored utilities, that are more likely to achieve a high global optimum.

Definition: A utility $g_\eta(z)$ is **factored** if any change agent η makes to the world line that results in a higher $g_\eta(z)$ also results in a high global utility:

$$g_\eta(z) \geq g_\eta(z') \Leftrightarrow G(z) \geq G(z') \tag{3.1}$$

where z' and z differ only in the state of agent η.

If an agent takes an action (i.e. changes the world in some way) that increases the value of a factored utility, then that action will increase the value of the global utility given every other agents' actions. It can be shown that if all the agents in a system greedily maximize a factored utility then the global utility will reach a local maximum. One important factored utility is the global utility, which is factored by definition. Systems that use the global utility as the private utility are team games.

A factored utility is only useful in maximizing the global utility if the agents in a system are able to learn to achieve high values for their private utilities. Highly learnable utilities are therefore preferable to less learnable ones. COIN theory defines the **learnability** of a utility U of an agent η as the ratio of the change in U caused by η's actions to the change in U caused by all the other agents' actions.

Definition: The **differential learnability** of a utility g_η for agent η is

$$\lambda_{\eta,g_\eta}(z) \equiv \frac{\|\vec{\nabla}_{z_\eta} g_\eta(z)\|}{\|\vec{\nabla}_{z_{\hat{\eta}}} g_\eta(z)\|} \,. \tag{3.2}$$

where $\hat{\eta}$ refers to all agents other than η.

Learnability represents an agent's ability to "see" the results of its actions. As a consequence agents with highly learnable private utilities should be able to maximize their utilities almost as well as if they were in a single agent system. Learnability can also be interpreted as a ratio of "signal to noise." Under this interpretation the portion of an agent's utility that it influences is the signal. In turn the part of the utility that an agent cannot influence is considered noise since it is not giving information about how the agent can improve its action. A highly learnable utility has a high signal to noise ratio, since the agent's action will have significant influence over the value of the utility compared to the background noise caused by the actions of other agents.

3.3 Intelligence

One way to measure how well an agent is performing its task, is to measure the value of utilities in systems that the agent performed in. Measuring an agent against its private utility may be useful in seeing how well the agent is learning its own utility. In contrast measuring an agent against the global utility may show how well the agent is contributing to the entire system, and may give an idea of how factored the agent's private utility is with respect to the global utility. However evaluating an agent based on its utility performance has numerous problems. One issue is that scaling the utility may influence the perception of how well the agent is performing. A more important problem is that the observer does not know how hard it is for an agent to reach certain level of utility. It is possible that a poorly designed agent could achieve high utility values because the problem was very easy. Conversely, a very good agent could be associated with low utility values, simply because the problem was hard.

Another way to measure the performance of an agent is with intelligence values

Definition: The **intelligence** of an agent η's action with respect to utility U is

$$\epsilon_{U,\eta}(z) \equiv \int dz'_\eta \Theta(U(z) - U((z_{\hat{\eta}}, z'_\eta))) \tag{3.3}$$

The function $\Theta(x)$ is the Heaviside function which is equal to 1 when $x \geq 0$ and equal to 0 when $x < 0$. The intelligence of an agent's action is always in the range $[0, 1]$ and can be interpreted as percentile rank of the performance of the action, with respect to all other possible actions that the agent could have taken. For example if an agent had an intelligence of 0.6 with respect to a utility, sixty percent of all possible other actions would achieve lower values of that utility (see Figure 3.1). Note that if utility U_1 is factored with respect to utility U_2 then $\epsilon_{U_1,\eta}$ is equal to $\epsilon_{U_2,\eta}$.

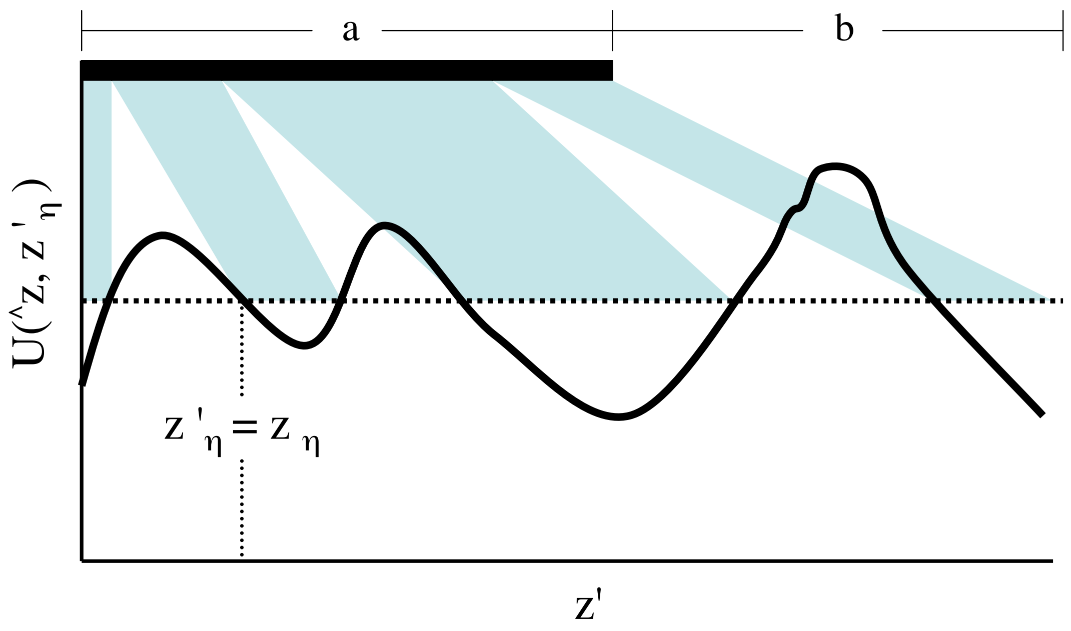


Figure 3.1: The intelligence of an agent's action is equal to the percentile rank of that action compared to all possible actions. Here the intelligence of η with respect to U is equal to $\frac{a}{a+b}$.

3.4 Central Equation

The probability of achieving a given value of global utility, G, given the parameter setting made by the collective designer, s, can be decomposed as follows:

$$p(G|s) = \int d\epsilon_G p(G|\epsilon_G, s) \int d\epsilon_g p(\epsilon_G|\epsilon_g, s) p(\epsilon_g|s) \tag{3.4}$$

where ϵ_g is the intelligence with respect to an agent's private utility and ϵ_G is intelligence with respect to the global utility. The vector s represents all the free parameters of the worldline that the collective designer is able to set. These include things such as the agent's reinforcement learning algorithm and the agent's private utility functions. The job of the collective designer is to set values of s so that the system has a high probability of achieving high values of G.

The three terms of this decomposition summarize the important issues in collective learning. Term three of the equation, $p(\epsilon_g|s)$, can be used to measure how well an agent is able to learn its utility. If the probability distribution of the third term is peaked around high values ϵ_g, then the private utility $g(z)$ is probably highly learnable. The second term, $p(\epsilon_G|\epsilon_g, s)$, can be used to measure how well the agents are coordinating their activities. If the probability distribution of the second term is peaked around high values of ϵ_G then the private utilities are likely to be close to being factored with respect to G. Finally the first term, $p(G|\epsilon_G, s)$, shows how well the system is overcoming local minima. This dissertation will be mostly concerned with terms two and three. The methods presented in this dissertation can usually be combined with standard optimization methods, such as random restarts to achieve high values of global utility.

3.5 Difference Utilities

There is an intrinsic tradeoff between factoredness and learnability. One of the largest issues in COIN research is to find a good compromise between the two. This issue is important because when using a utility with high factoredness and learnability, even systems that have agents with simple learning mechanisms can achieve high global utility. Other methods such as team games are factored, but often have low learnability. Especially when there are many agents, the influence that one agent has on its own utility is usually dominated by the influence of other agents. In contrast individualistic greedy utilities, such as the amount of time a driver takes to get himself home, are often highly learnable, but not factored.

One important class of utilities that makes a good compromise between factoredness and learnability is the class of difference utilities: **Definition:** the **Difference Utility** for agent η is:

$$DU_\eta(z) \equiv G(z) - \Gamma(f(z)) \tag{3.5}$$

where Γ is independent of η. Difference utilities have been proven to be factored with respect to $G(z)$. In addition, there are a number of choices for Γ and f which will cause the second term to subtract out much of the noise in the first term that is caused by the actions of other agents. When this happens the difference utility will be far more learnable than a team game.

One difference utility known to have high learnability is know as the AU [1] [78]: **Definition:** the **AU** for agent η is:

$$AU_\eta(z) \equiv G(z) - E_{z_\eta}[G((z_{\hat{\eta}}, z_\eta))|z_\eta] \tag{3.6}$$

This is the difference utility where $f(\cdot)$ is the global utility and $\Gamma(\cdot)$ is an expectation taken over the values of z_η. This utility can be shown to be the most learnable difference utility in some senses. However this utility is usually

[1]AU stands for "Aristocrat Utility" showing an agent's effect on G compared to an "average" effect on G.

not very practical to compute, especially if G is a complicated function. Even if it is approximated with Monte-Carlo methods, it takes a lot of computation time, requiring many evaluations of G.

3.5.1 Wonderful Life Utility

The Wonderful Life Utility (WLU) is a Difference Utility that is easier to compute than AU, and still has high learnability. To define the WLU one first defines the effect set of an agent η, $S^{eff}_{\eta}(z)$, as all the components in the worldline that are affected by a change in η.

Definition: The **effect set** for agent η is

$$S^{eff}_{\eta}(z) \equiv \{i | \vec{\nabla}_{z_\eta}(z_i) \neq \vec{0}\}. \tag{3.7}$$

where i is an index of vector z.

The clamping operator $CL_{\sigma}(z)$ is then defined as an operator that transforms z to z' such that all the components in z' that correspond to elements in a set σ are clamped to a vector $\vec{\kappa}$.

Definition: the **Wonderful Life Utility** for agent η is:

$$WLU_{\eta}(z) \equiv G(z) - G(CL_{S^{eff}_{\eta}}(z)) \tag{3.8}$$

where $\vec{\kappa}$ is a vector fixed ahead of time.

When $\vec{\kappa}$ is set to the zero vector, the clamped global utility approximates what the world would be like without agent η, therefore the WLU can be interpreted as approximating the agent's contribution to a system. Note that the WLU does not actually need to calculate what a system would be like without an agent. Instead $G(CL_{S_\eta^{eff}}(z))$ can be computed with little knowledge since the clamping is done irrespective of the system's dynamics. Since the WLU models the agent's own contributions, it is much easier to learn than the global utility, yet it can be proven that it still retains its desirable factoredness property. Using 3.2 the learnability for the WLU utility is:

$$\frac{\lambda_{\eta,WLU_\eta}(z)}{\lambda_{\eta,G}(z)} = \frac{\|\vec{\nabla}_{z_\eta} G(z)\|}{\|\vec{\nabla}_{z_\eta} G(z) - \vec{\nabla}_{z_\eta} G(CL_{S_\eta^{eff}}(z))\|} \,. \tag{3.9}$$

It can be seen from this equation that when $\vec{\nabla}_{z_\eta} G(z)$ and $\vec{\nabla}_{z_\eta} G(CL_{S_\eta^{eff}}(z))$ are close, the WLU will be highly learnable. This will usually happen when the number of agents is large compared to the effect set.

Note that the clamping vector, $\vec{\kappa}$, can be set to other values besides the zero vector. One choice that is in some ways optimal is the set the clamping vector the the expected action of agent η. This dissertation will refer to a WLU that uses this form of clamping vector as $WLU^{\vec{a}}$. In addition often an agent η only affects the worldline components in z_η. In these cases the clamping and effect set operators are not needed and the WLU can be defined as:

$$WLU_\eta(z) \equiv G(z) - G((z_{\hat{\eta}}, \vec{\kappa})) \tag{3.10}$$

3.5.2 Illustrative Example using WLU

This chapter will use several variants of the Minority Game to illustrate the value of the WLU [20]. In the classic Minority Game agents simultaneously choose to be a member of either team 1 or team 2. After the agents decide, the team with the least members wins, and all the agents of this team receives a utility of 1, and all the agents of the larger team receive a utility of zero. This problem poses many issues for agent learning since if too many agents learn to be part of the team with the highest utility, their actions will be self-defeating and will cause them to receive a low utility.

A related canonical problem in collective decision making is known as "Arthur's El Farol Bar Problem" [7]. In the classic version of the Bar Problem, 100 agents decide whether to go to a bar. The bar is most enjoyable when less than 60 agents attend. If more than 60 attend, the agents would have been better off staying home.

This chapter will focus on an extended version of the Bar Problem. In this problem each agent decides which night to attend a bar each week. Also each agent has a reinforcement learner that tries to maximize a reward it receives every week. The global objective of this problem is to maximize the total profit of the bar owner. The bar owner does not like the bar to be under-capacity since he will get very little revenue, however he does not want the bar to be too crowded since he then he will have to hire more people. This is an interesting problem because if all the agents develop the same strategy, then they will all attend on the same night, making the bar overly crowded and thereby hurting the global objective.

To put the n agent Bar Problem into the COIN framework, the world-line, z, can be defined as binary 7 by n matrix. If agent η goes to bar k, the matrix element $z_{k,\eta}$ will have a value of one, otherwise it will have a value of zero. Let $x_k(z) = \sum_\eta z_{k,\eta}$ be the attendance for night k. Total utility for a single night is $x_k(z)e^{(-x_k(z)/c)}$ and peaks when the number of agents attending that night is equal to c as shown in Figure 3.2. The global utility is then $G(z) = \sum_{k=1}^{7} x_k(z)u_k(z)$. To illustrate the benefits of applying certain COIN principles to maximize the global utility, three different trials were performed, each with their own utility system. The performance of each utility system was then measured (Figure 3.3).

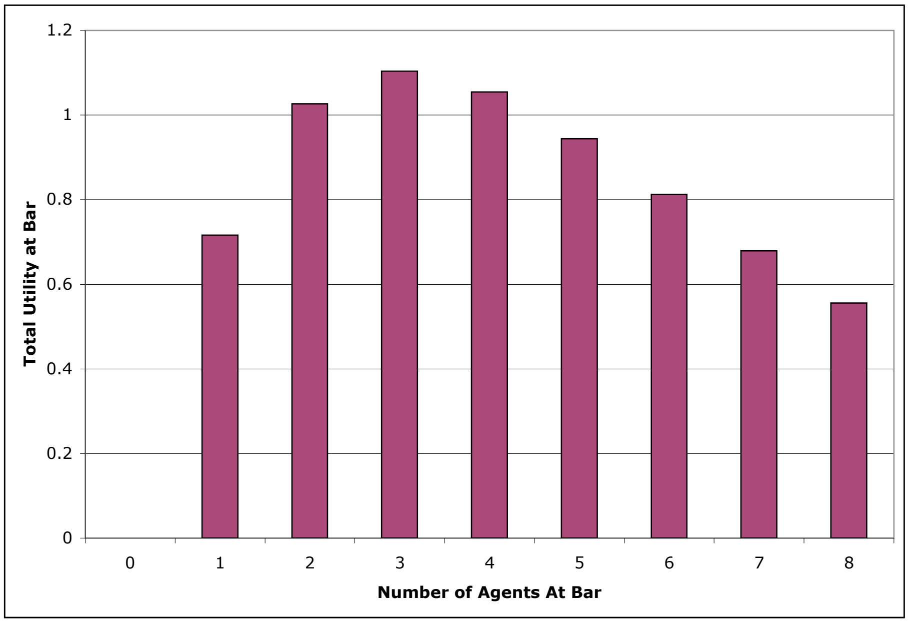


Figure 3.2: Utility for a single bar when $c = 3$.

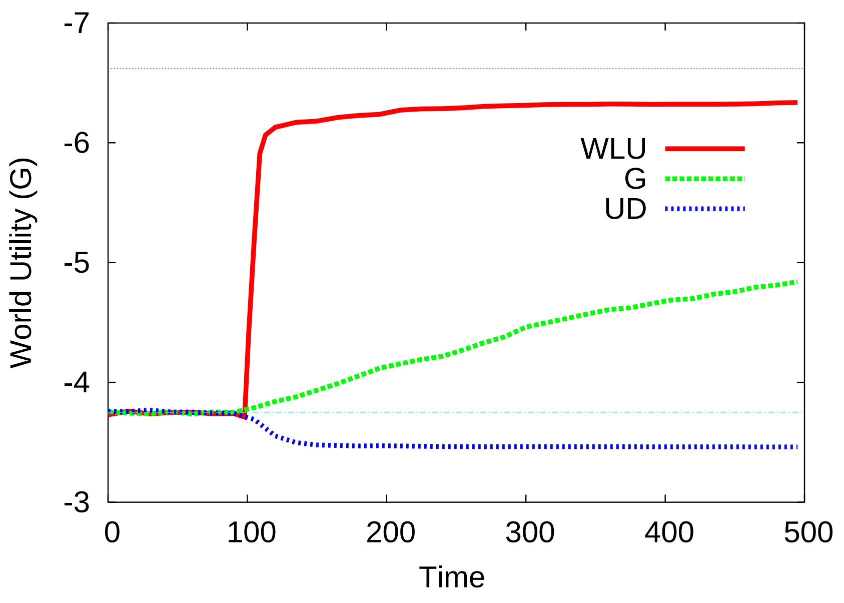


Figure 3.3: WLU displays superior performance in El Farol Bar Problem (random actions taken for first 100 time steps).

In the first trial each agent received the Wonderful Life Utility $WLU_\eta = G(z) - G((z_{\hat{\eta}}, \vec{0}))$. Since each patron makes its decision independently, the effect for an agent just includes itself, the column vector for agent η. The clamped worldline is therefore $z - z_\eta$ where z_η the z matrix with all of the columns other than column η set to the zero vector. The solution the agents found with this reward was for most of them to go to the bar on one night with a few going all the other nights. This turned out to be a very good solution, since the six nights where very few agents attended received very high rewards. The majority of the patrons that attended the same night would appear to be sacrificing themselves for the benefits of the others, since that night is overcrowded. However, the agents attending this night are still being greedy with respect to their own utility and are simply taking the action that maximizes the WLU.

In the second trial a team game solution was implemented where each patron received the global utility, $G(z)$. The agents using this utility were able to steadily improve, but learned very slowly because all the noise caused by the other agents. The third trial consisted of agents receiving a "selfish" utility which was equivalent the utility for a single night divided by the number of agents attending that night: $e^{-x_k(z)/c}$. The agents receiving a selfish utility did even worse as their performance actually decreased over time. The more competent an agent got at maximizing its reward the worse the system behaved, since the agents would lose their reward by making sacrifices.

3.5.3 Effect Sets

In the formulation of the Bar Problem there were no issues with effect sets for an agent η extending beyond z_η. As an example of a problem with effect set issues, suppose that the Bar Problem is changed so that all the patrons are divided into leader and follower agents. The leader agents will make all the decisions and each leader will have its own set of follower agents that will always go to the bar on the same night as the leader. Since the actions of the leader agent affects all of its followers, using the $WLU_\eta = G(z) - G((z_{\hat{\eta}}, \vec{0}))$ formulation is not possible, since a leader agent η will affect the components of $z_{\hat{\eta}}$ corresponding to the follower agents. In this case, the full formulation of WLU, which includes effect sets, has to be used for the WLU to be factored.

In this dissertation effect sets will not be an issue in most of the problems. The main exception is in Chapter 6, where effect sets need to be computed to determine an action's effect on future rewards.

3.6 COIN Theory Applied to Other Domains

COIN theory has been applied to data networking tasks where significant gains can be achieved over the popular Bellman-Ford algorithm [79]. In this application a collection of routing agents is created, where each agent learns a routing vector that controls the proportion of packets a router sends out on its various outbound links. Each agent has a simple reinforcement learner that has a routing vector as its output and the routing vectors from all the other agents as its input. The reward is created by applying the Wonderful Life Utility, so that when an agent tries to optimize its reward it will choose a routing vector that will not work cross-purposes with respect to the global objective of maximizing the network bandwidth.

COIN theory has also been applied to the NASA problem of helping minimize information loss of constellations of satellites around various planets communicating with earth [77]. In this problem a satellite could be forced to lose valuable data, when it could no longer communicate with Earth for a long enough period of time to cause its data buffer to overfill. Situations such as this happen often when satellites lose their direct line of site to Earth when going behind a planet (Figure 3.4). However when there are many satellites, a satellite that may not be able to communicate directly with Earth, may still be able to relay its information through other satellites nearby. This is a natural application of the distributed agent paradigm since these satellites need some fundamental degree of autonomy as a result of the high communications delay due to the speed of light. The collective solution to this problem addressed terms one and two of the central equation. In the solution, agents used simulated annealing, but instead of making random exploration steps they used a reinforcement learner, learning from a team game, to make the exploration step.

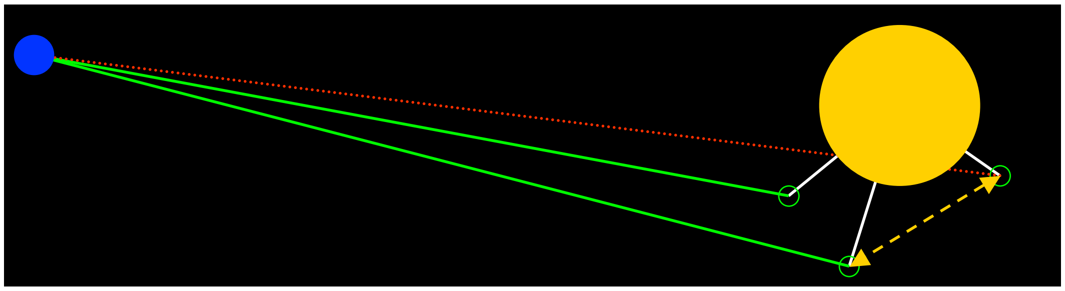


Figure 3.4: Satellites may be forced to throw out valuable data when they lose line-of-site to Earth (dotted line). However they may be able to relay information to another satellite (dashed line). The theory of collectives can be used to create a distributed algorithm that minimizes the satellites' information loss.

Chapter 4

Time-Extended Collectives

Collective learning in time-extended domains, such as Markov Decision Processes (MDPs), adds significant complications over collective learning in single time step environments. Both single time step rewards and time extended utilities must be used carefully. While single time step rewards are generally used with reinforcement learners in MDPs, in collective environments, many simple rewards including the "global reward" may not be markovian when all the agents are not completely observable. Also rolling up single time step rewards into a single time-extended utility will not always be useful since decisions must be made at each time step. This chapter explores some solutions to this problem using the Time Extended Bar Problem as an example.

In fully observable, deterministic domains, episodic time extended problems can be posed as a single step problem by rolling up all the states and actions into a single state and action in a larger space. The single combined action of the single step learner can then be decomposed into all the single time step actions that need to be taken. Since the environment is fully observable and deterministic, all the actions that need to be taken will be known from the start. A major difficulty with this approach is that it will not work in non-deterministic or partially observable environments. This is due to the fact that the action taken by the single step learner is not reactive and cannot be used to generate an action for a state that it does not expect to be in. Perhaps even a more significant difficulty with this approach is that the size of the single state and action space will grow exponentially with the number of time steps.

Single agent Partially Observable Markov Decision Process (POMDP) learning problems can be solved using reinforcement learners methods, such as Q-learning and Sarsa [19]. One solution to the multi-agent problem is to turn it into a single agent problem, by rolling up all the states and actions of all the agents into one state and action in a large space. This method however, leads to similar problems to rolling up the time steps, in that the state and action spaces will grow exponentially with the number of agents. Using individual learning agents, will allow for a manageable action space, but in a fully observable MDP, the state space will still grow exponentially with the number of agents. If the learners state is restricted to the "local" state of the agent, then the learning problem becomes manageable, but many rewards that would be markovian in a single agent system may not be markovian in the multi-agent one. Figure 4.1 shows a case when this is true. Suppose agent 1 always went from state A to B to C to D. Agent 2 moves from A to E to F to G or from A to F to G. On the transition from F to G, the *global reward* will be 10 if the previous state was A, since agent 1 would be making the transition from B to C at this time step. In contrast the global reward would be 0 if the previous state was E since agent 1 would be making the transition from C to D at this time step. In this case the global utility is non-markovian if the agents can only see their own states, since the reward probabilities depend upon previous states. However in many cases the Markov property can be violated in this way, and the system can still obtain high performance, since often the other agents act mostly as environmental noise. The global reward can also be made Markovian by restricting the transition diagram to a tree. This restriction can be interpreted as adding a time stamp to the state of an agent. Note that this will in general only work in episodic domains.

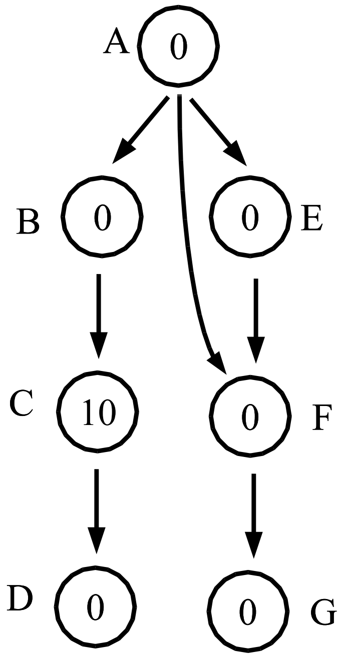


Figure 4.1: Global Utility may be non-markovian in multi-agent domain

4.1 Definitions

This sections will give the definition of the collective POMDP problem used in this chapter and more precisely define a single agent system. Let S be the set of states an agent can be in and T be the set of directed transitions between two states in S. The set T therefore determines the actions an agent can take. The global reward received after an action is a function of the new state that the agent is in after the action is performed. In the multi-agent case, each agent, η, is in a single state $s \in S$ and its actions are still restricted to those allowed by T. Let us define the matrix L_η to be a state by time matrix for agent η. An element $L_{\eta,t,s}$ is equal to one if agent η is in state s at time t and zero otherwise. This chapter, and much of this thesis will focus on three different forms of global reward for this multi-agent system.

- $R_t(z) = f(\sum_\eta L_{\eta,t})$
- $R_t(z) = \sum_{s \in S} f_s(\sum_\eta L_{\eta,t,s})$
- $R_t(z) = \sum_{s \in S} a_s \sum_\eta L_{\eta,t,s}$

where $f(\cdot)$ is an arbitrary function, $f_s(\cdot)$ is an s-indexed element of a set of arbitrary function, and a_s is an element of an s-indexed constant vector.

The first reward is the most general case when agent identities do not matter, being an arbitrary function of the number of agents at each state [1].

[1]The sum does not lose information if agent identities do not matter since the value of $L_{\eta,t}$ is binary.

This type of reward is very hard to learn in general, since it may not offer any form of locality. Many rewards of this type may confound agents using difference utilities, since nothing may be cancelled out. This thesis will be more focused on rewards of the second type, which are linear combinations of functions of the number of agents at each state. Note that the Bar Problem is in this form, since the global utility is a sum of the amount of agent satisfaction at each bar, which is a function solely of the number of agents at each bar. Congestion problems can usually be posed in this form too. The last reward is simply a linear combination of the number of agents at each state. Even though many multi-agent problems have this form, it is not an interesting multi-agent problem since an optimal solution is found when each agent greedily optimizes based on its own state. Congestion problems or the Tragedy of Commons do not arise in this form.

4.2 WL Rewards

For this problem, the time-extended WLU can be defined as:

$$WLU(z) = G(z) - G((L_\eta, C_\eta)) \tag{4.1}$$

where C_η is the fixed clamping matrix for agent η. The WLU can then be decomposed into single time step rewards as follows:

$$\begin{aligned} WLU(z) &= G(z) - G((L_\eta, C_\eta)) \\ &= \sum_t R(z_t) - \sum_t R(L_{\eta,t}, C_{\eta,t}) \\ &= \sum_t (R(z_t) - R(L_{\eta,t}, C_{\eta,t})) \end{aligned}$$

A WL reward can therefore be defined:

$$WLR_\eta(z_t) = R(z_t) - R(L_{\eta,t}, C_{\eta,t}) \tag{4.2}$$

Notice that this reward is a function of the world line for a single time step, therefore it is only useful in Markovian domains. It is also only factored through time since the second term of the WLR is independent of the agent's state. In Chapter 9, the second term of the obvious WLR is not independent of state and is only factored for a single time step. Chapter 9 will show how to construct a fully factored WLR for this situation.

The simplest way to maximize the WLU is through Monte-Carlo estimation. If a single step simple learner is used, the reward used at time t is simply the future WLRs:

$$R_{mc} = \sum_{t'>t} WLR_{\eta,t} \tag{4.3}$$

the simple learner then incorporates this reward into its tables as follows:

$$Q'(s,a) = \alpha R_{mc} + (1-\alpha) Q(s,a) \tag{4.4}$$

where the reward R_{mc} is received after the agent took action a in state s, and α is the learning rate. Q-learning and Sarsa can be viewed the same as the this simple learner, except that estimates of the future rewards are used. For example for Sarsa the reward would be:

$$R_{sarsa} = WLR_{\eta,t} + Q(s', a') \tag{4.5}$$

where s' and a' are the state and action for time step $t + 1$.

4.3 Time Extended Bar Problem

The TEBP is similar to the Bar Problem, except that the agents have to make a sequence of choices, and their utility is based upon the final bar they end up at. In the TEBP the agents have fewer choice of which bars to attend than in the Extended Bar Problem. Each day a player can only choose to attend the same bar, or the bars next to the one he attended the previous day. The actions space for the agent therefore has a size of three and the state space is equal to the number of bars. This is an episodic task where every four days the agents are reset to a random position, at which point they are given a reward based on their last choice of bar to attend. This task forces the agents to come up with a sequence of four actions that will maximize their final utility at the end of four days.

4.3.1 WLU with TEBP

This section shows that the WLU can also be applied to time extended version of the Bar Problem. Here the same utilities used in the single time

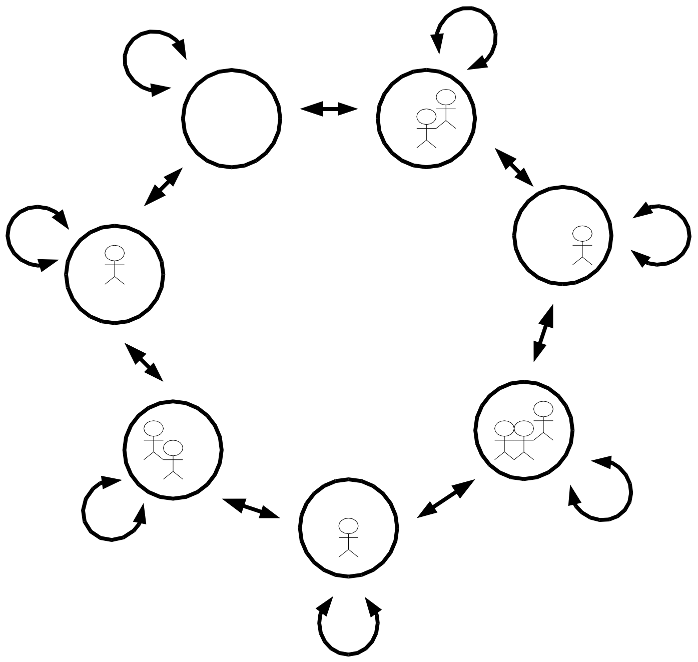


Figure 4.2: Time Extended Bar Problem. Circles represent bars, figures represent patrons attending a bar and arrows represent the transitions that patrons can take from week to week. Each week a patron can choose to attend the same bar or the bars next to the one he attended the previous week.

step version of the Bar Problem will be used by the agents at every time step of an episode. In future chapters this thesis will show why this is the correct way to apply a difference utility to a Markov Decision Problem. As in the bar problem, experiments were performed using WLU, team game utilities (TG) and selfish utilities (SU). The learning algorithm used by the agents is the Temporal Difference version of Sarsa. This section will show that the choice of utility is important and that the time extended nature of the problem cannot be ignored. However, the results will also show that the parameters of the learning algorithm is not significant.

4.3.1.1 TD(0) Sarsa Learner

Figure 4.3 shows the results of the comparison of the three utilities, using a TD(0) version of Sarsa. This version is equivalent to standerd Sarsa learning with one step look-ahead. As expected the SU reward does poorly, with performance actually going down with time. The TG reward also does very poorly, in fact much worse than in the standard Bar Problem. This lower performance is probably due to the Time Extended Bar Problem being much harder to learn. The WLU does well, achieving within 0.84 of optimal.

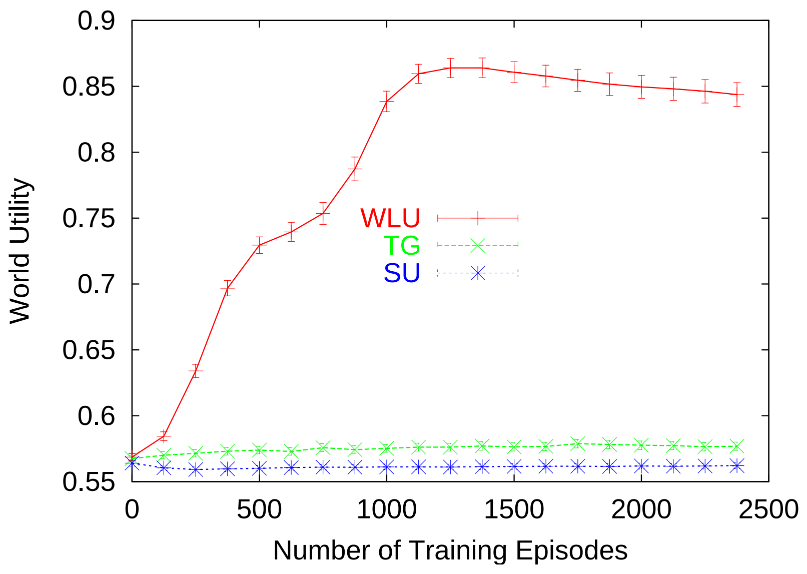


Figure 4.3: TD(0) Sarsa

4.3.2 TD(0) vs. TD(1) vs. TD(0.8)

Next how the parameters for temporal difference affects performance was investigated. TD(0) as discussed before is standard Sarsa learning with one step look-ahead. TD(1) is standard Monte-Carlo learning where all the future rewards are looked at equally. TD(0.8) is a compromise between the two. The results (Figure 4.4) show that WLU performs almost equally with the different values of lambda.

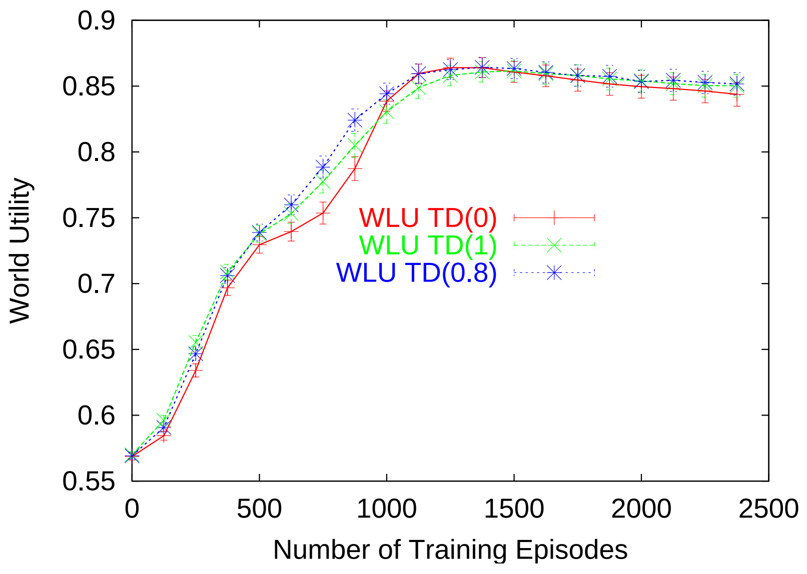


Figure 4.4: Comparison Between Values of Lambda

4.3.2.1 Learning while Ignoring Future Rewards

To test if the time extended learning was needed, experiment were ran using learning methods that only take into account immediate rewards. This is the same type of learner used in the original bar problem. This learner can be seen as a Sarsa learner with discount factor of 0. As expected the results (Figure 4.5) show that this type of learner is not able to learn as well. However the solution still performes well above random because the number of bars is small. Within one time step an agent is able to reach 3 out of seven bars.

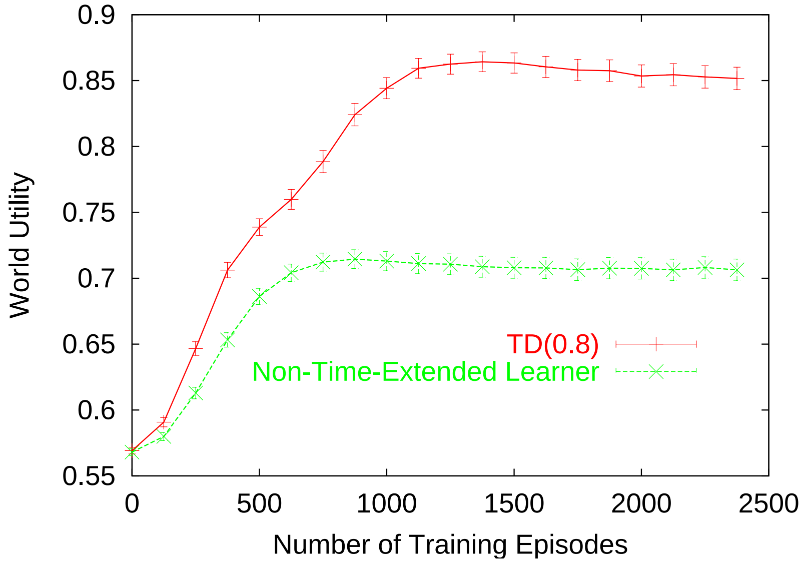


Figure 4.5: Performance of Learner with no Look-ahead

Chapter 5

Communication Restrictions

Many methods for coordinating the actions of autonomous agents in a large multi-agent system, including the framework of collectives, are designed to work in situations where those agents can fully communicate with one another [13, 23, 39, 54, 69, 78]. However, many problems impose communication restrictions among the agents, rendering the coordination problem more difficult [11]. Examples of these problems, include controlling collections of rovers, constellations of satellites and packet routers, where an agent may only be able to directly communicate with a small number of other agents. In addition, even if there are other indirect ways to share information, they may be costly and an agent may be unwilling to share, if doing so would hurt its private utility. In all of these problems, the collective's designer faces the following difficult task:

- ensuring that, as far as the provided world utility function is concerned, the agents do not work at cross-purposes (i.e., making sure that the private utilities of the agents and the world utility are "aligned").
- ensuring that agents can achieve their private utilities when they do not have access to a broad communication network allowing giving them

access to global information.

These issues are at odds with each other and in fact in many cases it will be impossible for agents to achieve high values of a private utility which is aligned with the world utility. In addition even if the world utility, computed with global information, can be broadcast to all the agents, agents may not be able to effectively use this information to select actions that will be useful to them and to the overall system. In fact many methods of incorporating local information into the world utility can lead to reduced performance as communication increases (Figure 5.1). This example shows the performance of a system (described in detail in Section 5.3) with respect to the amount of communication available to the agents. Note that increasing the amount of information to which the agents have access can have deleterious effects on the performance of the system. This chapter will discuss the reasons for this apparent paradox and show how some problems associated with communication restrictions can be overcome by modifying the agents' utility functions.

Furthermore, issues related to communication restrictions can also be addressed by agents aggregating into teams sharing a utility function. Many types of team formation have been shown to be effective in different domains [48, 51]. In this chapter's domain, utility sharing encourages team members to pool their information together, effectively reducing the impact of the communication restrictions. As the size of a team grows, the amount of information to which an agent has access also grows. However, even if large

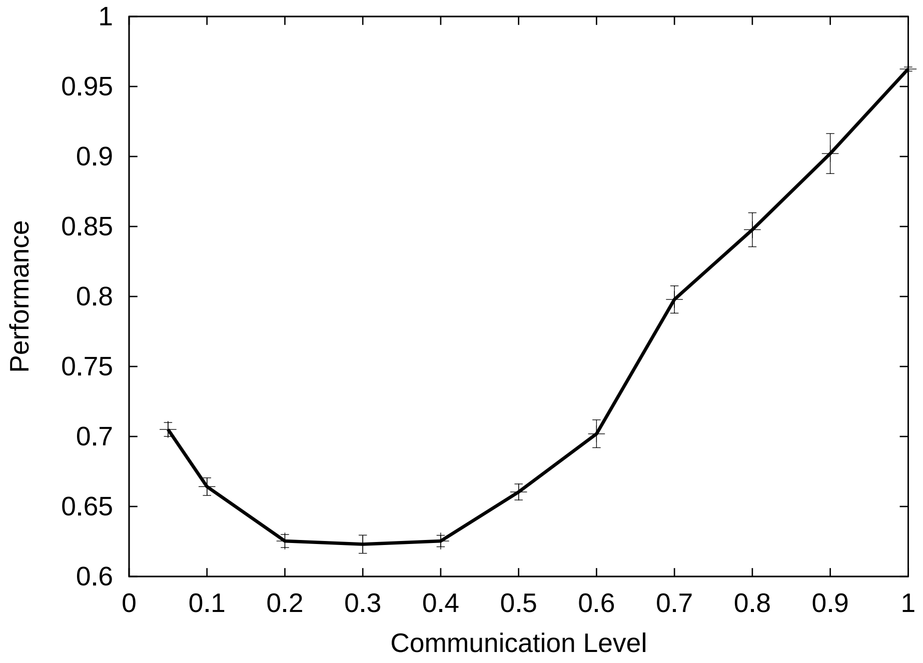


Figure 5.1: Performance vs. Communication Level when World Utility is combined with partial world information. When the world utility is known, adding additional information about the world can actually hurt performance. Using standard methods more than 60% of world information has to be revealed before improvements can be made on the world utility. With better designed utilities presented later in this chapter, even limited partial information can be used to increase performance.

teams have access to more information, the agents now face the problem of determining the contribution of their actions to the utility.

This chapter will explore how moderate communication restrictions can be overcome by modifying the agents' utilities. It will then show that team formation can be used when there are severe communication restrictions, and will explore the tradeoff between team size and communication restrictions.

Section 5.1 will give a brief description of existing methods for handling agent communications and team formation. Section 5.3 will describe the problem domain and present the collective-based solution to this problem. Section 5.4 will present the simulation results, which will then be discussed.

5.1 Existing Methods for Agent Communication and Team Formation

Issues related to agent communication and team formation have been studied separately for many years from a variety of viewpoints. Literature closer to the focus of this chapter, where teams are used to overcome limited communication is less common and tends to come form sensor-fusion research. In addition there is a large body of related work on multi-agent systems and how to coordinate multiple agents.

5.1.1 Communication Among Agents

The study of communication among agents has taken on many forms. Much work has been done on low level communication issues, such as agent communication languages and physical implementation of communications [24, 25, 57]. At a higher level Pynadath and Tambe have formalized many aspects of agent communications [49], including observability and explicit communication. For multi-agent Markov decision processes, Xule et al. dealt with the problem of partially hidden states of other agents [82]. In their system communication of an agent state had a cost and they presented a number of algorithms that traded off cost of communication versus the expected gain from the knowledge obtained from the communication. A number of researches have noted that often little communication is needed to coordinate agents [9], and that in many cases local communication is sufficient [27, 56]. However these observations are only true in certain specific domains.

5.1.2 Teams

The concept of teams can be found most often in human economies. For example, corporations are often setup with team structures where employees are members of a team or group (e.g., through sharing a bonus for successful completion of a project) and each team member benefits when the team successfully contributes to the goals of the corporation. Spontaneous team formation in agents has also been studied at a theoretical level. Axtell has shown that for small sizes of teams there can be a stable Nash equilibrium, but that the stability breaks down when teams go beyond a certain size [8]. Similarly this chapter will show that even when team formation is created in a top-down manner that there are still issues with how learnable a utility can be when there are large teams.

There has been extensive research on rule-based agent team formations. Tambe has shown that coordination rules can be used successfully in many fields including military engagement [67]. A common mechanism to coordinate team agents is for teams to have "joint intentions" [22] where team agents need to work for a common goal. Groz coins the term "SharedPlan" [32] to refer to this concept. As described later, this chapter will borrow from this concept by having team members share a common utility. Even more related to this chapter is work done in the field of sensor fusion. Fox has shown that when the amount of information that a robot receives is restricted, teams of robots, with different sensors, can work together to solve the robot localization problem [26]. In addition it has been shown that teams can share sensor information to estimate unobservable parts of the world in robotic soccer domains [62].

5.2 Communication Restricted Utilities

In general to compute a difference utility there may need to be enough communication to infer the value of the entire worldline. For some specific classes of utility such as the WLU, this communication demand may be relaxed, since many of the elements of the worldline cancel out and may be ignored. However in many real world problems there is not enough communication between agents to compute even the less demanding utilities. In these cases one must approximate the utility under the constraints of the communication restrictions.

Mathematically one can represent the communication restrictions for an agent η as elements of the worldline that are not observable. It is possible to decompose the worldline z into a component observable by agent η, z^{o_η}, and a component hidden from agent η, z^{h_η} (this chapter will denote the concatenated state z by $z = (z^{o_\eta}, z^{h_\eta})$). In this chapter, the communication level for agent

η will be defined as:

$$B_\eta = \frac{|z^{o_\eta}|}{|z|} \tag{5.1}$$

where $|z^{o_\eta}|$ is the number of observable elements in the worldline and $|z|$ is the total number of elements in the worldline. Note that B is always in the range $[0.0, 1.0]$.

If the WLU for agent η depends on any component of z^{h_η} then η cannot compute it directly. Instead this section will introduce different approximations to the WLU that vary in their balance between learnability and factoredness. In the four utilities discussed below, the first two letters of the utility represent how the two terms of the difference utility get their information. "B" stands for "broadcast" meaning that the world utility is broadcast to the system, "T" stands for "truncated" meaning that the hidden values are ignored, and "E" stands for "estimated" meaning that the hidden variable is estimated from the observed variables.

5.2.1 Broadcast/Truncated Utility (BTU)

The first private utility that will be presented for systems with communication restrictions is a variant of WLU, where the clamping removes not only agent η's contribution, but also the contribution of all agents that η cannot observe by clamping the state $(z^{o_\eta}, \vec{0})$:

$$BTU_\eta(z) \quad = \quad G(z) - G(CL_\eta((z^{o_\eta}, \vec{0}))) \tag{5.2}$$

where $\vec{0}$ is the vector whose components are all zero, and CL_η clamps all components of agent η to the zero vector. Note that BTU, as well as BEU

(discussed below), assume that the true world utility can be broadcast despite the communication restriction. In many applications, this is a reasonable assumption since the world utility can often be computed once and broadcast throughout the environment [29]. More complex forms of broadcasting are often used for distributed multi-agent systems [17], but in this chapter will assume a very simple global broadcast of a single number. In many domains it is also reasonable to assume world utility can even be obtained directly from the environment without broadcasting [56].

Note that despite this "broader" clamping, BTU is still factored. This is because BTU is in the form of a difference utility, which only specifies that the second term cannot depend on the state of η. Any clamping that *includes* the state of η will result in a factored utility, regardless of how much more is clamped. However, this utility may have much more noise than a pure WLU since much more than η has been clamped. Intuitively, only part of the noise, the part that was observable, has been removed from η's utility. The BTU is the utility used for Figure 5.1, and this noise issue causes the BTU to exhibit the poor performance shown in the graph.

As an example, consider a situation where agent η is an employee in a large company. Proper WLU would remove the impact of the other employees from employee η's private utility, since their general effect would be present both in the first term G and the second term $G(CL_\eta(z))$. But if employee η can only communicate with a fraction of the employees, all the employees with whom it cannot communicate will also also be clamped in the second term.

Then the subtraction will not remove their effect from employee η's utility. The influence those employee have on η's utility will be noise, and employee η will have a harder time seeing the effect of its actions on its utility.

5.2.2 Truncated/Truncated Utility (TTU)

The second private utility is conceptually similar to BTU except that both terms are computed under the communication restrictions:

$$TTU_\eta(z) \quad = \quad G((z^{o_\eta}, \vec{0})) - G(CL_\eta((z^{o_\eta}, \vec{0}))) \tag{5.3}$$

This utility is no longer factored with respect to the world utility, because the first term in the difference utility is $G((z^{o_\eta}, \vec{0}))$ instead of $G(z)$. While not being factored with world utility, TTU can have better learnability than BTU. This is because both terms are computed using the same truncated state, and thus the systematic error may be removed in the subtraction for certain types of world utility functions [76].

Continuing with the previous example, in this case the contribution of employees that are hidden from η will not appear in either term of TTU, since both terms are computed with the communication restriction. Therefore this utility will have good learnability, since the noise from the hidden employees will not clutter η's utility. As long as $G((z^{o_\eta}, \vec{0}))$ is sufficiently close to $G(z)$, this utility will be close to being factored and gains due to reduced noise will outweigh the loss in factoredness. However, if the assumption that $G((z^{o_\eta}, \vec{0}))$ is close to $G(z)$ does not hold (e.g, some hidden employees are crucial to the company's profit) then TTU will not produce good system performance.

5.2.3 Broadcast/Estimated Utility (BEU)

This utility is similar to BTU, except that instead of clamping the components of z^{h_η} to zero, their values are estimated given the values of z^{o_η}:

$$BEU_\eta(z) \quad = \quad G(z) - G(CL_\eta((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) \tag{5.4}$$

where $E[\cdot]$ is the expectation operator. As long as this estimate is not influenced by the actions of η beyond z_η, this utility is still factored, since the first term of the difference equation is still $G(\eta)$. While both BTU and BEU are factored, BEU may have less noise, depending on how good the estimate of z^{h_η} is.

As in the previous example, suppose that there are a large number of employees that are hidden from η, but that η can approximate their contribution to the company based on the employees that it can observe. In this case the first term of BEU will contain all of the employees' contribution to $G(z)$, but the second term will subtract out the hidden employees' inferred contribution. Even if effects of the hidden elements cannot be perfectly estimated, a lot of noise can still potentially be eliminated from the system. Note however that if the estimate is particularly poor, noise can also be introduced into the system.

5.2.4 Estimated/Estimated Utility (EEU)

This utility is similar to TTU, except that in both terms, the value of z^{h_η} is estimated:

$$EEU_\eta(z) = G((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}])) - G(CL_\eta((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) \qquad (5.5)$$

As was the case with TTU this utility is not factored with respect to the world utility G. However, with a good estimate of z^{h_η}, the value $G((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))$ will be much closer to $G(z)$ than $G((z^{o_\eta}, \vec{0}))$, so this utility can be much closer to being factored with respect to $G(z)$ than can TTU.

Following the example, EEU provides an advantage over TTU in that even if there are hidden employees whose actions strongly impact the company's profits, if the actions of those employees can be predicted, then EEU will be close to being factored. This utility retains the benefits of TTU (both terms computed the same way, leading to good learnability) while being much closer to being factored than TTU can be. Note that unlike with BEU, if the estimate of the hidden components is not particularly good, in general, noise will not be added to the system because both terms of the utility use the same estimate. Instead, the quality of the estimate only affects how close this utility is to being factored with respect to $G(z)$.

5.2.5 Team Formation

As discussed above, communication restrictions can have serious negative effects on the utility functions of the agents. One way to remedy this situation is to let agents form "teams" which "share" their knowledge of the worldline. More precisely the observable worldline for any member of a team will be the union of all the observable worldlines of the individual team members:

$$z^{o_T} = \bigcup_{\eta \in T} z^{o_\eta} \tag{5.6}$$

where T is the set of agents in the team and z^{o_T} is the observable worldline for the team. Note that team information sharing can be viewed as another form of communication and one can define the effective communication level of an agent in a team as:

$$B_{eff_\eta} = \frac{|z^{o_T}|}{|z|}. \tag{5.7}$$

However, this chapter will always refer to it as information sharing to differentiate from communication that happens between agents independent of the formation of teams. In real world situations team information sharing may have very different properties from general communication. It may have different constraints, different costs and may be imposed at different times in the creation of a system. There are also many different ways to form teams, but in this chapter will use a relatively simple model. A team is defined as an aggregation of agents where each agent:

- belongs to one and only one team;
- receives the utility of the team; and
- shares knowledge of the worldline with its team members.

Note that since all the members of the team share a utility, anything an agent does to help another agent in the team will also help itself. It will therefore assume that if possible the agents will share information with each other, whenever it is needed. Instead of having team members always share information, one could have each agent choose whether it wanted to share information. However the act of sharing would have to be added to the action space of the agent, possibly greatly increasing the size of that space, and therefore reducing the speed of learning. In addition adding information sharing to the action space would greatly increase the complexity of the problem as agents would gain and lose information continuously, based on the actions of other agents. Results for cases where agents can choose to share information are discussed in Section 5.5.

5.3 Bar Problem Experiments

To test the effectiveness of team formation and the new utilities, experiments were performed on the Modified Bar Problem, and the Time Extended Bar Problem (TEBP). Since this chapter concentrates on the effects of the utilities rather than on the RL algorithms that use them, very simple RL algorithms are used. One would expect that even marginally more sophisticated RL algorithms would give better performance.

For the Modified Bar Problem, each player η has a 7-dimensional vector giving its estimates of the utility it would receive for choosing each possible bar. The decisions are made using the vector, with an ϵ-greedy learner with ϵ

set to 0.05. All of the vectors are initially set to zero and there is a learning rate decay is 0.99. For the TEBP, each agent uses a Sarsa-learner. In every episode its 3 first rewards are zero and the last reward depends on the final bar attendance as computed in the single time step variant. The learner is an ϵ-greedy learner with ϵ set to 0.05 and $\gamma = 0.9$. The learning rate is set to $0.99^{v(s,a)}$ where $v(s, a)$ is a count of the number of times an agent took action a in state s.

5.3.1 Communication Restrictions and Team Formation

The communication restrictions are modeled in the bar problem by controlling how many other agents one agent can "talk" to. Without this communication the agent cannot know what the other agents have done. Here the communication level B will represent the fraction of all the agents to which an agent can talk. When $B = 1.0$ an agent can talk to the all other agents, whereas when $B = 0.0$ an agent has no communication, and thus is only aware of its own action. In the Bar Problem, communication restrictions are reduced to how $x_k(z)$ is computed. For truncated versions of the WLU, (BTU and TTU), $x_k(z^{o_\eta})$ is used, which returns how many of the observable patrons are going on bar k (note since in BTU the first term is broadcast, the agent does not need to compute it). For utilities using an estimate of the state (BEU and EEU), $x_k(z^{o_\eta})$ is scaled, and $\frac{1}{B}x_k(z^{o_\eta})$ represents the estimate of how many patrons actually went on bar k. For example when $B = 0.25$, it is assumed that $x_k(z^{o_\eta})$ is really only accounting for one quarter of the patrons, so it can be scaled by $\frac{1}{0.25} = 4$. Note this is an extremely simple estimation procedure and does not take any information an agent collects to modify how it forms this estimate.

Teams in the Bar Problem are modeled by creating disjoint groups of agents of approximately equal size. Every member of the team receives the same utility. In addition, the members of a team will be allowed to pool together all the information known by the team members. This means that each team member can get information about any agent that any of the team members can talk to. Therefore for the attendance for bar k that an agent η receives as a member of team k is: $x_k(z^{o_T})$ where $z^{o_T} = \bigcup_{\eta' \in T} z^{o_{\eta'}}$.

5.4 Results

The performance of the four versions of the WLU was tested with varying levels of communication, with and without teams. The test were conducted using the Bar Problem and Time Extended Bar Problem with 100 agents and with $c = 5$. All of the trials were conducted for 1000 episodes, and were run 25 times.

5.4.1 Communication Restrictions without Teams

The first set of experiments were conducted without teams (team size = 1). Figure 5.2 shows the performance of the four utilities with different levels of communication. When the communication level is high, the utilities converge to WLU so the resulting performance converges. When communication is very low, the BTU and BEU have the best performance because their first term G is not affected by the communication restriction. They essentially are reduced to using the global utility as their individual utility, and give moderately good performance. Note that the performance of BTU is worse at 50% communication than at 5%. This counterintuitive result is explained by how the utility is computed in the bar problem. With less communication, the total number of agents that can be seen is small, and the contribution of the second term is small. With 50% communication on the other hand, the second term will be large enough to have an impact on the utility. However, because both at 5% and 50% communication levels $x_k(z^{o_\eta})$ is significantly different than $x_k(z)$, neither provide a usable second term. In fact, rather than subtracting out noise, the second term adds noise.

For most levels of communication restriction, the EEU performs the best and performs up to 75% closer to optimal than utilities which use the same information. Recall that EEU and TTU are not factored, whereas BTU and BEU are. What helps EEU in this case is that though it is not factored, as long as the estimate for G in the first term is sufficiently close to G, it is close to being factored. Furthermore, because both the first and second terms use the same estimate for the state, the subtraction does remove noise, as intended. The utility TTU performs worst of all since, even though there may not be much noise in the utility, not only is it not factored with respect to the world utility, but due to the truncation, it may be very far from being factored.

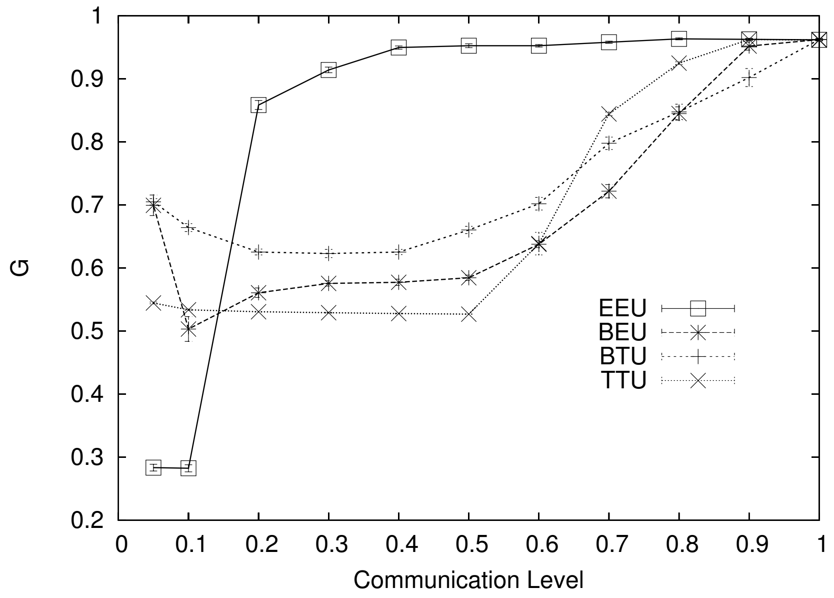


Figure 5.2: Performance of four utility functions without teams for a range of communication levels (including error bars). For moderate communication levels *EEU* performs best. For very low communication *BTU* performs best since, it uses information from world utility.

Figure 5.3 gives a clearer view of the performances at a fixed level of communication restriction (40% and 70%). *EEU* is clearly superior at 40% communication. At 70% communication *TTU* displays the problem with utilities that are not factored: the more the agents learn the worse the system performance becomes. Because this system is not factored (or in this case, not close to being factored), the agents maximizing their private utilities does not maximize the world utility. Ironically, because *TTU* has good learnability (i.e., the slope of *TTU* shows no sign of flattening out at $t = 1000$) the agents learn to do the wrong thing successfully. *BTU* and *BEU* on the other hand are factored so G does not decrease. However, because of learnability issues,

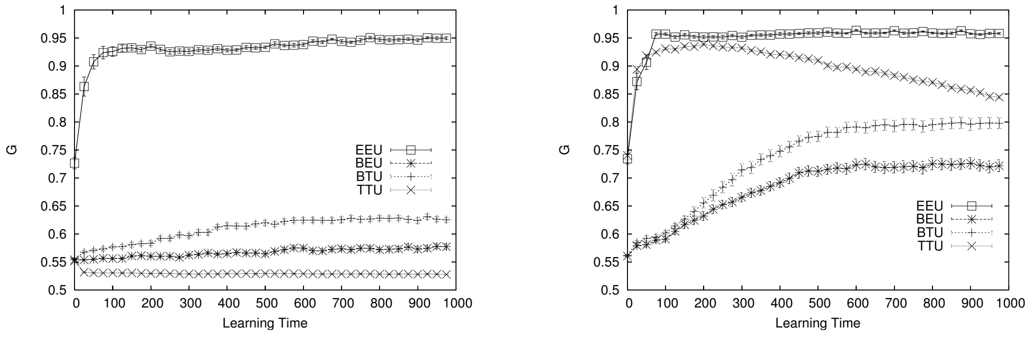


Figure 5.3: Learning rates of four utility functions at 40% communication (left). *EEU* learns far quicker, since it produces a much less noisy signal. Note that even though *TTU* is highly learnable, it is not close to being factored with respect to G, so it has a flat learning curve. At 70% communication (right) *TTU* is closer to factored and can learn quickly, but still sightly non-factored, causing performance to eventually go down.

after an initial period of improvement, the agents encounter a difficult signal to noise problem and the system performance stops improving.

5.4.2 Communication Restrictions with Teams

Even using the best utility, *EEU*, a high level of performance cannot be achieved if the communication level is too low. However if agents can form small teams where information sharing is allowed between team members, good performance is possible even when communication between teams is low. Figure 5.4 shows the tradeoffs between choices of team size at different levels of communication. At most communication levels, there is an optimal team size that lies between the extremes of not having teams (team size = 1), and only having a single team (team size = 100). The best team size is typically around 5 or 10 agents. This optimum represents to best balance between having small team sizes which produce a more learnable utility and large team sizes which allows for more information sharing.

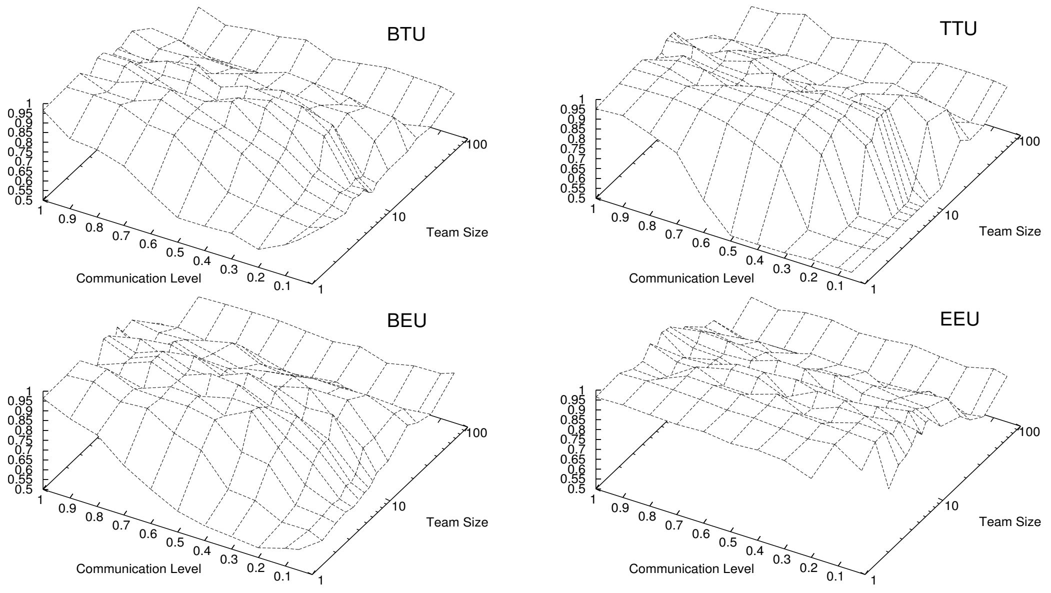


Figure 5.4: Performance with different team sizes and communication levels. Each graph is for a different utility. From top-left, clockwise the utilities used are: BTU, TTU, EEU, BEU. The two utilities, BEU and EEU, that estimate hidden values rather than ignoring them, perform much better than their conterparts, BTU and TTU.

With the non-factored utilities EEU and TTU this balance comes from the tradeoff between factoredness and learnability. Even though as team sizes get smaller, the utilities become more learnable, they also become less factored since as information sharing goes down, the first term in the difference equation diverges from G. For the factored utilities BEU and BTU there is a tradeoff between two different ways noise comes into the system. When teams are large, more components have to be clamped in the second term of the difference equation allowing more noise from the first term to remain. When teams are small, the lack of information sharing has a similar effect, in that many of the components in the second term are clamped because their values are unknown.

Figure 5.5 shows that even when having teams is possible, the choice of utility is still critical. As in the case without teams, the EEU tends to perform best under most team sizes. Even though it is not factored, it has up to 25% higher performance than a $g_\eta = G$ system. Only with very small team sizes do the factored utilities perform better. When team sizes are very large, there are no hidden agents, so all the utilities converge to the same values. Due to the high learnability of EEU, its superiority is even more pronounced when the agents do not have much time to learn as shown in figure 5.5 (right).

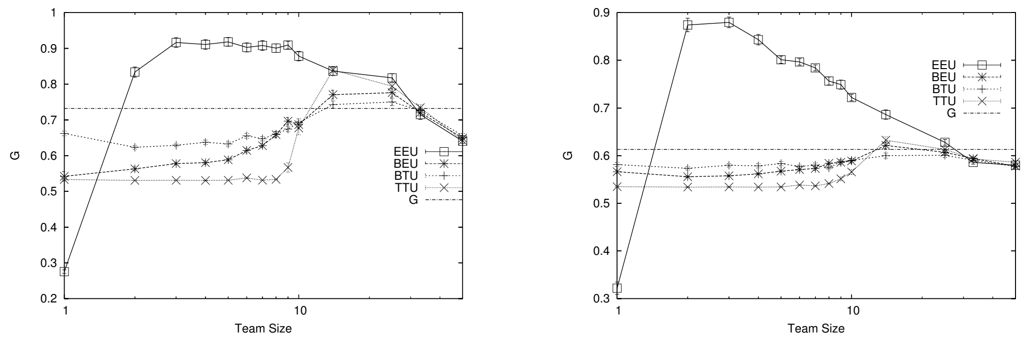


Figure 5.5: Performance of four utility functions at 10% communication. EEU performs best for most team sizes under normal learning time (left).The signal to noise advantages of EEU become more apparent when learning time is reduced to 1/8 of original time (right).

5.4.3 Time Extended Results

To test the effectiveness of our methods on a more difficult problem, the same experiments were performed on Time Extended Bar Problem. This problem is harder since it is a Markov Decision Process, unlike the conventional Bar Problem which is a single step problem. In this problem agents have to find the best sequence of four actions instead of just a single action. To maximize comparability between the two problems, the time expended problems were tested identically to the non-time extended problems, except that they were conducted over 4000 learning steps instead of 1000.[1]

Figure 5.6 shows that the time extended problem is significantly harder. In trials with large team sizes, the agents were unable to learn at all on this problem. This happens because when the teams were large, the signal-to-noise ratio of the agents' utilities went down since the utilities contain the noise from all the other agents on a team. The signal-to-noise problem is a bigger issue with the Time Extended Bar Problem than with the original Bar Problem, since the noise is compounded in every time step. Even if an agent were able to take the correct action in the last three time steps, it may perform poorly if a noisy utility caused it not to take the correct action in the first time step. Also agents using utilities BTU and BEU suffered at low communication levels because the amount of noise in these utilities goes up as the communication level goes down, since less of the noise from other agents gets subtracted out. Figure 5.7 shows that at 5% communication, the only effective utility is EEU. All the other utilities have very low performance, resulting in actions that are not much better than random for most team sizes. This situations contrasts to

[1]both experiments were conducted for 1000 episodes, but the Time Extended problem has four steps per episode.

the the easier non-time-extended problem, where many of the utilities would result in a reasonable performance, especially with large team sizes.

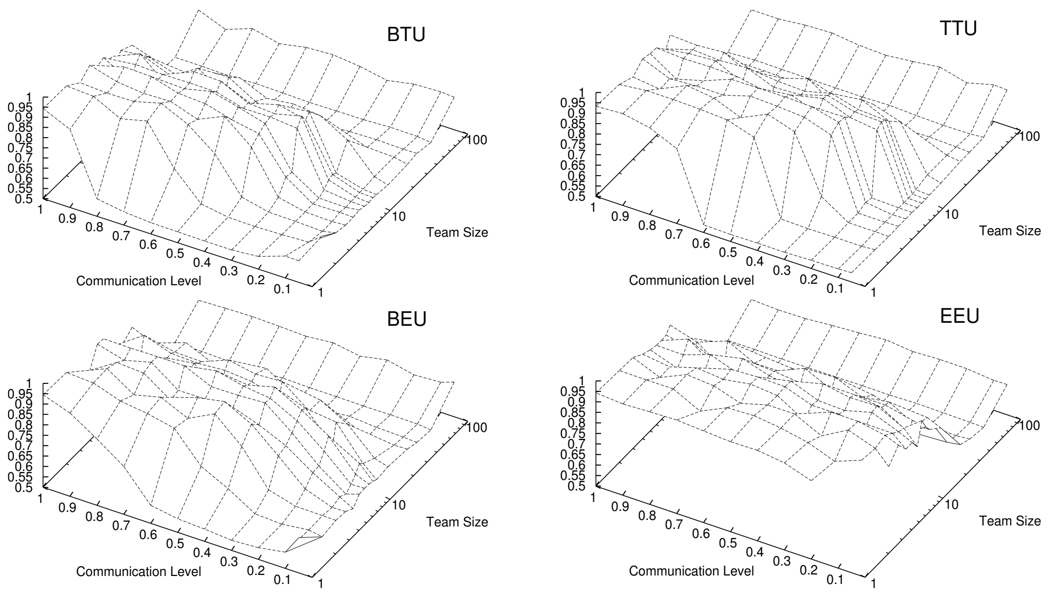


Figure 5.6: Performance with different team sizes and communication levels for Time Extended Bar Problem. Each graph is for a different utility. From top-left, clockwise the utilities used are: BTU, TTU, EEU, BEU. The two utilities, BEU and EEU, that estimate hidden values rather than ignoring them, perform much better than their conterparts, BTU and TTU. Note that with large team sizes, agents can not effectively learn in the time extended problem

5.4.4 Information Sharing Costs

In the previous experiments there is no cost for sharing information with team members. Since all agents always share information with fellow team members, any sharing cost will be constant and will cancel out in the difference utility. Therefore information sharing cost will not influence the

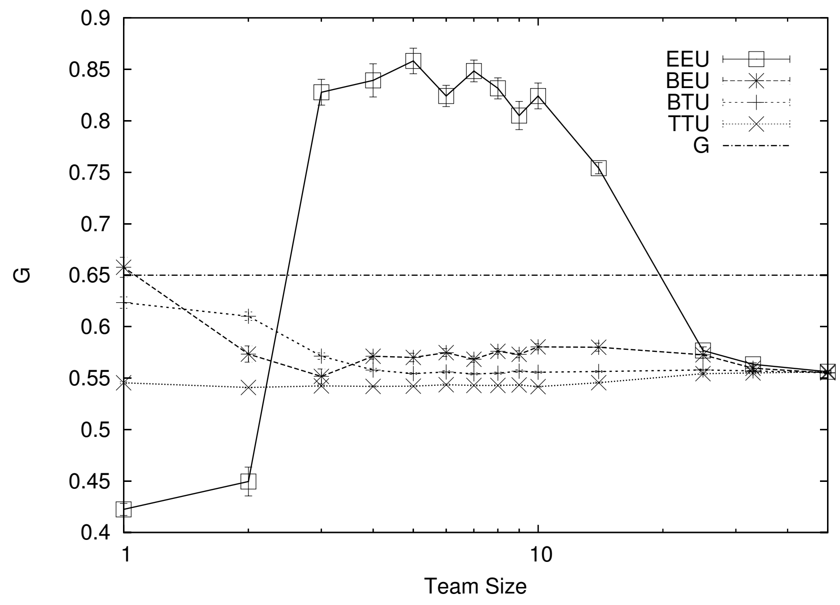


Figure 5.7: Performance of four utility functions at 5% communication. Superiority of *EEU* is even more clear in the time extended problem.

actions of the agents.

Another approach to information sharing is allowing the agents to decide whether or not they will share information, and charge them every time they decide to share. The difficulty of this approach is that it greatly increases the complexity of the system. Not only do the agents have an increase in their action space, but also their ability to compute their utility continuously changes as team members decide whether or not to share. To reduce noise one could abandon teams and let agents have different utilities. While not being in teams, agents could be in groups that share information. However, if each agent's utility is only factored to the world utility, and not factored to the group, then they may never choose to share information.

This approach was tested with two utilities:

$$TCEEU_\eta = EEU_{T_\eta} - |T_\eta| c \tag{5.8}$$

$$CEEU_\eta = EEU_\eta - c \tag{5.9}$$

where c is the information sharing costs, T_η is the set of agents that are on the same team as agent η and $| \cdot |$ is the set cardinality operator. The first utility is simply the EEU used previously minus the sharing cost. The second utility is like the $TCEEU$, except that only the individual agent is clamped. Agents using this second utility are not in teams, since each agent has its own utility. Note that the $CEEU$ is likely to have much less noise than the $TCEEU$ since only a single agent is clamped instead of an entire team. This difference in signal-to-noise level of the two utilities will increase with larger team sizes.

To test the effectiveness of these utilities, experiments were performed where the cost of information sharing was $c = \frac{m\hat{G}}{100n}$ where there are m agents in each group out of a total of n agents, and $\hat{G}$ is the maximum possible value of G.[2] Figure 5.8 shows that agents that could choose whether to share information performed worse than agents that always shared information. The agents that were not in teams (using $CEEU$) had the lowest performance, since they rarely chose to share information. This low performance may be surprising since the $CEEU$ has much less noise than the other utilities. However since these agents were not sharing information, their estimate of the worldline was

[2]The maximum average utility per agent is $\hat{G}/n$. With a 1% sharing cost per each agent, the cost is $\hat{G}/100n$. When sharing with m group members the cost is $m\hat{G}/100n$.

much worse, therefore the $CEEU$ was not be very close to being factored. Since the utility had less noise, the agents are learning quickly, but since the utility was not factored they were learning to take the wrong actions.

Agents in teams (using $TCEEU$) that chose whether to share information performed significantly better, since they eventually learned that sharing was helpful to the team, and therefore chose to share most of the time. However agents using $TCEEU$ that always shared information performed the best, since there was much less noise in this scenario. When agents could choose to share, the utility was less stable since it could change dramatically depending on whether other agents in the team chose to share at a particular time step. Note that these results shown here illustrate what can happen when the cost

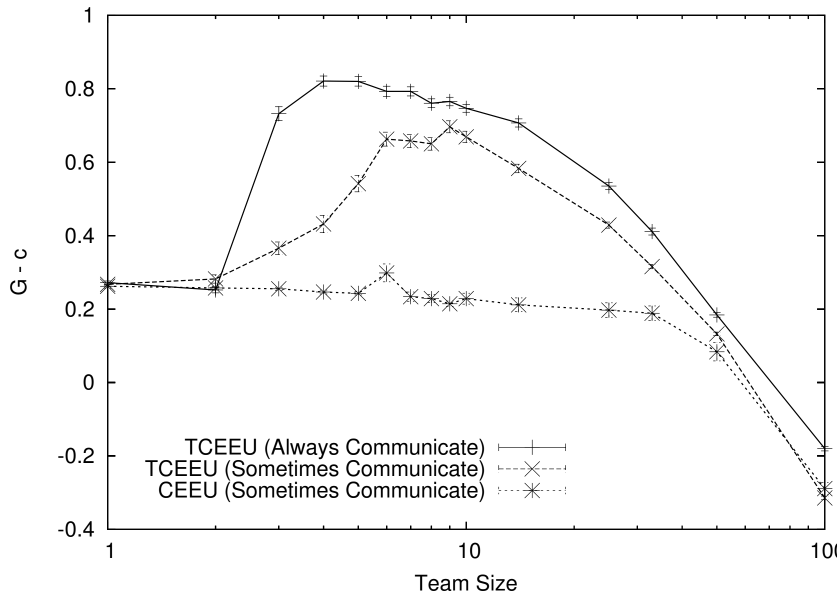


Figure 5.8: When agents can choose whether or not to share information, the problem becomes much more difficult and performance goes down.

of sharing is at a particular level. If the cost of sharing changed, the results could change. For instance, in a system that had a very high cost of sharing information, the the systems that always communicated could perform worse than systems that could choose whether or not to communicate.

5.5 Discussion

This chapter focuses on the problem of designing a collective of autonomous agents in the presence of severe communication restrictions. In such cases, private utilities which rely on agents having access to a fully connected communication network may break down. Four different utility functions were presented that each make different tradeoffs among what is available to an agent and how that information should be used. It was shown that one utility in particular, EEU, does far better than all the others in almost all experiments. Agents using this utility learn much faster and achieve better results on the traditional and time extended variants of the Bar Problem. Furthermore agents using this utility can perform well with far more restricted communication.

In addition, this chapter showed that team formation improves the performance of the system by as much as 25% on top of the 75% performance increase achieved by using better utilities. This was due to the increased information available to the members of a team, which alleviated the communication restriction imposed on the agents.

This performance boost was achieved using a simple team model, where team members which had a common utility always chose to share information. Also for simplicity, there were no costs to share information. In certain domains sharing costs may be significant, but in many cases this cost will not effect the applicability of our utility functions. For example if the sharing cost is solely a function of team size, it will cancel out in the difference utilities. In these domains the collective designer should include the cost of sharing in the global utility and adjust the group size so as to maximize performance in the specific domain.

Furthermore, in many problems agents can choose whether to share information or not, and consequently incur a cost or not. Results show that in such cases, agents have a difficult time in learning to maximize their utility functions. This is due to the constant change in how an agent perceives the world, which now depends on the sharing choices of many other agents. This in effect creates a more noisy learning environment.

Chapter 6

A Multi-agent View of Reinforcement Learning for Markov Decision Processes

Single agent reinforcement learnings in time-extended domains and multi-agent systems share a common dilemma, known as the credit assignment problem [63]. Multi-agent systems have the structural credit assignment problem of determining how an agent contributes to the global utility. Effective credit assignment is critical for the agents to learn quickly and to learn to collaborate. Utilities coming from the theory of collectives can be seen as tools for supplying credit assignment. For example the WLU in a loose sense returns the individual agent's contribution to the global utility without a lot of spurious noise.

In a single agent domain, the temporal credit assignment problem is concerned with how an action taken at a particular time step affects the final outcome. For example, if a player wins a game of checkers, it may be difficult for that player to determine which of his many moves were the most important in helping him win, and which moves may have actually been detrimental. Many reinforcement learning heuristics have been made to try to assign proper credit assignment. The goal of these heuristics is to make the learner converge to the correct policy, in a speedy manner, or to at least make a good tradeoff between correctness and speed. Many of these heuristics can be seen as a tradeoff between factoredness and learnability, where an agent designer may be satisfied with the agent not converging to the optimal policy as long as it converges quickly to a reasonable policy.

This chapter will pose the single agent time-extended problem as a multi-agent single time step problem, transferring the temporal credit assignment problem into a structural credit assignment problem. This credit assignment problem will then be solved using the theory of collectives. In many cases the collective solution will be equivalent to popular reinforcement learners. However this equivalence will be valuable, since it will allow users to pose many problems either as a structural or a temporal credit assignment problem, and choose the one that is easiest to work with. In addition, this equivalence lays the basis for solving time-extended collectives, where both credit assignment problems are present.

6.1 Setup

Consider a Markov Decision Process (MDP) for a single agent. This chapter will convert this into a multi-agent problem, but to avoid confusion, the term "node" will be used for the newly constructed agents, and the term "MDP agent" will always refer to the original single agent. Let the number of nodes in a collective equal the number of states in the MDP, with a one-to-one mapping between states and nodes. The state associated with a node η is indicated as $s(\eta)$. Let the action space of η be the transitions out of state $s(\eta)$. For each episode, all the nodes of the collective perform a single action. The action a node, η, takes is determined by the policy table for the node, $Q(s(\eta))$, where $Q(s(\eta), a)$ is the value of node η's table for action a. The collection of all of the nodes' policy tables will be referred to as Q. Note that Q can also be seen as the policy for the entire MDP.

The worldline for an episode is formed by having the MDP agent follow the actions specified by the nodes until a terminal state is reached. Let R be the vector of rewards received by the MDP agent and R_t be the reward received at time step t. Let S be the vector of states entered by the MDP agent and S_t be the state entered at time step t. The worldline for an episode is composed of all the rewards received by the MDP agent, the states entered by the MDP agent and the internal learning parameters of the nodes. Formally the worldline z is represented as:

$$z = (R, S, Q) \tag{6.1}$$

The world utility is equal to the undiscounted sum of rewards:

$$G(z) = \sum_t R_t \tag{6.2}$$

Now the utilities for the nodes need to be computed. Let $T(\eta)$ be the first time that the MDP agent was in state $s(\eta)$. It is assumed that all the rewards after time $T(\eta)$ are hidden from η [1]. It is also assumed that all compo-

[1]For simplicity it is assumed that even if state $s(\eta)$ is entered more than once that η only knows the value of the first reward received. If this assumption is not used, the utilities will represent "average visit" methods instead of "first visit" methods [65].

nents of S are hidden other than $S_{T(\eta)+1}$. Despite these hidden components it will be assumed that $G(z)$ may be known. Note that the observable worldline for a node η in which state $s(\eta)$ was never entered is undefined since $T(\eta)$ will be undefined. These nodes will receive no reinforcement at all.

As an example take the gridworld problem (Figure 6.1). In the classic problem the agent can move from grid square to grid square, until it reaches a terminal state. The agent can move in four directions, and the state is determined by which grid square the agent is in. This problem can be broken down into nodes, where each grid square is assigned a node. At the beginning of the episode each node independently chooses an action from one of four possible moves. The MDP agent then follows the actions chosen by the nodes, until it reaches the terminal square. All the nodes associated with grid squares that the MDP agent actually went through are updated at the end of the episode.

6.2 WLU and Monte-Carlo Estimation

While G could be used for the node utilities, it would cause them to learn very slowly. In fact this solution would solve the MDP much slower than standard reinforcement learning methods. A more learnable utility is the WLU, defined as:

$$WLU_{\eta} = G(z) - G(CL_{Eff_{\eta}}(z)) \tag{6.3}$$

where CL is the clamping operator and Eff_{η} is the effect set for node η. For now it will be assumed that the clamping is to the zero vector and that the

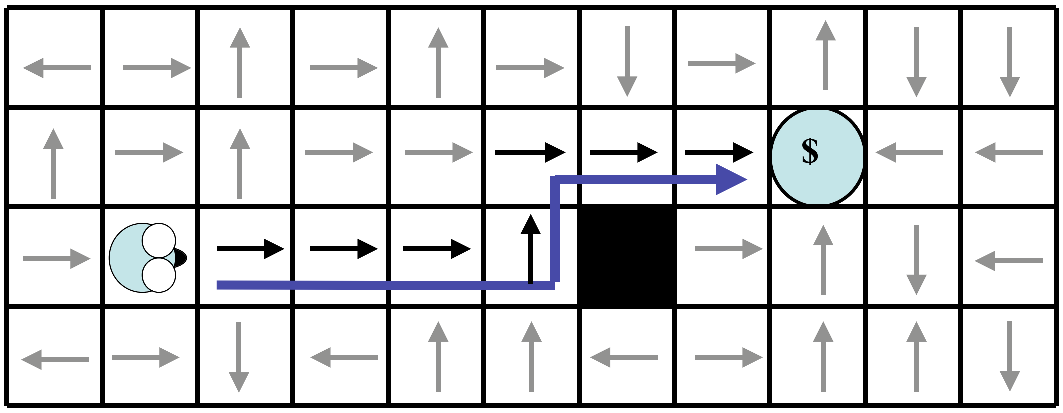


Figure 6.1: Classic Gridworld Problem broken down into multiple nodes. Each grid square is assigned a node, which chooses an action at the beginning of each episode. The agent then follows these actions. Nodes from squares agent went through are updated (black arrows).

effect set includes all rewards and states after and including time $T(\eta)$:

$$Eff_\eta = \bigcup_{t \geq T(\eta)} (R_t, S_t) \tag{6.4}$$

This effect set is a worst-case scenario since a decision a node makes at time step $T(\eta)$ could change the entire path of the MDP agent, and therefore change all of the future rewards. However an action by η cannot affect rewards that were already received, prior to time step $T(\eta)$. Note that the effect set includes all of the hidden components of z so that the second term of WLU can be computed despite the communication restriction[2]. From equations 6.2, 6.3,

[2]It is assumed that the Eff operator is defined for hidden elements. If not the BTU and BEU utilities can be used as shown later.

6.4 one can simplify WLU as follows:

$$\begin{aligned} WLU_\eta &= \sum_t R_t - \sum_{t<T(\eta)} R_t & (6.5) \\ &= \sum_{t \geq T(\eta)} R_t & (6.6) \end{aligned}$$

When WLU is used, the collection of nodes is equivalent to a system where the update rule for each state is the sum of the rewards after that state was entered. This corresponds to "first visit" monte-carlo estimation where Q-table updates are defined in the same way.

6.3 TTU and Immediate Rewards

It is also possible to have the nodes use the four utilities designed for domains with communication restrictions: BTU, TTU, BEU, EEU. Because of the clamping, the communication restriction has no effect on the second term, since all the hidden components are clamped in the second term. Therefore the second term of the four utilities is the same as the second term for WLU. The utilities WLU, BTU and BEU are equivalent. However the two utilities that do not use broadcast information, TTU and EEU, are different. TTU is defined as:

$$\begin{aligned} TTU_\eta(z) &= G((z^{o_\eta}, \vec{0})) - G(CL_{Eff_\eta}((z^{o_\eta}, \vec{0}))) & (6.7) \\ &= \sum_{t \leq T(\eta)} R_t - \sum_{t<T(\eta)} R_t & (6.8) \\ &= R_{T(\eta)} & (6.9) \end{aligned}$$

The second term is the same as the second term in WLU where all the rewards from times equal to or greater than $T(\eta)$ are clamped. In the first term all the rewards greater than $T(\eta)$ are truncated due to the communication restriction. This leaves only the immediate reward, $R_{T(\eta)}$, remaining after the subtraction. Note that this is not a good utility for an MDP, since it can cause the node to take actions that maximize the immediate reward at the cost of future rewards.

6.4 EEU and Q-Learning

The EEU can be computed as follows:

$$\begin{aligned} EEU_\eta(z) &= G(((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) - \\ &\qquad G(CL_{Eff(\eta)}((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) && (6.10) \\ &= G(((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) - \sum_{t<T(\eta)} R_t && (6.11) \end{aligned}$$

To compute the first term of this utility it is necessary to estimate the values for the set of rewards contained in the hidden components of z. The notation $\hat{R}_t$ will be used to refer to the estimate of the reward at time t. The EEU can now be simplified as follows:

$$\begin{aligned} EEU_\eta(z) &= G(((z^{o_\eta}, E[z^{h_\eta}|z^{o_\eta}]))) - \sum_{t<T(\eta)} R_t && (6.12) \\ &= \sum_{t\le T(\eta)} R_t + \sum_{t>T(\eta)} \hat{R}_t - \sum_{t<T(\eta)} R_t && (6.13) \\ &= R_{T(\eta)} + \sum_{t>T(\eta)} \hat{R}_t && (6.14) \end{aligned}$$

If the agent will follow the optimal policy it is possible to estimate $\sum_{t>T(\eta)} \hat{R}_t$ as $\max_a Q(S_{T(\eta)+1}, a)$. The utility EEU can then be represented as

$$EEU_\eta(z) = R_{T(\eta)} + \max_a Q(S_{T(\eta)+1}, a) \tag{6.15}$$

A system of nodes that uses this definition of EEU is performing Q-learning[3].

6.5 Discounting as Uncertainty in Credit Assignment

An alternative way to model effect sets is to use a worldline with an "effect mask". The new worldline will be referred to as $z' = (z, m)$ where z is the original worldline and m is the effect mask. The effect mask will consists of node effect masks, m_η, each of which has the same number of elements as z. Each element of the effect mask m_{η_i} will correspond to the element z_i. If node η effects worldline element z_i then $m_{\eta_i} = 0$, otherwise $m_{\eta_i} = 1$. The clamping operator, $CL_\eta(z, m)$, will return (z'', m) where for each element i, $z''_i = z_i m_{\eta_i}$. It will be assumed that the global utility will ignore the effect masks. If the elements of the effect masks are not observable then it is not possible to compute WLU. However, one can compute BEU and EEU. To compute the second term of these utilities it is necessary to estimate the values of the effect masks corresponding to reward values. The effect element for each reward element R_t will be referred to as $m_{\eta_{R_t}}$. Let p_t be the probability that $m_{\eta_{R_t}} = 0$. If it is assumed that the probability that a node affects a future

[3]Here the state is only being updated the first time it is entered. Most forms of Q-learning do an update every time the state is entered. However since Q-learning is off-policy, this makes no difference as far as convergence

reward will fall exponentially with time, then one can make the following probability assignments:

$$\forall_{t<T(\eta)}\, p_t \;=\; 1 \tag{6.16}$$

$$\forall_{t\geq T(\eta)}\, p_t \;=\; \gamma^{t-T(\eta)} \tag{6.17}$$

The expected value of $m_{\eta_{R_t}}$ is $1-p$, since it is a Bernoulli random variable. BEU can now be computed as follows:

$$\begin{aligned} BEU_\eta &= \sum_t R_t - (\sum_{t<T(\eta)} R_t + \sum_{t\geq T(\eta)} (1-\gamma^{t-T(\eta)})R_t) && (6.18)\\ &= \sum_{t\geq T(\eta)} R_t - \sum_{t\geq T(\eta)} (1-\gamma^{t-T(\eta)})R_t && (6.19)\\ &= \sum_{t\geq T(\eta)} \gamma^{t-T(\eta)} R_t && (6.20) \end{aligned}$$

Each node therefore will maximize a discounted sum of rewards, when using BEU. This provides another interpretation to the discounting of rewards other than the interest rate interpretation commonly used [36]. Here the discounting comes from uncertainty in how an agent's action affects future states. It is an uncertainty in credit assignment.

6.6 Discussion

This chapter unified the structural credit assignment problem present in single-time-step multi-agent systems and the temporal credit assignment present in time-extended single-agent systems. It did this by showing the relationship between the three utilities, WLU, EEU and TTU to the three different reinforcement learning methods, monte-carlo estimation, Q-learning and immediate reward learning, respectively. In each case the relation between the utility and the reinforcement learner was made through a specific construction of a collective. However the relation goes deeper than this specific construction as the utilities and their associated RL exhibit many of the same salient properties. The TTU which is not structurally factored is related to an immediate reward which will break a time-extended system, since it is not factored with time. Similarly the EEU depends on good estimates to become close to being factored, like the Q-learner which depends on good Q-function estimates. When the Q-function estimate is poor, Q-learners will often diverge, similar to the case where the hidden component estimate of the EEU is poor causing the performance of the collective to collapse.

The unification of the credit assignment problems can lead to ways of solving problems that are both time-extended and mutli-agent. However the temporal credit assignment problem and structural credit assignment problem is entangled. This leads to some difficulties as the complicated effect sets between nodes of the same MDP agent and nodes between different MDP agents are difficult to analyze. Chapter 8 will address this problem by separating the two credit assignment problems, allowing Q-learning to be used in a multi-agent system. However, many other solutions are possible, some of which cannot relate to Q-learning directly.

Chapter 7

Genetic Algorithms and Collectives

This chapter will show that if a singe-agent problem is structured in such a way that it can be solved with a genetic algorithm (GA), then it can naturally be solved by a collective, using simple evolutionary learners. Evolutionary algorithms and genetic algorithms usually work with strings of bits, with each string encoding a solution to a problem. A single bit-string is often referred to as a *chromosome* and is partitioned into *n genes.* Genes have a fixed location on a chromosome, which can be specified by a beginning and end index of the bit-string. The actual bit pattern for a particular gene location on a chromosome is referred to as an *allele.* In many GA problems, the genes are carefully chosen to refer to meaningful structures in the problem solution. While the structures, specified by the alleles, may be somewhat independent, the utility of one allele typically depends on other alleles present in the chromose. Due to this interdependence, how to chose a good collection of alleles is a natural problem for a collective, where each member of the collective can chose an allele. This chapter will refer to the bits that correspond to an allele in the GA solutions, instead as a *sub-bit-strings* while the full solution will be referred to as a *bit-string.* Each member of the collective will chose a sub-bit-string, and the concatenation of all the sub-bit-strings will form the bit-string for the full solution. The goal of each member of the collective will be to chose a sub-bit-string that will lead to the highest possible global utility, given the choices of all the other members.

Genetic algorithms are concerned with discovering a chromosome that works well for a particular task. This chapter will show how the search for high performing chromosomes can be cast as a multi-agent learning problem, with a mapping from agents to genes. It will then show how improvements can be achieved relative to existing genetic algorithms. To avoid confusion, this chapter will always refer to these agents as “nodes.” Instead, the term “agent” will be used in a traditional sense referring to the entity, such as a rover, that has some of its characteristics specified by a chromosome. For example, consider a simple problem where there is a rover that is defined by two genes: one for height and one for color. In this problem there would be two nodes. The action space of one of the nodes would be a choice of color and the action space of the other node would be a choice of height. The actions of the collective define a full solution. There are many ways that the nodes can make their choices and may be limited by the number of height sub-bit-strings and color sub-bit-strings available as shown in Figure 7.1.

7.1 Multi-Time-Step Episodic Tasks

Consider a task where an agent takes a series of actions over time until it reaches a goal state after T time steps. During the episode the agent enters T states and receives T rewards. In a general, possibly non-markovian domain,

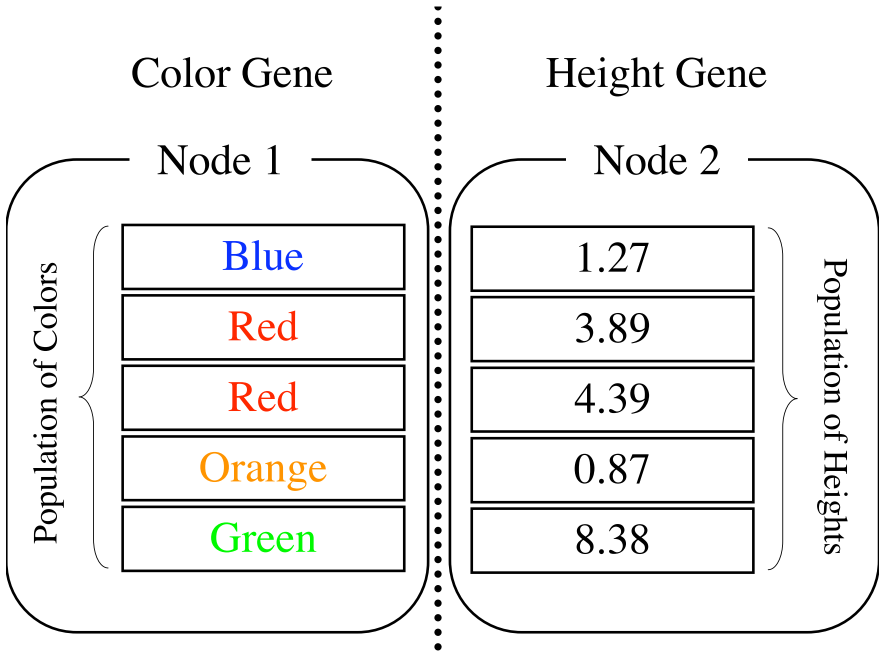


Figure 7.1: Simple genome with two genes. Each node in the collective choses a bit-string associated with a gene. The choice of the two nodes represents a full solution.

each reward is a function of all the states that preceded it. Each state is a function of all the actions that preceded it. Each action is a function a bit-string that controls the agent. In a GA, this bit-string is the chromosome that is produced by the system. In the collective the bit-string will instead be a concatenation of the bit-strings chosen by the nodes. Let B be this bit-string. This bit-string is a concatenation of n sub-bit-strings, where each sub-bit-string, B_η, is chosen at the beginning of the episode by a node η:

$$B = (B_1, B_2, ..., B_n) \tag{7.1}$$

Note that the location of each sub-bit-string, B_η, in the full bit-string corresponds to a gene. The global utility can now be written as follows:

$$\begin{aligned} G(z) &= \sum_t R_t(z) & (7.2) \\ &= \sum_t R_t(s_{\leq t}(z)) & (7.3) \\ &= \sum_t R_t(s_{\leq t}(a_{<t}(B))) & (7.4) \end{aligned}$$

where $s_{\leq t}(\cdot)$ is the state function, which computes a list of the agent's states before and including time t from the actions. In turn the action function, $a_{<t}(\cdot)$, computes the actions taken before time t as a function of B.

The task of the collective is to produce the bit-string, B, that induces the highest possible value of G. To do this each node has to be able to evaluate the B_η it chose at the beginning of the episode. In most genetic algorithms there are not separate evaluations for each allele. Instead there is an evaluation for an entire chromosome, which is usually the global utility for the episode.

7.2 Node Utilities

After an episode a node needs a way to evaluate its action (choice of B_η). A node can do this by calculating its utility, based on the rewards and state transitions that were recorded during the episode. One possible choice of utility is the global utility, G. This would be similar to the evaluation a genetic algorithm makes on a choice of chromosome. However, the global utility may be a poor choice, since as discussed in previous chapters, it is hard to learn from in many domains. If there are many nodes, the global utility may give little information about the contribution of a single node. Another choice of utility is the Wonderful Life Utility:

$$WLU_\eta(z) = \sum_t (R_t(B) - R_t((B_{\hat{\eta}}, C_\eta))) \tag{7.5}$$

where C_η is the clamping bit-string for agent η. If $R(B)$ is relatively smooth in the vicinity of B then the WLU will usually cancel out much of the noise caused by other nodes, and will return approximately η's contribution to the global utility. However there are many problems with this approach. First of all $R(B)$ is usually a very complex function of B, requiring the computation of the dynamics of the system. Also due to this complexity, $R(B)$ may not be very smooth in the vicinity of B and therefore WLU may not be able to eliminate much noise from the utility. In addition the dynamics of the system may not be known, or too cumbersome to compute. Therefore it may not possible to compute the rewards directly from the chromosome, making it impossible to compute the WLU in its current form.

Rather than computing WLU directly from B it is possible to make an estimate of WLU, using the reward and state values recorded during the episode. Let $R = \bigcup_t R_t$ be the set of rewards received during the episode and $S = \bigcup_t S_t$ be the states entered during the episode. The first term of the WLU can then be trivially computed as $\sum_t R_t$. Computing the second term is more difficult since it is hard to determine how replacing B_η with C_η affected the reward functions. However it is possible to find out for certain time steps which $R_t((B_{\hat{\eta}}, C_\eta))$ are almost equal to R_t, by looking at the sensitivity of the agents actions to B_η. Let the *action sensitivity* at time t be defined as:

$$L_{\eta,t}(B) = \frac{\delta a_t((B_{\hat{\eta}}, B_\eta))}{\delta B_\eta} \tag{7.6}$$

In addition define T_η as the first time step in which $L_{\eta,t}(B)$ is greater than a threshold τ. For all $t < T_\eta$, the choice of B_η has little influence on the agent's actions and therefore the agents rewards. It is therefore possible to approximate $R_t((B_{\hat{\eta}}, C_\eta)$ as R_t for all $t < T_\eta$. If the values of $R_t((B_{\hat{\eta}}, C_\eta)$ for all $t \geq T_\eta$ are crudely approximated to zero then the WLU can be computed as follows:

$$\begin{aligned} WLU_\eta &= \sum_t R_t - \sum_{t<T_\eta} R_t && (7.7) \\ &= \sum_{t \geq T_\eta} R_t && (7.8) \end{aligned}$$

7.2.1 Monte Carlo Estimation

Suppose that the state space is partitioned into n disjoint regions, with each region, S_η, being associated with node, η. Suppose that the action of the agent when it is in region S_η is solely determined by B_η. In this case the sensitivity $L_{\eta,t}(B)$ will be zero for every time step in which the agent is not in region S_η. Therefore T_η will be the first time step that the agent entered region S_η.

Consider the special case where the action and state space is discrete and each node is associated with a state. In addition let each sub-bit-string encode an action. In this case a node using WLU will be using the sum of future rewards to evaluate its action. Therefore a collective using WLU in this case will be performing “first visit” monte-carlo estimation.

7.3 RBFs for Markov Decision Problems

This section will discuss a collective based solution for continuous Markov Decision Problems, in domains where it is not possible to calculate $R_t((B_{\hat{\eta}}, C_\eta))$. In this solution, the actions that the agent takes will be determined by a neural network that maps states to actions. For simplicity it will be assumed that the action is one-dimensional therefore there is only one output unit. The weights of the neural network will be determined by the chromosome. The most important choice to be made in this problem, is the type of neural network that will be used.

7.3.1 Multi-Layer Perceptron

Consider a two layer Multi-Layer Perceptron (MLP) where each sub-bit-string, B_η, determines one of the network weights [1]. At each time step the

[1]Instead if each B_η determines a hidden unit, the collective will have a close similarity to the “Neuron” SANE (Symbiotic Adaptive Neural Evolution) and ESP (Enforced Sub-Populations) methods [31, 44]. Like ESP each node maintains its own “sub-population”, and no mixing of population members will ever be done between sub-populations, since the nodes are independent. A collective can be made closer to the SANE algorithm if nodes can share their populations. SANE and ESP use a global utility. Collectives that use more

current state is fed into the input layer of the MLP, and the action for the agent in that state is taken from the output layer as shown in Figure 7.2. For MLPs with sigmoid activation functions, the output of the network is:

$$a(s) = g(\sum_i w_i \phi_i(s)) \tag{7.9}$$

where $\phi_i(s)$ is the evaluation for hidden unit i and $g(x)$ is the sigmoid function which equals $\frac{1}{1+e^{-x}}$. The action sensitivity for a time t is therefore:

$$\begin{aligned} L_{\eta,t}(B) &= \frac{\delta a_t((B_{\hat{\eta}}, B_\eta))}{\delta B_\eta} \\ &= g(\sum_i w_i \phi_i(s_t))(1 - g(\sum_i w_i \phi_i(s_t))) \frac{\delta \sum_i w_i \phi_i(s_t)}{\delta w_\eta} \\ &= g(\sum_i w_i \phi_i(s_t))(1 - g(\sum_i w_i \phi_i(s_t))) \phi_\eta(s_t) \end{aligned}$$

Note that the action sensitivity will be low either when the network is saturated or the activation of the hidden unit is low. If the network is saturated, little information is going to be gained from the episode, and some external mechanism will usually have to be applied to get the system out of saturation. In general there is no reason to expect the output of the hidden unit to be low, so that the first time a node's action has significant impact on the output of the network is likely to be very early in the episode. Therefore the value of T_η is likely to be close to zero for most nodes. This problem arises from the highly distributed nature of MLPs, where every weight will have an influence on the network output at most time steps. In this case the second term of the

learnable utilities such as WLU more closely resemble Eugenic Algorithms [6].

WLU will be close to zero and the WLU will essentially be the global utility resulting in a credit assignment problem.

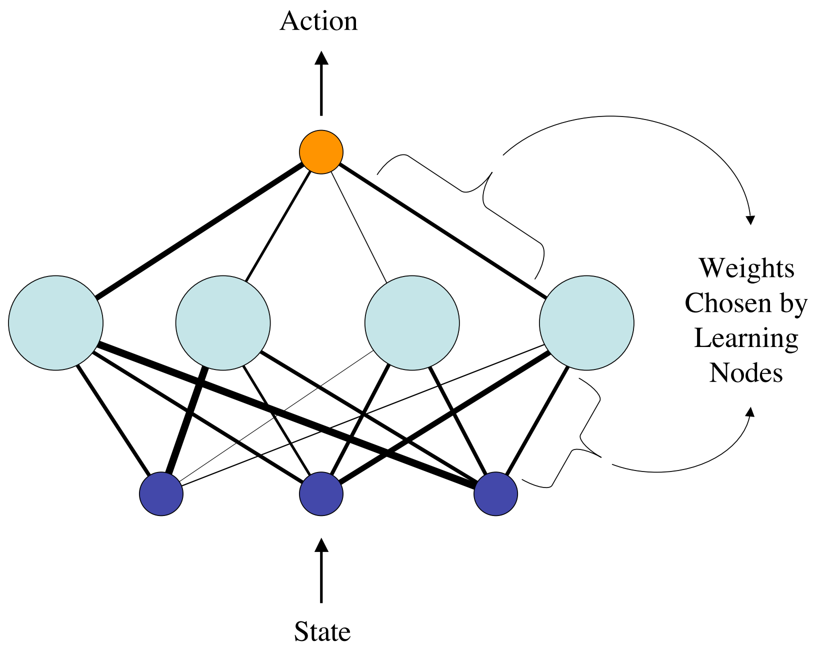


Figure 7.2: Multi-layer perceptron mapping states to actions. Here the state space is three-dimensional and the action space is one-dimensional. All the weights in the MLP are determined by the learning nodes.

7.3.2 Radial Basis Function Networks

To alleviate the credit assignment problem, radial basis function networks (RBFNs) can be used instead. Like the MLP, the state will be fed into the input layer of the RBFN and the action will be determined by the output of the RBFN (see Figure 7.3). Consider a standard RBFN with n bases with fixed width d. The output of the RBF is a linear sum of the basis activations:

$$a(s) = \sum_{\eta} w_i \phi_\eta(s) \tag{7.10}$$

where $\phi_\eta(s)$ is the basis function and weight w_η is the action of node η. For RBFNs, the action sensitivity at time t is simply equal to $\phi_\eta(s_t)$, the activation of the basis function. RBFNs typically use gaussian activation functions of the form:

$$\phi_\eta(s) = e^{\frac{1}{2}(s-c_\eta)^2/d^2} \tag{7.11}$$

where c_η is the centroid of the basis function. Due to the localized nature of this type of activation function, one can expect that the value of $L_{\eta,t}$ will be very low for most states. Only states that are close to the centroid will produce significant activation. Therefore T_η will be equal to the time that the agent entered a state that was close to the centroid ϕ_η.

One difficulty with using RBFNs in a reinforcement learning problem, is how to set the basis centers. Typically a clustering method is used such as k-means or EM, but in real-time domains, the RBFN needs to be used as data points are being produced. One solution to this problem is to periodically cluster the data points using all the available data. However many clustering algorithms make Gaussian assumptions that are often broken in reinforcement learning problems, such the pole-balancing problem (see Figure 7.4).

Instead of using clustering, basis centers can be added dynamically. With the dynamic method, a new basis is added at the location of the current state whenever the activation of the closest basis function is below a threshold.

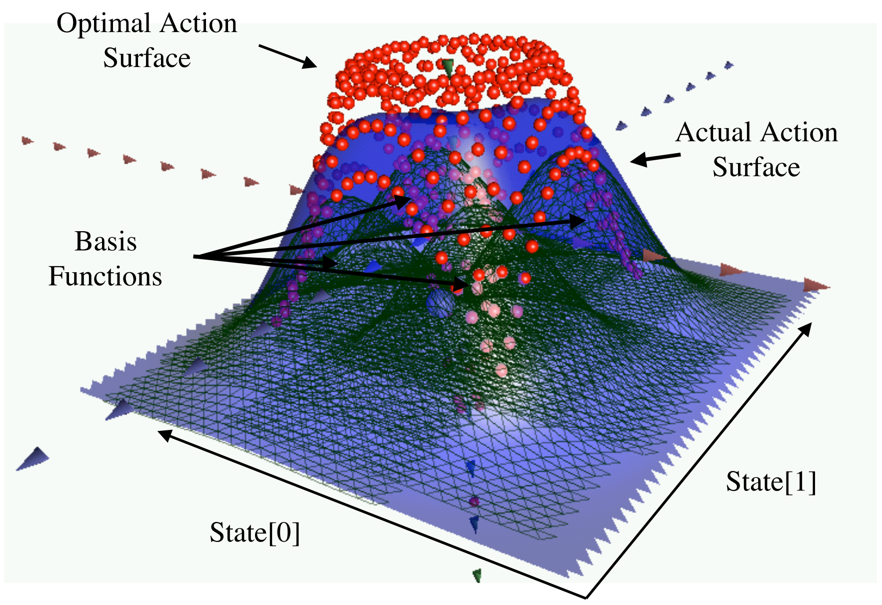


Figure 7.3: The RBFN maps states to actions. Here basis function are placed when needed. Weight of the basis function determines height of curve in diagram, and is set with evolutionary methods.

7.3.3 Results

RBFNs were tested in the double pole balancing experiment with both the team game utility and the WL utility. In this experiment there is a cart that can move in one axis (see Figure 7.5). Two poles of different length are attached to the cart, and can pivot at the attachment point. The agent can apply a positive or negative force to the cart. The goal of the agent is to keep the two poles from falling while keeping the position of the cart within a fixed set of bounds. The state space consists of six values: the position and velocity

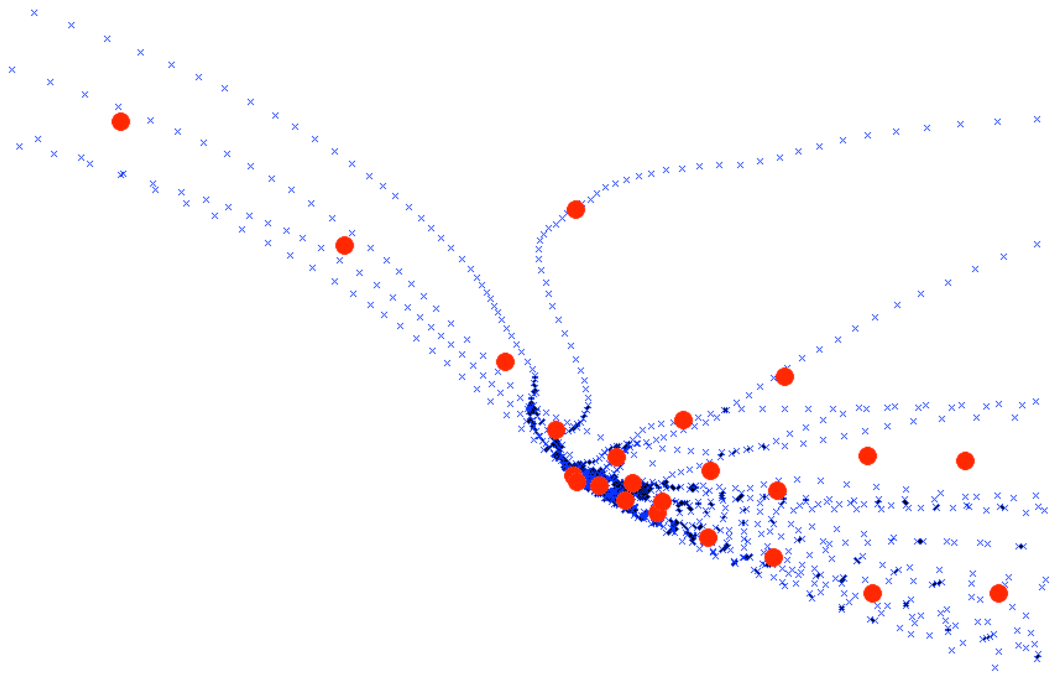


Figure 7.4: Data points of the states entered during a cart-pole experiment (x's) and cluster centers produced by k-means (circles). Algorithms that make a gaussian assumption do not form satisfactory clusters in this reinforcement learning problem

of the cart, and the angles and angular velocities of the two poles. At each time step a reward of 1 is received. The episode ends when the angles of either of the poles goes outside of fixed bounds or when the cart position goes outside of bounds.

The learning algorithms were evaluated based on the number of episodes that needed to be completed before the poles could be balanced for 50,000 time steps. In this particular problem, the length of one of the poles was one meter and the other was one tenth of a meter. The time resolution was 20 millisec-

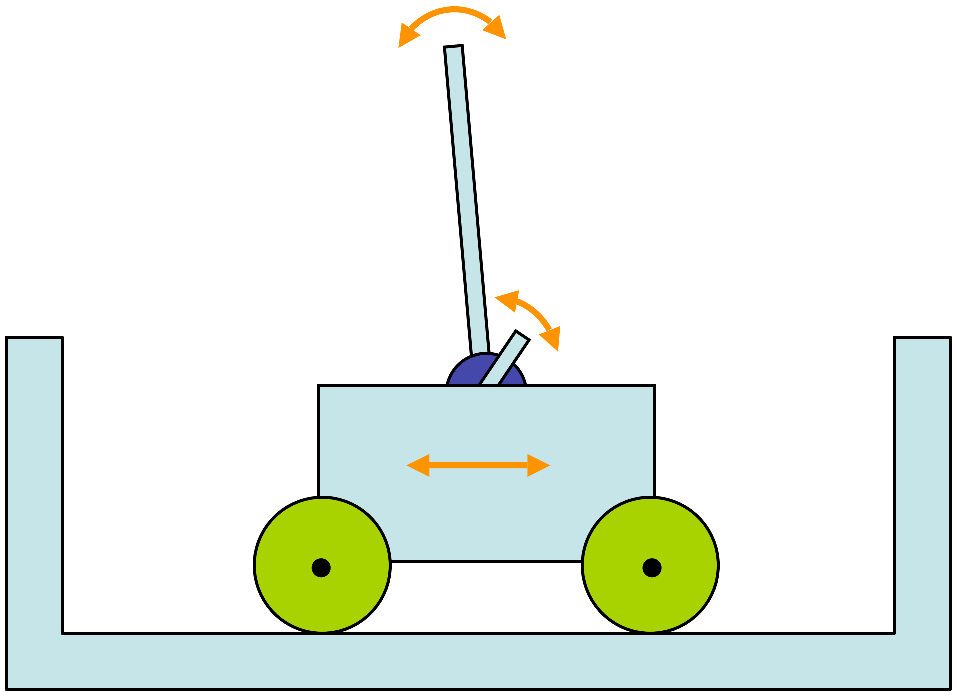


Figure 7.5: A cart with two poles attached can move with one degree of freedom and each pole can rotate with one degree of freedom. Since the poles have different lengths, by applying a force to the cart, both poles can be balanced indefinitely.

onds. For all five algorithms the same code base was used to simulate the pole.

The results for 400 runs are in the following table:

Algorithm	Average Episodes	Deviation in Mean
SANE	12,600	?
ESP	3,800	?
NEAT	3,578	257
RBF (G)	4,025	178
RBF (WLU)	2,815	91

The results show that the RBFN using a team game utility performs almost as well as the two existing high performance algorithms: ESP and NEAT [31, 58]. This is to be expected since these algorithms have some features in common. Similar to NEAT, the RBF used here grows as hidden nodes are needed. Also similar to ESP, the collective evolves separate "subpopulations." However the results for using WLU are significantly better than for G. This can be expected since the WLU for a node eliminates the reward values that the node could not possibly influence, therefore giving it a cleaner signal.

Chapter 8

Discrete Rover Problem

This chapter will present an application of the theory of collectives to a NASA domain. The abstract version of the problem will be presented as the "Multi-agent Gridworld." The abstract problem can then be applied to the more NASA specific example of multiple rovers collecting information on Mars. Even in the abstraction, the addition of multiple agents changes the gridworld problem significantly, since the reward collection for competing agents has to be defined. This problem is also significantly more complex than either traditional gridworld problems, or single time step multi-agent problems, since it involves credit assignment issues across both agents and time. Thus for the collective-based approach to work in such problems, two fundamental issues need to be addressed:

1. the agents need to learn the action *sequence* that will provide good values for their payoff utility functions, i.e., agents need to achieve their own goals; and

2. the agents' learning their own payoff utilities needs to benefit the world utility, i.e., the agents' utilities need to be "aligned" with the world

utility so that the agents do not work at cross-purposes, as far as the world utility is concerned.

The first of these issues has been dealt with extensively in the single agent context; there are many reinforcement learning systems [65], (e.g., Q-learners [74]) that have successfully been applied to real world problems [14, 15]. The second issue is addressed with the framework of collectives.

This chapter will show how the framework of collectives can be extended to a problem where agents need to maximize a time-extended utility function through selecting sequences of actions. It will show that in this significantly more complex domain, agents that use collective-based utilities provided solutions that are significantly superior to agents that either use team games or "natural" utilities. Section 8.1 will describe the gridworld problem domain and develop a collective based solution to the design of agents' payoff utilities. Section 8.2 will present simulation results that show that collective-based solutions significantly outperform both traditional and more "natural" solutions. Finally, Section 8.3, will show that by studying the Nash equilibrium of a simple system, one can demonstrate how collective-based algorithms achieve performance unattainable by systems using "selfish" payoff functions.

8.1 Multi-agent Grid World Problem

A common reinforcement learning problem is the Grid World Problem [65], where an agent navigates about a two-dimensional $n \times n$ grid. At each time step, the agent can move up, down, right or left one grid square, and receives a reward after each move. The observable state space for the agent is its grid coordinate, and the reward it receives depends on the grid square to which it moves. In the episodic version, which is the focus of this paper, the agent moves for a fixed number of time steps, and then is returned to its starting location. This problem typically requires the use of a reinforcement learner that can maximize a sum of rewards in contrast to one that maximizes an immediate reward, since the agent may have to cross squares of low reward value to enter the squares of high value. Q-learners or the Sarsa algorithm [65] are often used for this problem.

This chapter applies the theory of collectives to a multi-agent version of the grid world problem. In this instance of the problem there are multiple agents navigating the grid simultaneously interacting with each others' rewards. This reward interaction is modeled through the use of tokens that are distributed throughout the grid squares of the gridworld (Figure 8.1). Each token has a value between zero and one, and each grid square can have at most one token. When an agent moves into a grid square, it receives a reward for the value of the token and then removes the token so that a reward will no longer be received when another agent enters the grid square. However, all the tokens are reset at the end of an episode. The global objective of the Multi-agent Grid World Problem is to collect the highest aggregated value of tokens in a fixed number of time steps.

The Multi-agent Grid World Problem is an idealized version of many

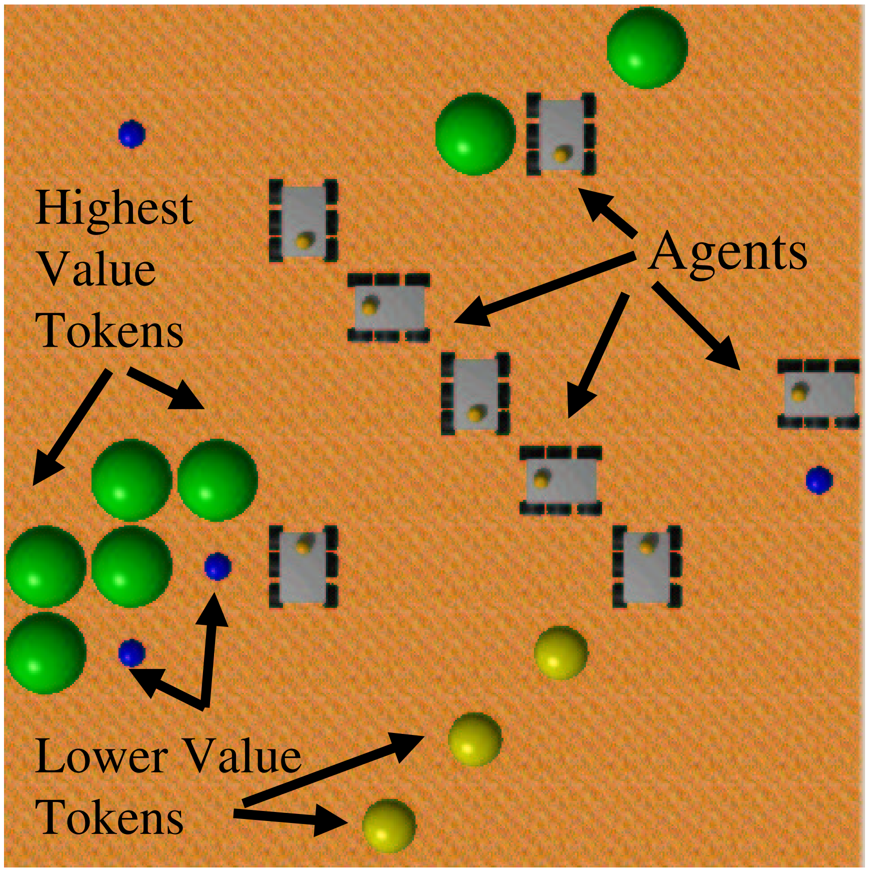


Figure 8.1: Agents Collecting Tokens of Varying Value.

real world problems, including the control of multiple planetary exploration vehicles (e.g., rovers on the surface of Mars, collecting rocks in an attempt to maximize total scientific return, submersible under Europa examining potential life signs). Furthermore, the agent interaction provides a critical study of coordination and interference, as the agents have the potential to work at cross-purposes. This problem is particularly interesting in a multi-agent setting because each agent attempting to maximize the value of the tokens it collects, can drive the world utility to severely sub-optimal values. As such, the design of the payoff functions is crucial in this problem, an issue which is address below.

8.1.1 Collective-Based Solution

To pose the Multi-agent Grid World Problem in the form of the collective framework it is necessary to define:

- $L_{\eta,t}$: The matrix representing the location of an agent. If agent η at time t is in location (x, y), then $L_{\eta,t,x,y} = 1$; otherwise $L_{\eta,t,x,y} = 0$. Furthermore, $\{L_{\eta,t}\}$ denotes the set all the agents' location matrices.
- $L^a_{\eta,t}$: The location matrix agent η would have had at time t, had it taken action a at time step $t - 1$.
- L_η: The location matrix of agent η across all time ($L_\eta = \sum_t L_{\eta,t}$).
- $L_{\eta,<t}$: The location matrix of agent η across times less than t ($L_{\eta,<t} = \sum_{t'<t} L_{\eta,t'}$).
- L_t: The location matrix of all agents at time t ($L_t = \sum_\eta L_{\eta,t}$).
- L: The location matrix of all agents across all time ($L = \sum_t L_t = \sum_t \sum_\eta L_{\eta,t}$).
- $L_{<t}$: The location matrix of all agents across times less than t ($L_{<t} = \sum_{t'<t} L'_t = \sum_{t'<t} \sum_\eta L_{\eta,t'}$).

- $L_{\hat{\eta}}$: The location matrix of all agents other than η across all time ($L_{\hat{\eta}} = L - L_{\eta}$).

- $L_{\hat{\eta},<t}$: The location matrix of all agents other than η across times less than t ($L_{\hat{\eta},<t} = L_{<t} - L_{\eta,<t}$).

- Θ: The initial token value matrix (e.g., $\Theta_{x,y}$ contains the initial value of the token at location (x, y)).

The space **Z** is composed of Θ and the set of all possible location matrices, $\{L_{\eta,t}\}$, given the length of an episode. A worldline z is a point in this space, i.e., the combination of the token configurations Θ, along with a particular set $\{L_{\eta,t}\}$. To facilitate utility computation, it is convenient to define $V(L, \Theta)$, which returns the value of a token received from a location matrix. Formally:

$$V(L, \Theta) = \sum_{x,y} \Theta_{x,y} \min(1, L_{x,y}). \tag{8.1}$$

The global utility $G(z)$ is the sum off all the tokens collected during an episode:

$$G(z) = V(L, \Theta). \tag{8.2}$$

Based on the definitions and world utility given above, it is possible now derive the collective-based utility functions for this domain. In this formulation, the AU (given in Chapter 3) becomes:

$$AU_\eta(z) = G(z) - \sum_{\vec{a} \in \vec{A_\eta}} p_{\vec{a}} V(L_{\hat{\eta}} + L_\eta^{\vec{a}}, \Theta) \tag{8.3}$$

where A_η is the set of possible action sequences agent η can take. The second term in the equation is the expected value of the global utility over all the possible actions of agent η.

Now, it is possible to formulate the WL utilities for this domain. First, setting the clamping parameter CL_η to the null vector, one obtains the WL utility, where the agent is removed from the worldline:

$$WLU_\eta^{\vec{0}}(z) = G(z) - V(L_{\hat{\eta}}, \Theta). \tag{8.4}$$

This utility returns an agent's contribution to the world utility. Note, this utility differs from one where the values of the tokens present in the locations visited by the agent are summed (i.e., a selfish utility). $WLU^{\vec{0}}$ gives the value of the tokens *in locations not visited by other agents*, i.e., the values of token that would not have been picked up had agent η not been in the system.

Next, define the WL utility resulting from agent η taking the fictitious average action, where it partially takes all possible actions:

$$WLU_\eta^{\vec{a}}(z) = G(z) - V(L_\eta + \sum_{\vec{a} \in \vec{A_\eta}} p_{\vec{a}} L_\eta^{\vec{a}}, \Theta) \tag{8.5}$$

Because these utilities are based on the performance on a full episode, they are problematic to work with directly. This section will therefore introduce single time step "rewards" that will help in learning the set of actions (e.g., through Q-learners or Sarsa learners) that will lead to good values for the utility. Note, that the utilities will be undiscounted sums of these rewards. To create rewards, first decompose an arbitrary utility U in the following manner:

$$U(L) = \sum_t U(L_{<t+1}) - U(L_{<t}). \tag{8.6}$$

Now, define the single time step reward R_t by:

$$R_t(L) = U(L_{<t+1}) - U(L_{<t}) \tag{8.7}$$

Now it is possible to generate the four single time step reward versions of the four utilities[1]:

$$\begin{aligned}
GR_t(z) &= V(L_{<t+1}, \Theta) - V(L_{<t}, \Theta) && (8.8)\\
AR_{\eta,t}(z) &= GR_t(z) - \sum_{\vec{a} \in \vec{A}_\eta} p_{\vec{a}}(V(L_{\eta,<t+1} + L^{\vec{a}}_{\eta,<t+1}, \Theta) \\
&\qquad - V(L_{\eta,<t} + L^{\vec{a}}_{\eta,<t}, \Theta)) && (8.9)\\
WLR^{\vec{0}}_{\eta,t}(z) &= GR_t(z) - (V(L_{\hat{\eta},<t+1}, \Theta) - V(L_{\hat{\eta},<t}, \Theta)) && (8.10)\\
WLR^{\vec{a}}_{\eta,t}(z) &= GR_t(z) - \left[V\left(L_{\eta,<t+1} + \sum_{\vec{a} \in \vec{A}_\eta} p_{\vec{a}} L^{\vec{a}}_{\eta,<t+1}, \Theta \right) \right. \\
&\qquad \left. - V\left(L_{\eta,<t} + \sum_{\vec{a} \in \vec{A}_\eta} p_{\vec{a}} L^{\vec{a}}_{\eta,<t}, \Theta \right) \right] && (8.11)
\end{aligned}$$

Note that as expressed above the formulation for AR and $WLR^{\vec{a}}$ has significant drawbacks. First, the set of all possible action sequences is very

[1]In the actual implementation there are some tie breaking rules if more than one agent goes into the same square at the same time.

large, and grows exponentially with t. Second, without prior information, the average path contains little information and for WLR is similar to clamping to zero. To side-step these issues, the approximation is made that each action is equally likely, and computes the average action over the actions available to the agent in the previous time step:

$$\begin{aligned}
AR_{\eta,t}(z) &= GR_t(z) - \sum_{a \in A_{\eta,t-1}} p_a(V\left(L_{\eta,<t+1} + L^a_{\eta,<t+1}, \Theta\right) \\
&\quad -V\left(L_{\eta,<t} + L^a_{\eta,<t}, \Theta\right)) \\
WLR^a_{\eta,t}(z) &= GR_t(z) - \left[V\left(L_{\eta,<t+1} + \sum_{a \in A_{\eta,t-1}} p_a L^a_{\eta,<t+1}, \Theta\right)\right. \\
&\quad \left. - V\left(L_{\eta,<t} + \sum_{a \in A_{\eta,t-1}} p_a L^a_{\eta,<t}, \Theta\right)\right]
\end{aligned}$$

Note, the average action sequence has been replaced with the sequence of average actions, that is the sequence where at each time step the average action has been taken. Because of the arbitrariness of the clamping operator (see discussion in Chapter 3) and the fact that theoretically, clamping to any fixed vector results in a factored system, this approximation is milder for WL^a than it is for AU.

8.2 Experimental Results

To evaluate the effectiveness of the collective-based approach in the Multi-agent Grid World, experiments on three different types of token distributions were conducted. The payoff utility function investigated included: (i) the Selfish Utility (SU), where each agent receives the weighted total of the tokens that it alone collected. It is the natural extension of the single agent problem, and represents the optimal utility in the single rover domain; (ii) the Team Game (TG) utility where each agent received the full world utility; (iii) the $WL^{\vec{0}}$ utility, where there clamping parameter is set to $\vec{0}$. Intuitively, this utility computes the contribution an agent makes to the token collection, by looking at the difference in the total token collection with and without that agent; (iv) the $WL^{\vec{a}}$ utility, where there clamping parameter is set to $\vec{a}$, representing the difference between the utility value resulting from an agent's actual action and its "smeared" action; and (v) the AU, where the agent's contribution is computed as the difference between the action it took and its expected action.

The following subsections evaluate the five payoff utilities using three different distributions of tokens. Since the author is unaware of any standard gridworld benchmarks that specify reward distributions, artificial distributions of tokens were created that could illustrate the important collective learning issues. The first set of tokens is the most "unnatural," but requires the agents to optimize a sum of rewards instead of an immediate reward, while working together, if high world utility is to be achieved. The second set of tokens has similar properties to the first set, but has smoother transitions in token values. The final set is randomly generated from Gaussian kernels on every trial, to illustrate that the collective-based principles still hold on less hand crafted

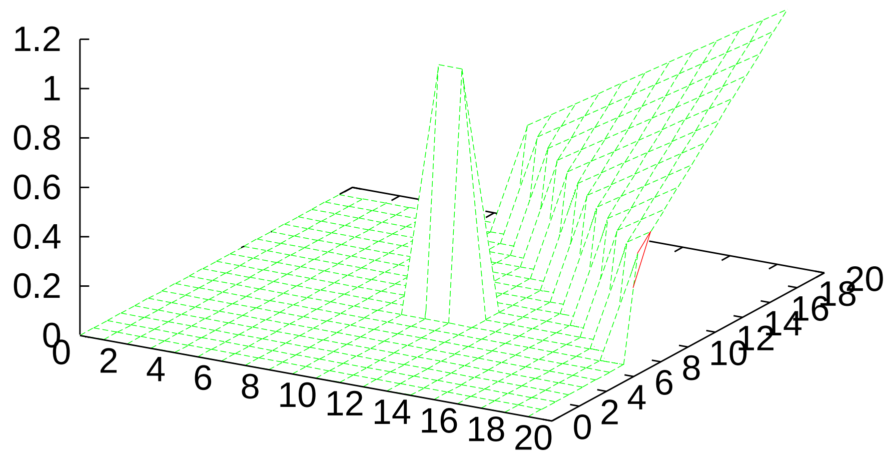


Figure 8.2: Distribution of Token Values in "Corner" World

token distributions.

8.2.1 Corner World Token Value Distribution

The first experimental domain investigated consisted of a world where the "highly valued" tokens are concentrated in one corner, with a second concentration near the center where the rovers are initially located. The rest of the world is uniformly filled with tokens of little importance. Figure 8.2 conceptualizes this distribution for a 20x20 world.

Figure 8.3 (left) shows the performance of the different payoff utilities

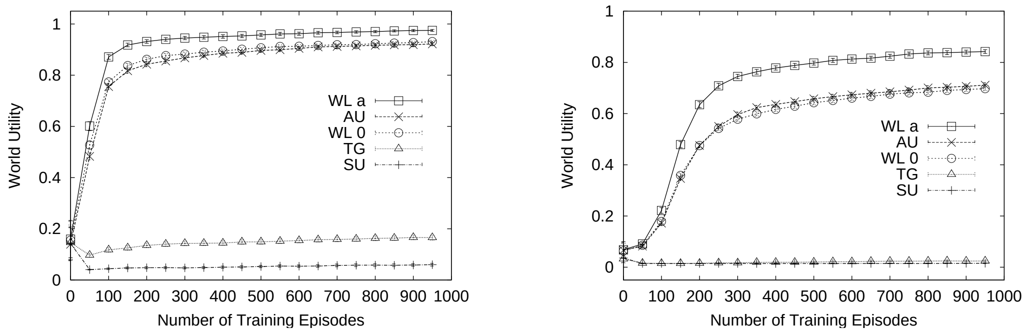


Figure 8.3: Effect of Payoff Utility on System Performance. Left: 40 Rovers on a 20x20 grid. Right: 100 Rovers on a 32x32 grid.

for 40 agents on a 400 unit-square world for the token value distribution shown in Figure 8.2, and where an episode consists of 20 time steps (error bars of $\pm$ one σ are included, though they are smaller than the symbols). The performance measure in these figures is "normalized" world utility given by $\frac{V(L,\Theta)}{V(1_n,\Theta)}$, where 1_n is the $n \times n$ matrix of ones. This normalized utility provides the fraction of token values that was collected by the agents (a value of 1 means all available tokens were collected).

The results show that SU produced poor results, results that were indeed worse than random actions. This is caused by all agents aiming to acquire the most valuable tokens, and congregating towards the corner of the world where such tokens are located. In essence, in this case agents using the SU payoff competed, rather than cooperated with one another. The agents using TG fared marginally better, but their learning was slow. This system was plagued by the signal-to-noise problem associated with each agent receiving the full world reward for each individual action they took. Notice both the

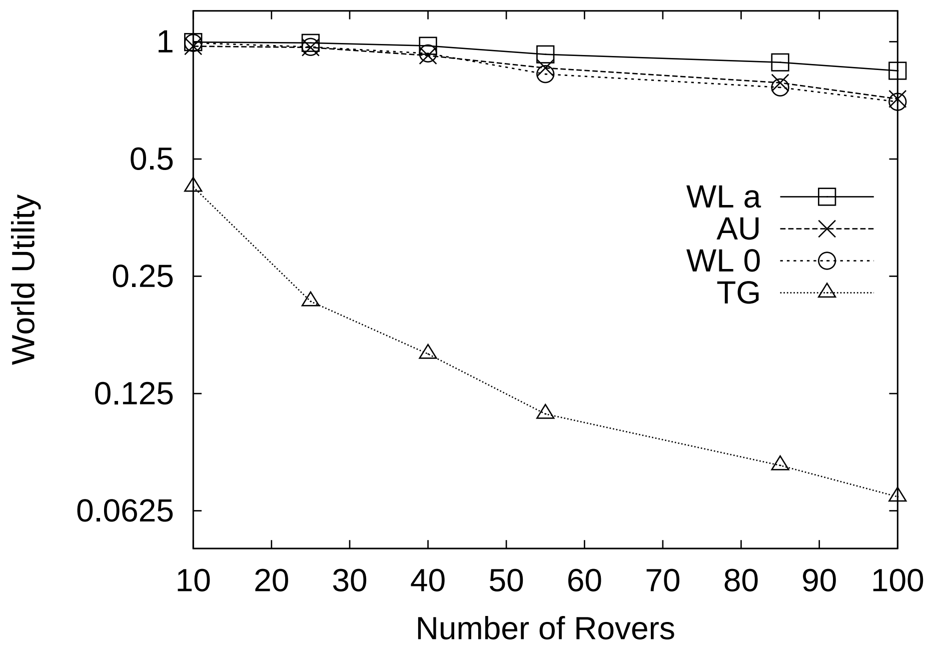


Figure 8.4: Scaling Properties of Different Payoff Functions.

selfish agents and those trained with TG had a drop in their performance in the early going, as they learned the "wrong" actions (as far as the world utility is concerned). Agents using TG payoffs overcame this early setback whereas selfish agents never did. In contrast, agents using $WL^{\vec{0}}$ and AU performed very well, and agents using $WL^{\vec{a}}$ performed almost optimally. In each of these three cases, the reinforcement signal the agents received was both factored and showed how their actions affected the world reward more clearly than did the TG reinforcement signal.

For a scaled up version of the same token value distribution, Figure 8.3 (right) shows the results for 100 agents on a 1024 unit-square grid, where an episode consists of 32 time steps. Qualitatively, the results are similar to the 40 agent case. However, note that the team game agents have a harder time learning, because in this case the reinforcement signal is even further diluted. Furthermore, the performance of $WL^{\vec{a}}$ is now clearly superior to that of $WL^{\vec{0}}$, showing that using the degree of freedom given by the clamping parameter provides significant improvements over solutions aimed at "endogenizing externalities" or similar to the Groves mechanism ($WL^{\vec{0}}$).

Figure 8.4 explores the scaling issue in more detail. As the number of agents was increased, the difficulty of the problem was kept the same by increasing the size of the gridworld, and allocating more time for an episode. Specifically the ratio of the number of agents to total number of grid squares and the ratio of the number of agents to total value of tokens was held constant. In addition the ratio of the furthest grid square from the agents' starting point to the total amount of time in an episode was also held constant. The scaling results show that agents using TG payoffs were not hampered as much by the noise associated with other agents when the number of agents was low. As the system scaled up however their performance deteriorated rapidly. Agents using WL and AU payoffs on the other hand were not strongly affected by the increase in the size of the problem. This underscores the need for a payoff utility function that has good signal-to-noise properties so that in large systems, the agents have an opportunity to learn the actions that will maximize their payoff utilities.

8.2.2 Incline World Token Value Distribution

The second experimental domain investigated was a world where the "highly valued" tokens are still concentrated in one corner, but where there is a "ridge" of moderately high values along a side, starting from the opposite corner. The actual distribution for the token values $\Theta_{x,y}$ on a two dimensional (x,y) map is given by:

$$\Theta_{x,y} = 1.0 - \left(\frac{x}{s} \; (1.0 \; - \; \frac{s-y}{s}) \right) - \left(0.5 \; \frac{s-y}{s} \right) \qquad (8.12)$$

where s is the one dimensional "size" of the map (i.e., $s = 20$ for 40 agents, and $s = 32$ for 100 agents). Figure 8.5 conceptualizes this distribution for a 10x10 world (s=10).

Figure 8.6 shows the performance of the different payoff utilities for 40 agents and 100 agents on 400 and 1024 unit-square worlds, respectively, on the token value distribution given by Equation 8.12. The results show that the WL payoff is unaffected by the change in the token value distribution. Both TG and SU payoffs in contrast perform better in this case, showing a much larger sensitivity to the token distribution. The improvements in SU are easily explained: The area surrounding the high token values contains sufficiently many tokens that even when the SU agents are all trying to reach the high valued tokens, they help the world utility.

Though agents using TG collect a larger portion of token as compared to the previous token configuration, the lack of improvement displayed in the system where agents use TG payoffs is noteworthy. Because of the noise in

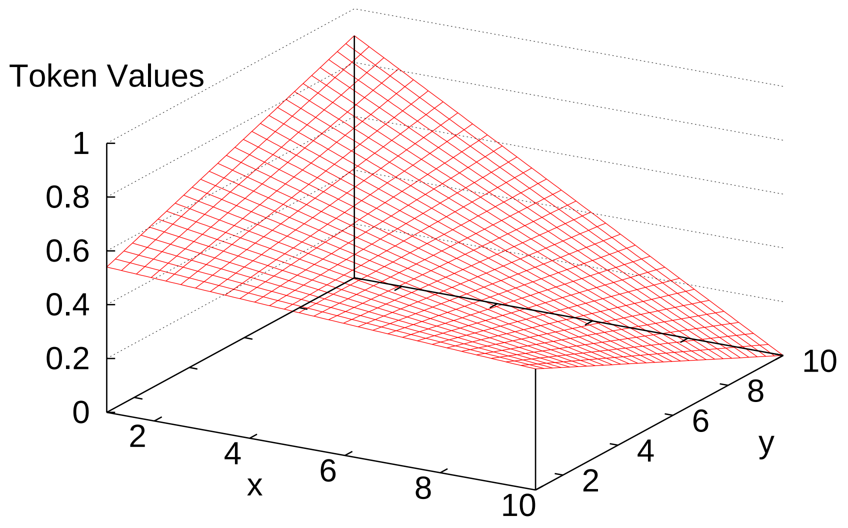


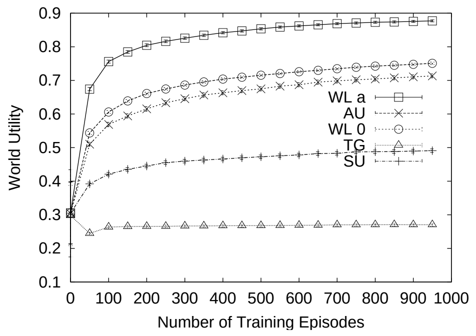
Figure 8.5: Distribution of Token Values in “Incline” World

the system, these agents do not even learn to “walk” in the right direction in the allotted number of episodes.

8.2.3 Random World Token Value Distribution

The final set of experiments investigates the behavior of agents in a gridworld where the token values are randomly distributed. In this world, for N agents, there are $N/3$ Gaussian ‘attractors’ whose centers are randomly distributed. Figure 8.7 shows an instance of the gridworld using this distribution for the 20x20 world, used in the experiments with 40 agents.

Figure 8.8 shows that the performance of the agents in the “random” world are very similar to the “incline” world, but for the poorer performance of



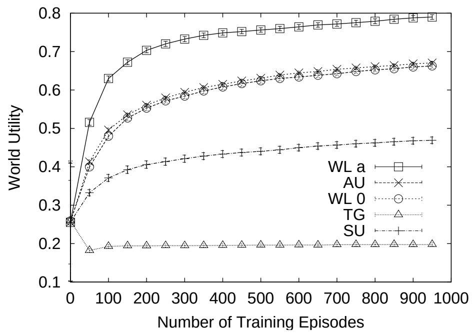


Figure 8.6: Effect of Payoff Utility on System Performance for Incline World Token Value Distribution. Left: 40 Rovers on a 20x20 grid. Right: 100 Rovers on a 32x32 grid.

the SU payoff function. This can be explained by the token value distributions: there are many "dry" patches, and agents aiming for the high valued token do not necessarily get the consolation of mid-valued tokens. The WL payoff again does well for both clamping parameters.

This experiment illustrates that when agents need to use a "divide and conquer" approach, the selfish utility performs poorly. Furthermore, these experiments illustrate that the collective-based payoff functions are superior across a wide range of token distributions ranging from smooth to irregular to random.

8.3 Nash Equilibrium and World Utility Optimum

Figure 8.9 illustrates a simple example with two agents in a six square world, where each agent can choose to move left or right for two time steps. There are two tokens, one of values 5 and the other of value 10 that the agents

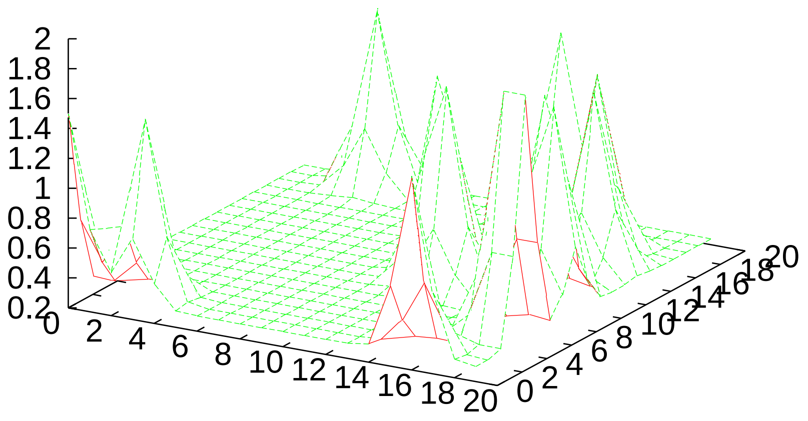
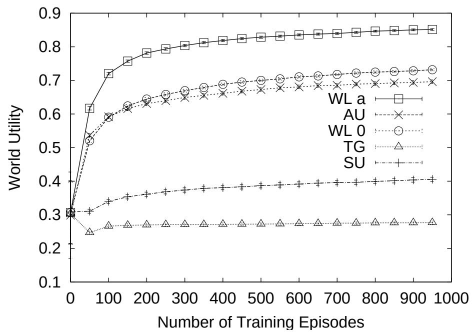


Figure 8.7: Distribution of Token Values in "Random" World

can pick up by entering the appropriate square.

Consider the joint set of moves where agent 1 moves right twice, and agent 2 moves left twice [2]. In this scenario agent 2 picks up a token worth 10 units on the first time step and then nothing. Agent 1 does not pick up any tokens. Figure 8.9 summarizes the reward and utility values associated with this move. Agent 1 receives an SU of 10 (10 for the first step, 0 for the second) whereas agent 2 receives an SU of 0. This results in a world reward of 10 for the first time step and 0 for the second, resulting in a world utility of 10. Incidentally, this is the solution the system settles into if the agents are indeed

[2]The second step of agent 2 is arbitrary.



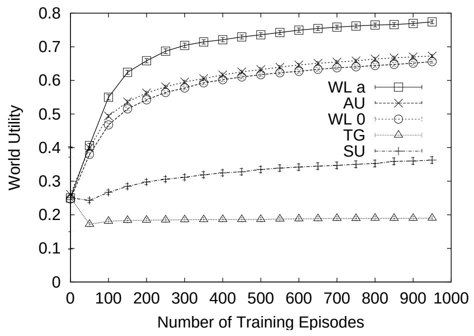


Figure 8.8: Effect of Payoff Utility on System Performance for Random World Token Value Distribution. Left: 40 Rovers on a 20x20 grid. Right: 100 Rovers on a 32x32 grid.

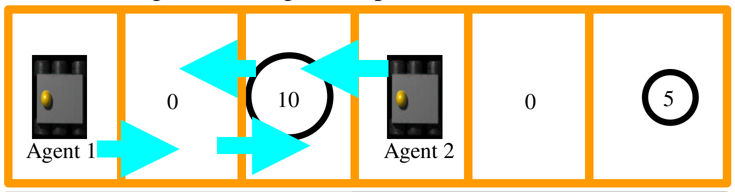
using SU as their payoff utilities. For SU, this is a Nash Equilibrium: There is no unilateral moves that a player can make that will improve its utility.

Now analyze the $WL^{\vec{0}}$ payoff utility for agent 2 for this set of moves:[3] For the first time step, the WL reward is the same as SU: agent 2 receives 10 for picking up the token. It is in the second time step though the differences emerges: The first term of WL as given in Equation 8.11 (i.e., the world reward for time step 2) is 0 for this time step as no tokens are picked up. However, in the worldline where agent 2 has been clamped to zero, the first parameter of the V function, L_{γ_η}, does not include the locations of agent 2, causing this function do disregard any tokens agent 2 previously picked up. This causes agent 2 to credit agent 1 for picking up the token of value 10 in the second time step. Agent 2 then computes a value of 10 for the world reward of this state where it's clamped its action to $\vec{0}$. The WL reward for agent 2 for this

[3]Both AU and WL with clamping to $\vec{a}$ provide similar results, but they are omitted them for clarity.

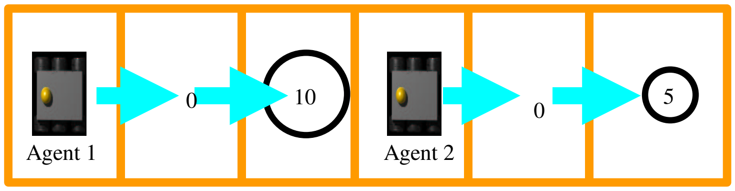
time step is then given by: $WLR_{\eta_2,t=2} = 0 - 10 = -10$.

Agents Taking Non-optimal Actions (G = 10)



	G	WLU		SU	
		η_1	η_2	η_1	η_2
t=1	10	0	10	0	10
t=2	0	0	**-10**	0	0
Total	10	0	**0**	0	10

Agents Taking Optimal Actions (G = 15)



	G	WLU		SU	
		η_1	η_2	η_1	η_2
t=1	0	0	0	0	0
t=2	15	10	5	10	5
Total	15	10	**5**	10	5

Figure 8.9: WLU Nash Equilibria and World Utility Optima

Now the time-extended WLU for agent 2 can be computed by summing the WLRs. This results in a WLU of 0, even though agent 2 picks up a token weighted 10 (10 for t=1 and -10 for t=2). The interpretation for this "counter-intuitive" utility value is clear: because that token would have been picked up by agent 1 at another time step, the net effect of agent 2's actions on the world utility was nil, resulting in a WLU value of 0 for agent 2 for this set of actions.

Because moving to the right twice provides a WLU value of 5 for agent 2 (as detailed in Figure 8.9), an agent maximizing its WL payoff utility will take this second action. Similarly agent 1 moving right twice will receive a WLU of 10. As this simple example shows, each agent maximizing its WLU leads the system to the world utility maximum where both tokens are picked up.

The game-theoretic equilibrium states for WLU and SU can be analyzed in these two solutions: The SU is in a Nash equilibrium for the first set of moves, in that neither agent can improve its SU by unilaterally changing its actions. Therefore, the system is "stuck" in this suboptimal solution. Furthermore, even if the agents stumble upon the second solution by accident, they will not remain there, as this solution is unstable with payoff utilities given by SU: Agent 2 can change its move (in future episodes) and improve its payoff utility from 5 to 10. That this move reduces agent 1's utility from 10 to 0, and the world utility from 15 to 10 has no influence on agent 2's actions. Furthermore, this solution is also Pareto-optimal, in that there is no set of joint moves that improves the utility of *both* agents. This example shows a simple case where Pareto optimality and optimum of the world utility do not necessarily coincide[4], and simply seeking a Pareto-optimal solution will not necessarily lead to high values of the world utility function.

On the other hand agents whose payoff utilities are given by the WLU are in a Nash Equilibrium in the second set of actions. They will therefore

[4]Note, in this case the world utility optimum is also a Pareto-optimal, but there are no guarantees that the system will stumble upon the "desirable" Pareto optimal solution.

seek this solution as it offers higher payoff utilities for each agent. The use of WLU has the net effect of "aligning" the Nash equilibrium of the agents with the world utility optimum, ensuring that when the agents optimize their payoff utilities, the world utility is also at a local – and in this case also the global – optimum.

8.4 Discussion

This chapter focuses on the problem of designing a collective of autonomous agents that individually learn sequences of actions such that the resultant sequence of joint actions achieves a predetermined global objective. In particular it addresses the problem of controlling multiple agents in a grid-world, a problem related to many real world problems including exploration vehicles trying to maximize aggregate scientific data collection (e.g., rovers on the surface of Mars).

This chapter illustrates how the theory of collectives can be used to leverage the work done on existing reinforcement learners that are able to work with sequences of actions, so that they can be extended to multi-agent problems. In this domain, the critical issue of what utility functions those agents should strive to maximize is addressed. This chapter extends the previous results on collective intelligence to agents attempting to maximize sequences of actions, and used Q-learning with rewards set by the theory of collectives. The results demonstrate that agents using collective-derived goals outperform both "natural" extensions of single agent algorithms and global reinforcement learning solutions based on “team games”.

Even the simplest collective-derived utility, $WL^{\vec{0}}$, showed marked improvement over greedy and team game utilities as it was able to scale well, while still directing agents to “work together.” To try to increase performance further, this chapter presented two utilities in addition to $WL^{\vec{0}}$: $WL^{\vec{a}}$ and AU. While $WL^{\vec{0}}$ performed well in these problems, $WL^{\vec{a}}$ provided further improvements, and approached the optimal solution in many cases. This improvement was due to $WL^{\vec{a}}$ returning an agents contribution compared to an average action rather than to the more extreme case comparing it to no action at all. The experimental results confirm the theoretical analysis that shows $WL^{\vec{a}}$ having a higher learnability than $WL^{\vec{0}}$.

While $WL^{\vec{a}}$ proved to be a superior utility, the investigations revealed an interesting situation where AU, the theoretically “best” strategy, was not necessarily the best approach in practice. Although AU is theoretically superior to WLU (higher learnability), two issues prevent one from fully exploiting its power: First, the “expected” action is impossible to compute in a time extended setting, since even a simple case where an agent has four actions and ten time steps leads to 4^{10} possible actions. Even Monte Carlo sampling of such a space will yield highly inaccurate estimates of the potential actions and their rewards. Second, estimating the correct probability distributions over the possible actions causes the utility values to change, creating a self-consistency problem. To sidestep both issues, the algorithm used in this chapter focused on the last time step (e.g., current step for the agent) and approximated the

AU with the agent taking each of the four actions possible in that time step with equal likelihood. The resulting utility function provided good solutions, but the performance of such a "handicapped" AU did not exceed those of the conceptually simpler $WL^{\vec{0}}$ and was well below that of $WL^{\vec{a}}$.

Finally, a game-theoretic analysis of the utilities provided by the theory of collectives sheds light on the reasons for the dramatic improvements obtained over selfish utilities and team games: While the Nash equilibrium of the system in which each agent pursues a selfish goal does not correspond to a globally good solution, the global optimum is indeed a Nash equilibrium of the system in which agents use collective-based utilities. Furthermore, such utilities provide "good" off-equilibrium signals to lead the system into desirable equilibrium states, whereas systems in which all agents use team game utilities fail to reach the desirable states (e.g., Nash equilibrium states which also are optima of the world utility) due to the excessive "noise" in the system.

Chapter 9

Continuous Rover Problem

The Multi-agent Gridworld Problem is a good problem to test many of the theories of collective learning over sequences of actions. However this problem is limited in two primary ways. The first way it is limited is that the problem is discrete in both its action space and its state space. In addition, it is difficult to generalize the solution to a continuous problem since the table-based learning methods used depend on this discreteness, since Q-values for all possible state and action pairs have to be recorded. Even though it is possible to extend the table-based methods to continuous domains, some theoretical difficulties with common approaches to doing this can lead to poor results. This makes it difficult to apply the solutions to the Multi-agent Gridworld Problem to continuous problems. Yet one expects to find continuous problems in real-world settings since many sensory inputs are continuous and actions that a rover needs to take may be continuous. The other limitation of the Multi-agent Gridworld Problem is that the solutions are hard to generalize. Solutions to this problem involve finding paths, using x,y coordinates as sensory input. A solution that a rover finds during a gridworld training phase is unlikely to apply to a real world application since the coordinates of objects will have changed. Algorithms based on naturally episodic domains, such as the Gridworld Problem, are difficult to apply to a real-world problem, such as controlling rovers on Mars. A rover on Mars is unlikely to move in an episodic manner, and the rover will have to generalize from the learning done before it was deployed to its current situation on Mars.

To address these issue, this chapter will introduce the Continuous Rover Problem. In this problem, there is a set of rovers on a two dimensional plane, which are trying to observe points of interests (POIs). A POI has a fixed position on the plane and has a value associated with it. The observation information from observing a POI is inversely proportional to the square of the distance the rover is from the POI. In this chapter the distance metric will be the squared euclidean norm, bounded by a minimum distance:

$$\delta(x, y) = min\{\|x - y\|^2, d^2\} \tag{9.1}$$

where d is the minimum distance[1]. While any rover can observe any POI, in this chapter it will be assumed that an observation will contribute no additional information than the closest observation. Therefore as far as the global utility is concerned, only the closest observation counts [2]. The global utility for a single time step is as follows:

$$R = \sum_i \frac{V_i}{\delta(L_i, L_{c(i)})} \tag{9.2}$$

[1]The square euclidean norm is appropriate for many natural phenomenon, such as light and signal attenuation. However any other type of distance metric could also be used as required by the problem domain. The minimum distance is included to prevent singularities when a rover is very close to a POI

[2]Similar utilities could also be made where there are many different levels of information gain depending on the position of the rover. For example 3-D imaging may utilize different images of the same object, taken by two different rovers.

where $c(i) = argmin_{\eta}\delta(L_i, L_{\eta})$. The global utility for the episode is the sum of rewards.

The rovers see the world through eight continuous sensors. From a rover's point of view, the world is divided up into four quadrants relative to the rover's orientation, with two sensors per quadrant (see Figure 9.1). For each quadrant, the first sensor returns a function of the POIs in the quadrant. Specifically the first sensor for quadrant q returns the sum of the values of the POIs in its quadrant divided by their squared distance to the rover:

$$s_{1,q,\eta} = \sum_{i \in I_q} \frac{V_i}{\delta(L_i, L_{\eta})} \tag{9.3}$$

where I_q is the set of POIs in quadrant q. The second sensor is similar except that it returns the sum of square distances from a rover to all the other rovers in the quadrant:

$$s_{2,q,\eta} = \sum_{\eta' \in N_q} \frac{1}{\delta(L_{\eta'}, L_{\eta})} \tag{9.4}$$

where N_q is the set of rovers in quadrant q. The sensor space is broken down into four regions, since it is fairly easy for a multi-layer perceptron to map inputs from four regions into a two-dimensional output action. In addition there is a trade-off between the granularity of the regions and the dimensionality of the input space. In some domains the tradeoffs may be such that it is preferable to have more or less than four sensor regions. Also, even though this paper assumes that there are actually two sensors present in each region at all times, in real problems there may be only two sensors on the rover, and they do a sensor sweep at 90 degree increments at the beginning of every time step.

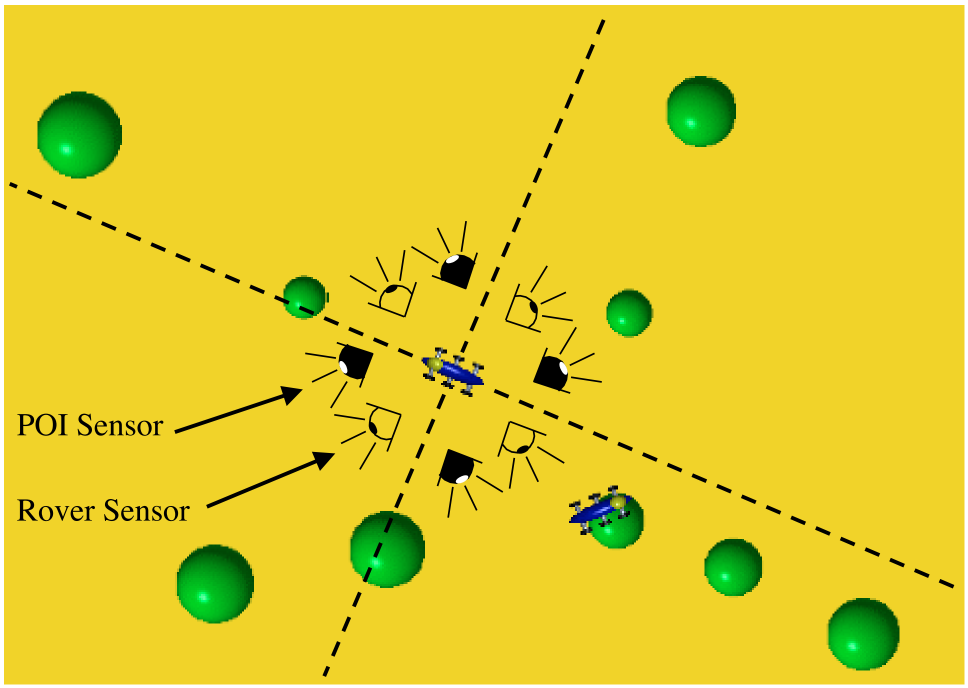


Figure 9.1: Diagram of a rover's sensor inputs. The world is broken up into four quadrants relative to rover's position. In each quadrant one sensor, senses points of interests, while the other sensor senses other rovers.

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional action. The action represents an x,y movement relative to the rover's location and orientation (see Figure 9.2). The mapping from state to action is done with a multi-layer-perceptron (MLP), with 8 input units, 10 hidden units and 2 output units. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actions, dx and dy, are determined from substracing 0.5 from the input and multiplying by the maximum distance the rover can move in one time step:

$$\begin{aligned} dx &= D(o_1 - 0.5) \\ dy &= D(o_2 - 0.5) \end{aligned}$$

where D is the maximum distance the rover can move in one time step, o_1 is the value of the first output unit, and o_2 is the value of the second output unit.

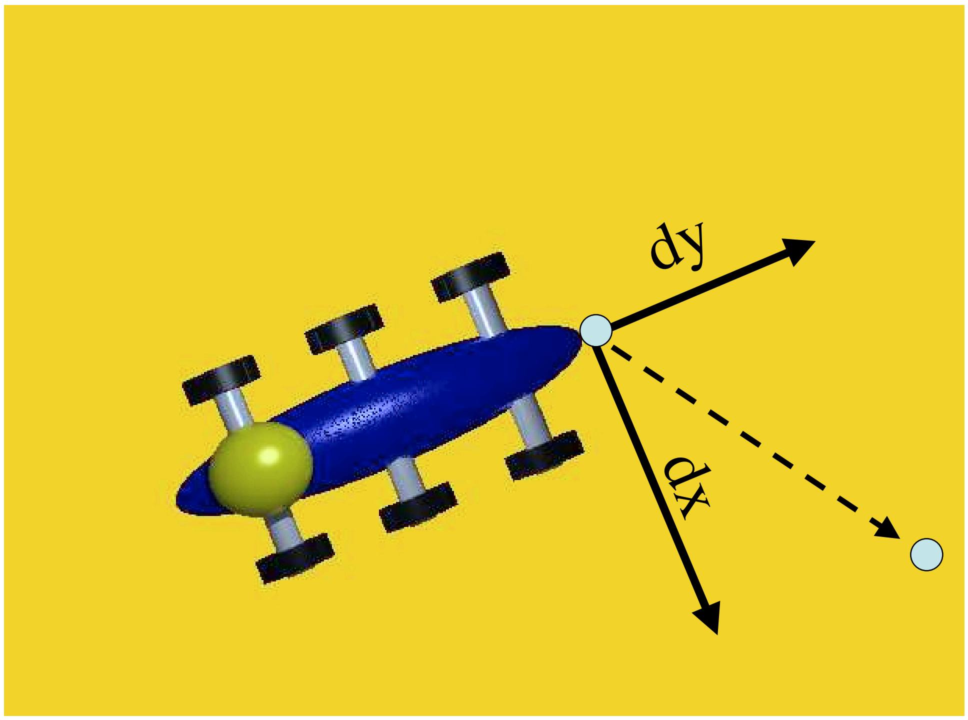


Figure 9.2: A rover moves a dx amount and a dy amount along a coordinate plane relative to the rovers orientation.

The MLP for a rover is chosen by an evolutionary algorithm. In this algorithm each rover has a population of MLPs. At the beginning of each episode step, an MLP for a rover is copied from its population using an ϵ-greedy selector. The copied MLP is then mutated and used for the entire episode. When the episode is complete, the MLP is evaluated by the rover's utility function and inserted into the population. The worst performing member of the population is then deleted.

9.1 Results

The Continuous Rover Problem was tested in three different scenarios. In each scenario, an episode consisted of 15 time steps, and each rover had a population of MLPs of size 10. The world was 100 units tall and 115 units wide. All of the rovers started an episode 65 units from the left boundary and 50 units from the top boundary. The maximum distance the rovers could move in one direction during a time step, D, was equal to 10. The rovers could not move beyond the bounds of the world. The minimum distance, d, used to compute δ was equal to 5. Note that since the rovers were learning through sensors, non-episodic scenarios could also have be created, where the environment changed through time. For simplicity however this chapter focused on episodic scenarios. In addition other forms of continuous reinforcement learners could have been used instead of the evolutionary neural networks.

In each of the three scenarios there were ten rovers and three different reward functions were tested. The first reward was the global reward (R):

$$R = \sum_i \frac{V_i}{\delta(L_i, L_{c(i)})} \tag{9.5}$$

The second reward was the selfish reward (SR):

$$SR_\eta = \sum_i \frac{V_i}{\delta(L_i, L_\eta)} \tag{9.6}$$

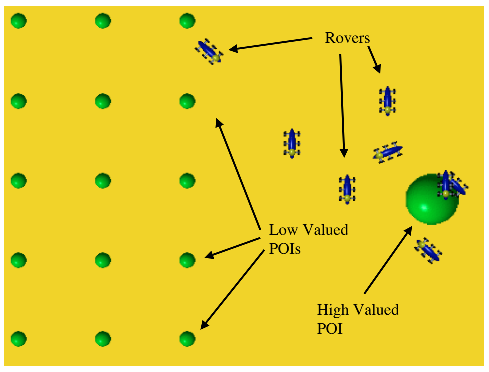
Note that the selfish reward is equivalent to the global reward when there is only one rover. The final reward was the WL reward:

$$\begin{aligned} WLR_\eta &= \sum_i \frac{V_i}{\delta(L_i, L_{c(i)})} - \sum_i \frac{V_i}{\delta(L_i, L_{c_{CL_\eta}(i)})} \\ &= \sum_i I_{i,\eta}(z) \frac{V_i}{\delta(L_i, L_\eta)} \end{aligned}$$

where $c_{CL_\eta}(i) = argmin_{\eta' \neq \eta} \delta(L_i, L_{\eta'})$ and $I_{i,\eta}(z)$ is an indicator function, returning one if and only if η is the closest rover to L_i. The second term of the WLR is equal to the value of all the information collected if rover η were not there. Note that for all POIs where η is not the closest, the subtraction leaves zero. The WLU can be computed easily as long as η knows the position and distance of the closest rover to each POI it can see. If η cannot see a POI then it is not the closest rover to it.

The first experiment was performed using a set of POIs that remained fixed for all episodes (see Figure 9.3). Results from Figure 9.3 show that the rovers using the WLR rewards performed the best, by a wide margin. Early in training, rovers using SR performed better than rovers using R. However since the learning curve of these rovers using SR remained flat, while the ones using

R increased, the rovers using R eventually overtook the ones using SR. This phenomenon can be explained with factoredness and learnability. The selfish reward tends to be highly learnable since it is only effected by the moves of a single rover. This high learnability enables the rover to learn basic tasks very quickly, such as moving towards a POI. However since the SR is not factored, it is unable to do more, since maximizing its own reward occasionally causes the rover to take actions that hurt the global reward. In contrast rovers using R learn slowly, since the global reward is effected by the actions of all the other rovers. With time, however, rovers using the global reward slowly learn to maximize to global utility.



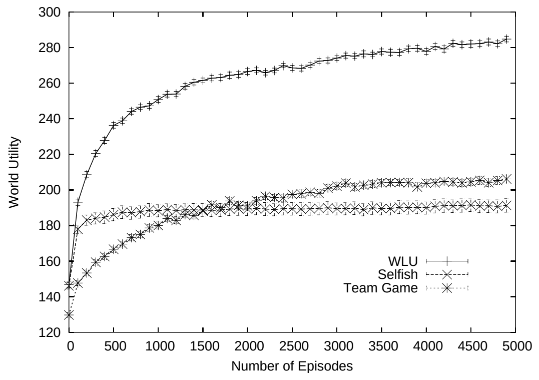


Figure 9.3: World where points of interests are at fixed locations for every episode. Diagram (left). Results for three different utilities (right).

The second experiment was similar to the first on except that the value of a POI went down each time it was observed. This was a harder problem than the previous one, since the state of the entire world changed more at every time step. Figure 9.4 shows that rovers using WLR, still performed best in this problem, but not by as wide as margin. The performance gap was probably not as large because the problem was more difficult, with more noise. In this type of problem, it was harder to do better than a greedy solution or even a random solution.

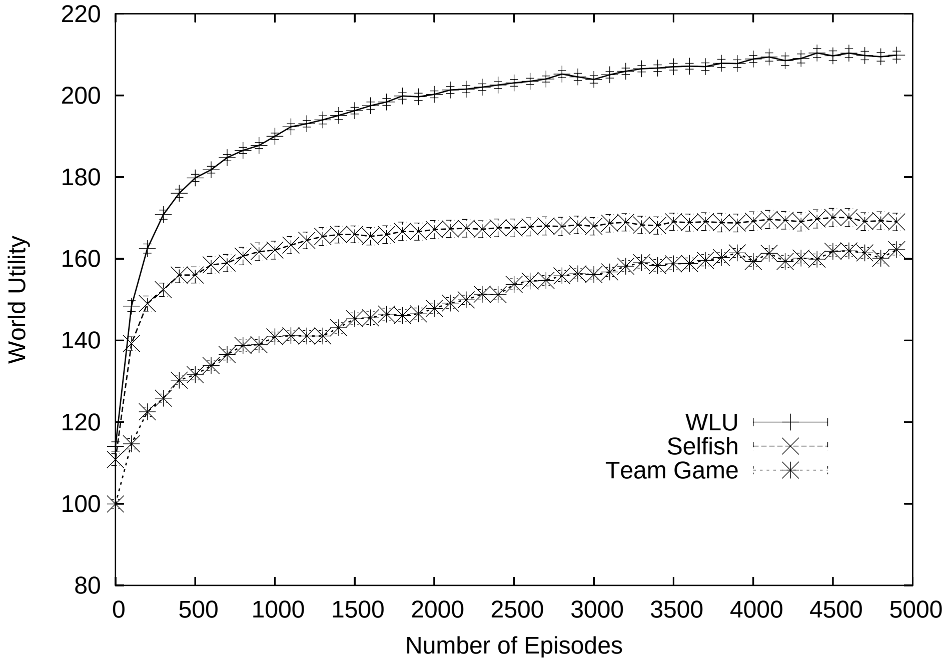


Figure 9.4: Results of rovers in world with fixed POIs where their value is decreasing as they are observed.

In the last experiment, the locations and values of ten POIs were set randomly at the beginning of each episode (see Figure 9.5). The locations were chosen from a uniform distribution, within the boundary of the scene. The changing of locations at each episode forced the rovers to create a general solution, based on their sensor inputs. Figure 9.5 shows that rovers using WLR still performed better than the other utilities, but this time there was an even smaller margin between their performance and that of rovers using SR. The results could come from this being a more difficult problem, even in the single agent case, making it harder to beat a greedy solution. Also since the POIs are more uniformly distributed in this problem, there may simply not be as much agent interaction as in the previous problem, allowing the greedy solution to perform well.

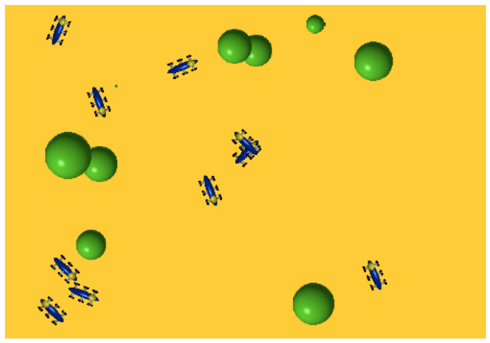
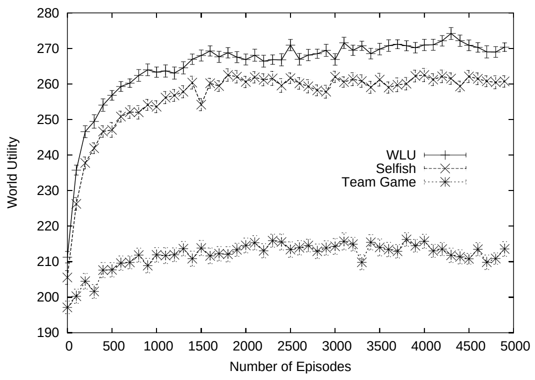



Figure 9.5: World where POIs are placed at random locations at the beginning of each episode. Diagram (left). Results for three different utilities (right).

9.2 Discussion

This chapter has shown that the theory of collectives can be applied to more realistic domains that are continuous in their state and action space. To this end, the chapter has also shown that the theory of collectives can be used with a much broader class of learners than table based reinforcement learners. In simple continuos problems, the neural network based system using utilities derived from the theory of collectives was able to show much better performance than a similar system using a selfish utility. In more difficult domains, the performance gap narrowed, but a performance gain was still achieved when one used alternate utilities to the selfish utility. It is likely that on difficult problems with increased agent interaction that the performance gap would again widen.

Chapter 10

Increasing Collective PageRank

The ranking of a page can be very important for the owner of that page. Since many pages are found through search engines, often only the highest ranked pages are read since they tend to be at the top of the list after a search engine query. The popular Google search engine uses an algorithm to rank pages called “PageRank,” which assigns a rank to each page, based on the topology of the web, i.e. its link structure [47]. Perturbation analysis shows that PageRank is robust against changes made to the directed web graph [46]. The robustness come from the fact that the PageRank algorithm uses information about the inlinks to a page to determine its rank. Since a page’s owner cannot normally change the inlinks to a page, he cannot manipulate the page to increase its rank. However, if a group of domains work together, the average PageRank for pages within the group can be increased significantly by changing the link structure within the group. Since PageRank only counts links between different network domains, this chapter will discuss links between domains instead of individual pages.

10.1 Reinforcement Learning Approach to PageRank Maximization

For this problem a collective learning approach can be implemented where each domain has its own reinforcement learning algorithm that learns which links to add to increase page rank, under a set of constraints. The reinforcement learners "learn," by first doing exploration, where they try a series of actions, consisting of adding links to various domains within the group. After each action, the reinforcement learner receives a reward based on how good the action was. The learner can then exploit the information it has gathered, and choose the actions that will result in the highest expected value of the objective function.

Traditional methods of increasing PageRank involve specific agreements between owners of domains on how to link their domains together. Often domain owners pay other domains to link to their domains. Alternatively some domains establish mutual linking agreements. Reinforcement learning has some significant advantages over traditional methods. First of all it can learn arbitrary objective functions. This property becomes very useful when the page owners want to put some soft constraints in the form of nonlinear penalties added to the ranking objective. Another advantage of using reinforcement learners is that they provide more robustness in this naturally distributed system. Since each domain has its own learner, it can adapt to problems caused by other domains. This ability to adapt is important since numerous problems could arise when a diverse set of domain owners is trying to work together. These problems could range from issues such as an owner neglecting to implement the algorithm correctly, or difficulties outside the owners' control such as a failure in a hosting service.

10.2 Experiments

Three experiments were conducted to determine how effective reinforcement learning is in increasing PageRank for a group of domains. The experiments were performed using artificial data with 50 domains where the number of outlinks of a domain was uniformly distributed in the range 0 to 25. In all three experiments, 10 random domains were chosen to comprise the set of cooperating domains. All links from a domain within this set to a domain outside of the set were removed. This was done to simulate the practice of many companies of avoiding linking to any page that is not within their own domain or a domain of a closely associated parter. Each of the 10 domains was given a simple immediate-reward reinforcement learner. The actions of the reinforcement learners was to choose how many links to add to domains within the set. These links were then made randomly since experiments showed that the exact target of the links did not influence PageRank significantly.

In each of the experiments, the overall goal was to maximize a global utility, $G(z)$, where z contains the link structure. An agent set the values of z_η, which were the additional links made by the agent. The experiment in Section 10.2.1 used a simple social welfare utility as the global utility. The experiments in Sections 10.2.2 and 10.2.3 used a more complex global utility that took into account a penalty for additional links. In the experiments, the agents tried to maximize their private utility. To test the performance of different types of private utilities, in each experiment three types of private utilities, $g_\eta(z)$, where used. The first utility was the Team Game (TG) utility where $g_\eta(z) = G(z)$. The second utility used was the Selfish Utility (SU), where a learner's reward was equal to its domain's PageRank. The final utility was the Wolderful Life Utility (WLU). Due to the nature of PageRank, the Selfish Utility is degenerate in many cased. An agent typically has very little control over the value of the SU, and in some cases it has no control at all. This causes the selfish utility to be nearly unlearnable, while retaining the property of not being factored. Due to its poor properties, it is reasonable that in many cases, a self interested agent may be willing to except an alternative to the SU as its private utility, even though in the end it wants to maximize its SU. A small external incentive mechanism such as reputation effect may be enough for an agent to accept an alternative to the SU. In the experiments it was just assumed that each agent would try to maximize the private utility given to them.

10.2.1 Social Welfare Objective

In this experiment the global utility, $G(z)$, is simply the average page rank, without any penalty on the number of links added:

$$G(z) = \sum_{i \in S} r_i(z) \tag{10.1}$$

The global utility here is a simple social welfare function with no "fairness" assumption. For this global utility, the optimal solutions was to trivially add as many links as possible. Despite having a staightforward solution, Figure 10.1 shows that learners using TG utilities and SU utilities could not achieve high values of the objective function. The TG utility failed because it was difficult for an individual learner to observe the influence of its action, among all the actions of the other learners. Even if a learner took an action that was good, some of the other learners may have taken bad actions at the same time, resulting in a local utility that had too small a value. The second utility tried was the Selfish Utility:

$$SU(z) = r_\eta(z) \tag{10.2}$$

This private utility was simply the rank of η's domain. The SU utility also failed, because a domain cannot change its own rank significantly by altering its links. Even if an agent could add inlinks and therefore significantly increase its SU, doing so would not necessarily lead to a higher global utility since the two utilities are not aligned. The next utility tested was the WLU, which can be expressed as follows:

$$WLU_\eta(z) = \sum_{i \in S} r_i(z) - \sum_{i \in S} r_i(z_{\hat{\eta}}) \tag{10.3}$$

The second term was computed by evaluating global utility when no links are added by η. The reinforcement learners using the WLU were able to learn to achieve high values for $G(z)$ very quickly. This is because the WLU was very learnable since an agent's choice of how many links to add significantly influenced the value of its utility. At the same time, the links added by the other agents did not influence the an agent's WLU as much as it influenced G.

In addition the WLU was factored so that when an agent tried to maximize its WLU, it tended to maximize G.

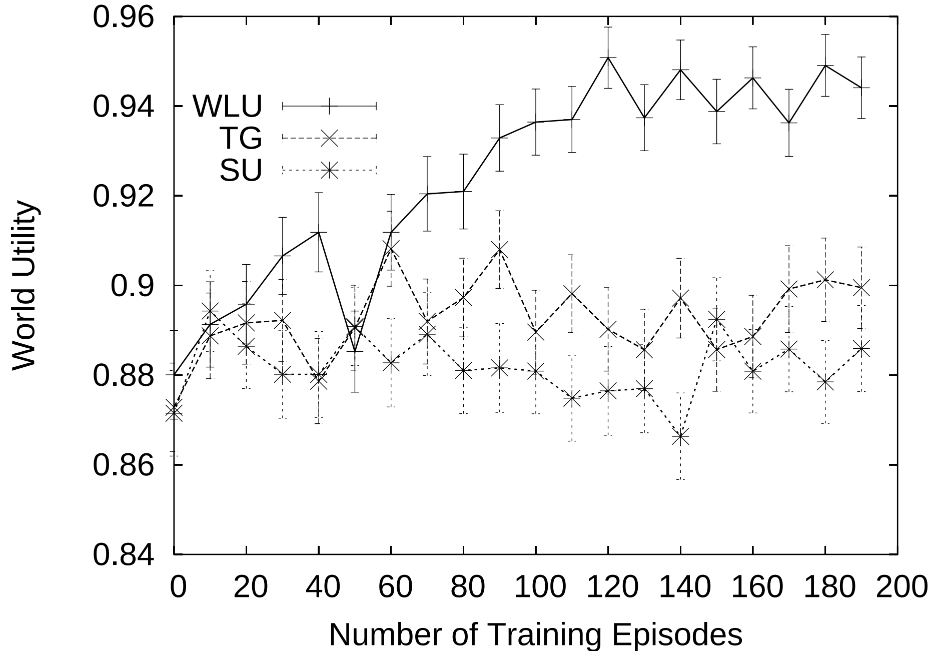


Figure 10.1: Performance of utilities, when there is no penalty for adding links

10.2.2 Link Penalties

To test reinforcement learning in a more interesting problem, the objective was modified to penalize the number of links added:

$$G(z) = (\sum_{i \in S} r_i(z))(1 - a \sum_{i \in S} e^{l_i(z)}) \tag{10.4}$$

where l_i is the number of links added from domain i in set S, a is a scaling factor, S is the set of domains, and r_i is the rank of domain i. Note also that this new objective was nonlinear and difficult to solve analytically. However the WLU could still be calculated in a similar manner as it was done in Section 10.2.1:

$$WLU_\eta(z) = (\sum_{i \in S} r_i(z))(1 - a \sum_{i \in S} e^{l_i(z)}) - (\sum_{i \in S} r_i(z_{\hat{\eta}}))(1 - a \sum_{i \in S} e^{l_i(z_{\hat{\eta}})}) \quad (10.5)$$

The results in Figure 10.2 show that reinforcement learners using the WLU were also able to learn the new objective function quickly.

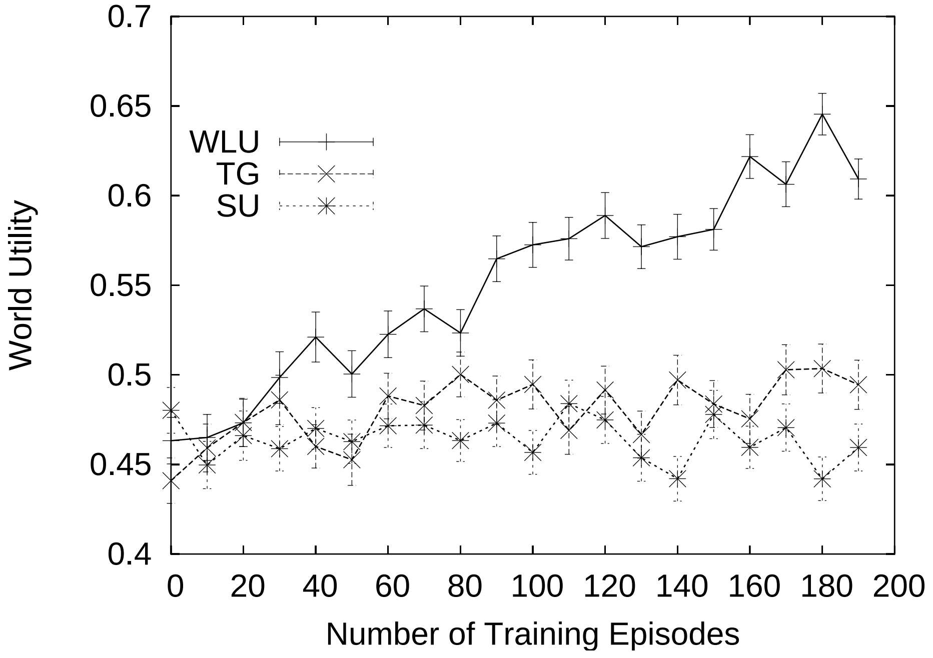


Figure 10.2: Performance of utilities, when there is a penalty for adding links

The solution formed by these learners was non-trivial with different domains choosing to add different numbers of links. Figure 10.3 shows outcome of one of the trials. The number of links chosen depends the PageRank of the domain as well as the initial link structure. For example, since the PageRank conferred to a target is divided by the number outlinks from the origin, one would expect that a domain with few outlinks would be more likely to add new links, since each new link would be more valuable to the target's PageRank. This is confirmed by the domain on the upper right of the figure, which started out with no outlinks, but added a large number of new links.

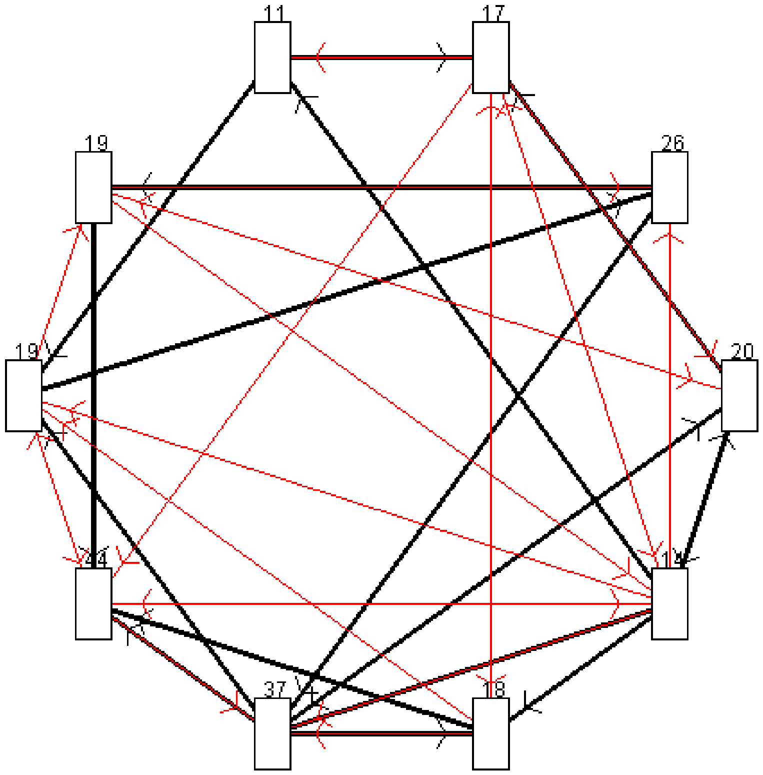


Figure 10.3: Final link structure when using WLU. Thick lines are the original links.

10.2.3 Noisy Agents

As a final experiment, how the reinforcement learners were able to learn when not all the domains cooperated well was tested. In this experiment, the same non-linear objective function was used, but this time 20% of the time the learners would not take any action. This would simulate a situation where a domain owner withdraws from the system unannounced, or neglects to modify his domain. Figure 10.4 shows that again the learners using WLU are able to achieve high values of the objective function, even when not every domain is cooperating.

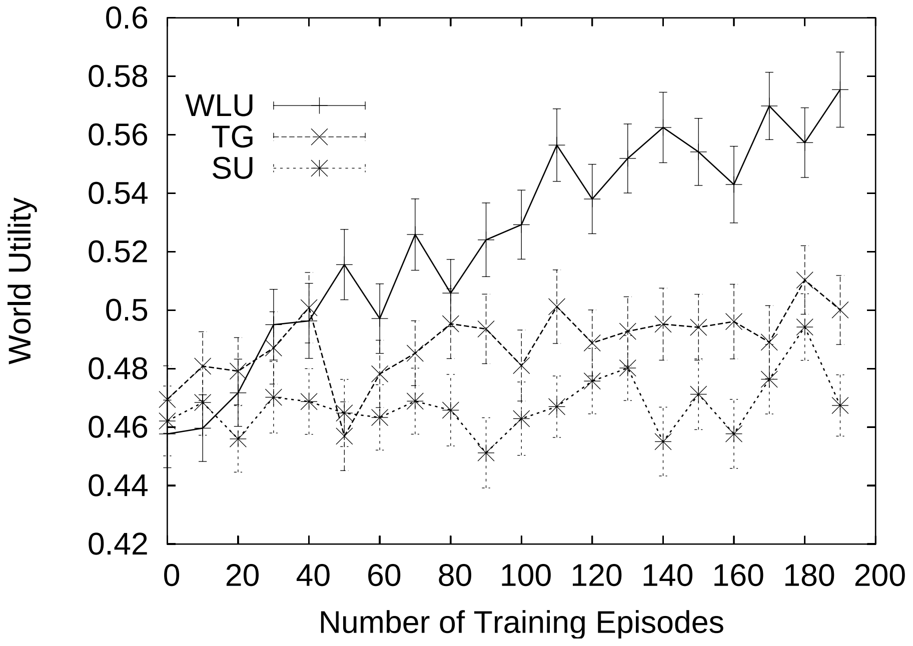


Figure 10.4: Performance when not all domains cooperate. Reinforcement learning approach is able to recover from misbehaving agents.

10.3 Discussion

Google proclaims that an individual page cannot significantly change its own PageRank value by adding selected keywords or outlinks. Previous perturbation analysis has supported this claim which helps Google to be robust against manipulation by individual commercial interests. However, it is possible that a set of domains "working together" can significantly improve their average PageRank, but this is a highly complicated problem involving distributed learning. This chapter presents a method, based on collective reinforcement learning, where a collection of domains can increase their average PageRank, under a wide set of constraints even while being subject to unreliable cooperation.

Chapter 11

Cluster Ensembles

Often there is a need to combine multiple clusterings, formed from different aspects of the same data set, into a single unified clustering. A system that does this is known as a *cluster ensemble*. Cluster ensembles are most useful when all of the original data points are not available to create a clustering, but separate clusterings of the data still exist. This situation could occur when some of the data points come from proprietary sources, where the data owners are willing to reveal their clustering of the data, but not the data itself. This situation can also occur if the original data points are simply lost, or have been thrown away, but the much smaller summery cluster data is saved. In addition even if all of the original data is available, it may be too big to store in one site of computation. In this case, it may be desirable to make separate clusterings of different parts of the data and combine them later.

Currently the best algorithms for cluster ensembles are graph-based methods. While these methods have shown to have good results on certain information theoretic measure, they tend to be inflexible. The graph-based methods require the cluster combining to happen on a single node of computation, creating a single point of failure. This makes them inappropriate for domains where there is high component failure such as in space-based operations. Instead of graph-based cluster ensemble methods, this chapter will propose using agents to solve the cluster ensemble problem. This agent-based method does not have a single point of failure. Instead the performance of the system will gracefully degrade with the number of failures. In addition the agent-based system is more flexible, allowing the cluster-combining computation to stop, when the solution is good enough for a particular task.

This chapter will first give a formal description of the cluster ensemble problem as well as a brief overview of a graph-based solution to the cluster ensemble problem. It will then describe a simple greedy approach, followed by the agent-based approach. Finally the chapter will show results for the performance of agent-based cluster ensemble methods, and how they are robust against agent failure.

11.1 Cluster Ensemble Problem

Suppose that there is a set of n data points. A clustering can be seen as a partition of the data, forming k subsets called clusters. This partition of the data will be referred to as a clustering. In general there can be many partitions of the same set, therefore the same data can lead to many different clusterings. To compare two clusterings, one can use information theoretic measures based on the sizes of the clusters within the clusterings (a formal justification for this is presented in [59]). First, the mutual information between clusterings X

and Y can be defined as:

$$I(X, Y) = \sum_{x \in X} \sum_{y \in Y} \frac{|x \cap y|}{n} log_2 \left(\frac{n|x \cap y|}{|x||y|} \right) \tag{11.1}$$

where $|\cdot|$ and $\cap$ are the set cardinality and intersection operators respectively. Next the entropy of a clustering, X, can be defined as:

$$H(X) = -\sum_{x \in X} \frac{|x|}{n} log_2 \left(\frac{|x|}{n} \right) \tag{11.2}$$

Now one can define the normalized mutual information (NMI) between two clusters, X and Y as:

$$\text{NMI}(X, Y) = \frac{I(X, Y)}{H(X)H(Y)} \tag{11.3}$$

which has the desirable property of being bounded by [0, 1] and having $\text{NMI}(X, X) = 1$. In addition a clustering X can be compared to a set of clusterings $\overline{Y}$ using average normalized mutual information (ANMI):

$$\text{ANMI}(\overline{Y}, X) = \frac{1}{|\overline{Y}|} \sum_{Y \in \overline{Y}} \text{NMI}(X, Y) \tag{11.4}$$

The goal of the cluster ensemble problem is to create a clustering that is as close as possible to the hidden "true" clustering in terms of NMI. In this chapter the problem will be approached by trying to find the clustering, X^*, that maximizes the ANMI between X^* and the set of available clusterings $\overline{Y}$.

11.2 Graph-based Approaches

The graph-based approaches to the cluster ensemble problem involve representing the data points as nodes in a hypergraph, and the clusters as undirected hyperedges of the hypergraph. This chapter will compare cluster ensemble performance with three of these algorithms, which are described in detail in [60]. The first algorithm called CSPA (Cluster-based Similarity Partitioning Algorithm) provides moderate performance, but has a computational complexity proportional to the square of the number of data points. CSPA works by creating a similarity measure between data points. Within a clustering, all data points in the same cluster have a similarity of 1, and data points from different clusters have a similarity of 0. If the similarities between data points are averaged over different clusterings, a new single clustering can be made with a similarity based clustering algorithm, such as METIS [37].

The second clustering algorithm is called HGPA (HyperGraph Partitioning Algorithm) and provides generally improved performance over CSPA and has a lower computational complexity, which is almost linear in the number of data points. This method creates the final combined clustering for the ensemble by using HMETIS [38] to perform hypergraph partitioning on the hypergraph that represents the clusterings. The objective of this method is to create clusters that break the least number of hyperedges. The final method, called MCLA (Meta-CLustering Algorithm), provides similar performance to HPGA and retains its low computational complexity. This algorithm works by collapsing a group of related hyperedges into a single hyperedge. This can be seen as clustering clusters.

11.3 Simple Greedy Optimization

Simple greedy optimization approaches have been applied to the cluster ensemble problem in [60]. In this method, one starts with a single representative clustering, X, which is usually the clustering that has the highest ANMI with respect to all the other clusterings:

$$X = argmax_{X' \in \overline{Y}} \text{ANMI}(\overline{Y} \backslash X', X') \tag{11.5}$$

where $\overline{Y}$ is the set of clusterings, and $\backslash$ is the set difference operator. For each data point, a new clustering is created from the previous clustering by moving the data point to a new cluster at random. If this new clustering has a higher ANMI than the previous one, then it is preserved, otherwise it is thrown away. The algorithm is repeated for each data point. When it has gone through all the data points, the algorithm stops if all of the new clusterings where thrown away. Otherwise it repeats through all the data points. This algorithm can be seen as a form of serial simulated annealing with zero exploration or as a Stackelberg game [28]. Since the exploration is zero, the system will only reach a local maximum. Also since for each loop, the ANMI has to be computed for every data point, the computational complexity is very high when there are many data points.

11.4 Collective-based Approach

The cluster ensemble problem can be approached by assigning one or more agents to each cluster in the original set of clusterings. If m agents are assigned to each cluster, then a collective for r clusterings with k clusters each would have mrk agents. The action of an agent is to vote on which cluster in the combined clustering, the data points in its assigned cluster should belong to. A data point will then belong to final cluster that got the most votes from agents that were assigned to a cluster containing the data point. The global utility of the system is the ANMI of the combined clustering with respect to the original clusterings. Agents can then try to maximize the ANMI, by using their private utilities to help them choose the best actions.

Formally the votes by the agents for a combined clustering Z can be represented by a two dimensional array of sets indexed by agents and clusters. If agent η chooses cluster $z \in Z$, then $V_{\eta,z}$ equals the set of data points in agent η's assigned cluster. If agent η did not choose z then $V_{\eta,z}$ equals the empty set. The number of votes for data point p to be put in cluster z can be formulated as follows:

$$N_{p,z} = \sum_{\eta} I_{p \in V_{\eta,z}} \tag{11.6}$$

where I is an indicator function. The assignments of data points to a cluster in the combined clustering can now be expressed as:

$$z = \{p | N_{p,z} = max_{z'} N_{p,z'}\} \tag{11.7}$$

The global utility can then be defined as the ANMI between Z and the initial set of clusterings $\overline{Y}$:

$$G(\overline{Y}, V) = \text{ANMI}(\overline{Y}, Z(V)) \tag{11.8}$$

Finally the WLU can be defined as:

$$\mathrm{WLU}(\overline{Y}, V) = \mathrm{ANMI}(\overline{Y}, Z(V)) - \mathrm{ANMI}(\overline{Y}, Z(V')) \tag{11.9}$$

where V' is the same as V, except that all of the elements of V' that are indexed by η are equal to the empty set.

11.5 Results

The performance of the collective approach to cluster ensembles relative to other methods was tested with three data sets. The first data set was artificial, while the other two came from real-world problems. The experiments showed that the collective approach achieved superior performance in some domains, but was inferior in others. However they also showed that the collective approach was able to recover from a number of types of agent failures. The cluster ensemble using this framework benefited from robustness, not necessarily performance. In addition it also provided decentralization, allowing for great flexibility in how this computation could have been performed.

All agents learned with a simple single-time-step reinforcement learner. This learner was equivalent to a Q-learner with γ equal to zero. At every time step, each agent would choose one of k clusters based on its estimates for the utility payback for that choice. These estimates were stored in a utility table of size k. After all the agents chose their cluster, each agent would compute their private utility and use it to modify the payback estimate in the utility table.

11.5.1 Noisy Ensembles

In the first experiment, four hundred data points where randomly placed into ten clusters to form an initial clustering. This initial clustering was duplicated eight times. In each duplicate clustering, random noise was added by moving a random subset of the data points to new clusters (for each data point in the subset, the cluster it was moved to was chosen independently over a uniform distribution over all the clusters). The amount of random noise was specified by B, the fraction of all the data points contained in the subset. These duplicate clusterings were used as the ensemble clusterings and experiments were performed with B ranging from 0.0 to 1.0. The performance of the algorithms were determined by the NMI between the clustering produced by the algorithm and the initial clustering. Figure 11.1 shows the results for agents using WLU as their utility along with results obtained from the graph-based methods and the simple greedy method.

The results show that the agent-based method performs about the same as the simple greedy method. However this performance was achieved with much less computation. The greedy method was reported to take an hour on a 1Ghz PC. The agent-based method took only 45 seconds on a similar computer. The agent-based method performed slightly worse than the best graph-based method, MCLA, but performed considerably better than the worst graph based method, CSPA. It also performed about about the same as HGPA (not shown in graph).

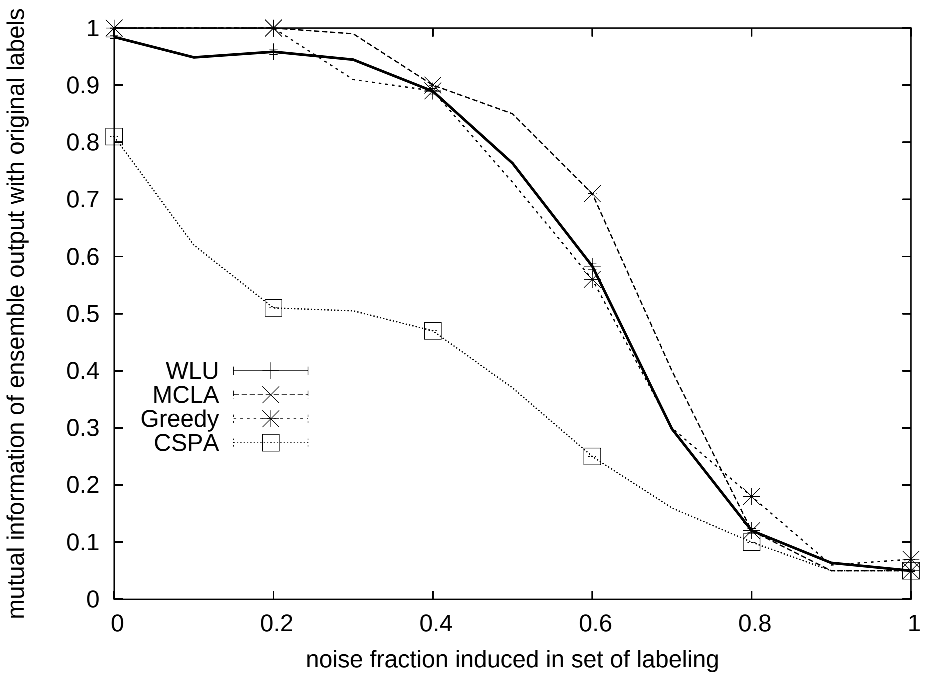


Figure 11.1: Results of ensemble methods with noisy clusterings.

11.5.2 Yahoo Data Set

In this experiment a processed version of the Yahoo! data set was used. This data set has also been used in [61] and [12]. The data set contained 2340 data points, with each data point having 2903 features. Each data point represented a document and the features were a pruned set of word frequencies contained in the document. The data set was clustered into twenty clusters, based on which Yahoo! news category they were originally placed in. The data set was then broken down into twenty separate data sets. Each new data set contained all of the 2340 data points, but each data point only had a 128 element subset of the original features. Each new data set was then clustered using METIS, forming twenty clusterings. These new clusterings were then used for the ensemble in the experiment. The output of the ensemble algorithm was then compared to the original Yahoo! clustering using NMI. Figure 11.2 shows the results between an agent-based method using WLU as its utility, an agent-based method using G as its utility and the graph-based method, MCLA.

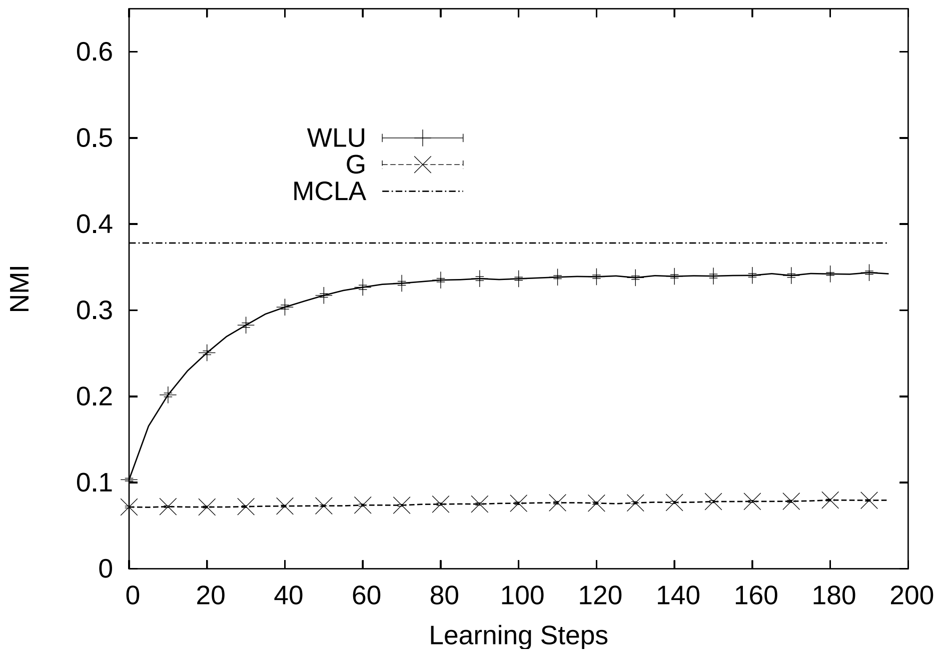


Figure 11.2: Results of ensemble methods for the Yahoo data set.

The results show that agents using the global utility were unable to produce an adequate clustering. This is not surprising since there were 400 agents in the system. In this case, the learnabity of the global utility was very poor since an agent cannot easily see the contribution of its action on the global utility, amongst the 399 actions of the other agents. In contrast the agents using WLU were able to do much better. However these agents were not able to do as well as the MCLA algorithm, even after running for 200 time steps [1].

11.5.3 Pendig Data Set

The Pendig data set contained one thousand data points, with each data point having sixteen features. Each data point represented a processed handwritten digit. The data points were clustered into ten clusters, with each cluster representing a digit from '0' to '9.' Similar to the Yahoo data set, the Pendig data set was broken down into ten data sets. Each point in the new data sets contained four features sampled from the original sixteen. Ten clusterings were then produced by clustering each of the new data sets with METIS. Using these clustering for the ensemble, the performance of MCLA was compared to an agent-based system using WLU. In this experiment three agents were used for every cluster to provide redundancy. This redundancy was tested under three failure scenarios. In the first scenario 50% of the agents were "broken." These broken agents (called "random agents") chose a random cluster at every time step. In the second scenario 50% percent

[1]Note that the results for MCLA presented here for the Yahoo data set or the Pendig data set cannot be directly compared to similar results shown in [59, 60], since the ensemble clusterings are different. Because the METIS algorithm used to create the clusterings is random, different sets of clusterings can be created from the same data set. Higher performance was shown in [59, 60] for MCLA since in some sense the clusterings that were created for those papers happened to be "easier" to combine.

of the agents (called “fixed random agents”) chose a random cluster at the beginning of a trial and kept making the same choice throughout the trail. In the final scenario 10% of the agents always chose the first cluster (these are called “0” agents). The results shown in Figure 11.3, reveal that agent-based cluster ensembles can perform well in a number of adverse conditions. The agent-based system where none of the agents were broken was able to perform significantly better than the best graph-based cluster ensemble. In addition, even when half of the agents were broken, the system could still out-perform the graph-based methods. Note that the agent-based system in the first scenario performed worse than the one in the second scenario, even though the same number of agents were broken. This can be explained by the adaptively of the agents. In the second scenario the broken agents always took the same wrong move, so the working agents could adapt to overcome the adversity. In the first scenario, the broken agents simply added random noise to the system, which could not be easily adapted to. Despite the inherent failure tolerance of the collective, there was still a possibility of catastrophic failure if too many of the agents failed in the same way. While the system was able to recover when 10% of the agents choose the same wrong cluster, the performance drops to zero if 50% percent of the agents choose the same wrong cluster. This failure is caused by the voting scheme, since when 50% percent of the agents vote for the same cluster, that cluster will always win. In this case, the final clustering will always have just one cluster.

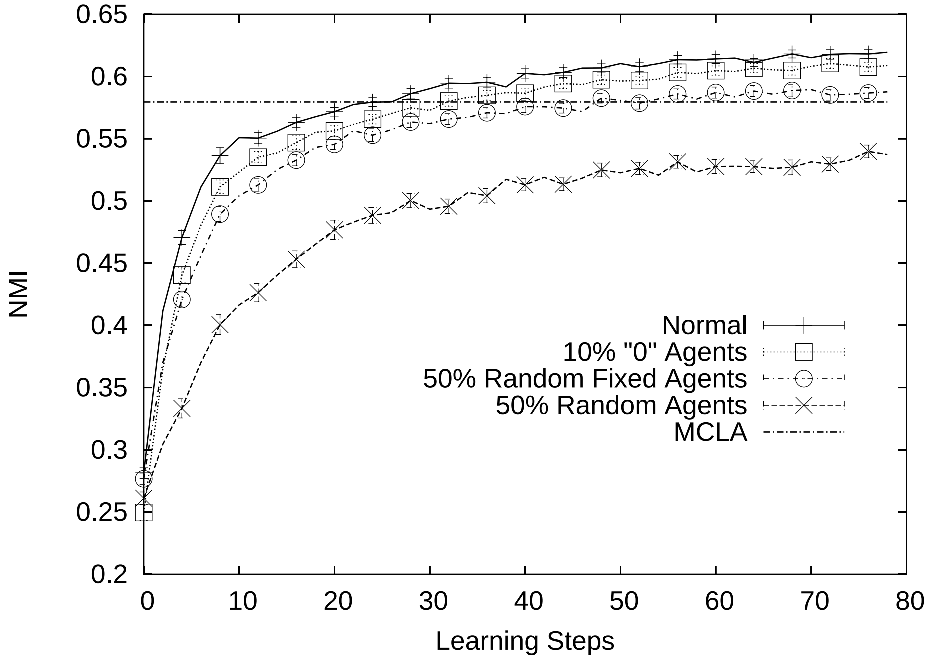


Figure 11.3: Results of ensemble methods for the Pendig data set.

11.6 Discussion

Using the WL Utility, agent-based cluster ensembles showed that they could perform comparably to the best existing cluster ensemble methods. In addition they were able to achieve this level of performance with relatively little computation as compared to simple greedy optimization methods. Also the agent-based cluster ensembles were able to show a high level of fault tolerance. This property allows them to be used in domains that have a high component failure rate, such as in space-based systems. In addition the agent-based cluster ensembles have the flexibility to be stopped early in domains where a fixed level of performance is needed. This ability can save computational costs, even when the difficulty of the ensemble problem is not known ahead of time.

Chapter 12

Visualization of a Collective

For collectives where agents have complex interactions, such as in the continuous rover problem, visualization can be essential to understanding how the system behaves. Proper visualization provides far more information than summary utility values, and is necessary for debugging and for designing high performance utilities. For domains with physical agents, one of the most important forms of visualization is simply the animation of the agents interacting in the environment through time. Snapshots of this type of visualization for the continuous rover problem have already been provided in this chapter, such as in Figure 9.5. This form of visualization is essential in determining how well the agents are coordinating. In addition it shows the agent designer how well the collective is accomplishing its task, which is especially important when the highest value of global utility that the collective can expect to reach is unknown. Domain animation also shows how well the agents are learning through time, and often suggest areas in which their learning can be improved. Lastly domain animation provides a sanity check to ensure to the collective designer, that the system is operating as expected.

While useful, domain animation has a number of deficiencies. Foremost, visualization often has to be presented in a static form. While single time snapshots of domain animation can be used, they may convey very little information or may even be misleading. Furthermore, additional information may need to be visualized that is not present in domain animation, such as utility values. Many forms of agent visualization have been researched, but they range from being too domain specific [16] to being too general [55] to give useful information about the collective. This chapter will focus on two related forms of visualization, specific to domains where agents move on a two-dimensional surface. The first one will involve direct utility visualization, and the second one will involve intelligence visualization.

12.1 Reward Visualization

One method of visualizing rewards is to plot the values of the rewards received at the locations where they were received. For example if a rover were in location (x, y) and received a reward value of r, then the value of r would be plotted at location (x, y) (after the appropriate affine transformation to put it in screen coordinates). If very few rewards were received then the value of the reward may simply be printed as text. However, usually there will be too many utilities values computed to do this. For example a typical experiment in the Continuous Rover Problem has 10 rovers, a 15 step episode and a 5000 episode learning period. This experiment would result in 750,000 reward values that need to be plotted. Even if there were very few reward values that need to be plotted, they may have been computed at nearby locations, and text printing of reward values will overlap.

In most situation it is preferable to plot the reward value as a light intensity instead of printing out its value as text. The first issue that needs to be addressed is the mapping of the continuous domain coordinates to the discrete pixel coordinates in the visualization. For a pixel size of p let the set of all reward values associated with that pixel be:

$$U_{x,y} = \left\{ u_i : \left\| L_i - \begin{bmatrix} x \\ y \end{bmatrix} \right\|_1 < \frac{p}{2} \right\} \tag{12.1}$$

where $\| \cdot \|_1$ is the L1 norm and L_i is the location of rover i when it received reward u_i. The reward plot can now be made by assigning the pixel value at each point $V_U(x,y)$ the average values of the utilities received at that point:

$$V_U(x,y) = \frac{\sum_{u \in U_{x,y}} u}{|U_{x,y}|} \tag{12.2}$$

where $|U_{x,y}|$ is the number of elements of the set $U_{x,y}$. Since visual displays have a maximum range of intensity, the reward values need to be normalized. However this normalization can cause difficulties. If the maximum and minimum reward values are not known ahead of time, real-time plotting of data may force the image to be re-normalized in real-time, causing disturbing visual effects. In addition even if the maximum and minimums are known, reward normalization may produce an unacceptable image if there are outliers or extremes in the data. For example if a single data point had a much higher value than all the other data points, that single data point would appear white, while the rest of the image would appear black. Often the normalization will have to be done on a case by case bases, using appropriate monotonic transforms. Often simple smoothing and a change of the pixel size can be used as an alternative to non-linear scaling of utilities. Increasing the pixel size will tend to smooth over extreme values. In addition it can be helpful when comparing two visualizations as shown in Section 12.2.

Using simple affine normalization with a large pixel size, the global reward for the Continuous Rover Problem can be plotted as shown in Figure 12.1. This visualization is not very useful since it shows that high reward values are received throughout the space. This happens because the global reward is plotted, which is the same for each rover. In some ways this graph shows little more then the distribution of agent locations. However it also shows that this reward is hard to learn from due to the credit assignment problem inherent in the global reward. At each location that the rover is in, it receives about the same reward, which is dominated by the noise of other agents.

As an alternative to the global reward, the selfish reward (SR) for each rover can be plotted instead as shown in Figure 12.2. This visualization shows a more local picture of the rewards received throughout the landscape. As expected, the visualization shows that high rewards are received close to POIs. Less expected is that high rewards are also received in locations fairly far from a POI. This happens because a rover still receives information from a POI when it is far away. However no additional information is gained over rovers that are closer to the POI. Since rovers far away from a POI are unlikely

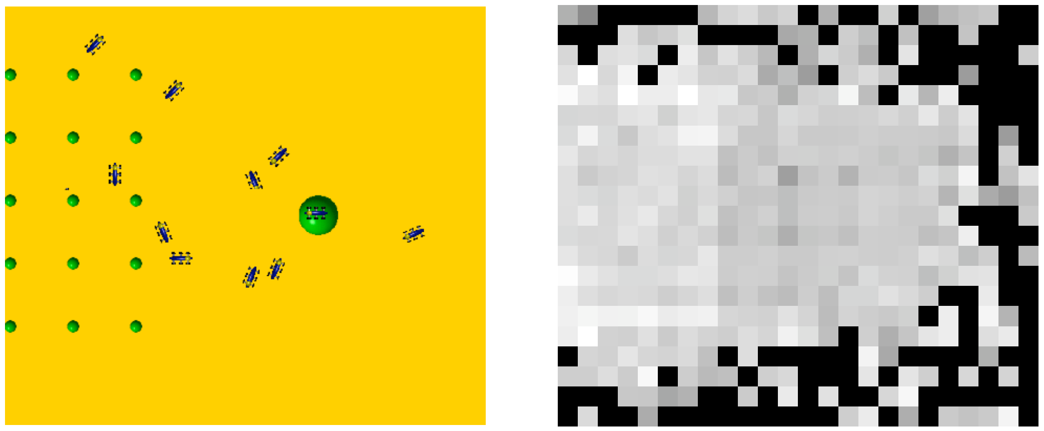


Figure 12.1: The values of the global reward received by the rovers are visualized on the right from the domain shown on the left. The global reward visualization is not very useful, because of the credit assignment problem.

to contribute to the global reward, this diagram shows that the rovers are receiving rewards that are not factored in practice (it was already known that the selfish reward is not factored in principle). The diagram reveals that using the SR, rovers may be encouraged to explore regions that are useless with respect to reward. In contrast to the selfish reward, the WLR values match closely with the locations of the POIs as shown in Figure 12.3.

12.2 Intelligence Visualization

Reward visualization imposes a number of problems related to how to interpret the scale of reward values. Reward values are not only hard to draw, but also hard to compare. For example if the collective designer wants to see how well the rovers are learning to maximize their rewards at specific locations, he may want to perform a reward visualization. However some rewards, such

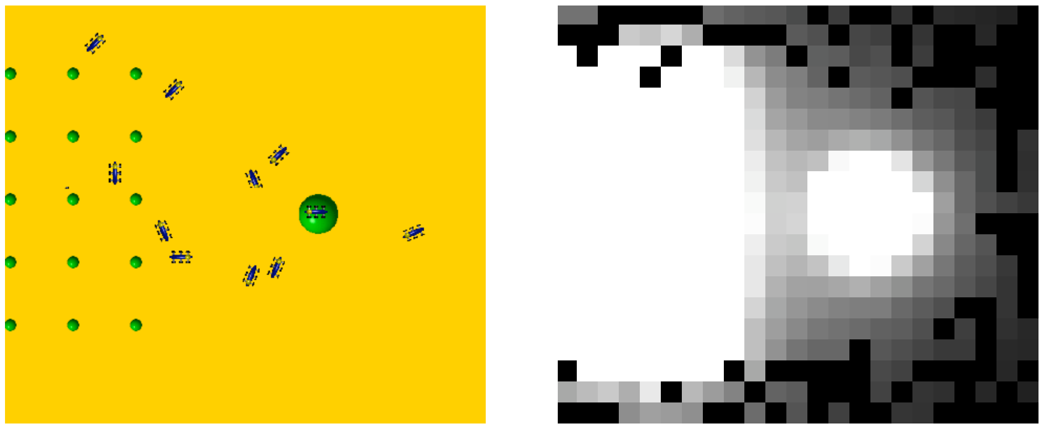


Figure 12.2: The values of the selfish rewards received by the rovers are visualized on the right from the domain shown on the left. The visualization shows how the selfish reward is not very factored, since high reward values are received far from POIs, where rovers are unlikely to contribute to the global reward.

as the upscaled global reward, will tend to give higher values than the WLU. In addition it may be hard to scale the rewards appropriately.

An alternative to rewards are intelligences, which were introduced in Chapter 3. Since intelligences measure a percentile rank of how good an action is compared to all other actions, it is invariant to scaling the reward. The intelligence of an agent's action at a particular location can be approximated through sampling actions at the agents location. For example for a rover η, the estimate for the intelligence of its action can be computed as follows:

$$\epsilon = \frac{\sum_{z'_\eta \in A} \Theta(U(z) - U((z_{\hat{\eta}}, z'_\eta)))}{|A|} \tag{12.3}$$

where A is a random set of actions that rover η could have taken. Note that intelligence values are bounded between zero and one, making them possible to visualize without scaling.

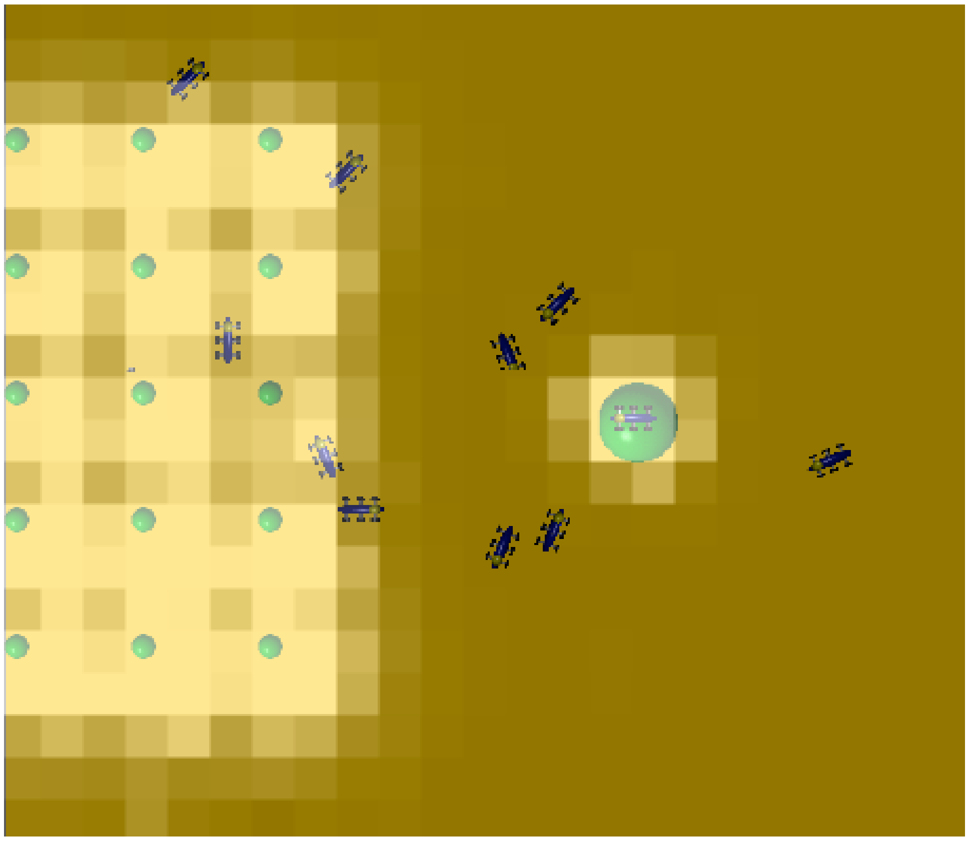


Figure 12.3: WLR values overlaid on the diagram of the domain. The reward values closely match areas in which one would expect a rover to be able to contribute to global reward.

Once the intelligences are computed, they can be mapped to pixels in the same way that rewards are. Figure 12.4 shows the intelligence visualizations of three rewards in the Continuous Rover Problem. Notice that the intelligence visualization for the WLR is nearly the same as for the global reward. This similarity occurs since the intelligence of actions with respect to factored rewards are the same, and WLR is factored with respect to the global reward. The differences arise due to the sampling error in the estimate the intelligences. The intelligence visualization of the selfish reward however is much different. This visualization tells the collective designer that while the agents have done well at maximizing their selfish rewards, in many locations they are not acting intelligently with respect to the global reward.

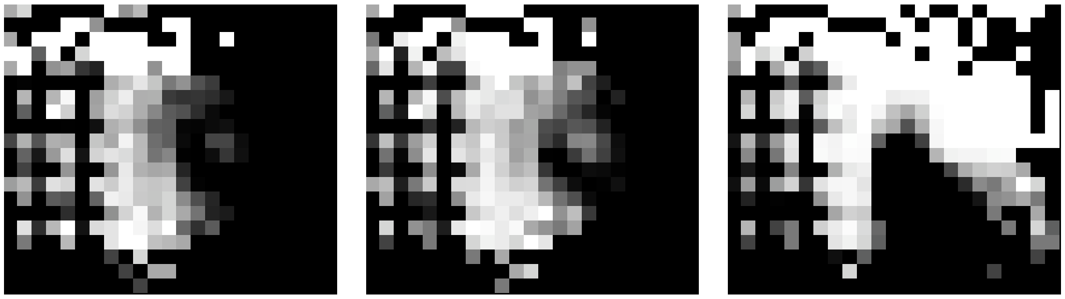


Figure 12.4: From left to right the intelligence visualizations for global reward, WLR and selfish reward. With perfect estimation, intelligence visualizations of factored rewards should be identical. For approximations they are still close such as in the left and middle figure. The right figure is very different showing that the selfish reward is not aligned with the global reward in many places.

By subtracting the intelligence visualization of the global reward, from the intelligence visualization of the reward the rovers are using, it is possible to get a picture of the locations where the reward used is not well aligned with the global reward. This chapter will call this a *Difference Visualization* of reward U, where each point of the difference visualization is equal to the absolute value of the difference between the intelligence with respect to U and intelligence with respect to G:

$$V_{diff}(x,y) = |V_{\epsilon_U(x,y)} - V_{\epsilon_G}(x,y)| \tag{12.4}$$

Figure 12.5 shows the results of the difference visualization for the WLR and the SR. As expected visualization for WLR does not show much of a difference. The small difference that exists are uniformly distributed across the domain, which is consistent with the differences being caused by random sampling error. In contrast the difference visualization for the SR shows that there are many areas in which the SR is not aligned with the global reward. Furthermore, the areas of misalignment are localized to certain regions. This information can tell the collective designer in what parts of the domain the selfish reward may be appropriate, and in which parts it should be changed.

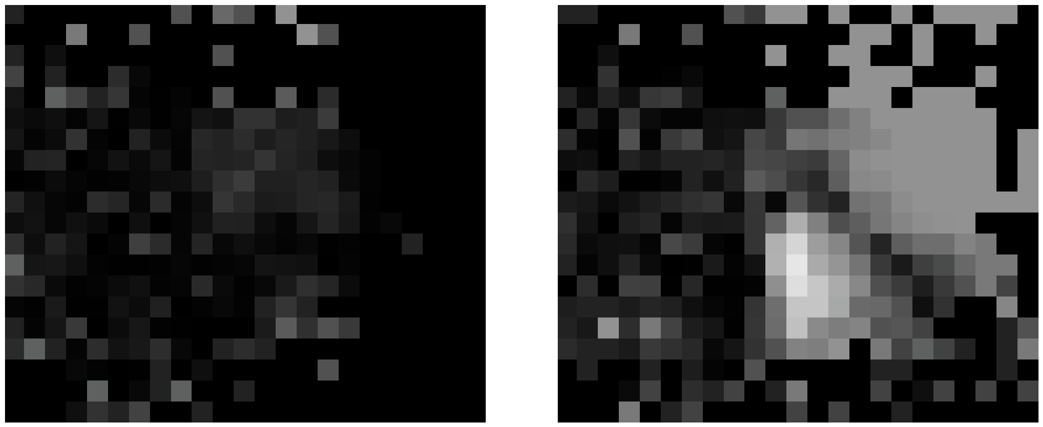


Figure 12.5: In both figures the intelligence diagram for the global reward is subtracted from the intelligence diagram of WLU (left) or selfish reward (right). Areas in which the rewards are not aligned with the global reward show up as brighter.

Figure 12.6 gives more insight into the misalignment between the selfish reward and the global reward. The left figure shows that most of the misalignment occurs near the POI that has a large value. This is expected since it is an attractive target for rovers, yet having many rovers in that area will not help the global reward. The figure also shows that the selfish reward is satisfactorily aligned in areas where there are more POIs than rovers. This suggest that in domains where there are many POIs of uniform value, that the selfish reward should work well enough. The graph on the right gives a little more insight into how the selfish reward is misaligned. In areas to the top/right side of the domain, the selfish reward consistently causes the rover to over-predict the intelligence of their actions with respect to global reward. This happens since the selfish reward will return high values when the rovers head towards the large POI, even though it is not a useful action as far as the global reward is concerned. More towards the center of the graph, the selfish reward causes the rovers to under-predict their intelligence with respect to global reward. This happen when rovers occasionally move away from the large POI towards many of the small POIs. The selfish reward returns low values for this type of action, when in reality it is a very good action with respect to global reward.

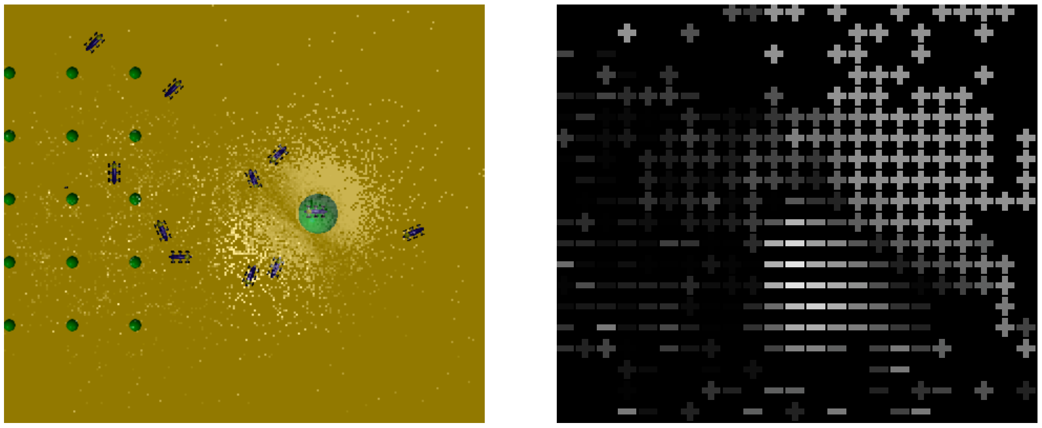


Figure 12.6: The left figure shows a high resolution version of the difference visualization for the selfish reward, overlaid on the diagram of the domain. The figure on the left shows "+" or "-" symbols for areas where the selfish reward causes intelligence over-prediction or under-prediction respectively.

12.3 Discussion

This chapter introduced a visualization method applicable to domains similar to the Continuous Rover Problem in Chapter 9, where the location a reward was received has some significance. In this problem, the locations were directly visualizable on a two dimensional surface; however, the same visualization technique could be used in higher dimensions with the appropriate projection. In the Continuous Rover Problem, the visualization showed in a spacial manner, what parts of the rover domain were causing the most difficulties. Also the intelligence visualization was able to show how well rovers using different private utilities where able to act towards the global objective. Intelligence visualization may also be used to visualize other large collective systems as well. Since intelligence values are bounded between zero and one, their visualization can be easily interpreted even if agents are not naturally embedded in a two dimensional domain. Intelligence visualization is most useful when there are so many agents that it is difficult to analyze agent performances individually. In addition intelligence visualization could be used to visualize traditional systems such as MLPs and RBFs that have been decomposed into agents as shown in Chapter 7. These intelligence visualization may provide alternatives to existing visualization methods for RBFs [1, 3] and MLPs [34].

Chapter 13

Conclusions and Future Work

This dissertation greatly expands the scope, power and flexibility of the theory of collectives, while offering increased interpretability. These contributions make collectives easier to use and makes the theory of collectives a more valuable tool in the solution to many real-world multi-agent problems. Collectives can now be used in a myriad of domains, including ones with any combination of missing information, time-extended rewards and continuous state/action spaces. In addition these problems can be tackled using a variety of tools, ranging from standard reinforcement learners to complex genetic algorithms to very simple evolutionary computation.

The initial contribution of this dissertation was to show how missing information can be handled while preserving the best properties of factordness and learnability. New utilities were created that were shown to preserve good performance characteristics under adverse communication conditions. These utilities were then leveraged to show how the structural credit assignment problem in collectives was related to the temporal credit assignment problem in reinforcement learning. Later solutions were provided for multi-agent time-extended domains, which contain both types of credit assignment problems.

The utilities in these solutions were shown to be factored structurally and across time. In addition the relationship was shown between the types of problems genetic algorithms and collectives can solve. These were important continuous problems, that were often well solved by neural evolution. It was shown that a complicated single-agent genetic algorithm problem could be better addressed by a multi-agent collective with simple evolutionary learner, using improved utilities. It was also shown that neuro-evolution could also be effectively leveraged in complex, continuous multi-agent problems.

Despite these improvements, there are still additions that can be made to the theory of collectives. In domains with complex communication restriction issues, a number of concepts used in this dissertation can be pushed further. Team formation may be a powerful tool in handling missing information and may be able to leverage such principles as small world structures to greatly increase the amount of information available while needed very little communication infrastructure. Dynamic team formation may even be able to optimize local communication networks, through learning. In addition to new methods in handling missing information, many new utilities can be made to deal with time-extended domains. This dissertation provided one solution to this problem, but many alternatives are possible. Time-extended utilities can differ in terms of bias, depending on how much local interaction there is in the collective and how distant through time rewards are dependent on earlier actions. Also some utilities may be able to use different informations depending how much is known about the structure of the environment and the global utility. With improvements in communication methods and an increasing assortment of utilities, collectives will become an even more valuable in multi-agent domains.

Bibliography

[1] A. Agogino and J. Ghosh. Visualization of RBF networks. In *Intl. Joint Conf. on Neural Networks*, Washington, DC, July 1999.

[2] A. Agogino and J. Ghosh. Increasing pagerank through reinforcement learning. In *Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks*, volume 12, pages 27–32, St. Louis, Missouri, Nov 2002. ASME Press.

[3] A. Agogino, J. Ghosh, S. J. Perantonis, V. Virvilis, S. Petridis, and P. Lisboa. The role of multiple, linear-projection based visualization techniques in rbf-based classification of high dimensional data. In *Intl. Joint Conf. on Neural Networks*, Como, Italy, July 2000.

[4] A. Agogino, K. Stanley, and R. Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38, 2000.

[5] A. Agogino and K. Tumer. Team formation and communication restrictions in collectives. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, July 2003.

[6] M. Alden, Aard-Jan van Kesteren, and R. Miikkulainen. Eugenic evolution utilizing a domain model. In *Proceedings of the Genetic and Evo-*

lutionary Computation Conference (GECCO-2002), San Francisco, CA, 2002.

[7] W. B. Arthur. Complexity in economic theory: Inductive reasoning and bounded rationality. *The American Economic Review*, 84(2):406–411, May 1994.

[8] Robert Axtell. Non-cooperative dynamics of multi-agent teams. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1082–1089, Bologna, Italy, July 2002.

[9] Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

[10] J. Bates, B. Loyall, and S. Reilly. Broad agents. In *Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures*, Stanford, CA, 1991.

[11] Liad Blumrosen and Noam Nisan. Auctions with severely bounded communication. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, Vancouver, Canada, November 2002.

[12] D. Boley, M. Gini, R. Gross, S. Han, K. Hastings, G. Kary pis, V. Kumar, B. Mobasher, and J. Moore. Partitioning-based clustering for web document categorization. *Decision Support Systems*, 27(3):329–341, 1999.

[13] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge*, Holland, 1996.

[14] J. A. Boyan and M. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems - 6*, pages 671–678. Morgan Kaufman, 1994.

[15] J. A. Boyan and A. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.* AAAI Press, 1998.

[16] R. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. Feature-interaction visualization and resolution in an agent environment. In *Fifth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, Lund, Sweden, 1998.

[17] Paolo Busetta, Antonia Dona, and Michele Nori. Channeled multicast for group communications. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1280–1287, Bologna, Italy, July 2002.

[18] G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communications networks. In *Proceedings of Fifth International Conference on Parallel Problem Solving from Nature*, pages 673–682, 1998.

[19] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1994.

[20] D. Challet and Y. C. Zhang. Emergence of cooperation and organization in an evolutionary game. *Physica A*, 246(3-4):407, 1997.

[21] J. Cheng and M. Wellman. The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computation Economics*, 1998.

[22] P. Cohen and H. Nous Levesque. Teamwork. *Special Issue on Cognitive Science and AI*, 25(4):487–512, 1991.

[23] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.

[24] M. Dastani, J. van der Ham, and F. Dignum. Communication for goal directed agents. In *Proceedings of the Agent Communication Languages and Conversation Policies*, Bologna, Italy, July 2002.

[25] Frank Dignum, Barbara Dunin-Keplicz, and Rineke Verbrugge. Agent theory for team formation by dialogue. In *Proceedings of the Agents, Theories, Architectures and Languages (ATAL 2000)*, pages 150–166, Boston, MA, July 2000.

[26] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, 2000.

[27] Jakob Fredslund and Maja J Mataric. Robots in formation using local information. In *Proceedings, 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, pages 100–107, Marina del Rey, CA, March 2002.

[28] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[29] D. Gage. How to communicate with zillions of robots. In *Proceedings of SPIE Mobile Robots VIII*, pages 250–257, Boston, MA, 1993.

[30] A. Goel, T. Liu, and K. Barber. Conflict resolution in sensible agents. In *Semiotic Modeling for Sensible Agents Workshop, Intelligent Systems Conference*, pages 80–85, Gaithersburg, MD, 1996.

[31] F. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1356–1361, Stockholm, Sweden, 1999.

[32] Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 2(86):269–357, 1996.

[33] B. Hayes-Roth, L. Brownston, and R. van Gent. Multi-agent collaboration in directed improvisation. In *Proc. of the First Int. Conference on Multi-Agent Systems*, pages 148–154, San Francisco, 1995.

[34] G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1986.

[35] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, June 1998.

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

[37] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[38] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. In *Proceedings of the Design and Automation Conference*, 1997.

[39] S. Kraus. Negotiation and cooperation in multi-agent environments. *Artificial Intelligence*, pages 79–97, 1997.

[40] C. Martin and K. Barber. Representation of autonomy in distributed agent-based systems. In *Semiotic Modeling for Sensible Agents Work-*

shop, Intelligent Systems Conference, pages 67–72, Gaithersburg, MD, 1996.

[41] Maja J Mataric. Reward functions for accelerated learning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–189, San Francisco, CA, 1994.

[42] Michael Mateas. An oz-centric review of interactive drama and believable agents. Technical Report CMU-CS-97-156, Carnegie Mellon University, Pittsburgh, PA, 1997.

[43] T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, Boston, MA, 1997.

[44] D. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, pages 2:11–32, 1996.

[45] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(48-49), 1950.

[46] A. Ng, A. Zheng, and M. Jordan. Stable algorithms for link analysis. In *Proceedings of SIGIR'01*, 2001.

[47] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[48] S. A. Petersen and M. Divitini. Using agents to support the selection of virtual enterprise teams. In *Proceedings of Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) (at AAMAS 2002)*, Bologna, Italy, July 2002.

[49] D. Pynadath. and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[50] D. Pynadath, M. Tambe, M. Arens, and Y Chalupsky. Electric elves: Immersing an agent organization in a human organization. In *Proceedings of AAAI Spring Symposium on Agents in Cyberspace*, 2000.

[51] David Pynadath, Milind Tambe, Nicolas Chauvat, and Lawrence Cavedon. Toward team-oriented programming. In *Proceedings of the Agents, Theories, Architectures and Languages (ATAL'99)*, pages 77–91, Orlando, Florida, July 1999.

[52] S. Rumeliotis, P. Pirjanian, and M. Mataric. Ant-inspired navigation in unknown environments. In *Proceedings, Autonomous Agents 2000*, Barcelona, Spain, 2000.

[53] T. Sandholm and V. R. Lesser. Issues in automated negotiations and electronic commerce: extending the contract net protocol. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 328–335. AAAI Press, 1995.

[54] T. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[55] M. Schroeder and P. Noy. Multi-agent visualization based on multivariate data. In *Proceedings of Autonomous Agents*, Montreal, Canada, 2001.

[56] Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, Seattle, WA, 1994.

[57] Ira A. Smith and Philip R. Cohen. Toward a semantics for a speech act based agent communications language. In Tim Finin and James Mayfield, editors, *Proceedings of the CIKM '95 Workshop on Intelligent Information Agents*, Baltimore, Maryland, 1995.

[58] K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002.

[59] Alexander Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*. PhD thesis, The University of Texas at Austin, May 2002.

[60] Alexander Strehl and Joydeep Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 3:583–617, December 2002.

[61] Alexander Strehl, Joydeep Ghosh, and Raymond J. Mooney. Impact of similarity measures on web-page clustering. In *Proc. AAAI Workshop on AI for Web Search (AAAI 2000), Austin*, pages 58–64. AAAI/MIT Press, July 2000.

[62] Ashley Stroupe, Martin C. Martin, and Tucker Balch. Distributed sensor fusion for object position estimation by multi-robot systems. In *IEEE International Conference on Robotics and Automation, May, 2001.* IEEE, May 2001.

[63] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[64] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 1038–1044, 1996.

[65] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[66] S.Thrun and A.Schwart. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263, Hillsdale, NJ, 1993.

[67] Sorosh Talukdar, Lars Baerentzen, Andrew Gove, and Pedro de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, pages 295–321, Dec 1998.

[68] M. Tambe, W. Shen, M. Mataric, D. Pynadath, D. Goldberg, J. Modi, Z. Qiu, and B. Salemi. Teamwork in cyberspace: Using teamcore to make agents team-ready. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop*, Stanford, CA, 1999.

[69] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 378–385, Bologna, Italy, July 2002.

[70] K. Tumer and D. Wolpert. A survey of collectives. In *Collectives and the Design of Complex Systems*. Springer, 2003. (to appear).

[71] K. Tumer and D. H. Wolpert. Collective intelligence and Braess' paradox. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 104–109, Austin, TX, 2000.

[72] R. Vaughan, K. Stoy, G. Sukhatme, and M. Mataric. Blazing a trail: insect-inspired resource transportation by a robot team. In *Proc. Int. Symp. Distributed Autonomous Robot Systems*, Knoxville, TN, 2000.

[73] L. Walras. *Elements of Pure Economics*. Allen and Unwin, 1874.

[74] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[75] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[76] D. Wolpert and J. Lawson. Designing agent collectives for systems with markovian dynamics. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002.

[77] D. H. Wolpert, J. Sill, and K. Tumer. Reinforcement learning in distributed domains: Beyond team games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 819–824, Seattle, WA, 2001.

[78] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.

[79] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems - 11*, pages 952–958. MIT Press, 1999.

[80] D. H. Wolpert, K. Wheeler, and K. Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Third International Conference of Autonomous Agents*, pages 77–83, 1999.

[81] D. H. Wolpert, K. Wheeler, and K. Tumer. Collective intelligence for control of distributed dynamical systems. *Europhysics Letters*, 49(6),

March 2000.

[82] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623, Montreal, January 2001. ACM Press.

[83] R.M. Zlot, A. Stentz, M.B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *IEEE International Conference on Robotics and Automation*, 2002.

Vita

Adrian Kujaneck Agogino was born in Albuquerque New Mexico, but was mostly raised in the San Francisco Bay Area. He received a BS in Computer Engineering in 1996 from the University of California at San Diego and an MSE in Electrical and Computer Engineering in 1999 from the Univeristy of Texas at Austin. During his studies he designed graphical user interfaces, graphical simulation environments and worked as a consultant designing hardware. His interests include machine learning, computer vision, computer graphics, data mining and user interface design.

Permanent address: 3514 Dwight Way
Berkeley, CA 94704

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.