Matthew E. Taylor
Karl Tuyls (Eds.)

LNAI 5924

Adaptive and Learning Agents

Second Workshop, ALA 2009
Held as Part of the AAMAS 2009 Conference
in Budapest, Hungary, May 2009
Revised Selected Papers

Lecture Notes in Artificial Intelligence 5924

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Matthew E. Taylor Karl Tuyls (Eds.)

Adaptive and Learning Agents

Second Workshop, ALA 2009
Held as Part of the AAMAS 2009 Conference
in Budapest, Hungary, May 12, 2009
Revised Selected Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Matthew E. Taylor
The University of Southern California
Los Angeles, CA, USA
E-mail: taylorm@usc.edu

Karl Tuyls
Maastricht University
Maastricht, The Netherlands
E-mail: k.tuyls@maastrichtuniversity.nl

Library of Congress Control Number: 2010921153

CR Subject Classification (1998): I.2.6, D.2, K.2, K.8, I.6.8

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-11813-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-11813-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12990634 06/3180 5 4 3 2 1 0

Preface

This book presents selected and revised papers of the Second Workshop on Adaptive and Learning Agents 2009 (ALA-09), held at the AAMAS 2009 conference in Budapest, Hungary, May 12.

The goal of ALA is to provide an interdisciplinary forum for scientists from a variety of fields such as computer science, biology, game theory and economics. This year's edition of ALA was the second after the merger of the former workshops ALAMAS and ALAg. In 2008 this joint workshop was organized for the first time under the flag of both events. ALAMAS was a yearly returning European workshop on adaptive and learning agents and multi-agent systems (held eight times). ALAg was the international workshop on adaptive and learning agents, which was usually held at AAMAS. To increase the strength, visibility and quality of the workshop it was decided to merge both workshops under the flag of ALA and to set up a Steering Committee as an organizational backbone.

This book contains six papers presented during the workshop, which were carefully selected after an additional review round in the summer of 2009. We therefore wish to explicitly thank the members of the Program Committee for the quality and sincerity of their efforts and service. Furthermore we would like to thank all the members of the senior Steering Committee for making this workshop possible and supporting it with sound advice. We also thank the AAMAS conference for providing us a platform for holding this event. Finally we also wish to thank all authors who responded to our call-for-papers with interesting contributions.

Contributions in this book cover a variety of themes: single and multi-agent reinforcement learning, the evolution and emergence of cooperation in agent systems, sensor networks and coordination in multi-resource job scheduling. The book starts of with an overview paper on generalization and abstraction techniques in reinforcement learning. This article is meant to provide a road map for newcomers in the field and give an elementary introduction to the most commonly used techniques. Next, in "The Effects of Evolved Sociability in a Commons Dilemma," Howley et al. study the evolution of cooperation in n-player dilemma games. They introduce an evolutionary model capable of modeling sociability within the agent strategy genome and show the influence of tagging on agents interactions, leading to cooperation in a population of agents. Kaiser and Tuyls introduce a perspective on time-dependant replicator dynamics and apply it to an example game, in "Replicator Dynamics for Multi-agent Learning: An Orthogonal Approach." In "Decentralized Learning in Wireless Sensor Networks" Mihaylov et al. introduce a new reinforcement learning algorithm that is able to prolong the autonomous lifetime of a sensor network in a distributed fashion. Noda introduces an innovative principled method in "Recursive Adaptation of Stepsize Parameter for Non-Stationary Environments" to appropriately

adapt the step size parameter of reinforcement learning algorithms in response to changes in the environment. In "Multiagent Reinforcement Learning Model for the Emergence of Common Property and Transhumance in sub-Saharan Africa," Pinter *et al.* examine the emergence of common property and transhumance in sub-Saharan Africa and show why Hardin's tragedy of the commons is not applicable. They do this by simulating specific real-world scenarios with adaptive learning agents. In "Learning to Locate Trading Partners in Agent Networks" Porter et al. investigate the effects of exploration in a rewiring strategy for locating good trading partners within agent-organized networks in production and exchange economies. Finally Tumer and Lawson propose a multiagent coordination approach to multi-resource job scheduling across heterogeneous servers in "Coordinating Learning Agents for Multiple Resource Job Scheduling" and illustrate the feasibility of the approach in a number of settings varying in complexity.

November 2009

Matthew E. Taylor
Karl Tuyls

Organization

ALA 2009 Senior Steering Committee

Franziska Klügl	University of Orebro, Sweden
Daniel Kudenko	University of York, UK
Ann Nowé	Vrije Universiteit Brussels, Belgium
Lynne E. Parker	University of Tennessee, USA
Sandip Sen	University of Tulsa, USA
Peter Stone	University of Texas at Austin, USA
Kagen Tumer	Oregon State University, USA
Karl Tuyls	Eindhoven University of Technology, The Netherlands

ALA 2009 Program Committee

Eduardo Alonso	City University, UK
Bikramjit Banerjee	The University of Southern Mississippi, USA
Ana L.C. Bazzan	UFRGS, Porto Alegre, Brazil
Marek Grzes	University of York, UK
Zahia Ghuessoum	University of Paris 6, France
Franziska Klügl	University of Orebro, Sweden
Daniel Kudenko	University of York, UK
Ann Nowé	Vrije Universiteit Brussels, Belgium
Liviu Panait	Google Inc Santa Monica, USA
Lynne Parker	University of Tennessee, USA
Jeffrey Rosenschein	The Hebrew University of Jerusalem, Israel
Michael Rovatsos	Centre for Intelligent Systems and their Applications, UK
Sandip Sen	University of Tulsa, USA
Kagan Tumer	Oregon State University, USA
Katja Verbreek	KaHo Sint-Lieven, Belgium

Table of Contents

Abstraction and Generalization in Reinforcement Learning: A Summary and Framework

Marc Ponsen[1], Matthew E. Taylor[2], and Karl Tuyls[1]

[1] Universiteit Maastricht, Maastricht, The Netherlands
{m.ponsen,k.tuyls}@maastrichtuniversity.nl
[2] The University of Southern California, Los Angeles, CA
taylorm@usc.edu

Abstract. In this paper we survey the basics of reinforcement learning, generalization and abstraction. We start with an introduction to the fundamentals of reinforcement learning and motivate the necessity for generalization and abstraction. Next we summarize the most important techniques available to achieve both generalization and abstraction in reinforcement learning. We discuss basic function approximation techniques and delve into hierarchical, relational and transfer learning. All concepts and techniques are illustrated with examples.

1 Introduction

In this chapter we provide an introduction to the concepts of generalization and abstraction in reinforcement learning (RL). Abstraction is a technique to reduce the complexity of a problem by filtering out irrelevant properties while preserving all the important ones necessary to still be able solve a given problem. Generalization is a technique to apply knowledge previously acquired to unseen circumstances or extend that knowledge beyond the scope of the original problem. Humans show great capability in abstracting and generalizing knowledge in everyday life. RL needs abstraction and generalization as well to deal successfully with contemporary technological challenges, given the huge state and action spaces that characterize real world problems. Recently, abstraction and generalization have received significant attention in the machine learning research community, resulting in a variety of techniques.

We start by introducing the preliminaries of RL itself in Section 2. We will discuss Markov decision processes, policy and value iteration and model-free solution techniques. In Section 3 we define both abstraction and generalization, capturing common features of both found in different definitions in literature, and then describe different operators in a concrete domain, the video-game Wargus. Section 5 gives a concise introduction to function approximation, one of the most commonly used types of methods in RL for generalization and abstraction. Sections 6-8 go into greater detail discussing three classes of techniques used for abstraction and generalization in RL: hierarchical, relational, and transfer learning. In addition to outlining the ideas behind each of these classes of techniques, we present results to assist the reader in understanding how these ideas may be applied in practice, and provide multiple references for additional exposition. Finally, Section 9 concludes.

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 1–32, 2010.

The goals of this survey are to provide an introduction to, and framework for, discussing abstraction and generalization in RL domains. The article does not provide discussions at an advanced level but merely tries to combine the basics into one coherent structure, such that newcomers to the field easily understand the elementary concepts of abstraction and generalization in RL and have pointers available to more elaborate and detailed expositions in the literature.

2 Reinforcement Learning

This section introduces basic reinforcement learning concepts and notation.

2.1 Markov Decision Processes

Most RL research is framed as using a Markov decision processes (MDP) [29]. MDPs are sequential decision making problems for fully observable worlds. They are defined by a tuple (s_0, t, S, A, T, R). Starting in an initial state s_0 (or set of states) at each discrete time-step $t = 0, 1, 2, \ldots$ an adaptive agent observes an environment state s_t contained in a set of states $S = \{s_1, s_2, \ldots, s_n\}$, and executes an action a from a finite set of admissible actions $A = \{a_1, a_2, \ldots, a_m\}$. The agent receives an immediate reward $R : S \rightarrow \mathbb{R}$, that assigns a value or reward for being in that state, and moves to a new state s', depending on a probabilistic transition function $T : S \times A \times S \rightarrow [0, 1]$. The probability of reaching state s' after executing action a in state s is denoted as $T(s, a, s')$. For all actions a, and all states s and s', $0 \leq T(s, a, s') \leq 1$ and $\sum_{s' \in S} T(s, a, s') = 1$. An MDP respects the *Markov property*: the future dynamics, transitions and rewards fully depend on the current state: $T(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = T(s_{t+1}|s_t, a_t)$ and $R(s_{t+1}|s_t, s_{t-1}, \ldots) = R(s_{t+1})$. The transition function T and reward function R together are often referred to as the *model* of the environment. The learning task in an MDP is to find a policy $\pi : S \rightarrow A$ for selecting actions with maximal expected (discounted) reward. The quality of a policy is indicated by a *value function* V^π. The value $V^\pi(s)$ specifies the total amount of reward which an agent may expect to accumulate over the future, starting from state s and then following the policy π. Informally, the value function indicates the long-term desirability of states or state-action pairs after taking into account the states that may follow, and the rewards available in those states. In a discounted infinite horizon MDP, the expected cumulative reward (i.e., the value function) is denoted as:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)|s_0 = s\right] \tag{1}$$

A discount factor $\gamma \in [0, 1\rangle$ may be introduced to ensure that the rewards returned are bounded (finite) values. The variable γ determines the relevance of future rewards in the update. Setting γ to 0 results in a *myopic* update (i.e., only the immediate reward is optimized), whereas values closer to 1 will increase the contribution of future rewards in the update.

The value for a given policy π, expressed by Equation 1, can iteratively be computed by the *Bellman Equation* [3]. One typically starts with an arbitrarily chosen value function, and at each iteration for each state $s \in S$, the value function is updated based on the immediate reward and the current estimate of V^{π}:

$$V_{t+1}^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_n^{\pi}(s') \tag{2}$$

The process of updating state value functions based on current estimates of successor state values is referred to as *bootstrapping*. The depth of successor states considered in the update can be varied, i.e., one can perform a shallow bootstrap where one only looks at immediate successor states or a deep bootstrap where successors of successors are also considered. The value functions of successor states are used to update the value function of the current state. This is called a *backup* operation. Different algorithms use different backup strategies, e.g., sample backups (sample a single successor state) or full backups (sample all successor states).

The solution to an MDP is the *optimal policy*, i.e., the policy that receives the maximum reward. The optimal policy $\pi^*(s)$ is defined such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and all policies π. The optimal value function, often abbreviated as V^*following Bellman optimality criterion:

$$V^*(s) = R(s) + \gamma \max_{\alpha \in A} \left[\sum_{s' \in S} T(s, a, s') V^*(s') \right] \tag{3}$$

Solving Equation 3 can be done in an iterative manner, similar to the computation of the value function for a given policy such as expressed in Equation 2. The Bellman optimality criterion is turned into an update rule:

$$V_{t+1}^{\pi}(s) = R(s) + \gamma \max_{\alpha \in A} \left[\sum_{s' \in S} T(s, a, s') V_n^{\pi}(s') \right] \tag{4}$$

The optimal action can then be selected as follows:

$$\pi^*(s) = \arg\max_{a} \left[R(s) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right] \tag{5}$$

Besides learning state-values, one can also define state-action value functions, also called *action-value functions*, or *Q-functions*. Q-functions map state-action pairs to values, $Q : S \times A \rightarrow \mathbb{R}$. They reflect the long term desirability of performing action a in state s, and then performing policy π thereafter. Learning Q-functions is particularly useful when T is unknown. The Q-function is defined as follows:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \tag{6}$$

The optimal policy π^* selects the action which maximizes the optimal action value function $Q^*(s, a)$ for each state $s \in S$:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a) \tag{7}$$

Algorithm 1. Policy Iteration

REQUIRE initialize $V(s)$ and $\pi(s)$ arbitrarily;
POLICY EVALUATION;
repeat
    $\Delta = 0$;
    foreach $s \in S$ **do**
        $v = V(s)$;
        $V(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s')V(s')$;
        $\Delta = \max(\Delta, |v - V(s)|)$;
    end
until $\Delta < \sigma$;
POLICY IMPROVEMENT;
policy-stable = true;
foreach $s \in S$ **do**
    $b = \pi(s)$;
    $\pi(s) = \arg\max_a \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s')\right]$;
    if $b \neq \pi(s)$ **then** policy-stable = false
end
if *policy-stable* **then** stop **else** go to POLICY EVALUATION

2.2 Solution Techniques

When an environment's model (i.e., transition function T and reward function R) is known, the optimal policy can be computed using a dynamic programming approach, such as in *policy iteration* and *value iteration*. Policy iteration [18] consists of two steps, a policy *evaluation* and policy *improvement* step. It starts with an arbitrary policy and value functions. It then updates the value functions under the given policy (the evaluation step), and uses the new value functions to improve its policy (the improvement step). Each policy is guaranteed to be a strict improvement over the previous one. The algorithm requires an infinite number of iterations to converge, but in practice the algorithm can be stopped when value functions only change by a small amount. A complete description is given in Algorithm 1.

The drawback of policy iteration is that it requires a complete evaluation of the current policy before improvements are made. Another possibility is to make improvements after a single sweep (a single backup of a state). This particular case is called *value iteration* [3]. Value iteration (or greedy iteration) starts with an arbitrary action-value function and for each state it iterates over all actions (unlike policy iteration which only evaluates the action as indicated by the policy) and updates the action-value function. The value iteration backup is identical to the policy evaluation backup except that it requires the maximum to be taken over all actions. Similar to policy iteration, the algorithm can be stopped when the change in policy is within a certain bound. Algorithm 2 gives a complete description of value iteration.

There exist several model-based learning methods, such as Dyna-Q [38,51] and R-Max [6], but we will not go into much detail here because we are most interested in domains where the model is assumed to be both unknown and too complex to easily learn. When the model of the environment is unknown, as it usually is, we can use RL

Algorithm 2. Value Iteration

```
REQUIRE initialize V(s) arbitrarily;
repeat
    Δ = 0;
    foreach s ∈ S do
        v = V(s);
        foreach a ∈ A(s) do
            Q(s,a) = R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')
        end
        V(s) = max_a Q(s,a);
        Δ = max(Δ, |v − V(s)|);
    end
until Δ < σ ;
```

as a viable alternative. RL does not depend on a model but rather collects samples from the environment to estimate the environment's model. Therefore, the crucial distinction between model-free and model-based methods is that the first samples future states whereas the second does a full sweep of successor states. Through exploration the reinforcement learner gathers data (i.e., rewards and future states) and uses this to learn a policy. An important issue that occurs is the exploration and exploitation dilemma, i.e., when to cease exploration and to start exploiting acquired knowledge. Various exploration and exploitation strategies exist, such as ϵ-greedy and Boltzmann exploration. For a thorough overview, we refer interested readers elsewhere [52,39]. Temporal difference learning methods such as Q-learning [50] and SARSA [33] are model-free solution methods. The algorithms are described in detail in [39]. The update rule for one of the most popular algorithms, *one-step* Q-learning is:

$$Q(a,s) \rightarrow (1-\alpha)Q(a,s) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(a',s') \right] \tag{8}$$

where α is the step-size parameter, and γ the discount-rate. This algorithm is proven to converge to an optimal policy in the limit (under reasonable conditions). Unfortunately, for many complex, real-world problems, solving the MDP is impractical and complexity must be reduced in order to keep learning tractable.

In the following sections we will discuss several ways to reduce the search space, so that learning with RL is still possible in more challenging domains (i.e., domains with large or infinite state spaces).

3 Abstraction and Generalization

In order to make RL feasible in complex domains, abstraction or generalization operators are often applied to make the problem tractable. We describe these operators in the current section and then give concrete examples in the following section.

Abstraction and generalization are important concepts in artificial intelligence (AI). Some claim that the ability to abstract and generalize is the essence of human intelligence [7] and that finding good representations is the primary challenge in designing

intelligent systems. However, systems that learn and discover useful representations automatically are scarce. Instead, this problem is often tackled by the human designer.

A consistent definition of abstraction in the AI literature is not available: typically the definitions are tailored to specific subfields of AI, e.g, planning and problem solving [17], theorem proving [16], knowledge representation (e.g., spatial and temporal reasoning), machine learning, and computer vision [53]. The general principle underlying all these definitions is that an abstraction operation maps a representation of a problem onto a new representation so as to simplify reasoning while preserving useful properties. One only considers what is relevant and ignores many less important details for solving a particular task. Readers interested in a survey of state abstraction techniques in MDPs, as well an initial attempt to unify them, are referred elsewhere [24].

In problem solving and theorem proving, abstraction may be associated with a transformation of the problem representation that allows a theorem to be proved (or a problem to be solved) more easily with reduced computational complexity. This form of abstraction first abstracts a goal, proves or solves the abstracted goal, and then uses the structure of this abstracted proof to help construct the proof of the original goal. This method relies on the assumption that the structure of the abstracted proof is similar to the structure of the original goal. Another form of abstraction, as used in knowledge representation, machine learning, and computer vision, focuses more on the conceptualization of a domain, i.e., finding appropriate concepts or features of a domain. In this paper we will adopt the following definition for abstraction:

Definition 1 (Abstraction). *An abstraction operation changes the representation of an object by hiding or removing less critical details while preserving desirable properties. By definition, this implies loss of information.*

This definition is rather general and covers several different abstraction operations. In this paper we will adopt Zucker's taxonomy [53] to further categorize the different abstraction types. These abstraction operations are defined and explained with the help of a concrete example in the next section.

For generalization we employ the following definition:

Definition 2 (Generalization). *A generalization operation defines similarities between objects. This operation does not affect the object's representation. By definition, this implies no loss of information.*

For example, we may hypothesize that all rectangles are similar in some way. A strict definition of generalization states that all rectangles are a subset of its generalized hypothesis (e.g., all rectangles have 4 sides), but typically in machine learning, hypothesis are approximated and allow errors. For example, when stating that all rectangles have equal length sides, it is possible that some rectangles are outside of the hypothesis space (namely, all non square rectangles). Therefore, a weaker definition of generalization states that we have good evidence that all rectangles behave in a similar way. The generalization power measures the quality of the hypothesis on future examples.

We will next describe abstraction and generalization opportunities for RL in a concrete example, namely for learning a policy for a virtual agent in the computer game of Wargus.

4 An Illustrative Example

One example application that can benefit from reinforcement learning is computer games. Figure 1 is a screen shot of the computer game Wargus. In this figure we see an agent that is surrounded by bushes and buildings. This agent's responsibility (a peasant in the game) is a typical resource gathering task: it must travel to the goldmine (situated in the top right corner) and gather gold. We will tackle this learning task within the RL framework. The action space will contain the actions for moving in all directions. We assume that the transition function is unknown due to the complex and dynamic nature of the game environment. We define the agent's reward signal to be as follows: a small negative reward for each step and a positive (or zero) reward when completing the task. The difficult part is finding an appropriate state representation for this task. The state complexity in our world can be expressed by m^n, where n represents the number of grid cells and m the number of objects in the world. The state complexity is thus exponential in the dimension of the world and polynomial in the number of objects. A naive state representation (see Figure 2) for our example application would be to consider the smallest particle of this world (in this case a pixel in this 2-dimensional computer game world with dimension 500×500) to be a single grid cell, and then assuming that each grid cell can be part of any of five different objects (which is already a simplification). For example, in Figure 2 the first row indicates that the first pixel is part of a forest object. When using this representation, learning a policy that directs the agent to the goldmine would be infeasible, due to the large state space, requiring the value function to contain 25000^5 distinct values. For any complex computer game, when modeling the world described as above, none of the standard RL approaches will converge to a decent policy in a reasonable amount of time. Rather than devising new update rules for RL,

Fig. 1. A complex learning task

$$w = \begin{bmatrix} agent & structure & forest & rock & sand \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 2. A naive state representation, where rows represent the observations of states (in this case, a pixel) and the columns represent the features used to describe the world

a more promising approach is to find more compact task representations (i.e., make the problem space simpler) and generalize over similar states. In other words, we need to apply appropriate abstractions and generalizations. We will next describe five different abstraction operations as defined by Zucker [53] that can scale down the problem complexity. We will apply these five techniques consecutively to our challenging problem to reduce complexity.

4.1 Domain Reduction

Domain Reduction is an abstraction operation that reduces part of the domain (i.e., content or instances) by grouping content together. Content refers to the observations of states (i.e., the row vectors in our world matrix). Before we evaluated each single pixel, so that our world matrix contained 25000 state observations (one for each grid cell in our world). The matrix in Figure 2 is a reformulation of the image in Figure 1: it applies a different notation for the same object without losing any information. We can reduce the content, i.e., reduce the number of state observations by making sets of grid cells indistinguishable. In our example we can choose to group neighboring pixels together to form a larger prototype grid cell. As a result, the world is divided in larger grid cells, as illustrated in Figure 3. An observation in our world matrix now covers several pixels, and therefore attribute values are real-valued percentages (averages over the covered pixels) rather than booleans (see Figure 4). The number of pixels grouped together to form a grid cell can be increased, but a coarser view of the world necessitates information loss. The tradeoff between information loss and the quality of the learned policy can be tuned, depending on task requirements.

4.2 Domain Hiding

Domain hiding is an abstraction operation that hides part of the domain, focusing on relevant content or objects in the domain. This is one of the most common form of abstraction. As mentioned before, content refers to the state observations (row vectors

Fig. 3. Domain reduction

$$w = \begin{bmatrix} agent & structure & forest & rock & sand \\ 0 & 0 & .7 & .3 & 0 \\ 0 & .3 & .3 & 0 & .4 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & .9 & .1 \end{bmatrix}$$

Fig. 4. State representation of world after domain reduction

in our matrix). Rather than reducing the number of state observations (by grouping them), domain hiding simply ignores less relevant state observations. For example, in our task we want the agent to learn a policy that directs it to the gold mine. Therefore, we are not necessarily interested in some parts of the world, and we hide these state observations. We take the world that was the result of domain reduction as our input and apply domain hiding. The result is shown in Figure 5. In our matrix representation, a domain hiding operation can be performed by deleting observations, whereas domain reduction averages observations together.

Fig. 5. Domain hiding

$$w = \begin{bmatrix} agent & structure & forest & rock & sand \\ 0 & 0 & .7 & .3 & 0 \\ 0 & .3 & .3 & 0 & .4 \\ \dots & \dots & \dots & \dots & \dots \\ \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} \\ \cancel{0} & \cancel{0} & \cancel{0} & .9 & .1 \end{bmatrix}$$

Fig. 6. State representation of world after domain hiding

4.3 Co-domain Hiding

Co-domain hiding is an abstraction operation that hides part of the co-domain (i.e., description) of an object by selectively paying attention to subsets of useful features in a given task. With the co-domain, we refer to the features of our world. In the state matrix, this is represented by the column vectors. Co-domain hiding ignores columns that are not relevant for the task. For example, in Figure 7, the sand feature is removed from the description since it is believed this feature does not contribute to an improvement for the agent's policy. Notice that the **sand** in Figure 7 and the sand column in Figure 8's matrix have been removed.

4.4 Co-domain Reduction

Co-domain reduction is an abstraction operation that reduces part of the co-domain (i.e., description) by making sets of attribute values indistinguishable. This implies reducing the range of values an attribute may take. In Figure 8 we see attribute values ranging from 0 to 1. We can apply abstractions by reducing the range of attribute values.

This can be achieved by applying some threshold function (injective mapping). For example, if a certain cell is covered with an object by more than 50 percent, in our new world representation this cell is now covered completely with this object, whereas objects that cover less than 50 percent are abstracted away from the matrix representation

Fig. 7. Co-domain hiding

$$w = \begin{bmatrix} agent & structure & forest & rock & sand \\ 0 & 0 & .7 & .3 & 0 \\ 0 & .3 & .3 & 0 & .4 \\ ... & ... & ... & ... & ... \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & .9 & .1 \end{bmatrix}$$

Fig. 8. State representation of world after co-domain hiding

Fig. 9. Co-domain reduction

$$w = \begin{bmatrix} agent & structure & forest & rock & sand \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 10. State representation of world after co-domain reduction

(see Figure 10). Effectively, we transform real numbers (i.e., percentages of objects in a grid cell) to boolean values, just as we saw in Figure 2, but now the boolean values do not correspond to pixels, but to composite grid cells.

4.5 Domain Aggregation

Domain Aggregation is an abstraction operation that aggregates (combines) parts of the domain (i.e., content). Content (or objects) are grouped together and form a new object with its own unique properties and parameters. In our example, we can choose to group objects together that obstruct the agent such as forests, structures or rocks. We group these objects together to form a complete new object, namely an obstacle (see Figure 11).

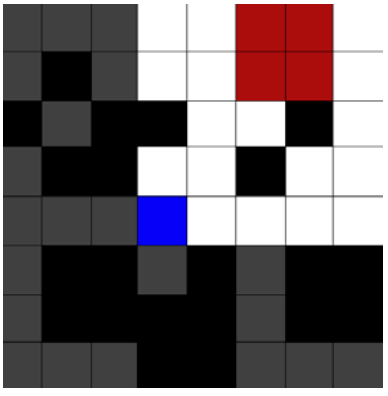


Fig. 11. Domain Aggregation

$$w = \begin{bmatrix} agent & obstacle \\ 0 & 1 \\ 1 & 0 \\ \dots & \dots \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Fig. 12. State representation of world after domain aggregation

4.6 Generalization

A generalization operation is different from an abstraction operation in that it does not change an object's representation and, therefore, does not lose any information. Instead it claims generalities between objects, leaving the original objects untouched. In our example we can make a generalized hypothesis that forests and rocks are equivalent in that they obstruct the agent from moving there. This is illustrated in Figure 13: our generalized hypothesis claims that the light-grey parts of the world are equivalent (i.e., trees and rocks combined), and similarly for the dark-grey parts of the world (i.e., grass and sand combined). The effect is (roughly) similar to the effect of an aggregation operation in Figure 11. However, it is possible that at some point our generalization that forests and rock are equivalent proves to be faulty. Say, the agent has learned to chop trees down so it can move through forest locations. In the case of generalization,

Fig. 13. Generalization

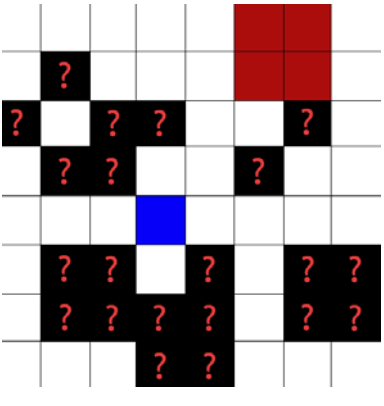


Fig. 14. Difference between abstraction and generalization: with an abstraction operation, information is lost

we can simply remove or reformulate our hypothesis and return to the original world, whereas with abstraction the original information is lost and we can not turn back to the original world. In our example (see Figure 14), it is unclear whether an obstacle used to be part of a forest or rock. We threw away that information during our abstraction process. Therefore, we claim that generalization is more flexible and less conclusive than abstraction.

5 Function Approximation

The previous section introduced many different generalization and abstraction operators. In this section, we discuss a commonly used approach, where information gathered by an agent is used to tune a mathematical function that represents the agent's gathered knowledge.

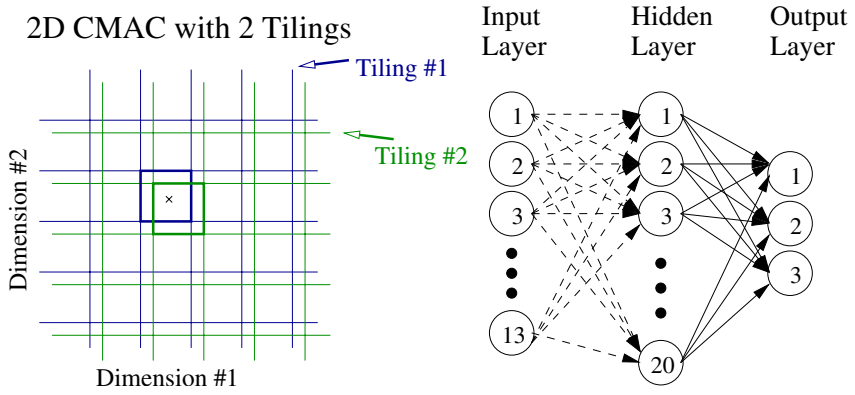
In tasks with small and discrete state spaces, the functions V, Q, and π can be represented in a table, such as discussed in the previous section. However, as the state space grows using a table becomes impractical (or impossible if the state space is continuous). In such situations, some sort of *function approximator* is necessary, which allow the agent to use data to estimate previously unobserved (s, a) pairs.

How to best choose which function approximator to use, or how to set its parameters, is currently an open question. Although some work in RL [11, 24, 25] has taken a more systematic approaches to *state abstractions* (also called *structural abstractions*), the majority of current research relies on humans to help bias a learning agent by carefully selecting a function approximator with parameters appropriate for a given task. In the remainder of this section we discuss three popular function approximators: Cerebellar Model Arithmetic Computers (CMACs), neural networks, and instance-based approximation.

The first two methods, CMACs and neural networks, may be considered both approximation and generalization operators. Rather than saving the data gathered in the world, the agent tunes its function approximator and discards data, losing some information (abstraction), but it is then able to calculate the value of the function for values that have not been experienced (generalization). Many methods for instance-based approximation also discard data, but some do not; while all instance-based function approximators are generalizers, not all are abstractors.

Cerebellar Model Arithmetic Computers. CMACs [1] take arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them (see Figure 15(a)). This allows discretization of continuous state space into tiles while maintaining the capability to generalize via multiple overlapping tilings. Increasing the tile widths allows better generalization; increasing the number of tilings allows more accurate representations of smaller details. The number of tiles and the width of the tilings are generally handcoded: this sets the center, c_i, of each tile and dictates which state values will activate which tiles. The function approximation is trained by changing how much each tile contributes to the output of the function approximator. Thus, the output from the CMAC is the computed sum:

$$f(x) = \sum_i w_i f_i(x) \tag{9}$$



(a) CMAC (b) Neural Network

Fig. 15. CMAC's value, shown in (a), is computed by summing the weights, w_i, from multiple activated tiles (outlined above with thicker lines). State variables are used to determine which tile is activated in each of the different tilings. The diagram in (b) shows an artificial feedforward 13-20-3 network, suggesting how Q-values for three actions can be calculated from 13 state variables.

but only tiles which are activated by the current state features contribute to the sum:

$$f_i(x) = \begin{cases} 1, \text{ if tile } i \text{ is activated} \\ 0, \text{ otherwise} \end{cases}$$

Weights in a CMAC are typically initialized to zero and are changed over time via learning.

Artificial Neural Networks. The neural network function approximator similarly allows a learner to approximate the action-value function, given a set of continuous, real valued, state variables. Although neural networks have been shown to be difficult to train in certain situations on relatively simple RL problems [5,30], they have had notable successes on some RL tasks [9,46]. Each input to the neural network is set to the value of a state variable and each output corresponds to an action. Activations of the output nodes correspond to Q-values (see Figure 15(b) for a diagram).

When used to approximate an action-value function, neural networks often use non-recurrent feedforward networks. Each node in the input layer is given the value of a different state variable and each output node corresponds is the the calculated Q-value for a different action. The number of inputs and outputs are thus determined by the task's specification, but the number of hidden nodes is specified by the agent's designer. Note that by accepting multiple inputs the neural network can determine its output by considering multiple state variables in conjunction (as opposed to a CMAC consisting of a separate 1-dimensional tiling for each state variable). Nodes often have either sigmoid or linear transfer functions. Weights for connections in the network are typically initialized to random values near zero. The networks are often trained using backpropagation, where the error signal to modify weights is generated by the learning algorithm, as with the other function approximators.

Instance-based approximation. CMACs and neural networks aim to represent a complex function with a relatively small set of parameters that can be changed over time. In contrast, instance-based approximation stores *instances* experienced by the agent (i.e., $\langle s, a, r, s' \rangle$) to predict the underlying structure of the environment. Specifically, this approximation method can be used by model-learning methods (c.f., [19,20]), which learn to approximate T and R by observing the agent's experience when interacting with an environment.

Consider the case where an agent is acting in a discrete environment with a small state space. The agent could record every instance that it experienced in a table. If the transition function were deterministic, as soon as the agent observed every possible (s, a) pair, it could calculate the optimal policy. If the transition function were instead stochastic, the agent would need to take multiple samples for every (s, a) pair. Given a sufficient number of samples, as determined by the variance in the resulting r and s', the agent could again directly calculate the optimal policy via dynamic programming.

When used to approximate T and R for tasks with continuous state spaces, using instances for function approximation becomes significantly more difficult. In a stochastic task the agent is unlikely to ever visit the same state twice, with the possible exception of a start state, and thus approximation is critical. Furthermore, since one can never gather "enough" samples for every (s, a) pair, such methods generally need to determine which instances are necessary to store so that the memory requirements are not unbounded. Creating efficient instance-based function approximators, and their associated learning algorithms, are topics of ongoing research in RL.

Now that the basic concepts of abstraction, and generalization have been introduced in the context of RL, the next section describes our own contribution in the field of abstraction and generalization in RL through action abstraction via hierarchical RL techniques. Later sections will then discuss work on generalization using relational reinforcement learning and transfer learning.

6 Hierarchical Reinforcement Learning

There exist many extensions to standard RL that make use of the abstraction and generalization operators mentioned in Section 4. One such method is hierarchical reinforcement learning (HRL), which essentially aggregates actions. HRL is an intuitive and promising approach to scale up RL to more complex problems. In HRL, a complex task is decomposed into a set of simpler subtasks that can be solved independently. Each subtask in the hierarchy is modeled as a single MDP and allows appropriate state, action and reward abstractions to augment learning compared to a flat representation of the problem. Additionally, learning in a hierarchical setting can facilitate generalization: knowledge learned in a subtask can be transferred to other subtasks. HRL relies on the theory of Semi-Markov decision processes (SMDPs) [40]. SMDPs differ from MDPs in that actions in SMDPs can last multiple time steps. Therefore, in SMDPs actions can either be primitive actions (taking exactly 1 time-step) or temporally extended actions. While the idea of applying HRL in complex domains such as computer games is appealing, with the exception of [2], there are few studies that examining this issue.

We adopted a HRL method similar to Hierarchical Semi-Markov Q-learning (HSMQ) described in [12]. HSMQ learns policies simultaneously for all non-primitive subtasks in the hierarchy, i.e., $Q(p, s, a)$ values are learned to denote the expected total reward of performing task p starting in state s, executing action a, and then following the optimal policy thereafter. Subtasks in HSMQ include termination predicates. These partition the state space S into a set of active states and terminal states. Subtasks can only be invoked in states in which they are active, and subtasks terminate when the state transitions from an active to a terminal state. We added to the HSMQ algorithm described in [12] a pseudo-reward function [12] for each subtask. The pseudo-rewards tell how desirable each of the terminal states are for this subtask. Algorithm 3 outlines our HSMQ-inspired algorithm. Q-values for primitive subtasks are updated with the one-step Q-learning update rule, while the Q-values for non-primitive subtasks are updated based on the reward $R(s, a)$, collected during execution of the subtask and a pseudo reward $\hat{R}$.

Algorithm 3. Modified version of the HSMQ algorithm: The update rule for non-primitive subtasks (line 13) differs from the original implementation

1 **Function** HSMQ(state s,subtask p) returns float;
2 Let *Totalreward* $= 0$;
3 **while** *(p is not terminated)* **do**
4 Choose action $a = \Pi(s)$;
5 **if** *a is primitive* **then**
6 Execute a, observe one-step reward $R(s, a)$ and result state s';
7 **else if** *a is non-primitive subtask* **then**
8 $R(s, a) :=$ HSMQ(s, a) , which invokes subtask a and returns the total reward received while a executed
9 *Totalreward* $=$ *Totalreward* $+ R(s, a)$;
10 **if** *a is primitive* **then**
11 $$Q(p, a, s) \rightarrow (1 - \alpha)Q(p, a, s) + \alpha \left[R(s, a) + \gamma \max_{a'} Q(p, a', s') \right];$$
12 **else if** *a is non-primitive subtask* **then**
13 $$Q(p, a, s) \rightarrow (1 - \alpha)Q(p, a, s) + \alpha \left[R(s, a) + \hat{R} \right];$$
14 **end**
15 **return** *Totalreward*;

6.1 Reactive Navigation Task

We applied our HSMQ algorithm to the game of Wargus. Inspired by the resource gathering task, we created a world wherein a peasant has to learn to navigate to some location on the map while avoiding enemy contact (in the game of Wargus, peasants have no means for defending themselves against enemy soldiers). More precisely, our simplified game consists of a fully observable world that is 32×32 grid cells and includes two units: a peasant (the adaptive agent) and an enemy soldier. The adaptive agent has to move to a certain goal location. Once the agent reaches its goal, a new goal is set at random. The enemy soldier randomly patrols the map and will shoot at the peasant if it is in firing range. The scenario continues for a fixed time period or until the peasant is destroyed by the enemy soldier.

Relevant properties for our task are the locations of the peasant, soldier, and goal. All three objects can be positioned in any of the 1024 locations. A propositional format of the state space describes each state as a feature vector with attributes for each possible property of the environment, which amounts to 2^{30} different states. As such, a tabular representation of the value functions is too large to be feasible. Additionally, such encoding prevents any opportunity for generalization for the learning algorithm, e.g., when a policy is learned to move to a specific grid cell, the policy can not be reused to move to another. A deictic state representation identifies objects relative to the agent. This reduces state space complexity and facilitates generalization. This is a first step towards a fully relational representation, such as the one covered in Section 7. We will discuss the state features used for this task in Section 6.2.

The proposed task is complex for several reasons. First, the state space without any abstractions is enormous. Second, the game state is also modified by an enemy unit. (The enemy executes a random move on each timestep unless the peasant is in sight, in which case it moves toward the peasant.) Furthermore, each new task instance is generated randomly (i.e., random goal and enemy patrol behavior), so that the peasant has to learn a policy that generalizes over unseen task instances.

6.2 Solving the Reactive Navigation Task

We compare two different ways to solve the reactive navigation task, namely using flat RL and HRL. For a **flat representation** of our task, the deictic state representation can be defined as the Cartesian-product of the following four features: `Distance(enemy,s)`, `Distance(goal,s)`, `DirectionTo(enemy,s)`, and `DirectionTo (goal,s)`. The function `Distance` returns a number between 1 and 8 or a string indicating that the object is more than 8 steps away in state s, while `DirectionTo` returns the relative direction to a given object in state s. Using 8 possible values for the `DirectionTo` function, namely the eight compass directions, and 9 possible values for the `Distance` function, the total state space is drastically reduced from 2^{30} to only 5184 states. The size of the action space is 8, containing actions for moving in each of the eight compass directions. The scalar reward signal $R(s, a)$ in the flat representation should reflect the relative success of achieving the two concurrent sub-goals (i.e., moving towards the goal while avoiding the enemy). The environment returns a $+10$ reward whenever the agent is located on a goal location. In contrast, a reward of -10 is returned when the agent is being fired at by the enemy unit, which occurs when the agent is in firing range of the enemy (i.e., within 5 steps). Each primitive action always yields a reward of -1. An immediate concern is that both sub-goals are often in competition. We can certainly consider situations where different actions are optimal for the two sub-goals, although the agent can only take one action at a time. An apparent solution to handle these two concurrent sub-goals is applying a hierarchical representation, which we discuss next.

In the **hierarchical representation**, the original task is decomposed into two simpler subtasks that solve a single sub-goal independently (see Figure 17). The *to goal* subtask is responsible for navigation to goal locations. Its state space includes the `Distance(goal,s)` and `DirectionTo(goal,s)` features. The *from enemy* subtask is responsible for evading the enemy unit. Its state space includes

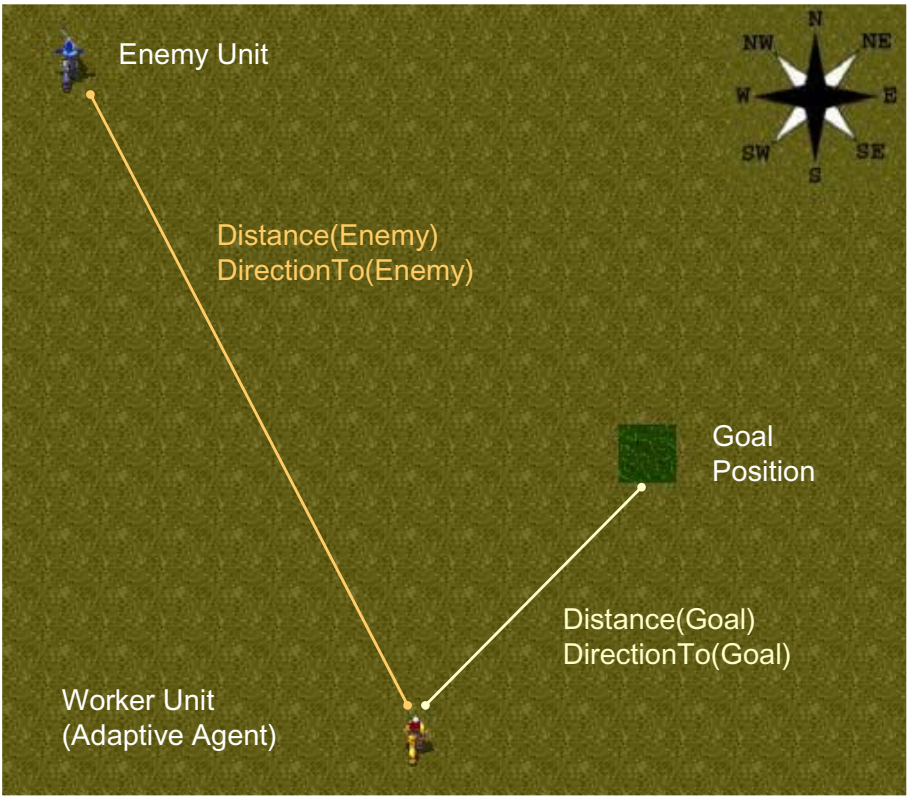


Fig. 16. This figure shows a screenshot of the reactive navigation task in the Wargus game. In this example, the peasant is situated at the bottom. Its task is to move to a goal position (the dark spot right to the center) and avoid the enemy soldier (situated in the upper left corner) that is randomly patrolling the map.

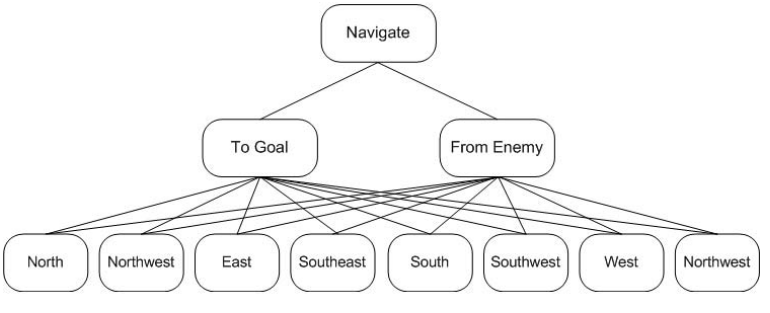


Fig. 17. Hierarchical decomposition of the reactive navigation task

the `Distance(enemy,s)` and `DirectionTo(enemy,s)` features. The action spaces for both subtasks include the primitive actions for moving in all compass directions. The two subtasks are hierarchically combined in a higher-level *navigate* task. The state space of this task is represented by the `InRange(goal,s)` and `InRange(enemy,s)` features, and its action space consists of the two subtasks that can be invoked as if they were primitive actions. `InRange` is a function that returns *true* if the distance to an object is 8 or less in state s, and *false* otherwise. Because these new features can be defined in terms of existing features, we are not introducing any additional domain knowledge compared to the flat representation. The *to goal* and *from enemy* subtasks terminate at each state change on the root level, e.g., when the enemy (or goal) transitions from in range to out of range and vice versa. We choose to set the pseudo-rewards for both subtasks to $+100$ whenever the agent completes a subtask and 0 otherwise. The *navigate* task never terminates, but the primitive subtasks always terminate after execution. The state spaces for the two subtasks are of size 72, and four for *navigate*. Therefore, the state space complexity in the hierarchical representation is approximately 35 times less than with the flat representation. Additionally, in the hierarchical setting we are able to split the reward signal, one for each subtask, so they do not interfere. The *to goal* subtask rewards solely moving to the goal (i.e., only process the $+10$ reward when reaching a goal location). Similarly, the *from enemy* subtask only rewards evading the enemy. Based on these two reward signals and the pseudo-rewards, the root *navigate* task is responsible for choosing the most appropriate subtask. For example, suppose that the peasant at a certain time decided to move to the goal and it took the agent 7 steps to reach it. The reward collected while the *to goal* subtask was active is -7 (reward of -1 for all primitive actions) and $+10$ (for reaching the goal location) resulting in a $+3$ total reward. Additionally, a pseudo-reward of $+100$ is received because *to goal* successfully terminated, resulting in a total reward of $+103$ that is propagated to the *navigate* subtask, that is used to update its Q-values. The Q-values for the *to goal* subtask are updated based on the immediate reward and estimated value of the successor state (see equation 8).

6.3 Experimental Results

We evaluated the performance of flat RL and HRL in the reactive navigation task. The step-size and discount-rate parameters were set to 0.2 and 0.7, respectively. These values were determined during initial experiments. We chose to use more exploration for the *to goal* and *from enemy* subtasks compared to *navigate*, since more Q-values must be learned. Therefore, we used Boltzmann action selection with a relatively high (but decaying) temperature for the *to goal* and *from enemy* subtasks and ϵ-greedy action selection at the top level, with ϵ set to 0.1 [39].

A "trial" (when Q-values are adapted) lasted for 30 episodes. An episode terminated when the adaptive agent was destroyed or until a fixed time limit was reached. During training, random instances of the task were generated, i.e., random initial starting locations for the units, random goals and random enemy patrol behavior. After each trial, we empirically tested the current policy on a test set consisting of five fixed task instances that were used throughout the entire experiment. These included fixed starting locations for all objects, fixed goals and fixed enemy patrol behavior. We measured the

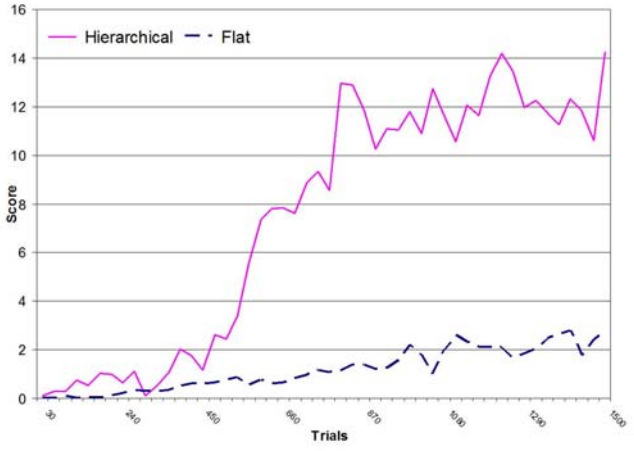


Fig. 18. This figure shows the average performance of Q-learning over 5 experiments in the reactive navigation task for both flat and HRL. The x-axis denotes the number of training trials and the y-axis denotes the average number of goals achieved by the agent for the tasks in the test set.

performance of the policy by counting the number of goals achieved by the adaptive agent (i.e., the number of times the agent successfully reached the goal location before it was destroyed or time ran out) by evaluating the greedy policy. We ended the experiment after 1500 training episodes (50 trials). The experiment was repeated five times and the averaged results are shown in Figure 18.

From this figure we can conclude that while both methods improve with experience, learning with the HRL representation outperforms learning with a flat representation. By using (more human-provided) abstractions, HRL represented the policy more compactly than the flat representation, resulting in faster learning. Furthermore, HRL is more suited to handling concurrent and competing subtasks due to the split reward signal. We expect that even after considerable learning with flat RL, HRL will still achieve a higher overall performance. This experiment shows that when goals have clearly conflicting rewards and the overall task can be logically divided into subtasks, HRL could be successfully applied.

7 Relational Reinforcement Learning

Relational reinforcement learning [14] (RRL) combines the RL setting with relational learning or inductive logic programming [26] (ILP) in order to represent states, actions, and policies using the structures and relations that identify them. These structural representations allow generalization over specific goals, states, and actions. Because relational reinforcement learning algorithms try to solve problems at an abstract level, the solutions will often carry to different instantiations of that abstract problem. For example, resulting policies learned by an RRL system often generalize over domains with varying number of existing objects.

A typical example is the blocks world. A number of blocks with different properties (size, color, etc.) are placed on each other or on the floor. It is assumed that an infinite number of blocks can be put on the floor and that all blocks are neatly stacked onto each other, e.g., a block can only be on one other block. The possible actions consist of moving one clear block (e.g., a block with no other block on top of it) onto another clear block, or onto the floor. It is impossible to represent such world states with a propositional representation without an exponential increase of the number of states. Consider as an example the right-most state in Figure 19. In First-Order Logic (FOL), this state can be represented, presuming this state is called s, by the conjunction

$$\{on(s,c,floor) \wedge clear(s,c) \wedge on(s,d,floor) \wedge on(s,b,d) \wedge on(s,a,b) \wedge clear(s,a) \wedge on(s,e,floor) \wedge clear(s,e)\}.$$

s is reached by executing the move action (indicated by the arrow), noted as *move(r,s,a,b)*, in the previous state (on the left in Figure 19).

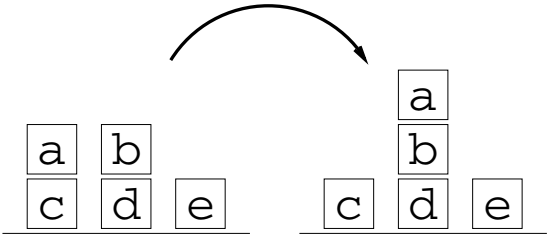


Fig. 19. The blocks world

One of the most important benefits of the relational learning approach is that one can generalize over states, actions, objects, but one is not forced to do so (one can abstract away selectively only these things that are less important). For instance, suppose that all blocks have a size property. One could then say that "there exists a small block which is on a large block" $(\exists B1, B2 : block(B1, small, \_), block(B2, large, \_), on(B1, B2))$. Objects $B1$ and $B2$ are free variables, and can be instantiated by any block in the environment. RRL can generalize over blocks and learn policies for a variable number of objects without necessarily suffering from the "curse of dimensionality" (where the size of the value function increases exponentially with the dimension of the state space).

Although it is a relatively new representation, several approaches to RRL have been proposed during the last few years. One of the first methods developed within RRL was relational Q-learning [14], described in Algorithm 4. Relational Q-learning behaves similarly to standard Q-learning, but is adapted to the RRL representation. In relational reinforcement learning, the representation contains structural or relational information about the environment. Relational Q-learning employs a relational regression algorithm to generalize over the policy space. Learning examples, stored as a tuple $(a, s, Q(s, a))$, are processed by an incremental relational regression algorithm to produce a relational value-function or policy as a result. So far, a number of different relational regression learners have been developed.[1]

[1] A thorough discussion and comparison can be found in [13].

Algorithm 4. The Relational Reinforcement Learning Algorithm

REQUIRE initialize $Q(s, a)$ and s_0 arbitrarily;
$e \leftarrow 0$;
repeat {for each episode}
  Examples $\leftarrow \emptyset$;
  $i \leftarrow 0$;
  repeat {for each step $\in$ episode}
    take a for s using policy $\pi(s)$ and observe r and s';
    $Q(a, s) \rightarrow (1 - \alpha)Q(a, s) + \alpha \left[r + \gamma \max_{a'} Q(a', s') \right]$;
    Examples $\leftarrow$ *Examples*$\cup\{a, s, Q(s, a)\}$;
    $i \leftarrow i + 1$;
  until s_i is terminal ;
  Update $\hat{Q}_e$ using *Examples* and a relational regression algorithm to produce $\hat{Q}_{e+1}$;
  $e \leftarrow e + 1$;
until done ;

In [28], we demonstrated how relational reinforcement learning and multi-agent systems techniques could be combined to plan well in tasks that are complex, multi-state, and dynamic. We used a relational representation of the state space in multi-agent reinforcement learning, as this has many benefits over the propositional one. For instance, it handled large state spaces, used a rich relational language, modeled other agents without a computational explosion (generalizing over agents' policies), and generalized over newly derived knowledge. We investigated the positive effects of relational reinforcement learning applied to the problem of agent communication in multi-agent systems. More precisely, we investigated the learning performance of RRL given some communication constraints. Our results confirm that RRL can be used to adequately deal with large state spaces by generalizing over states, actions and even agent policies.

8 Transfer Learning

This section of the chapter chapter focuses on transfer learning (TL) and its relationship to generalization and abstraction. The interested reader is referred elsewhere [44] for a more complete treatment of transfer in RL.

8.1 Transfer Learning Background

All transfer learning algorithms for reinforcement learning agents use one or more *source tasks* to better learn in a *target task*, relative to learning without the benefit of the source task(s). Transfer techniques assume varying degrees of autonomy and make many different assumptions. For instance, one way TL algorithms differ is in how they allow source and target tasks to differ. Consider the pairs of MDPs represented in Figure 20. The source and target tasks could differ in any portion of the MDP: the transition function, T, the reward function, R, what states exist, what actions the agent can perform, and/or how the agent represents the world (the state is represented here in

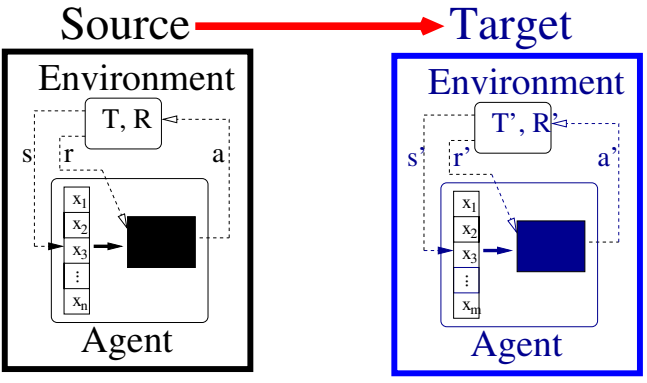


Fig. 20. Simple transfer schematic

terms of *state variables* $\langle x_1, x_2, \ldots, x_n \rangle$ in the source task MDP and $\langle x_1, x_2, \ldots, x_m \rangle$ in the target task MDP).

The use of transfer in humans has been studied for many years in the psychological literature [34,47]. More related is sequential transfer between machine learning tasks (c.f., [48]), which allows the learning of higher performing classifiers with less data. For instance, one could imagine using a large training corpus from a newspaper to help learn a classifier such that when the classifier is presented with training examples from a magazine, it can learn with fewer examples than if the newspaper data had not been used. Another common approach is that of learning to perform multiple tasks simultaneously (c.f., [8]). The motivation in this case is that a classifier capable of performing multiple tasks in a single domain will be forced to capture more structure of the domain, performing better than if any one classifier was trained and tested in isolation.

8.2 Transfer as Generalization

Generalization may be thought of as the heart of machine learning: given a set of data, how should one perform on novel data? Such questions are prevalent in RL as well. For instance, in a continuous action space, an agent is unlikely to visit a single state more than once and it must therefore constantly generalize its previous data to predict how to act.

Transfer learning may be thought of as a different type of generalization, where the agent must generalize knowledge *across tasks*. This section examines two strategies common to transfer methods. The first strategy is to learn some low-level information in the source task and then use this information to better bias learning in the target task. The second strategy is to learn something that is true for the general domain, regardless of the particular task in question.

Example Domain: Keepaway. One popular domain for demonstrating TL in RL tasks is that of *Keepaway* [37], a sub-task of the full 11 vs. 11 simulated soccer, which uses the RoboCup Soccer Server [27] to simulate sensor and actuator noise of physical robots.

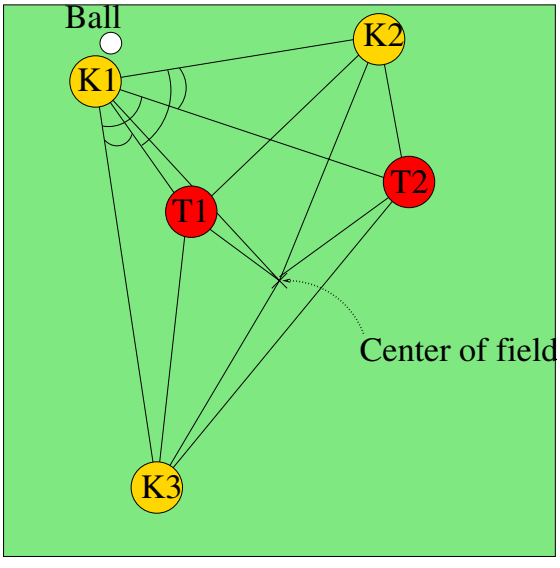


Fig. 21. 3 vs. 2 Keepaway uses multiple distances and angles to represent state, enumerated in Table 1

Typically, n teammates (the *keepers*) attempt to keep control of the ball within a small field area, while $n-1$ teammates (the *takers*) attempt to capture the ball or kick it out of bounds. The keepers typically learn while the takers follow a fixed policy. Only the keeper with the ball may select actions intelligently: all keepers without the ball always executes a `getopen` action, where they attempt to move to an area of the field to receive a pass.

Keepers learn to extend the average length of an episode over time by selecting between executing the `hold` ball action (attempting to maintain possession), or they may `pass` to a teammate. In 3 vs. 2 Keepaway, there are three keepers and two takers, and thus the keeper with the ball has 3 macro[2] actions (hold, pass to closest teammate, and pass to second closest teammate). The keeper with the ball represents its state with 13 (rotational invariant) state variables, shown in Figure 21. This is also a deictic representation, similar to the one discussed in Section 6.1, because players are labeled relative to their distance to the ball, rather than labeled by jersey number (or another fixed scheme).

The 4 vs. 3 Keepaway task, which adds an additional keeper and taker, is more difficult to learn. First, there are more actions (the keeper with the ball selects from 4 actions) and keepers have a more complex state representation (due to the extra players, the world is described by 19 state variables). Second, because there are more players on the field, passes must be more frequent. Each pass has a chance of being missed by the intended receiver, and thus each pass brings additional risk that the ball will be lost, ending the episode.

[2] Note that because the actions may last more than one timestep, Keepaway is technically an SMDP, rather than an MDP, but such distinction is not critical for the purposes of this chapter.

Given these two tasks, there are (at least) two possible goals for transfer. First, one may assume that a set of keepers have trained on the source task. Considering this source task training a sunk cost, the goal is to learn better/faster on the target task by using the source task information, compared to learning the target task while ignoring knowledge from the source task. A second, more difficult, goal is to explicitly account for source task training time. In other words, the second goal of transfer is to learn the source task, transfer some knowledge, and then target task better/faster than if the agents had directly trained only on the target task.

Transferring Low-Level Information: Ignoring novel structure. When considering transfer from 3 vs. 2 to 4 vs. 3, one approach would be to learn a Q-value function in the source task and then copy it over into the target task, ignoring the novel state variables and actions. This is similar to the idea of domain hiding, discussed earlier in Section 4.2. For instance, the `hold` action in 4 vs. 3 can be considered the same as the `hold` action in 3 vs. 2. Likewise, the 4 vs. 3 actions `pass` to closest teammate and `pass` to second closest teammate may be considered the same as the 3 vs. 2 actions `pass` to closest teammate and `pass` to second closest teammate. The "novel" 4 vs. 3 action, `pass` to third closest teammate, is ignored. Table 1 shows the state variables from the 3 vs. 2 and 4 vs. 3 tasks, where the novel state variables in the 4 vs. 3 task are in bold.

This is precisely the approach used in *Q-value Reuse* [45]. Results (reproduced in Table 2) show that the Q-values saved after training in the 3 vs. 2 task can be successfully used directly in the 4 vs. 3 task by ignoring the novel state variables and actions. Specifically, the source task action-value function is used as an initial bias in the target task, which is then refined over time with SARSA learning (e.g., Q-values for the novel 4 vs. 3 action are learned, and existing Q-values are refined). Column 2 of the table shows that the source task knowledge can be successfully reused, and column 3 shows that the total training can be successfully reduced via transfer.

Lazaric et al. [23] also focuses on transferring very low-level information by using source task *instances* in a target task. After learning one or more source tasks, some experience is gathered in the target task, which may have a different state space or transition function, but the state variables and actions must remain unchanged. Saved instances (that is, observed $\langle s, a, r, s' \rangle$ tuples) are compared to recorded instances in the target task. Source task instances that are very similar, as judged by their distance and alignment with target task data, are transferred. A batch learning algorithm (*Fitted Q-iteration* [15], which uses instance-based function approximation, then uses both source instances and target instances to achieve a higher total reward (relative to learning without transfer).

Region transfer, introduced in the same chapter [23], calculates the similarity between the target task and different source tasks per sample, rather than per task. Thus, if source tasks have different regions of the state space which are more similar to the target, only those most similar regions can be transferred. In this way, different regions of the target task may reuse data from different source tasks, and regions of the target task that are completely novel will use no source task data. Although this work did not use Keepaway, one possible example would be to have source tasks with different coefficients of friction: in the target task, grass conditions in different sections of the

Table 1. This table lists the 13 state variables for 3 vs. 2 Keepaway and the 19 state variables for 4 vs. 3 Keepaway. The distance between a and b is denoted as $dist(a, b)$; the angle made by a, b, and c, where b is the vertex, is denoted by $ang(a, b, c)$; and values not present in 3 vs. 2 are in bold. Relevant points are the center of the field C, keepers K_1-K_4, and takers T_1-T_3, where players are ordered by increasing distance from the ball.

Description 3 vs. 2 and 4 vs. 3 State Variables

3 vs. 2 state variable	4 vs. 3 state variable
$dist(K_1, C)$	$dist(K_1, C)$
$dist(K_1, K_2)$ $dist(K_1, K_3)$	$dist(K_1, K_2)$ $dist(K_1, K_3)$ $\mathbf{dist(K_1, K_4)}$
$dist(K_1, T_1)$ $dist(K_1, T_2)$	$dist(K_1, T_1)$ $dist(K_1, T_2)$ $\mathbf{dist(K_1, T_3)}$
$dist(K_2, C)$ $dist(K_3, C)$	$dist(K_2, C)$ $dist(K_3, C)$ $\mathbf{dist(K_4, C)}$
$dist(T_1, C)$ $dist(T_2, C)$	$dist(T_1, C)$ $dist(T_2, C)$ $\mathbf{dist(T_3, C)}$
Min($dist(K_2, T_1)$, $dist(K_2, T_2)$) Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)	Min($dist(K_2, T_1)$, $dist(K_2, T_2)$, $\mathbf{dist(K_2, T_3)}$) Min($dist(K_3, T_1)$, $dist(K_3, T_2)$, $\mathbf{dist(K_3, T_3)}$) Min($\mathbf{dist(K_4, T_1)}$, $\mathbf{dist(K_4, T_2)}$, $\mathbf{dist(K_4, T_3)}$)
Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$) Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$)	Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$, $\mathbf{ang(K_2, K_1, T_3)}$) Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$, $\mathbf{ang(K_3, K_1, T_3)}$) Min($\mathbf{ang(K_4, K_1, T_1)}$, $\mathbf{ang(K_4, K_1, T_2)}$, $\mathbf{ang(K_4, K_1, T_3)}$)

field would dictate which source tasks(s) are most similar and where data should be transferred from.

Transferring Low-Level Information: Mapping novel structure. *Inter-task mappings* [45] allow a TL algorithm to explicitly state the relationship between different state variables and actions in the two tasks. For instance, the novel 4 vs. 3 action, `pass` to third closest teammate, may be mapped to the 3 vs. 2 action `pass` to second closest teammate. When such an inter-task mapping is provided (and is correct), transfer may be even more effective than if the novelty is ignored. Such an approach generalizes the source task knowledge to the target task. By generalizing over different target task state variables and actions, the TL method can initialize all target task Q-values to values learned in the source task, biasing learning and resulting in significantly faster learning. The Value Function Transfer method in Table 2 uses inter-task mappings (columns four and five), and outperforms Q-value Reuse, which had ignored novel state variables and actions.

While inter-task mappings are a convenient way to fully specify relationships between MDPs, it is possible that not enough information is known to design a full

Table 2. Results in columns two and three (reproduced from [45]) show learning 3 vs. 2 for different numbers of episodes and then using the learned 3 vs. 2 CMAC directly while learning 4 vs. 3. Minimum learning times for reaching a preset 4 vs. 3 performance threshold are bold. All times reported are "simulator hours," the number of playing hours simulated, as opposed to wall-clock time. The top row of results shows the time required to learn in the 4 vs. 3 task without transfer. As source task training time increases, the required target task training time decreases. The total training time is minimized with a moderate amount of source task training. The results of using Value Function Transfer (with inter-task mappings) are shown in columns four and five. Q-value Reuse provides a statistically significant benefit relative to no transfer, and Value Function Transfer yields an even higher improvement.

Transfer Results between 3 vs. 2 and 4 vs. 3 Keepaway

	Q-value Reuse		Value Function Transfer	
# of 3 vs. 2 Episodes	Avg. 4 vs. 3 Time (sim. hours)	Avg. Total Time (sim. hours)	Avg. 4 vs. 3 Time (sim. hours)	Avg. Total Time (sim. hours)
0	30.84	30.84	30.84	30.84
10	28.18	28.21	24.99	25.02
50	28.0	28.13	19.51	19.63
100	26.8	27.06	17.71	17.96
250	24.02	24.69	16.98	**17.65**
500	22.94	24.39	17.74	19.18
1,000	22.21	**24.05**	16.95	19.70
3,000	**17.82**	27.39	**9.12**	18.79

mapping. For instance, an agent may know that a pair of state variables describe "distance to teammate" and "distance from teammate to marker," but the agent is not told *which* teammate the state variables describe. *Homomorphisms* [31] are a different abstraction that can define transformations between MDPs based on transition and reward dynamics, similar in spirit to inter-task mappings, and have been used successfully for transfer [35]. However, discovering homomorphisms is NP-hard [32]. Work by Soni and Singh [35] supply an agent with a series of possible state transformations (i.e., potential homomorphisms) and an inter-task mapping for all actions. One transformation, X, exists for every possible mapping between target task state variables to source task state variables. The agent learns in the source task as normal. Then the agent must learn the correct transformation: in each target task state s, the agent must choose to randomly explore the target task actions, or choose to take the action suggested by the learned source task policy using one of the existing transformations, X. Q-learning allows the agent to select the best state variable mapping, defined as the one which allows the player to accrue the most reward, as well as learn the action-values for the target task. Later work by Sorg and Singh [36], extend this idea to that of learning "soft" homomorphisms. Rather than a strict surjection, these mappings assign probabilities that a state in a source task is the same as a state in the target task. This added flexibility allows the authors to provide bounds on their algorithm's performance, as well as show that such mappings are indeed learnable.

The MASTER algorithm [42] transfers instances similar to Lazaric [23], except that novel state variables and actions may be explicitly accounted for by using inter-task

mappings. MASTER uses an exhaustive search to generate all possible inter-task mappings, and then selects the one that best describes the relationship between the source and target task, learning the best inter-task mapping. Then, data from the source task is mapped to the target task, and learning can continue to refine the source task data. Although this method currently does not scale to Keepaway (due to its reliance on model-learning methods [19] for continuous state variables), but is similar in spirit to the above two methods.

Transferring High-Level Information. This section discusses four methods which transfer higher-level information than the previously discussed transfer methods. One example is an *option*, where a set of actions are composed into a single high-level macro-action that the agent may choose to execute. Thus far, researchers have not quantified "high-level" and "low-level" information well, nor have they made convincing arguments to support the claim that high-level information is likely to generalize to different tasks within a similar domain. However, this claim makes intuitive sense and is an interesting open question.

Section 7 introduced Relational RL (RRL). Recall that the propositional representation allows state to be discussed in terms of objects and their properties. Actions in the RRL framework typically have pre- and post-conditions over objects. When the state changes such that there are more or fewer objects, learned Q-values for actions may be very similar, as the object on which the action acts has not changed. One example of such transfer is by Croonenbourghs et al. [10], where they first learn a source task policy with RRL. This source task policy then generates state-action pairs, which are in turn used to build a relational decision tree. This tree predicts which action would be executed by the policy in a given state. As a final step, the trees produce *relational options*. These options are directly used in the target task with the assumption that the tasks are similar enough that no translation of the relational options is necessary.

Konidaris and Barto [22] also consider transferring options, but do so in a different framework. Instead of an RRL approach, they divide problems into *agent-space* and *problem-space* representations [21]. Agent-space defines an agent's capabilities that remains fixed across all problems (e.g., it represents a robot's physical sensors and actuators); agent-space may be considered a type of domain hiding. The problem-space may change between tasks (e.g., there may be different room configurations in a navigation task). By assuming "pre-specified salient events," such as when a light turns on or an agent unlocks a door, agents may learn options to achieve these events. Options succeed in improving learning in a single task by making it easier for agents to reach such salient events (which are assumed to be relevant for the task being performed). Additionally, the agent may train on a series of tasks, learning options in both agent- and problem-space, and reusing them in subsequent tasks. The authors suggest that agent-space options will likely be more portable than problem-space options, and it is likely that problem-space options will only be useful when the source and target tasks are very similar.

Torrey et al. [49] also consider transferring option-like knowledge. Their method involves learning a *strategy*, represented as a finite-state machine, which can be applied to a target task with different state variables and actions. Strategies are learned in the source task and then remapped to the target task with inter-task mappings. These

transferred strategies are then demonstrated to the target task learner. Instead of exploring randomly, the agent is forced to execute any applicable strategy for the first 100 episodes, learning to estimate the value of executing different strategies. After this demonstration phase, agents may then select from all of the MDP's actions. Experiments show that learning first on 2-on-1 BreakAway (similar to 2 vs. 1 Keepaway, but where the "keepers" are trying to score a goal on the "takers"), improves learning in both 3-on-2 Breakaway and 4-on-3 BreakAway.

8.3 Abstraction in Transfer

The majority of the TL methods in the above section focused on the first goal of transfer: showing that source task knowledge can improve target task learning, when the source task is treated as a sunk cost. (An exception is Taylor et al. [45], where they demonstrate that learning a pair or sequence of tasks can be faster than directly learning the target task.) In this section, we discuss abstraction in the context of transfer. Specifically, if the source task is an abstract version of the target task, it may be substantially faster to learn than the target task, but still provide the target task learner with a significant advantage.

Taylor and Stone [43] introduced the notion of *inter-domain* transfer in the context of *Rule Transfer* (similar in spirit to Torrey et al.'s method [49]). The chapter showed two instances where transfer between very different domains was successful. In both cases, the source tasks are discrete and fully observable, and one is deterministic. The target task was 3 vs. 2 Keepaway, which has continuous state variables, is partially observable, and is stochastic (due to sensor and actuator noise).

One of the source tasks is shown in Figure 22. In this task, the player begins at one end of the 25×25 board and the opponent begins at the opposite end of the board. The goal of the player is to reach the opposite end of the board without being touched by the opponent (either condition will end the episode). The player's state is represented by three state variables and it has three actions available. The player may move North (forward) or perform a knight's jump: North + East + East, or North + West + West. Although this task is very different from Keepaway, there are some important similarities. For instance, when the distance between the keeper with the ball and the closest taker is small, the player is likely to lose the ball (and thus end the episode). In Knight's Joust, when the distance between the player and the opponent is small, the episode is also likely to end.

Most significant with respect to transfer is the fact that an agent sees an average of only 600 distinct states over a 50,000 episode learning trial of Knight's Joust. This makes learning a good source task policy relatively easy: learning in the Knight's Joust abstraction takes on the order of a minute, whereas learning in 3 vs. 2 Keepaway takes hours of wall clock time. Using an abstract source task that allows very fast learning makes the task of reducing the total training time must easier: if the goal is to learn 3 vs. 2 Keepaway, it makes sense to spend a minute or two learning an abstract source task because it can save hours of learning in the target task (relative to directly learning 3 vs. 2 Keepaway).

While this result is encouraging, it can be regarded as a proof of concept: both source tasks used by Taylor and Stone [43] were carefully designed to be useful for transfer

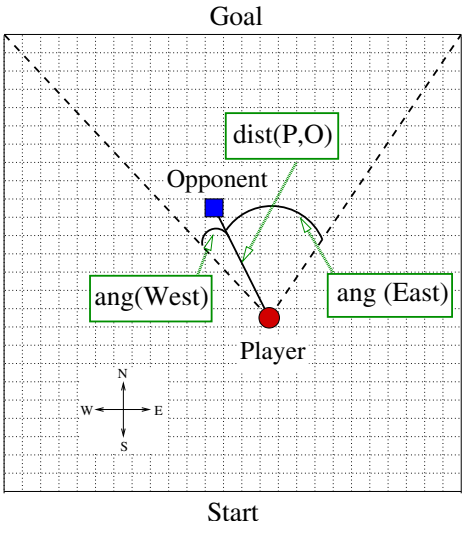


Fig. 22. Knight's Joust: The player attempts to reach the goal end of a a 25×25 grid-world while the opponent attempts to touch the player

into the Keepaway task. The idea of constructing sequences of tasks which are fast to learn is very appealing [4,41], but there are currently no concrete guidelines. This is due, in part, to the absence of a general method for deciding when transfer from a given source task will help learn a target task.

For instance, consider an agent that first learns the game of "Giveaway," where the goal is to loose the ball as fast as possible. If the agent then transfers this knowledge to Keepaway, it will perform worse than if it had ignored its previous knowledge [45]. While this may appear "obvious" to a human, such differences may be opaque to an agent and its learning algorithm. In the above example, the source task and target task differ only in reward function: the transition model, state variables, actions, etc. are all identical. Similarly, if an agent learns a specific path out of a maze in a source task and then uses this policy to navigate a target task maze, a malicious designer of the target task may make the source task policy perform arbitrarily poorly. Protecting against such negative transfer is an important open question.

9 Conclusions

In this survey we investigated generalization and abstraction in Reinforcement Learning. We provided the fundamentals of RL, introduced definitions of generalization and abstraction, and elaborated on the most common techniques to achieve both. These techniques include function approximation, hierarchical reinforcement learning, relational reinforcement learning and transfer learning. With novel and existing examples from the literature, we illustrated these techniques and provided many references to the literature. Our hope is that this chapter has provided a solid introduction and structure

to the concepts of abstraction and generalization in RL, encouraging additional work in this exciting field.

Acknowledgements

The authors would like to thank Scott Alfeld and Jason Tsai for useful comments and suggestions. Marc Ponsen is sponsored by the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

References

1. Albus, J.S.: Brains, Behavior, and Robotics. Byte Books, Peterborough (1981)
2. Barto, A., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems: Theory and Application 13(4), 341–379 (2003)
3. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
4. Bengio, Y., Collobert, J.L.R., Weston, J.: Curriculum learning. In: Proceedings of the Twenty-Sixth International Conference on Machine Learning (June 2009)
5. Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 369–376. MIT Press, Cambridge (1995)
6. Brafman, R.I., Tennenholtz, M.: R-max - a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research 3, 213–231 (2003)
7. Brooks, R.A.: Intelligence without representation. Artificial Intelligence (47), 139–159 (1991)
8. Caruana, R.: Multitask learning. Machine Learning 28, 41–75 (1997)
9. Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems, vol. 8, pp. 1017–1023. MIT Press, Cambridge (1996)
10. Croonenborghs, T., Driessens, K., Bruynooghe, M.: Learning relational options for inductive transfer in relational reinforcement learning. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 88–97. Springer, Heidelberg (2008)
11. Dean, T., Givan, R.: Model minimization in Markov decision processes. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, pp. 106–111 (1997)
12. Dietterich, T.: An overview of MAXQ hierarchical reinforcement learning. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 26–44. Springer, Heidelberg (2000)
13. Driessens, K.: Relational Reinforcement Learning. PhD thesis, DEPTCW (2004), http://www.cs.kuleuven.be/publicaties/doctoraten/cw/CW2004_05.abs.html
14. Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning 43, 7–52 (2001)
15. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556 (2005)
16. Giunchiglia, F., Walsh, T.: A theory of abstraction. Artificial Intelligence 57(2-3), 323–389 (1992)
17. Holte, R.C., Chouiery, B.Y.: Abstraction and reformulation in ai. Philosophical transactions of the Royal Society of London 358(1435:1), 197–204 (2003)

18. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
19. Jong, N.K., Stone, P.: Model-based exploration in continuous state spaces. In: The Seventh Symposium on Abstraction, Reformulation, and Approximation (July 2007)
20. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: Proc. 15th International Conf. on Machine Learning, pp. 260–268. Morgan Kaufmann, San Francisco (1998)
21. Konidaris, G., Barto, A.: Autonomous shaping: Knowledge transfer in reinforcement learning. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 489–496 (2006)
22. Konidaris, G., Barto, A.G.: Building portable options: Skill transfer in reinforcement learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 895–900 (2007)
23. Lazaric, A., Restelli, M., Bonarini, A.: Transfer of samples in batch reinforcement learning. In: Proceedings of the 25th Annual ICML, pp. 544–551 (2008)
24. Li, L., Walsh, T.J., Littman, M.L.: Towards a unified theory of state abstraction for MDPs. In: Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics, pp. 531–539 (2006)
25. Mahadevan, S., Maggioni, M.: Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. Journal of Machine Learning Research 8, 2169–2231 (2007)
26. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19(20), 629–679 (1994)
27. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence 12, 233–250 (1998)
28. Ponsen, M., Croonenborghs, T., Ramon, J., Tuyls, K., Driessens, K., van den Herik, J., Postma, E.: Learning with whom to communicate using relational reinforcement learning. In: International Conference on Autonomous Agents and Multi Agent Systems, AAMAS (2009)
29. Puterman, M.: Markov decision processes: Discrete stochastic dynamic programming. John Wiley and Sons, New York (1994)
30. Pyeatt, L.D., Howe, A.E.: Decision tree function approximation in reinforcement learning. In: Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation & Probabilistic Graphical Models, pp. 70–77 (2001)
31. Ravindran, B., Barto, A.: Model minimization in hierarchical reinforcement learning. In: Proceedings of the Fifth Symposium on Abstraction, Reformulation and Approximation (2002)
32. Ravindran, B., Barto, A.: An algebraic approach to abstraction in reinforcement learning. In: Twelfth Yale Workshop on Adaptive and Learning Systems, pp. 109–114 (2003)
33. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical report, Cambridge University Engineering Department (1994)
34. Skinner, B.F.: Science and Human Behavior. Colliler-Macmillian (1953)
35. Soni, V., Singh, S.: Using homomorphisms to transfer options across continuous reinforcement learning domains. In: Proceedings of the Twenty First National Conference on Artificial Intelligence (July 2006)
36. Sorg, J., Singh, S.: Transfer via soft homomorphisms. In: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems, May 2009, pp. 741–748 (2009)
37. Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway soccer: From machine learning testbed to benchmark. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 93–105. Springer, Heidelberg (2006)
38. Sutton, R.: Dyna, an integrated architecture for learning, planning, and reacting. SIGART Bulletin 2, 160–163 (1991)

39. Sutton, R., Barto, A.: Reinforcement Learning: an introduction. MIT Press, Cambridge (1998)
40. Sutton, R., Precup, D., Singh, S.: Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112, 181–211 (1999)
41. Taylor, M.E.: Assisting transfer-enabled machine learning algorithms: Leveraging human knowledge for curriculum design. In: The AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers (March 2009)
42. Taylor, M.E., Jong, N.K., Stone, P.: Transferring instances for model-based reinforcement learning. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 488–505. Springer, Heidelberg (2008)
43. Taylor, M.E., Stone, P.: Cross-domain transfer for reinforcement learning. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (June 2007)
44. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research 10(1), 1633–1685 (2009)
45. Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning Research 8(1), 2125–2167 (2007)
46. Tesauro, G.: TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation 6(2), 215–219 (1994)
47. Thorndike, E., Woodworth, R.: The influence of improvement in one mental function upon the efficiency of other functions. Psychological Review 8, 247–261 (1901)
48. Thrun, S.: Is learning the n-th thing any easier than learning the first? In: Advances in Neural Information Processing Systems, vol. 8, pp. 640–646 (1996)
49. Torrey, L., Shavlik, J.W., Walker, T., Maclin, R.: Relational macros for transfer in reinforcement learning. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 254–268. Springer, Heidelberg (2008)
50. Watkins, C.: Learning with Delayed Rewards. PhD thesis, Cambridge University (1989)
51. Weiss, G.: A multiagent variant of dyna-q. In: Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000), pp. 461–462 (2000)
52. Wiering, M.: Explorations in Efficient Reinforcement Learning. PhD thesis, Universiteit van Amsterdam (1999)
53. Zucker, J.D.: A grounded theory of abstraction in artificial intelligence. Philosophical transactions of the Royal Society of London 358(1435:1), 293–309 (2003)

The Effects of Evolved Sociability in a Commons Dilemma

Enda Howley and Jim Duggan

System Dynamics Research Group,
Department of Information Technology,
National University of Ireland, Galway, Ireland
{enda.howley,jim.duggan}@nuigalway.ie

Abstract. This paper explores the evolution of strategies in an n-player dilemma game. These n-player dilemmas provide a formal representation of many real world social dilemmas. Those social dilemmas include littering, voting and sharing common resources such as sharing computer processing time. This paper explores the evolution of altruism using an n-player dilemma and our results show the importance of sociability in these games. We propose a novel tag-mediated mechanism to allow for n-player interactions. This paper provides an examination of the interaction dynamics that occur in these n-player games when sociability is an evolved trait. Our results show how the agent population changes and evolves rapidly in response to the strategies of their peers in the population.

Keywords: Evolution, Learning, Cooperation, Agent Interactions, Tragedy of the Commons, Tag-Mediated Interaction Models.

1 Introduction

When a common resource is being shared among a number of individuals, each individual benefits most by using as much of the resource as possible. While this is the individually rational choice, it results in collective irrationality and a non Pareto-optimal result for all participants. These n-player dilemmas are common throughout many real world scenarios. For example, the computing community is particularly concerned with how finite resources can be used most efficiently where conflicting and potentially selfish demands are placed on those resources. Those resources may range from access to processor time or bandwidth.

One example commonly used throughout existing research is the *Tragedy of the Commons* [7]. This outlines a scenario whereby villagers are allowed to graze their cows on the village green. This common resource will be over grazed and lost to everyone if the villagers allow all their cows to graze, yet if everyone limits their use of the village green, it will continue to be useful to all villagers. Another example is the *Diners Dilemma* where a group of people in a restaurant agree to equally split their bill. Each has the choice to exploit the situation and order the most expensive items on the menu. If all members of the group apply this strategy, then all participants will end up paying more [4].

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 33–48, 2010.

These games are all classified as n-player dilemmas, as they involve multiple participants interacting as a group. N-player dilemmas have been shown to result in widespread defection unless agent interactions are structured. This is most commonly achieved through using spatial constraints which limit agent interactions through specified neighbourhoods on a spatial grid. Limiting group size has been shown to benefit cooperation in these n-player dilemmas [22].

In this paper we will examine an n-player dilemma, and study the evolution of strategies when individuals can bias their interactions through a tag mediated environment. Furthermore, we will show how certain strategies evolve with respect to their sociability towards their peers. The simulations presented in this paper use the n-player Prisoner's Dilemma (NPD). The purpose of this paper is to examine the impact sociability on cooperation throughout the agent population in the NPD. The research presented in this paper will deal with a number of specific research questions:

1. Can a tag-mediated interaction model be used to determine group interactions in a game such as the NPD?
2. If agents have an evolvable trait which determines their sociability, then will this trait prove significant to the emergence of cooperation in the agent society?

The following section of his paper will provide an introduction to the NPD and a number of well known agent interaction models. In the experimental setup section we will discuss our simulator design and our experimental parameters. Our results section will provide a series of game theoretic simulations. Finally we will outline our conclusions and future work.

2 Background Research

A number of researchers have used social dilemmas in a number of contexts including trust [17] [3], social capital [19] and solidarity [20]. Social dilemmas are particularly useful as analytic tools through which large groups of agents can be studied. These individuals can have significant interdependencies and interact through complex social structures. These agent interactions have been identified as being very significant to the study of social dilemmas [10] [1]. The importance of these interaction choices motivates our interest in agent sociability and the evolution of cooperation in an n-player dilemma.

A number of investigations into social dilemmas have attempted to study ways of evolving cooperation. For example, Suzuki extended traditional studies by allowing individuals become charging agents in the tragedy of the commons [18]. In other work Yamashita et al, have examined the effects of group dynamics in the NPD [21]. Individuals come together to form groups and then participate in NPD games with each other. Yamashita studied a number of group formation mechanisms, which included unilateral choice and mutual choice. Unilateral stated that once an agent wanted to join a group then ie is admitted into the group. In mutual choice the agent must be accepted in by a majority of the

existing group members. This mutual choice mechanism was then augmented to include group splitting which allowed dissenting individuals in the group to split away on their own when a group applicant divided the voting group members. The agreeing agents proceeded to accept the new individual into their group, while the opposing agents would leave to form a new group of their own [21].

The work outlined in this paper differs through attempting to examine group dynamics through a tag-mediated interaction model. Existing research has not examined the NPD with respect to tags and their known ability to effectively bias agent interactions and engender cooperation in two player games such as the Prisoner's Dilemma. By proposing a tag mediated interaction model for n-player games, we hope to bridge the gap between the research already conducted involving tags in two player games [16] [12], and the need for more research involving group structures in many n-player games [1] [21].

2.1 The N-Player Prisoner's Dilemma

The n-player Prisoner's Dilemma is also commonly known as the Tragedy of the Commons [7] and the payoff structure of this game is shown in Figure 1.

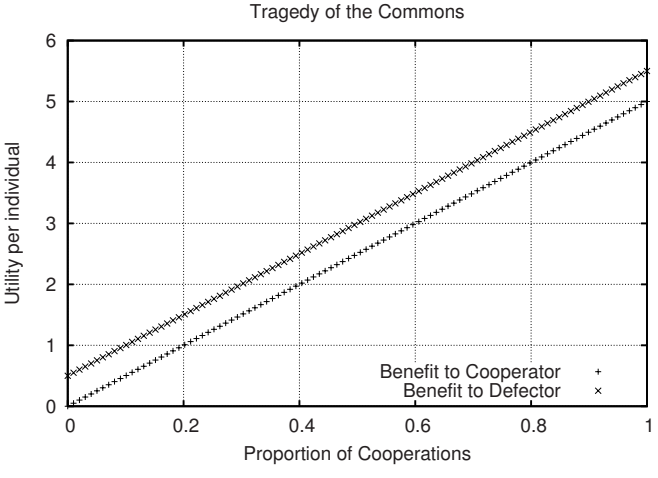


Fig. 1. The N-Player Prisoner's Dilemma

The x-axis represents the fraction of cooperators in the group of n players in a particular game. The vertical axis represents the payoff for an individual participating in a game. There is a linear relationship between the fraction of cooperators and the utility received by a game participant. Importantly, the payoff received for a defection is higher than for a cooperation. The utility for defection dominates the payoff for cooperation in all cases. Therefore, an individual that defects will always receive a higher payoff than if they had chosen to cooperate. The result of this payoff structure should result in an advantage to defectors in the agent population. Despite this, a cooperator in a group of cooperators will do much better than a defector in a group of defectors.

This game is considered a valid dilemma due to the fact that individual rationality favours defection despite this resulting in state which is less beneficial to all participants. In our case where all individuals defect they all receive 0.5. This state is a non-pareto, sub-optimal, and collectively irrational outcome for the agent population. For all values of x this can be expressed as follows:

$$U_d(x) > U_c(x) \tag{1}$$

x represents the fraction of cooperators while U_d and U_c are utility functions based on the fraction of cooperators in the group.

2.2 Agent Interaction Models

A number of alternative agent interaction models have been proposed and examined, such as spatial constraints [14] [13] and tag mediated interactions [16]. The importance of group size has been demonstrated in the NIPD [22] and also in the PD using tags [9]. Yao and Darwin demonstrated the effects of limiting group size, which was shown to benefit cooperation. Increasingly complex aspects of agent interactions have been examined by a number of authors, these include the effects of community structure on the evolution of cooperation [15] [2]. These have shown that neighbourhood structures benefit cooperation.

In this paper we are most concerned with tag-mediated interactions. Tags are visual markings or social cues which can help bias social interactions [8]. They are a commonly used agent interaction model and can be considered akin to football supporters identifying each other through wearing their preferred team colours. Similarly individuals can identify each other in conversations through a common language, dialect, or regional accent. Tag-mediated interaction models are often considered as more abstract interaction models, and thereby useful to represent agent interactions more abstractly without the specific characteristics of a specific topology or implementation. The research presented by Riolo demonstrated how tags can lead to the emergence of cooperation in the Prisoner's Dilemma [16]. Riolo investigated both a fixed and an evolved tag bias. More recently tags have been successfully applied to multi-agent problems [5] [6]. Tags have been shown to promote mimicking and thereby have major limitations where complimentary actions are required by agents. Cooperation that can be achieved through identical actions is quite easily achieved using tags, yet behaviours that require divergent actions are problematic [12] [11].

In this paper we will augment existing research to show the effects of using tags to determine group interactions in the NPD. The following section will provide a detailed specification of our simulator and the overall design of our experiments.

3 Experimental Setup

In this section we will outline our agent structure, our agent interaction model and our evolutionary algorithm. The following diagram shows the sequence of events as they happen in our simulations from generation G to generation $G+1$.

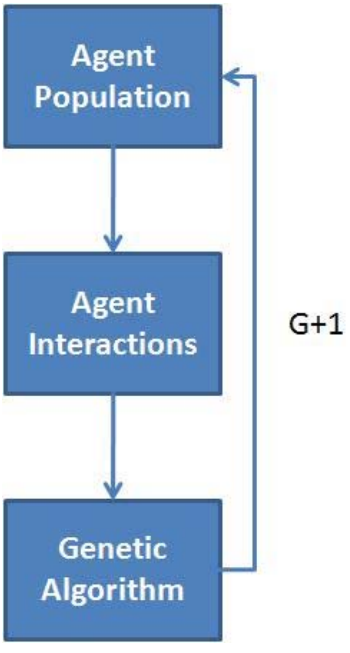


Fig. 2. Simulation Sequence

In the first generation the initial agent population is initialised. In each generation the agent population interacts through the agent interaction model, where through game interactions individuals can achieve fitness scores. These fitness scores are then used to evolve the population using a genetic algorithm for the subsequent generation.

3.1 Agent Genome

In our model each agent is represented through an agent genome. This genome holds a number of genes which represents how that particular agent behaves.

$$Genome = G_C, G_T, G_S, \tag{2}$$

The G_C gene represents the probability of an agent cooperating in a particular move. The G_T gene represents the agent tag. This is represented in the range $[0 \ldots 1]$ and is used to determine which games each agent participates. Finally, the G_S gene represents the sociability of each agent. This gene is also a number in the range $[0 \ldots 1]$ which acts as a degree of sociability for that individual agent. Initially these agent genes are generated using a uniform distribution for the first generation. Over subsequent generations new agent genomes are generated using our genetic algorithm. Each of these genes are evolved attributes and are fixed for that individual's lifetime, therefore changes in the population only occur through new offspring which have evolved genetic traits.

3.2 Agent Interactions

In our simulations each agent interacts through a tag mediated interaction model. We adopt a similar tag implementation as that outlined by Riolo [16]. In our model each agent has a G_T gene which is used as their tag value. Each agent A is given the opportunity to make game offers to all other agents in the population. The intention is that this agent A will host a game and the probability other agents will participate is determined using the following formulation.

$$d_{A,B} = 1 - |A_{GT} - C_{GT}| \tag{3}$$

This equation is based on the absolute value between the tag values of two agents A and B. This value is used to generate two roulette wheels R_{ab} and R_{ba} for A and B. These two roulette wheels will then be used to determine agent A's attitude to B and agent B's attitude to A. An agent B will only participate in the game when both roulette wheels have indicated acceptance. The distribution of these roulette wheels are also influenced by each agents sociability gene. This gene acts like a scalar value which is used to reflect that some agents are more sociable than others and will therefore be more willing to play with their peers. This is shown in the following equation, where R_{ab} represents the roulette wheel probability representing agent A's attitude towards B.

$$R_{ab} = d_{A,B} \times A_{GS} \tag{4}$$

The following pseudocode shows how agents are allowed to determine their game interactions through their tag values and their evolved sociability genes. These values are used to calculate the probabilities used in two roulette wheels which determine the final interaction decision.

```
WHILE(Agent_A < POPULATION_SIZE)

    Game_Participant_Set.Add(Agent_A)

        WHILE(Agent_B < POPULATION_SIZE)

            Decision_A = Decision_Roulettewheel(Agent_A, Agent_B)

            Decision_B = Decision_Roulettewheel(Agent_B, Agent_A)

            IF(Decision_A == TRUE && Decision_B == TRUE)

                Game_Participant_Set.Add(Agent_B)

        ENDWHILE

ENDWHILE
```

As the pseudocode outlines each agent in the population is given the opportunity to host a game. By default the offering agent A is a participant in the game. It makes game offers to all other agents B in the population. Two roulette wheels are then used to determine whether each agent B is added to the game. Equation 4 is used to determine the two roulette wheels R_{ab} and R_{ba}. Both roulette wheels are necessary in order to include the sociability of each agent towards their peers. Once completed the final set of participant agents then play the NPD game together.

3.3 Genetic Algorithm

In our simulator we have implemented a simple genetic algorithm. In each generation individuals participate in a variable number of games. Therefore, fitness is determined by summing all their payoffs received and getting an average payoff per game. In each generation, the top 5% of agents are copied directly into the following generation. The other 95% of the agent population in generation $G+1$ are generated through evolving new strategies based on agent fitness in G. Individuals are selected through roulette wheel selection based on their fitness from generation G. Parent pairs are selected based on their fitness and then these are used to generate a single new agent offspring for generation $G+1$. Each genome contains only three genes $G_C, G_T, G_S,$. We apply crossover by selecting two random genes from the fittest parent, and the remaining gene from the other parent. A 3% chance of mutation on each of these strategy genes is also applied, and once this occurs a gaussian distribution is used to determine the degree of change.

4 Experimental Results

In this section we will present a series of simulations showing the results of our experiments. Our results depict data from a single run over 10000 Generations. The aim of this single run is to show the inherent links between certain agent gene values and the overall cooperation throughout the agent population. Later in this section we will also present simulations involving subset of individuals in the population, based on their location in the tag space. All our simulations were conducted using an agent population of 100 agents.

4.1 Benchmark Simulation

In this section we briefly examine a base case simulation which ignores the sociability gene. The agents bias their interactions based only on their tag values. Using the formula in Equation 3, the agent population will actually fail to evolve cooperation. The following graph shows the results of a simulation run involving just this equation and no sociability gene.

The simulation shown in Figure 3 shows the case where the sociability gene is not used and the tag distribution is used solely. Due to the size of the population

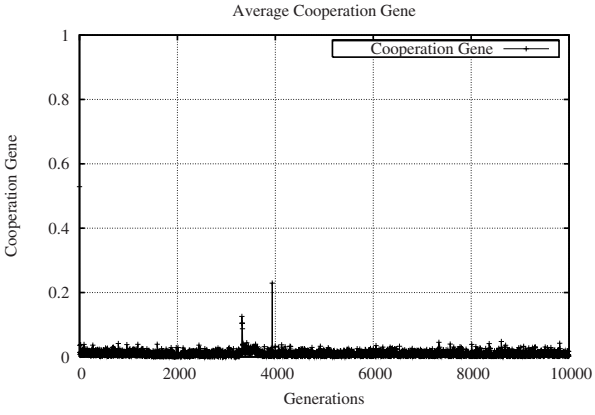


Fig. 3. Average Cooperation Gene - No Sociability Gene (1 Run)

(100) and the probability distribution of the tag space represented by Equation 3, defection is widespread. This is due to the large numbers of individuals participating in games and the inability of the tags to partition the population for these specific settings. A smaller population would help increase the chance of cooperation emerging, or alternatively altering the probability distribution presented in Equation 3. Through multiplying the probability of two individuals interacting by a value such as 0.25, it is possible to evolve total cooperation quite simply through this tag interaction model. This would have the effect of changing the probability distribution, and help partition the population using this fixed parameter. This paper explores the effects of avoiding such parameter setting, and the effects of evolved sociability in a game such as the n-player Prisoner's Dilemma.

4.2 Experiment 1

The simulation in Figure 3 shows the base case where all individuals using just their tags and Equation 3, to bias their interactions and the subsequent levels of cooperation that are evolved.

Figure 4 shows the emergence of cooperation throughout the agent population, when the sociability gene is used to determine game participation. This graph depicts the average G_C gene throughout the agent population in each generation. The results shows the emergence of cooperation as individuals in the population participate in various NPD games. The data reflects the emergence of cooperation which is quickly invaded and undermined by less cooperative individuals. This pattern continues throughout the simulations as cooperation emerges and is then undermined.

The data shown in Figure 5 shows the corresponding levels of sociability that are evolved throughout the agent population. This data indicates the overall dominance of strategies that limit their interactions. This strategy parameter

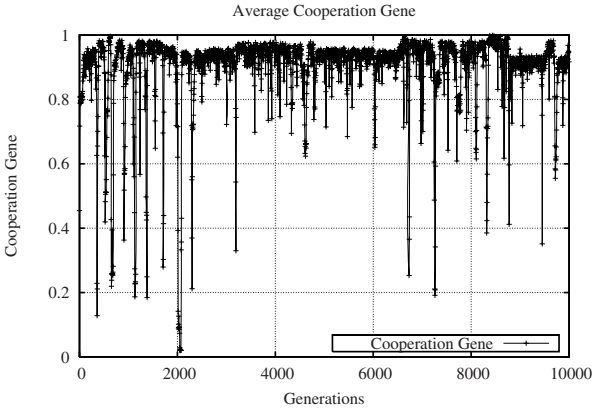


Fig. 4. Average Cooperation Gene (1 Run)

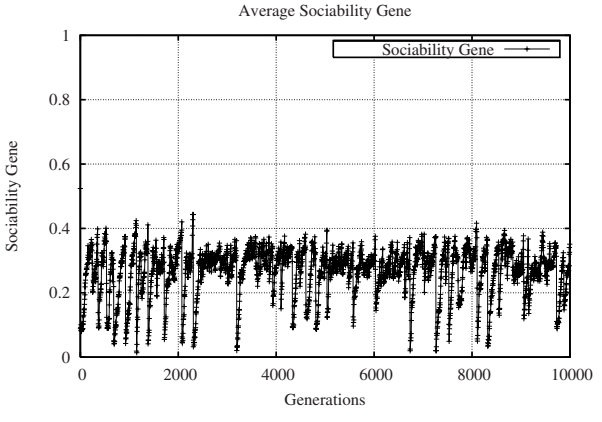


Fig. 5. Average Sociability Gene(1 Run)

has a direct effect on the number of game interactions each individual participates in. Similarly, this strategy trait also impacts on the number of individuals participating in each NPD game. The trends observed in Figure 4, can also be noticed in this data, where heightened instability reflects the pressure on the agents to void exploitation through being less sociable. Lower levels of cooperation occur through individuals are heavily exploited for their cooperative behaviour, and as a result those who are less sociable have an advantage. This results in the dominance of less sociable strategies which can maintain cooperative interactions without being exploited.

The results in Figure 6, depict the average number of games each agent participates in throughout successive generations. These results show the numbers of games entered into, which resulted in the simulations shown in Figures 4, 5.

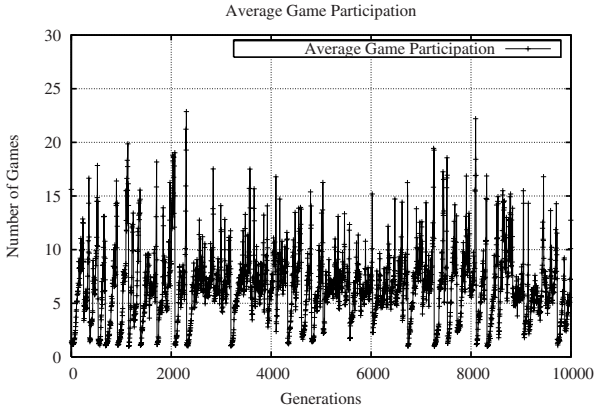


Fig. 6. Average Number of Games (1 Run)

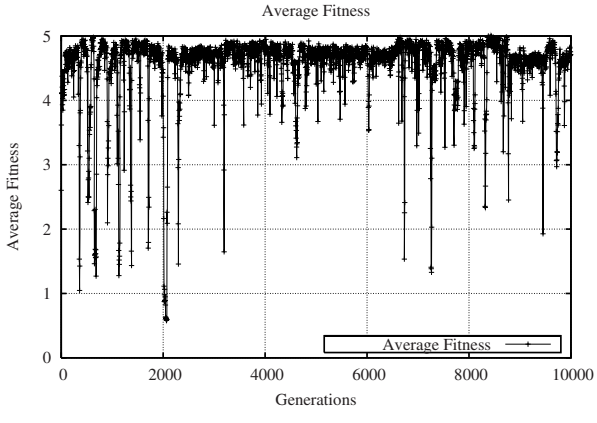


Fig. 7. Average Fitness (1 Run)

We note the features shown in Figures 4, 5, and 6 correlate strongly and show the strong links between sociability, game participation and cooperation.

In Figure 7 we show the levels of fitness recorded throughout the agent population. This data shows the high levels of fitness achieved through the evolved sociability and cooperative traits of the population. This fitness indicates the benefit to the entire agent population of high levels of cooperation between the participating players. This fitness is only undermined when individuals in the population become more sociable, and are then exposed to exploitation. The data shown in Figures 4, 5, 6, and 7 are typical results and were confirmed over multiple runs of our simulations. The following data shown in Figure 8 shows data averaged over 25 experimental runs.

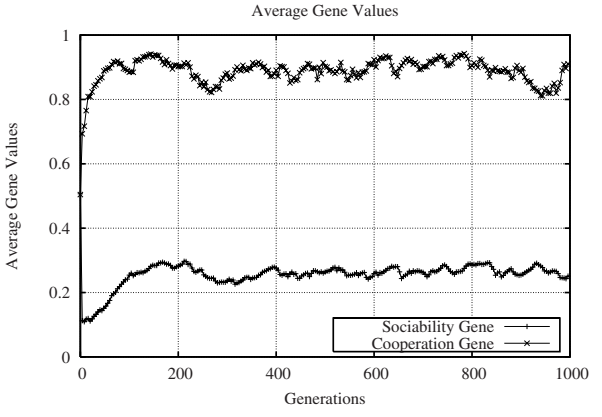


Fig. 8. Average Strategy Genes (25 Runs)

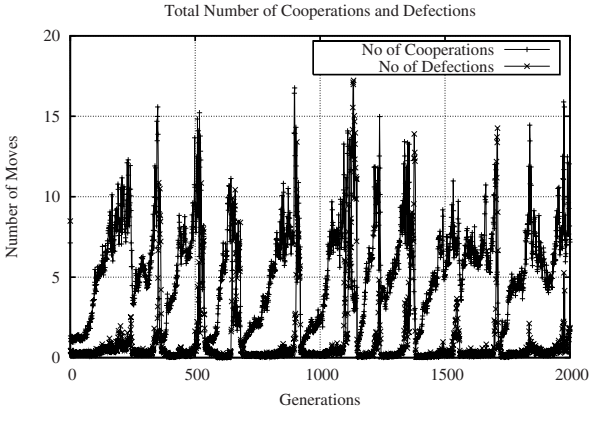


Fig. 9. Number of Cooperations and Defections (1 Run)

Figure 8 shows the emergence of strategies with higher levels of cooperation and lower levels of sociability throughout successive simulations. The following two graphs examine in greater detail the actions of the agent population in the initial 2000 generations of this sample run.

Figure 9 shows the numbers of cooperations and defections that occurred on average per agent in each generation of the population. These show the effects of increased and decreased sociability in the population and also the subsequent relationship between levels of cooperation, defection and the numbers of moves. As expected from the results shown earlier in Figures 4 and 8 the levels of cooperation are generally greater than the levels of defection. Furthermore, the numbers of moves reflect the levels of sociability and game participation identified in Figures 5 and 6, over the first 2000 generations. The following figure

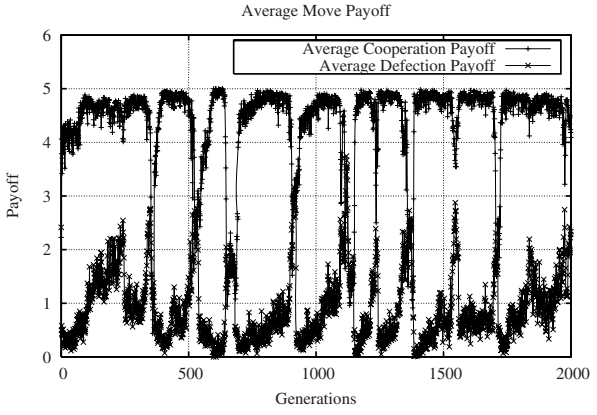


Fig. 10. Average Move Payoff (1 Run)

shows the average levels of payoff achieved by these cooperators and defectors in this simulation.

The data shown in Figure 10 shows the levels of payoff receives on average for the cooperators and defectors depicted previously in Figure 9. The payoff relationship between cooperation and defection in this game is quite clear from this data. The inverse relationship between cooperation and defection is shown, and the high payoffs achieved for cooperation are dramatic. These high payoff levels would not be possible without cooperators avoiding exploitation from defectors through limiting their interactions.

4.3 Experiment 2

In the following set of graphs we examine a portion of the agent population. We examine the evolution of agent cooperation and sociability within a small segment of the agent population. The data presented shows only individuals who have tag values between 0.1 and 0.15. This represents a sample of 5% of the potential tag space. The individuals located in this segment of the tag space may interact with many other individuals outside this tag range, depending on their respective sociability gene.

Figure 11 shows the average sociability among the individuals in the tag space between 0.1 and 0.15. This indicates the preference for peer interactions among this subset of the population. We observe that this sociability gene spikes to quite high levels at certain points, but in general remains at quite low levels. Interestingly, as with the previous data, the sociability of these individuals cannot be sustained at a heightened level for any significant period of time. This is due to the effects of mutation in the population, and drift where individuals are moving in the tag space and influencing the what peers are exploited or not.

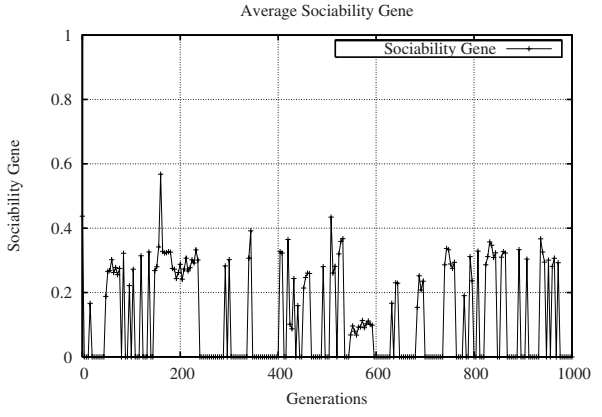


Fig. 11. Reduced Sample - Average Sociability Gene (1 Run)

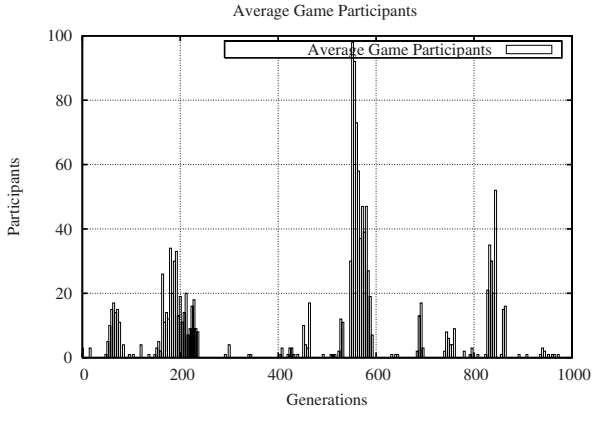


Fig. 12. Reduced Sample - Number of Agents (1 Run)

The data observed in Figure 11 is clarified through the results shown in Figure 12. Here we observe the numbers of individuals in this tag space. The number of individuals in this tag space increases to higher levels at certain points due to the convergence of the population to a particular tag value for a period of time. These higher levels are not sustained for long periods of time. We note that heightened levels of sociability in Figure 11 are mainly achieved when there are a small number of individuals in the tag space. Furthermore, when high numbers of individuals were identified, sociability levels were low. These results confirm our belief that once individuals begin to participate in a larger number of interactions, they expose themselves to exploitation and therefore achieve lower fitness. This cycle occurs repeatedly throughout the population with individuals benefitting from being less sociable as they are exploited for being sociable.

5 Conclusions

This paper has examined the NPD game with respect to group participation. For the first time this game has been investigated using a tag-mediated interaction model. Our results demonstrate that despite there being a clear incentive to defect, cooperation can still emerge. This stems from the ability of individuals in our agent population to determine their degree of sociability towards their peers. This reinforces much of the existing literature involving the traditional Prisoner's Dilemma [9] and also the NIPD [22]. Our model reinforces these observations through an alternative approach. In our case we have not explicitly predefined the sociability of our agent population. We have allowed the agent population to evolve with respect to their cooperative and sociability genes. The results have demonstrated the significance of sociability in games such as the NPD. Furthermore, we have also demonstrated the advantage to cooperative individuals who act less sociably towards their peers. Limiting game participation provides a very effective defence against exploiters. Earlier in our introduction we posed two specific research questions.

1. Our results show that tags can successfully bias interactions in the the NPD. We believe this is the first time a tag model has been applied to the NPD. Our results show the resulting levels of cooperation that emerge.
2. The significance of the sociability gene in our simulations is clear from the strong link between cooperation and sociability in our simulations.

This paper has presented an evolutionary model capable of modeling sociability within the agent strategy genome. We have also shown how tags can be used to determine n-player games. Our results have shown that occurrences of sociability are rapidly suppressed through exploitation and a subsequent return to less sociability. Finally, our results have shown through an evolutionary model that there is a strong benefit to agent strategies who are cooperative and less sociable through limiting their exposure to exploitation.

6 Summary and Future Work

This paper has shown that tags can be successfully adapted to bias agent interactions in a n-player game such as the NPD. Furthermore, we have demonstrated how an agent population can engender and maintain cooperation through an evolvable sociability trait. In future work we hope to examine how cooperation can be engendered while also increasing game participation.

Acknowledgments

The authors would like to gratefully acknowledge the continued support of Science Foundation Ireland (SFI) in this research.

References

1. Benard, S.W.: Adaptation and network structure in social dilemmas. Paper presented at the annual meeting of the American Sociological Association, Atlanta Hilton Hotel, Atlanta, GA (2003)
2. Chiong, R., Dhakal, S., Jankovic, L.: Effects of neighbourhood structure on evolution of cooperation in n-player iterated prisoner's dilemma. In: Yin, H., Tino, P., Corchado, E., Byrne, W., Yao, X. (eds.) IDEAL 2007. LNCS, vol. 4881, pp. 950–959. Springer, Heidelberg (2007)
3. Coleman, J.: Foundations of Social Theory. Belknap Press (August 1998)
4. Glance, N.S., Huberman, B.A.: The dynamics of social dilemmas. Scientific American 270(3), 76–81 (1994)
5. Hales, D., Edmonds, B.: Evolving social rationality for mas using "tags". In: AAMAS 2003: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pp. 497–503. ACM, New York (2003)
6. Hales, D., Edmonds, B.: Applying a socially inspired technique (tags) to improve cooperation in p2p networks. IEEE Transactions on Systems, Man, and Cybernetics, Part A 35(3), 385–395 (2005)
7. Hardin, G.: The tragedy of the commons. Science 162(3859), 1243–1248 (1968)
8. Holland, J.: The effects of labels (tags) on social interactions. Working Paper Santa Fe Institute 93-10-064 (1993)
9. Howley, E., O'Riordan, C.: The emergence of cooperation among agents using simple fixed bias tagging. In: Proceedings of the 2005 Congress on Evolutionary Computation (IEEE CEC 2005), vol. 2, pp. 1011–1016. IEEE Press, Los Alamitos (2005)
10. Macy, M., Flache, A.: Learning dynamics in social dilemmas. P Natl. Acad. Sci. USA 99(3), 7229–7236 (2002)
11. Matlock, M., Sen, S.: Effective tag mechanisms for evolving coordination. In: AAMAS 2007: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, pp. 1–8. ACM, New York (2007)
12. McDonald, A., Sen, S.: The success and failure of tag-mediated evolution of cooperation. In: Tuyls, K., 't Hoen, P.J., Verbeeck, K., Sen, S. (eds.) LAMAS 2005. LNCS (LNAI), vol. 3898, pp. 155–164. Springer, Heidelberg (2006)
13. Nowak, M., May, R.: The spatial dilemmas of evolution. Int. Journal of Bifurcation and Chaos 3, 35–78 (1993)
14. Oliphant, M.: Evolving cooperation in the non-iterated prisoner's dilemma: the importance of spatial organisation. In: Proceedings of Artificial Life IV (1994)
15. O'Riordan, C., Sorensen, H.: Stable cooperation in the n-player prisoner's dilemma: The importance of community structure. In: Tuyls, K., Nowe, A., Guessoum, Z., Kudenko, D. (eds.) ALAMAS 2005, ALAMAS 2006, and ALAMAS 2007. LNCS (LNAI), vol. 4865, pp. 157–168. Springer, Heidelberg (2008)
16. Riolo, R.: The effects and evolution of tag-mediated selection of partners in populations playing the iterated prisoner's dilemma. In: ICGA, pp. 378–385 (1997)
17. Sato, Y.: Trust, assurance, and inequality: A rational choice model of mutual trust 1. The Journal of Mathematical Sociology 26(1), 1–16 (2002)
18. Suzuki, K.: Effects of conflict between emergent charging agents in social dilemma. In: Kurumatani, K., Chen, S.-H., Ohuchi, A. (eds.) MAMUS 2003. LNCS (LNAI), vol. 3012, pp. 120–136. Springer, Heidelberg (2004)
19. Torsvik, G.: Social Capital and Economic Development: A Plea for the Mechanisms. Rationality and Society 12(4), 451–476 (2000)

20. Yamagishi, T., Cook, K.S.: Generalized exchange and social dilemmas. Social Psychology Quarterly 56(4), 235–248 (1993)
21. Yamashita, T., Axtell, R.L., Kurumatani, K., Ohuchi, A.: Investigation of mutual choice metanorm in group dynamics for solving social dilemmas. In: Kurumatani, K., Chen, S.-H., Ohuchi, A. (eds.) MAMUS 2003. LNCS (LNAI), vol. 3012, pp. 137–153. Springer, Heidelberg (2004)
22. Yao, X., Darwen, P.J.: An experimental study of n-person iterated prisoner's dilemma games. Informatica 18, 435–450 (1994)

Replicator Dynamics for Multi-agent Learning: An Orthogonal Approach

Michael Kaisers and Karl Tuyls

Maastricht University, P.O. Box 616, 6200 MD Maastricht

Abstract. Today's society is largely connected and many real life applications lend themselves to be modeled as multi-agent systems. Although such systems as well as their models are desirable, e.g., for reasons of stability or parallelism, they are highly complex and therefore difficult to understand or predict. Multi-agent learning has been acknowledged to be indispensable to control or find solutions for such systems. Recently, evolutionary game theory has been linked to multi-agent reinforcement learning. However, gaining insight into the dynamics of games, especially if time dependent, remains a challenging problem. This article introduces a new perspective on the reinforcement learning process described by the replicator dynamics, providing a tool to design time dependent parameters of the game or the learning process. This perspective is orthogonal to the common view of policy trajectories driven by the replicator dynamics. Rather than letting the time dimension collapse, the set of initial policies is considered to be a particle cloud that approximates a distribution and we look at the evolution of this distribution over time. First, the methodology is described, then it is applied to an example game and viable extensions are discussed.

Keywords: Reinforcement learning, Evolutionary game theory.

1 Introduction

The world of today is full of networks and connected systems, varying from stock exchanges and swarm robots to political networks [2, 7, 9, 11]. As a consequence, the assumption that a system runs in actual isolation of any other actor does not withstand reality. Hence, many domains need to be modeled as multi-agent systems in order to account for their inherent complexity. However, the models yield a complexity that makes them hard to understand, predict or control. As this is realized, multi-agent learning gains popularity to find solutions to or control these systems [10, 12].

Learning in multi-agent environments is significantly more complex than single-agent learning as the dynamics to learn change by the learning processes of other agents. This makes predicting learning behavior of learning algorithms in multi-agent environments difficult. They are not only situated in a non-stationary environment but also need to deal with incomplete information and communication limits. In non-stationary environments the Markov property does

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 49–59, 2010.

not hold, which makes all proofs of convergence to optimal policies from single-agent learning that are based on that assumption inapplicable. This reduces the theoretical framework available for multi-agent learning. More recently, evolutionary game theory with less strong assumptions than classical game theory could be linked to reinforcement learning and provides useful insights into the learning dynamics [1, 4, 15].

The learning dynamics are commonly visualized by showing the directional field plot of the replicator dynamics or showing policy trajectories with the time dimension collapsed into a surface. Both views work well for dynamics that do not change over time but provide little guidance when the game or the learning algorithm uses a parameter that is time dependent. In particular, the directional field plot can only capture the dynamics at one point in time. Hence, several independent plots are needed for changing dynamics and a gap remains in the transition between them. The trajectory view becomes unclear when circles occur or the dynamics change, in which case lines may intersect and clutter the plot. Furthermore, reducing the time dimension into a flat surface hinders the interpretation of time dependent artifacts. In addition, the higher the resolution (the more trajectories are plotted), the more crowded the plot and the harder it becomes to interpret. As a result, parameter tuning is a cumbersome task that often results in ad hoc trial and error approaches.

In order to tackle these problems, this article will answer the question how to extract more information from dynamical systems, especially for time dependent dynamics, with the goal of facilitating the systematical design of time dependent parameters. This is achieved by taking on a new perspective that is orthogonal to the common view of policy trajectories.

The remainder of this article is structured as follows: Section 2 introduces relevant concepts from game theory and reinforcement learning, and Section 3 proposes a new perspective on the process driven by the replicator dynamics. Section 4 demonstrates the new methodology on an example game and Section 5 concludes the paper with a discussion of viable extensions.

2 Background

2.1 Game Theory

Classical game theory is the mathematical study of strategic conflicts of rational agents. The central concept is the *game*, which comprises a set of players $I = \{1, 2, \ldots, n\}$ and a set of available pure strategies $S_i = \{1, 2, \ldots, k_i\}$ for each player i, for n and k_i some finite integer. For a more general introduction we refer the interested reader to [4, 16].

The players of normal form games are assumed to choose their pure strategies simultaneously and independently and receive a payoff that is dependent on the joint strategy profile $s \in S_1 \times \ldots \times S_n$. The payoff for two-player normal form games can be described by two matrices A and B, where for any joint strategy (i, j), A_{ij} denotes the payoff to player one and B_{ij} describes the payoff to player two.

As we are only considering repeated stateless games, the policy of each player can be described by a probability distribution over the available actions at each point in time t. The two-player game in this example will use the notation x and y for the policy vectors of the first and second player respectively.

2.2 Reinforcement Learning

Reinforcement Learning (RL) has originally been studied in the context of single-agent environments. An agent receives a numeric reward signal, which it seeks to maximize. The environment provides this signal as a feedback on the sequence of actions that has been executed by the agent. Learners relate the reward signal to previously executed actions to learn a policy that maximizes the expected future reward [13].

The environment is defined by the normal form game and the reinforcement learner updates the policy. A very simple policy iterator is the *Cross Learning* algorithm, a specific type of learning automata [1]. When action i is selected and reward $r(t) \in [0, 1]$ is received at time t, then policy x is updated according to the following equation:

$$x_i(t+1) \leftarrow (1 - r(t))x_i(t) + r(t)$$
$$x_j(t+1) \leftarrow (1 - r(t))x_j(t), \text{ for all } j \neq i$$

The policy change induced by this learner has been shown to converge under infinitesimal time steps to the replicator dynamics [1]. For the sake of clarity, this model is used for the further study in this article. However, it is worth mentioning that other learning algorithms can be described by similar differential equations, e.g., Q-learning with a Boltzmann exploration scheme has been shown to converge to an extension of the replicator dynamics [15]. The evolutionary description of reinforcement learning is detailed in the following subsections using the example of Cross learning.

2.3 Evolutionary Game Theory

Evolutionary game theory takes a rather descriptive perspective, replacing hyper-rationality from classical game theory by the concept of natural selection from biology [8]. The two central concepts of evolutionary game theory are the replicator dynamics and evolutionary stable strategies. The replicator dynamics presented in the next subsection describe the evolutionary change in the population. They are a set of differential equations that are derived from biological operators such as selection, mutation and cross-over. The evolutionary stable strategy describes the resulting asymptotic behavior of this population. However, their examination is beyond the scope of this article. For a detailed discussion, we refer the interested reader to [5, 6].

2.3.1 Replicator Dynamics

The replicator dynamics from evolutionary game theory formally define the population change over time. A population comprises a set of individuals, where

the species that an individual can belong to represent the pure strategies. The utility function can be interpreted as the Darwinian fitness of each species. The distribution of the individuals on the different strategies can be described by a probability vector that is equivalent to a policy. Hence, there is a second view on the evolutionary process: the population may also represent competing strategies within the mind of one agent, who learns which one to apply. The evolutionary pressure by natural selection can be modeled by the replicator equations. They assume this population evolve such that successful strategies with higher payoffs grow while less successful ones decay. These dynamics are formally connected to reinforcement learning [1, 14, 15]. Assume a two-player normal form game with payoff matrices A and B for player one and two respectively, and let the policies of player one and two be represented by the probability vectors $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$, where x_i and y_i indicate the probability to play action i. The two-player replicator dynamics that relate to the learning process of *Cross Learning*, a simple learning automaton, are given by the following set of differential equations, where e_i is the i^{th} unit vector:

$$\begin{aligned} \dot{x}_i &= x_i \left[e_i A y^T - x A y^T \right] \\ \dot{y}_i &= y_i \left[x B e_i^T - x B y^T \right] \end{aligned} \tag{1}$$

The change in the fraction playing action i is proportional to the difference between the expected payoffs $e_i A y$ and $x B e_i$ of action i against the mixing opponent, and the expected payoff xAy and xBy of the mixed strategy x against the mixed strategy y. Hence, above average actions strive while below average actions decay. The replicator dynamics maintain the probability distribution, thus $\sum_i \dot{x}_i = 0$. For the sake of clarity, the examples used in this article are constrained to two actions, which implies $\dot{x}_1 = -\dot{x}_2$ and $\dot{y}_1 = -\dot{y}_2$.

2.3.2 Policy Trajectories

The replicator dynamics describe how the policy changes over time, dependent on the game and the policy itself. Starting with a set of policies, we can follow this change over a period of time. The path that is taken is referred to as the policy trajectory. In a game with k actions for each player, the policy trajectory of p players can be specified with $(k-1)^p+1$ dimensions. In the case of two-player two-action games, the unit square yields the policy space, as we can specify x_1 as a full characterization of $x = (x_1, 1 - x_1)$, and y_1 similarly. One more dimension is optionally needed for an exploded view showing the time dimension.

3 Method

This section shows the learning process in a new perspective which is orthogonal to viewing policy trajectories in the classical way. Figure 1 shows 20 trajectories in an expanded view of policy space against time. Instead of looking at it from the top down, we suggest cutting slices at different points in time and looking at the distribution of trajectory points where they intersect these slices, which are orthogonal to the top-down view.

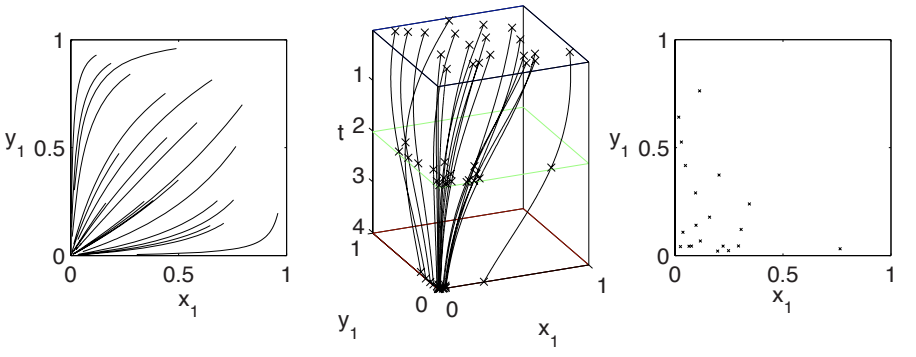


Fig. 1. An expanded view of 20 policy trajectories (middle), the common perspective collapsing the time dimension (left), showing the trajectories as a flat image, and the proposed orthogonal perspective (right), looking at the second slice that intersects the trajectories at the indicated points

The idea behind considering distributions rather than single trajectories is to obtain a more holistic view of the learning process. In the end, learning is a homeomorphic time dependent transformation of the policy space. As such, we can look at its influence on the whole space, e.g., by looking at the spacing between the trajectories, rather than only looking at individual policy trajectories. In order to do so, a set of particles is drawn from an initial distribution, in the examples given a uniform distribution, and subjected to a velocity field defined by the replicator dynamics. As time evolves, the distribution is transformed and the density of the particles changes. This allows to make statements of the following kind: assuming any policy was initially equally likely and these policies evolve according to the replicator dynamics, then after time t has passed, p percent of the policies have converged to attractor a with at most distance ϵ.

After some time, the simulation can be stopped and labels can be applied according to the eventual distribution. A certain percentage of particles can be considered converged to some attractors, assuming they are in the neighborhood of a stable point and that point is attracting in that neighborhood. Other particles can be labeled as not converged. Finally, these labels can be applied to earlier slices including the initial slice, revealing the basins of attraction. Although these basins can also be read from the directional field plot of the replicator dynamics, this approach is more general as it can be applied to dynamics that are controlled by a time dependent parameter.

In addition, this allows judging the convergence of a fraction of the policy space that is bound by a surface by considering the velocity field only on that surface. Due to the fact that the dynamics describe a continuous process and the transformation by the replicator dynamics is a homeomorphism, everything that is added or substracted from the trapped percentage has to go through the surface. This is related to the *divergence theorem* from physics [3]. It allows focussing attention on the surface that may be just a small subspace of the

whole policy space, e.g., a hypersphere with radius ϵ around an attractor. In many cases, the velocity field in this small neighborhood can be guaranteed to be rather static although the dynamics of other areas of the policy space may change quite substantially.

It is common to assume every policy initially equally likely, i.e., applying an initially uniform distribution. However, this approach allows to use an arbitrary initial distribution which can be used to model specific knowledge about the initial learning behavior. Furthermore, the policy distribution can also be generated from Q-value distributions, in case a Q-learning algorithm should be modeled. Using a similar evolution as the replicator dynamics in the Q-value space, the distribution can be evolved which allows comparing Boltzmann exploration to other exploration schemes that do not have a bijective action selection function[1] and can therefore not be solely described by dynamics in the policy space.

4 Experiments

This section demonstrates the proposed methodology on an example game that is controlled by a parameter that may change its value at one point in time. The game describes the following situation:

> There are two new standards that enable communication via different protocols. The consumers and suppliers can be described by probability vectors that show which standard is supported by which fractions. One protocol is 20% more energy efficient, hence the government wants to support that standard. Usually, the profit of the consumers and suppliers are directly proportional to the fraction of the opposite type that supports their standard. However, the government decides to subsidize early adopters of the better protocol.
>
> Such subsidies are expensive and the government only wants to spend as much as necessary. They have no market research information and consider any distribution of supporters on both sides equally likely. Furthermore, they know that the supporters are rational and their fractions will change according to the replicator dynamics. The question is, how long is the subsidy necessary to guarantee that the better standard is adopted in 95% of the possible initial policies.

This is a variation of the pure coordination game. A subsidy parameter $s \in \{0, 11\}$ is added, which can be used to make one action dominant. As a result, coordination on the Pareto optimal equilibrium is facilitated. Figure 2 displays the payoff bi-matrix numerically.

[1] Strictly speaking, Boltzmann action selection is also not a bijection, as it leaves one degree of freedom when computing Q-values from policies. However, each policy change relates to a Q-value change and vice versa, which is not the case in other exploration schemes such as epsilon-greedy.

	S_1	S_2
S_1	$10, 10$	$0, s$
S_2	$s, 0$	$12, 12$

	S_1	S_2
S_1	$10, 10$	$0, 0$
S_2	$0, 0$	$12, 12$

	S_1	S_2
S_1	$10, 10$	$0, 11$
S_2	$11, 0$	$12, 12$

Fig. 2. The payoff bi-matrix for the subsidy game (left) and its realizations for $s = 0$ (middle) and $s = 11$ (right). Player one chooses a row, player two chooses a column. The first number of the selected action combination represents the payoff to player one and the second number the payoff to player two.

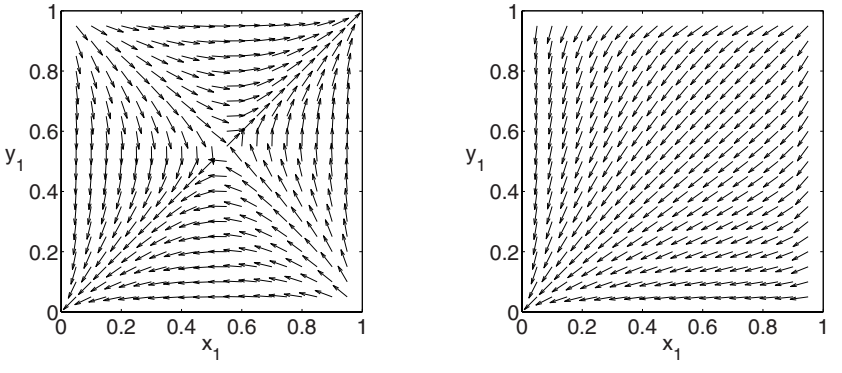


Fig. 3. The dynamics of the game without subsidy (left) and with subsidy (right)

The dynamics of the game can be visualized by showing the directional field plot of the replicator dynamics as shown in Figure 3. It can be observed that a large fraction of the policy space would converge to the suboptimal standard in the unsubsidized game, while all policies would converge to the optimum in the subsidized game. However, it is difficult to derive the correct time to switch between the two games.

The second classical way to look at the dynamics are policy trajectories. These will follow the directional change and are depicted in Figure 4. Similar to the replicator dynamics, this view neatly explains the dynamics of the individual parts of the game, but does not allow to infer the right time to switch from the one to the other.

Another possible approach is the visualization of trajectories with transitions from one game to the other at different points in time. Figure 5 shows the trajectories of the subsidy game when transition from $s = 11$ to $s = 0$ takes place at $t = \{0.1, 0.3, 0.5\}$. Although it can be observed that fewer trajectories converge suboptimal the later the switch occurs, this approach requires to guess the right time of transition. Furthermore, the view is cluttered by intersecting lines and readability does not allow to increase the number of trajectories.

In order to obtain insight into the time dependent artifacts of these dynamics, the new perspective will be applied. Answering the question when to switch requires 2 steps:

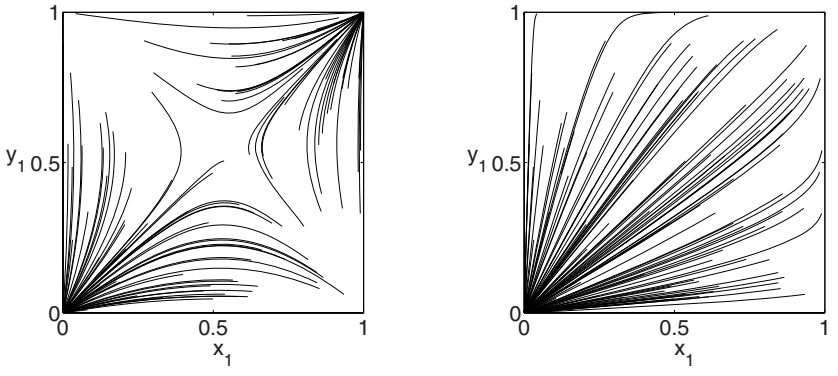


Fig. 4. Trajectories with a length of 4 units of continuous time in the game without subsidy (left) and with subsidy (right)

- Determine the part of the policy space for which trajectories converge optimally in the unsubsidized game.
- Determine the time when the subsidized dynamics have driven 95% of the initial policies into the previously derived subspace.

Step one is shown in Figure 6. Particles are drawn from a uniform initial distribution and evolved according to the replicator dynamics. After $t = 1.2$ the particles are considered converged and receive a label (a dot for the optimum and an x for the suboptimal attractor). Subsequently, the label is applied to all slices before plotting. From the labels on the initial slice, the basin boundary is deduced using a linear best fit, which is marked by the dashed line.

In step 2, shown in Figure 7, the boundary that has been inferred from step one is used to monitor the percentage of the initial policy space that would converge to the optimum if the game was switched at that time instance. The simulation advances until the subsidized dynamics have pushed 95% of the initial policies into the basin of attraction of the global optimum in the unsubsidized

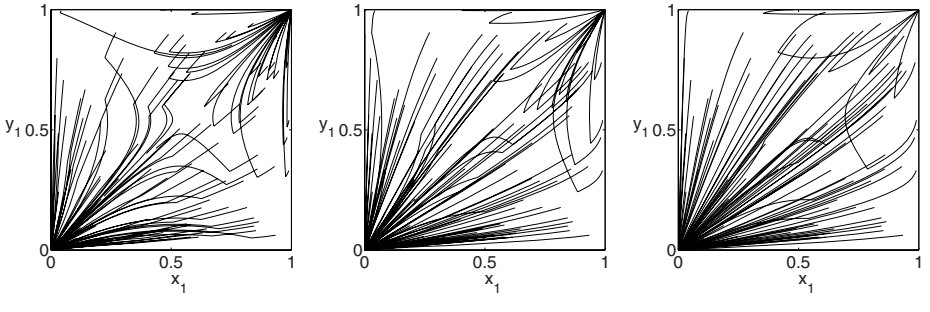


Fig. 5. The trajectory plot for the subsidy game with transition from the subsidized to the unsubsidized game at $t = \{0.1, 0.3, 0.5\}$ (left to right)

game. Then, the game is switched and the simulation shows convergence to the according attractors. Repeating the experiment $n = 1000$ times, we find that the time to bring 95% to the basin is 0.495 ± 0.0357 (indicating one standard deviation). A histogram of the distribution is given in Figure 8.

This section has demonstrated the advantages of the proposed perspective. It has been shown that the new methodology allows the systematic study and design of time dependent parameters in order to achieve desired effects, in the example a specific convergence behavior. The next section concludes the article with a discussion of contributions and viable extensions to the introduced approach.

5 Discussion and Conclusions

The contributions of this article can be summarized as follows: a new perspective on dynamical systems driven by time dependent replicator dynamics has been proposed, allowing to extracting the information encoded in the spacing between the particles. An illustrative example of a two-agent two-action game has been discussed, and the method has been shown to naturally reveal time dependent properties of the system. This facilitates designing parameters with a systematic approach rather than setting them ad hoc. While a rather simple example game was studied for the sake of clarity, the approach is general in the number of actions and can be applied to arbitrary initial distributions. In addition, it naturally generalizes to any number of agents when the reward matrix is considered to be a reward function on two strategy variables.

The parameter design methodology can be transferred to other parameters that change the replicator dynamics, most prominently the temperature function for Q-learning with a Boltzmann exploration scheme. Choosing an appropriate temperature function has long been a heuristic search and can now be tackled systematically to achieve a desired convergence distribution.

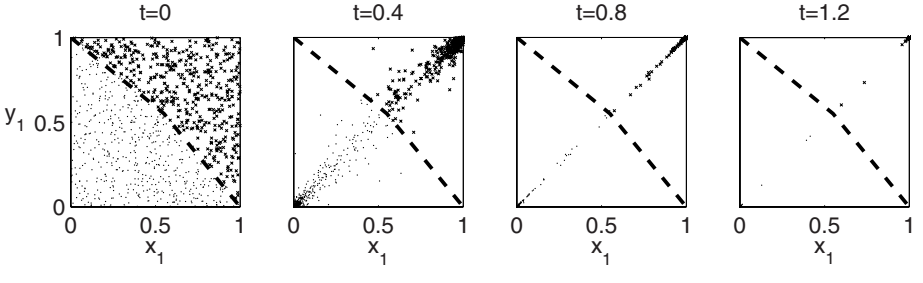


Fig. 6. This figure shows the evolution of particles drawn from a uniform initial distribution, revealing the basins of attraction of the unsubsidized game. Labels are applied according to the last slice and the dashed line is inferred from the labels to be the boundary between the basins of attraction.

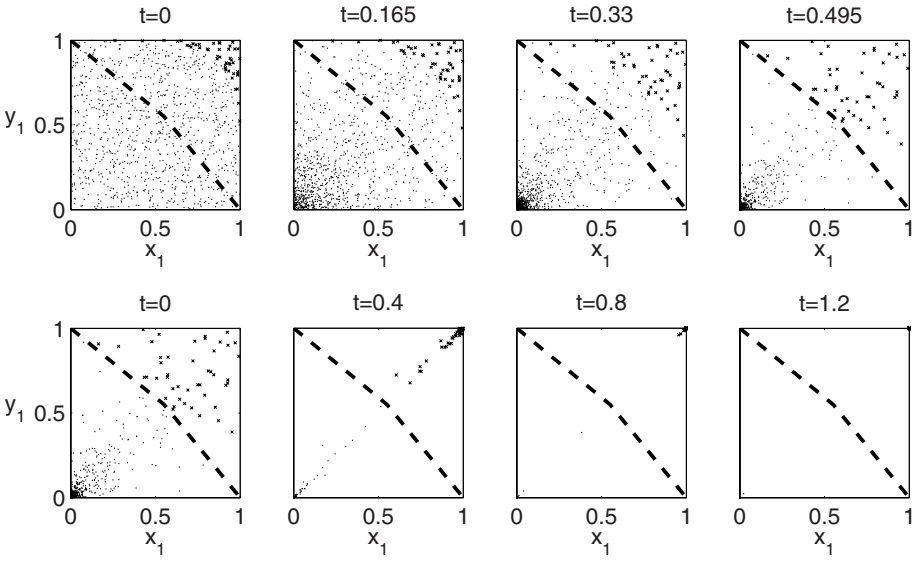


Fig. 7. The top row shows the evolution under the subsidized game until 95% of the policy space are in the basin for the global optimum of the unsubsidized game. The lower row shows the further evolution in the unsubsidized game.

Furthermore, the ideas presented in this paper have the strong potential to be further developed. The current approach can be seen as a particle simulation, where the replicator dynamics determine the velocity field that describes the movement of each particle, and the particle density describes a probability distribution. The authors have taken preliminary steps to make the transition to describe this probability density function as a continuous function, deriving the density change directly from the replicator dynamics. This will remove the stochasticity introduced by approximating the probability density by quantized

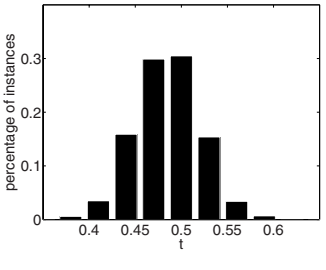


Fig. 8. Histogram of times at which the velocity field of the subsidized game has driven 95% of the particles into the basin of attraction of the global optimum in the unsubsidized game. The sample size is $n = 1000$, with a mean of 0.495 and a standard deviation of 0.0357.

particles. Another viable extension considers a distribution of Q-values and the distribution's evolution, deriving the according policy distributions from it. This enables the comparison of exploration schemes such as Boltzmann and epsilon-greedy exploration. Finally, this model is extendable to multiple states and continuous strategy spaces, which will compliment the theoretical framework for multi-agent learning.

Acknowledgements

This research was partially sponsored by a TopTalent2008 grant of the Netherlands Organisation for Scientific Research (NWO).

References

[1] Börgers, T., Sarin, R.: Learning through reinforcement and replicator dynamics. Journal of Economic Theory 77(1) (November 1997)
[2] Bueno de Mesquita, B.: Game theory, political economy, and the evolving study of war and peace. American Political Science Review 100(4), 637–642 (2006)
[3] Feynman, R.P., Leighton, R.B., Sands, M.: The Feynman Lectures on Physics including Feynman's Tips on Physics: The Definitive and Extended Edition. Addison Wesley, Reading (2005)
[4] Gintis, C.M.: Game Theory Evolving. University Press, Princeton (2000)
[5] Hirsch, M.W., Smale, S., Devaney, R.: Differential Equations, Dynamical Systems, and an Introduction to Chaos. Academic Press, London (2002)
[6] Hofbauer, J., Sigmund, K.: Evolutionary Games and Population Dynamics. Cambridge University Press, Cambridge (2002)
[7] Hon-Snir, S., Monderer, D., Sela, A.: A learning approach to auctions. Journal of Economic Theory 82, 65–88 (1998)
[8] Maynard-Smith, J.: Evolution and the Theory of Games. Cambridge University Press, Cambridge (1982)
[9] Nouyan, S., Groß, R., Bonani, M., Mondada, F., Dorigo, M.: Teamwork in self-organized robot colonies. Transactions on Evolutionary Computation 13(4), 695–711 (2009)
[10] Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems 11(3), 387–434 (2005)
[11] Phelps, S., Marcinkiewicz, M., Parsons, S.: A novel method for automatic strategy acquisition in n-player non-zero-sum games. In: AAMAS 2006: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, Hakodate, Japan, pp. 705–712. ACM, New York (2006)
[12] Shoham, Y., Powers, R., Grenager, T.: If multi-agent learning is the answer, what is the question? Artificial Intelligence 171(7), 365–377 (2007)
[13] Sutton, R., Barto, A.: Reinforcement Learning: An introduction. MIT Press, Cambridge (1998)
[14] Tuyls, K., Parsons, S.: What evolutionary game theory tells us about multiagent learning. Artificial Intelligence 171(7), 406–416 (2007)
[15] Tuyls, K., 't Hoen, P.J., Vanschoenwinkel, B.: An evolutionary dynamical analysis of multi-agent learning in iterated games. Autonomous Agents and Multi-Agent Systems 12, 115–153 (2005)
[16] Weibull, J.W.: Evolutionary Game Theory. MIT Press, Cambridge (1996)

Decentralized Learning in Wireless Sensor Networks

Mihail Mihaylov[1], Karl Tuyls[2], and Ann Nowé[1]

[1] Vrije Universiteit Brussel, Brussels, Belgium
{mike,ann}@como.vub.ac.be
[2] Maastricht University, Maastricht, The Netherlands
ktuyls@gmail.com

Abstract. In this work we present a reinforcement learning algorithm that aims to increase the autonomous lifetime of a Wireless Sensor Network (WSN) and decrease its latency in a decentralized manner. WSNs are collections of sensor nodes that gather environmental data, where the main challenges are the limited power supply of nodes and the need for decentralized control. To overcome these challenges, we make each sensor node adopt an algorithm to optimize the efficiency of a small group of surrounding nodes, so that in the end the performance of the whole system is improved. We compare our approach to conventional ad-hoc networks of different sizes and show that nodes in WSNs are able to develop an energy saving behaviour on their own and significantly reduce network latency, when using our reinforcement learning algorithm.

1 Introduction

An increasingly popular approach for environmental and habitat monitoring is the use of Wireless Sensor Networks (WSNs) [Car04, Rog06, Yic08]. The nodes in such a, WSN are limited in power, processing and communication capabilities, which requires that they optimize their activities, in order to extend the autonomous lifetime of the network and minimize latency. A complicating factor is communication, because some nodes can fall outside the transmission range of the base station, or can belong to different stakeholders, serving various purposes, thus rendering the common centralized approach inapplicable for large networks.

This article extends the work done in [Mih08] to a random network topology, reduces the communication overhead and significantly improves the results. In this work we use a reinforcement learning algorithm to optimize the energy efficiency of a WSN and reduce its latency in a decentralized manner. We achieve that by making nodes (hereby regarded as agents) develop energy-saving schemes by themselves without a central mediator. In many cases such a central authority is undesirable or simply impossible to implement [Rog06]. The idea behind our approach is that agents learn, by themselves, to reduce the negative effect of their actions on other agents in the system, based on a certain reward function. We investigate the performance of our algorithm in two networks of different

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 60–73, 2010.

sizes. We show that when agents learn to optimize their behaviour, they can increase the energy efficiency of the system and significantly decrease its latency with minimal communication overhead.

The outline of this chapter is as follows: Section 2 presents the background of our approach by describing the basics of a wireless sensor network and the MAC communication protocol. Section 3 describes the idea behind our algorithm and its application to the energy efficiency optimization of nodes. In Section 4 we explain the experiments and discuss our findings. Lastly, Section 5 presents our conclusions from this research and suggests some areas for improvement in the future.

2 Background

In this section we describe the basics of a Wireless Sensor Network and the MAC communication protocol. Subsection 2.1 elaborates on WSNs and Subsections 2.2 and 2.3 explain the working of the MAC protocol and the way nodes communicate. Lastly, Subsection 2.4 presents some related work in this field.

2.1 Wireless Sensor Networks

A Wireless Sensor Network is a collection of densely deployed autonomous devices, called sensor nodes, that gather environmental data with the help of sensors. The untethered nodes use radio communication to transmit sensor measurements to a terminal node, called the sink. The sink is the access point of the observer, who is able to process the distributed measurements and obtain useful information about the monitored environment. Sensor nodes communicate over a wireless medium, by using a multi-hop communication protocol that allows data packets to be forwarded by neighbouring nodes to the sink. This concept is illustrated in Figure 1. The environmental or habitat monitoring is usually done over a long period of time, taking into account the latency requirements of the observer.

The WSN can vary in size and topology, according to the purpose it serves. The sensor network is assumed to be homogeneous where nodes share a common communication medium (e.g. air, water, etc.). We further assume that the communication range is equal in size and strength for all nodes. They have a single

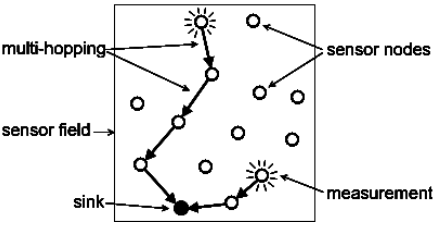


Fig. 1. Wireless Sensor Network

omnidirectional antenna that can only *broadcast* a message, delivering it to all nodes in range. In our network, sensor nodes can neither vary their transmission power, nor are they able to estimate their distance from the transmitting node by measuring the signal strength – such features are not generally available in sensor nodes and therefore are not considered here. The motivation to use such simple devices is to reduce the overall cost of nodes and to keep our solution applicable to the most general sensor network.

In this work we show that the selfish and computationally bounded agents can optimize their own performance, in a decentralized manner, in order to reduce both their own energy consumption and the latency of the network. We assume that communication between the agents is limited and that central control is not possible. We further require that the communication protocol considers not only energy efficiency, but also scalability and fault tolerance, so that our approach is able to adapt to a dynamic topology, where nodes may move, fail or new nodes may be added to the system. The communication protocol, therefore, constitutes an important part of the WSN design.

2.2 The MAC Protocol

The Medium Access Control (MAC) protocol is a data communication protocol, concerned with sharing the wireless transmission medium among the network nodes. Typical MAC protocols, used by ad-hoc networks, cannot be applied to WSNs, due to a number of differences between the two types of networks. Some differences include the large number and density of sensor nodes in a WSN, compared to the nodes in ad-hoc networks; the frequently changing topology of sensor nodes and their power constraints, etc.

We use a simple asynchronous MAC protocol that divides the time into small discrete units, called frames. Each node independently determines its sleep duration (or schedule), i.e. the amount of time in a frame that the node's antenna will be turned off. During that time the agent is not able to communicate with other nodes and therefore saves energy. Nevertheless, the agent continues its sensing and processing tasks. Our protocol allows nodes to synchronize their schedules prior to communication and thus avoid collisions and overhearing – typical sources of energy waste.

Since communication is the most energy expensive action [Dam03], it is clear that in order to save more energy, a node should sleep more. However, when sleeping, the node is not able to send or receive any messages, therefore it increases the latency of the network, i.e., the time it takes for messages to reach the sink. On the other hand, a node does not need to listen to the channel when no messages are being sent, since it loses energy in vain. As a result, nodes should learn on their own the number of time slots they should spend sleeping within a frame. For example, nodes far away from the sink may learn to sleep more, since they will have fewer messages to forward, while nodes close to the sink should learn to listen more, because the workload near the sink is usually heavier. Learning to optimize nodes' own schedules will ensure good energy efficiency of the network, while minimizing the latency. The MAC protocol should therefore support the exchange of

additional information, necessary for the algorithm for optimization. It is clear that the amount of this information within message packets should be kept as little as possible, in order to minimize the energy waste by control packet overhead. A brief description of the communication protocol is presented next.

2.3 Communication and Routing

When the WSN is deployed, nodes first need to determine their hop distance to the sink, i.e. the minimum number of nodes that will have to forward their packets. This is achieved by broadcasting SYNchronization (SYN) packets in the following way: the sink broadcasts a SYN packet, containing a counter, initially set to 0; all receivers set their hop equal to the counter, increment it and broadcast the new SYN packet further on, with a small random delay to avoid collisions. For example, a node right next to the sink will receive a SYN packet with `hop=0` and will broadcast a new one with `hop=1`.

When a node has a message to send[1], it broadcasts a Request To Send (RTS) packet to all nodes within range, which we call neighbours (or neighbouring nodes). All neighbours at an equal or higher hop simply go to sleep, since they do not need to forward the sender's message. All lower-hop neighbours wait a small random amount of time before replying with a Clear To Send (CTS) packet. Once one node broadcasts a CTS packet, all its neighbours go to sleep, except the sender of the RTS, who in turn broadcasts the actual data. In other words, all immediate neighbours of the two communication partners are sleeping during the broadcast of the data, in order to avoid collisions and overhearing. Once the receiver obtains the data packet, it replies with an ACKnowledgment (ACK) and thus the communication is over.

2.4 Related Work

A good literature survey on various aspects of Wireless Sensor Networks is presented in [Yic08]. The authors of [Ai04] present a method that adapts the duty cycle scheme of nodes to the traffic pattern of the network in order to minimize latency and maximize throughput. They, however, use a synchronous protocol (AC-MAC). In [Bar05] a new MAC protocol is presented (μ-MAC) with high sleep ratios while keeping latency and reliability at acceptable levels. This protocol, however, uses synchronous schedules, relies on relatively static environment and requires prior knowledge of the traffic pattern. An asynchronous protocol (X-MAC) is presented in [Bue06] that optimizes the energy efficiency of nodes, but not the latency of the network. A decentralized coordination protocol is presented in [Far08], that maximizes the social welfare within a group of interacting agents through local message passing. This approach, however, comes at a higher communication cost. A somewhat different application of decentralized learning is presented in [Jai09], where the authors use distributed constraint optimization techniques to make mobile wireless sensors maximize the sum of signal strengths on all network links over time.

[1] We assume that all messages are forwarded toward the sink.

3 Learning Algorithm

Besides on its hardware, the energy consumption of a node is also dependent on its position in the WSN. Nodes, closer to the sink have to forward more messages and therefore need to listen more, while those far away from the sink could spend more time sleeping. For this reason, the behaviour of agents cannot be the same for all (e.g. all listen and sleep the same amount of time in a frame). Each node needs to *learn* what behaviour is energy efficient in the network. To achieve that, we make nodes adopt an algorithm for optimization in order to improve the performance of the whole system.

Each agent in the WSN uses a reinforcement learning (RL) algorithm to learn an optimal schedule (i.e. sleep duration in a frame) that will maximize the energy efficiency and minimize the latency of the system in a decentralized manner. The main challenge in such a decentralized approach is to define a suitable reward function for the individual agents that will lead to an effective emergent behaviour as a group. Another challenge is that agents in a WSN can obtain only local information from surrounding nodes, due to their small transmission range. To tackle these challenges, we proceed with the definition of the basic components of the reinforcement learning algorithm.

3.1 Actions

The actions of each agent are restricted to selecting a sleep duration for a frame. The action space consists of a discrete number of sleep durations at equal increments within one frame length. Defining the size of the increment constitutes a tradeoff, since a rather large value will result in only few actions for the agent to choose. On the other hand, a small increment will result in a large action set, which makes it difficult for the algorithm to converge [Len08]. Agents choose their actions according to a probability distribution and use that action for a certain number of frames, which we call a frame window. The reason for using an action for more than one frame is that the agent will thus have enough time to experience the effect of that action on the system. The size of the frame window and the discretization increment will be discussed in Section 4.1.

3.2 Rewards

Before proceeding with the formulation of the reward signal, we first need to define what Energy Efficiency (EE) of a single agent is.

Energy Efficiency. We consider an agent to be energy efficient when it minimizes most of the major sources of energy waste in WSN communication – idle listening, overhearing and unsuccessful transmissions, while quickly forwarding any packets in its queue to ensure low network latency. Formally, the energy efficiency for agent i in frame f is:

$$\begin{aligned} EE_{i,f} = \quad & \alpha(1 - IL_{i,f}) + \beta(1 - OH_{i,f}) + \\ & + \gamma(1 - UT_{i,f}) + \delta(1 - DQ_{i,f}) + \epsilon BL_i \end{aligned}$$

where:

- $IL_{i,f}$ is the duration of idle listening of agent i within frame f;
- $OH_{i,f}$ is the duration of overhearing of agent i within frame f;
- $UT_{i,f}$ is the amount of unsuccessful transmissions of agent i within frame f;
- $DQ_{i,f}$ is the sum of the durations that each packet spent in the queue of agent i within frame f;
- BL_i is the remaining battery life of agent i;
- the constants α, β, γ, δ and ϵ weight the different terms accordingly.

All values are in the unit interval.

It is easy to show that if agents try to increase simply their own energy efficiency, they will prefer to sleep until they obtain a measurement (thus minimizing energy waste) and then wake up only to broadcast it (to ensure low latency). That will not lead to high global efficiency, due to the high number of collisions and unsuccessful transmissions that nodes will experience. Therefore, individual agents should also consider other agents in the system when optimizing their own behaviour. A similar approach was undertaken by Wolpert and Tumer in [Wol08], where they apply their Collective Intelligence framework to align the selfish agents' goals with the system goal.

Effect Set. Our hypothesis is that if each agent "cares about others" that will improve the performance of the whole system. To achieve that, we introduce the concept of an Effect Set (ES) of a node, which is the subset of that node's neighbourhood, with which it communicates within a frame window. In other words, the ES of agent i is the set N_i of nodes, whose messages agent i (over)hears within a frame window. Thus, the energy efficiency of agent i is directly dependent on the actions of all agents in N_i and vice versa.

Effect Set Energy Efficiency. As a result of the influence of agents on each other's performance, we form our hypothesis: if each agent seeks to increase not only its own efficiency, but also the efficiency of its ES, this will lead to higher energy efficiency of the whole system. For this reason, we set the reward signal of each agent to be equal to its mean Effect Set Energy Efficiency (ESEE) over a frame window of size $|F|$. We define the ESEE of agent i in the frame window F as

$$\mathrm{ESEE}_{i,F} = \frac{1}{|F|} \cdot \sum_{f}^{F} \frac{\mathrm{EE}_{i,f} + \sum_j \mathrm{EE}_{j,f}}{|N_i| + 1} \qquad \forall j \in N_i$$

where $EE_{i,f}$ is the energy efficiency of agent i in frame f and $|N_i|$ is the number of agents in the effect set of agent i. In other words, the reward signal that each agent receives at the end of each frame window is the mean energy efficiency of its effect set and of itself, averaged over the size of the frame window. Thus, agents will try to increase the value of their ESEE by optimizing their own behaviour.

Challenge. One challenge in our reward signal is that nodes cannot compute their ESEE directly, because to do so, they would have to obtain the efficiency

of each agent in N_i. To achieve that, nodes simply include the value of their own EE in the three control packets – RTS, CTS and ACK, so that neighbouring agents can (over)hear these values and compute their ESEE. This is the only information that nodes need to exchange for our algorithm to work. Although including additional information in control packets is expensive, we will show that the network performs still better than one without learning. We will now show how each agent can learn to optimize its ESEE.

3.3 Update Rule

At the end of each frame window, agents compute the average ESEE from the past frames and use this value to learn the best sleep duration that will maximize efficiency and minimize latency. Agents use the update rules of a classical learning automaton to update their action probabilities. More specifically, after executing action x in every frame of F, its probability $p_i(x)$ is updated in the following way

$$p_i(x) \leftarrow p_i(x) + \lambda \cdot ESEE_{i,F} \cdot (1.0 - p_i(x))$$

where λ is a user-defined learning rate. The probability $p_i(y)$ for all other actions $y \neq x$ in the action set of agent i then becomes

$$p_i(y) \leftarrow p_i(y) - \lambda \cdot ESEE_{i,F} \cdot p_i(y) \qquad \forall y \neq x$$

At the beginning of each frame, agents select their actions according to the updated probabilities and execute them in that frame window. As a result, the learning process is done on-line – the algorithm adapts to the topology of the network and the traffic pattern, which typically cannot be known in advance in order to train nodes off-line.

3.4 Discussion

Although a comparison with other existing learning approaches would provide good insights into coping with decentralized learning, a comparative study is not the topic of this paper. Rather, our aim is to show whether an efficient decentralized learning *can* be achieved by selfish and computationally bounded agents. The motivation behind the selection of our learning approach is twofold. Firstly, learning automata has very appealing theoretical guarantees for convergence [Tha04] and has been successfully applied in different multi-agent settings [Ver04, Tuy06, Vra08]. In other words, the latter property of learning automata suggests that the policy of our learning agents, as a team, will converge to an equilibrium. Secondly, this policy iteration technique uses an implicit exploration strategy that is "built-in" the algorithm itself, i.e. it does not require an additional aspect that needs to be tuned.

4 Results

4.1 Experimental Setup

We applied our algorithm on two networks of random topology and different sizes – one small network with 10 nodes and a large one with 50 nodes. The density of

both networks was the same, i.e. on average each node had 4 neighbours, because we found out empirically that it influences the speed of learning. In this work we focus on how well learning scales in terms of the number of nodes, rather than in terms of the density. The reason for the slower learning in more dense networks is the higher degree of interdependence of the actions of neighbouring agents. In other words, agents in dense networks have to consider more neighbours in optimizing the performance of their ESEE and thus converge to an optimal action slower than agents in less dense networks. An in-depth study of the optimal density of sensor networks is presented in [Ess08].

We considered networks of random topology, rather than organized in a grid structure (as in [Mih08]), so that the WSN can be deployed more freely (e.g. nodes can be scattered from a moving vehicle). The synchronization phase of the network was set to 20 seconds – this duration was enough for all nodes to find their hop distance to the sink in both networks. During this phase, agents do not learn to optimize their behaviour, since the resulting traffic pattern is independent of that from the actual data. We set the duration of a frame to 0.5 seconds and the message rate – to 1 sensor measurement in a frame on average. We chose this high message rate to make the effect of agents' actions more apparent and to give agents enough information in order to learn a good policy. A sufficient frame window size was found to be 4, i.e. agents repeat their selected action for 4 times, before obtaining a reward signal. The discretization coefficient (Subsection 3.1) was selected such that it results in 11 different actions (or sleep durations). The five terms in the computation of the EE (Subsection 3.2) are from a network perspective interdependent (yet not redundant) and their weighting coefficients are up to the network designer to set. We carried out an exhaustive brute-force search to determine a set of values that is most suitable for our network settings. Thus, the five weighting coefficients were empirically chosen in the following way: $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.1$, $\delta = 0.3$ and $\epsilon = 0.1$. The same brute-force search was carried out to determine the best learning rate. Small coefficient results in a relatively slow learning process, while large learning rates make the network unstable. Thus, the best learning rate λ was found to be 0.280 for the small network and 0.299 for the large one, where in both cases the initial action probability was uniform. Finally, the networks were allowed to run for 500 seconds, i.e. 1000 frames, before the simulation was terminated. This duration was sufficient to produce valuable results.

4.2 Experiments

As stated above, we evaluated our algorithm on two random topology networks of the same density, but of different sizes. We compared the performance of each setting to a network of the same size where agents do not optimize their behaviour, but rather all sleep the same pre-defined amount of time, as it is usually done in practice [Mar04]. In each experiment we measured six performance criteria:

1. Average **remaining battery** at the end of the simulation (i.e. after 1000 frames). This value shows what the battery levels of nodes will be after 500 seconds of runtime with the selected settings.

2. **Standard deviation of the average remaining battery** – indicates the difference between the most and the least efficient nodes. Here a small deviation is desirable, since it signifies a rather equal dissipation of energy over time.
3. **Average latency** of the network over all packets delivered to the sink. This criterion measures the average time a message takes from the moment it was generated to the time it reaches the sink.
4. **Standard deviation of the average latency** of the network. Again, a small deviation is preferable, because it signifies consistent traffic latency.
5. **Maximum latency** of the network, i.e. the latency of the packet that took the most time to be delivered to the sink. This value indicates the worst case scenario for the latency that the user of the WSN can experience for a packet.
6. Number of **received packets** by the sink within 500 seconds. This is an inverse indication of latency and it shows how many messages actually reached the sink during the simulation runtime.

The sleep duration of the two non-learning networks was selected such that it maximizes the above six performance criteria. The same technique was used to select the best learning rate of the networks with optimization. In other words we compared the optimal "non-learning" system to the optimal one *with* learning. This comparison is displayed in Figure 2. The first column shows the above six performance criteria, where the last two rows indicate the average sleeping time of the agents and the standard deviation. The second column indicates the objective (*obj.*) of the corresponding performance criterion – whether it should be maximized (*max*) or minimized (*min*). The third and forth column display the results from our experiments when agents are not learning and when they are learning, respectively. The column labeled *improvement* displays the percentage increase of the six performance measures when agents adopt our learning algorithm.[2]

As it can be seen from Figure 2, in both cases our learning agents sleep on average less than those in the non-learning network. One would expect that less sleeping results in lower battery level, due to idle listening and overhearing, and higher latency, due to collisions. However, our learning algorithm aims to reduce precisely those sources of energy waste, by making nodes optimize their behaviour, based on the actions of neighbouring nodes. Thus, agents learn to avoid "harming" other agents by adapting to the traffic pattern and therefore learning the optimal sleep duration in their neighbourhood. In other words, agents learn to sleep when their neighbours communicate (so as to avoid overhearing); stay awake enough to forward messages quickly (and thus decrease latency); and yet sleep enough (to ensure longer network lifetime). Figure 3 shows agents' actions (sleep durations) (vertical axis) over time (horizontal axis). Each dot represents that agent's selected action at the corresponding time in the simulation. The graph indicates that in the small network agents learn, as the time progresses, to sleep less and listen more, so that they reduce the latency of the network,

[2] The concept of "improvement" is not applicable to the last two rows.

Small Network (10 nodes)

performance criteria	obj	non-learning	learning	improvement
End battery - mean (%)	max	23.283	25.706	10.4% (increased)
End battery - std. dev. (%)	min	4.514	2.220	50.8% (decreased)
Latency - mean (sec.)	min	11.413	3.937	65.5% (decreased)
Latency - std. dev. (sec.)	min	8.455	3.348	60.4% (decreased)
Latency - max (sec.)	min	62.359	18.975	69.6% (decreased)
Packets arrived at Sink	max	2007	2167	8.0% (increased)
Sleeping time - mean (sec.)	n/a	0.120	0.094	n/a
Sleeping time - std. dev. (sec.)	n/a	0.000	0.136	n/a

Large Network (50 nodes)

performance criteria	obj	non-learning	learning	improvement
End battery - mean (%)	max	22.375	22.789	1.9% (increased)
End battery - std. dev. (%)	min	4.362	5.251	20.4% (increased)
Latency - mean (sec.)	min	20.552	5.823	71.7% (decreased)
Latency - std. dev. (sec.)	min	14.768	5.850	60.4% (decreased)
Latency - max (sec.)	min	88.669	50.892	42.6% (decreased)
Packets arrived at Sink	max	544	2296	322.1% (increased)
Sleeping time - mean (sec.)	n/a	0.220	0.166	n/a
Sleeping time - std. dev. (sec.)	n/a	0.000	0.176	n/a

Fig. 2. Comparison between non-learning and learning in the small and large networks

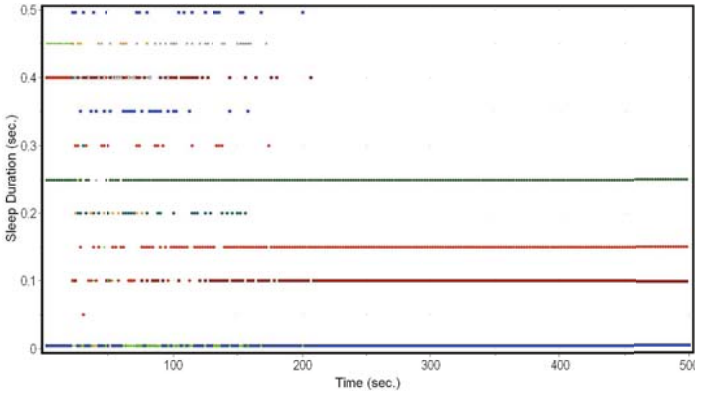
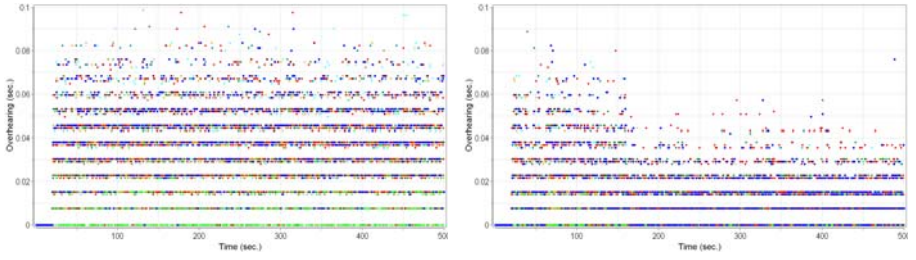


Fig. 3. Sleep Duration over Time when learning, Small Network (10 nodes)

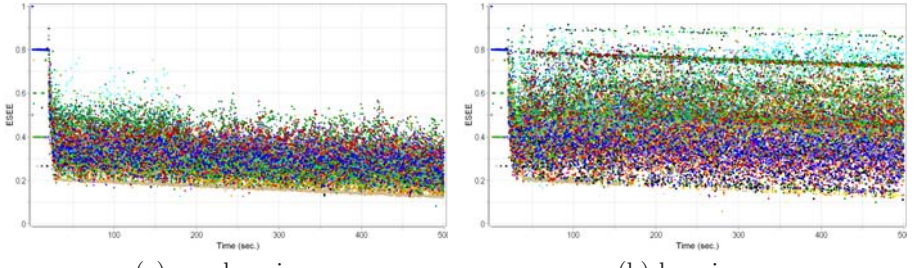
while increasing its lifetime.[3] The figure also shows that in the beginning of the simulation agents explore their action set and after approximately 200 seconds, the policy of all agents converges to an optimal action. In other words, after 400 frames, each agent finds the sleep duration that maximizes its ESEE and then sticks to it. The effect of adapting to the traffic pattern is even more

[3] Due to the discrete values in this graph, some values overlap and thus not all of them can be displayed at the same time.



(a) non-learning (b) learning

Fig. 4. Overhearing duration over Time, Small Network (10 nodes)



(a) non-learning (b) learning

Fig. 5. Effect Set Energy Efficiency over Time, Large Network (50 nodes)

apparent in the large network, where agents are able to decrease the average latency with over 70%, resulting in three times more packets delivered to the sink (cf. Figure 2).

Figure 4 compares the overhearing duration of nodes over time in the small network when all agents sleep the same amount of time (4(a)) and when they learn their optimal sleep duration (4(b)). Each dot represents that agent's overhearing duration within a frame at the corresponding time in the simulation. It is evident that when learning, agents reduce this source of energy waste, resulting in higher end battery level.[4] In other words, as the time progresses, agents learn to sleep when their neighbours are communicating, in order to reduce the amount of packets they overhear. This is evident from the fewer dots in Figure 4(b). As a consequence of the convergence to an optimal policy (explained above), one can see a large reduction in overhearing duration after approximately 200 seconds of network runtime. However, we did not measure significant decrease in the overhearing duration of the large network, as it can be predicted from Figure 2. The end battery level of the large network increased with only 2%. This was a result of the large number of nodes and consequently

[4] The discrete steps in the graph are a result of the fixed control and data packet lengths that nodes overhear.

the time they need to find an optimal action. Nevertheless, our learning agents had higher overall efficiency, due to the lower amount of unsuccessful transmission and the shorter stay of packets in the queues of the nodes.

The improved ESEE of agents in the large network can be seen in Figure 5(b), as compared to their non-learning counterparts (5(a)). Each dot represents that agent's ESEE within a frame window at the corresponding time in the simulation. In other words, the graph shows the relative energy efficiency of each node's neighbourhood over time. Although the efficiency of the worst performing nodes is comparable, the average ESEE of the learning agents is higher, than that of the non-learning nodes. This means that when using our algorithm for optimization, on average agents are more energy efficient than when they are not learning. The mean ESEE of both graphs, however, is constantly decreasing, since the remaining battery level of nodes is included in this reward signal (cf. Subsection 3.2). In other words, since battery level is inevitably decreasing, so is the ESEE of both networks.

5 Conclusion

In this article we used a reinforcement learning algorithm to improve the performance of Wireless Sensor Networks (WSN) in a decentralized manner, in order to prolong the autonomous lifetime of the network and reduce its latency. We were able to show that when agents in a WSN use an algorithm for optimization, they can learn to reduce the negative effect of their actions on other agents in the system, without a central mediator. Our results indicate that both in a small and large network, agents can learn to optimize their behaviour in order to increase the energy efficiency of the system and significantly decrease its latency with minimal communication overhead. Our results outperformed a conventional ad-hoc network, where all agents equally listen and sleep for a pre-defined amount of time. Thus, based on our experiments we can conclude that it is more beneficial for the sensor network when nodes *learn* what actions to take, rather than follow a pre-defined schedule. In our algorithm each node seeks to improve not only its own efficiency, but also the efficiency of its neighbourhood, which ensures that the agents' goal is aligned with the system goal of higher energy efficiency and lower latency.

Based on our empirical data, we can also generalize that a crucial point in achieving global efficiency in decentralized learning is aligning the agents' goals with the system goal. Letting each agent selfishly pursue its own objectives may simply lead to a suboptimal solution. In contrast, we were able to achieve global efficiency by making each agent consider a small group of surrounding agents. Thus, the objectives of each agent become identical with the goals of the "team" and therefore – of the whole system.

We are currently focusing on comparing the performance of our algorithm to the X-MAC protocol [Bue06], which aims to increase energy efficiency in a decentralized way without any communication overhead. Additionally, we aim to extend our approach, presented in this work, to make it suitable for a larger

set of WSN applications, where the network will adapt to the latency requirement of the user directly. We are also aiming towards more sophisticated parameter studies (e.g. genetic algorithms) to ensure a (nearly) optimal parameter setting.

Future work involves computing the energy requirements of the algorithm itself and experimenting with different network topologies and reward functions to obtain a yet bigger improvement in energy efficiency and latency.

References

[Car04] Carle, J., Simplot-Ryl, D.: Energy-Efficient Area Monitoring for Sensor Networks. IEEE Computer Society 47, 40–46 (2004)

[Rog06] Rogers, A., Dash, R.K., Jennings, N.R., Reece, S., Roberts, S.: Computational Mechanism Design for Information Fusion within Sensor Networks. In: 9th FUSION (2006)

[Mih08] Mihaylov, M., Nowé, A., Tuyls, K.: Collective IntelligentWireless Sensor Networks. In: Proc. of the 20th BNAIC, pp. 169–176 (2008)

[Dam03] van Dam, T., Langendoen, K.: An Adaptive Energy-Efficient Mac Protocol For Wireless Sensor Networks. In: Proceedings of The 1st SenSys, pp. 171–180 (2003)

[Yic08] Yick, J., Mukherjee, B., Ghosal, D.: Wireless Sensor Network Survey. Computer Networks 52, 2292–2330 (2008)

[Ai04] Ai, J., Kong, J., Turgut, D.: An adaptive coordinated medium access control for wireless sensor networks. In: Proceedings of 9th ISCC, vol. 2, pp. 214–219 (2004)

[Bar05] Barroso, A., Roedig, U., Sreenan, C.J.: μ-MAC: An Energy-Efficient Medium Access Control for Wireless Sensor Networks. In: Proceedings of the 2nd EWSN (2005)

[Bue06] Buettner, M., Yee, G., Anderson, E., Han, R.: X-MAC: A Short Preamble MAC Protocol For Duty-Cycled Wireless Sensor Networks. University of Colorado at Boulder (2006)

[Far08] Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th AAMAS, pp. 639–646 (2008)

[Jai09] Jain, M., Taylor, M., Yokoo, M., Tambe, M.: DCOPs Meet the Real World: Exploring Unknown Reward Matrices with Applications to Mobile Sensor Networks. In: Proceedings of the 21st IJCAI (2009)

[Len08] Leng, J.: Reinforcement learning and convergence analysis with applications to agent-based systems. University of South Australia (2008)

[Wol08] Wolpert, D.H., Tumer, K.: An Introduction To Collective Intelligence. NASA Ames Research Center (2008)

[Mar04] Martinez, K., Ong, R., Hart, J.: Glacsweb: a sensor network for hostile environments. In: The 1st IEEE Secon (2004)

[Ess08] Esseghir, M., Bouabdallah, N.: Node density control for maximizing wireless sensor network lifetime. Int. J. Netw. Manag. 18, 159–170 (2008)

[Ver04] Verbeeck, K.: Coordinated Exploration in Multi-Agent Reinforcement Learning. Ph.D Thesis, Computational Modeling Lab, Vrije Universiteit Brussel (2004)

[Vra08] Vrancx, P., Verbeeck, K., Nowe, A.: Decentralized Learning in Markov Games. IEEE Transactions on Systems, Man and Cybernetics 38, 976–981 (2008)

[Tha04] Thathachar, M.A.L., Sastry, P.S.: Networks of learning automata: Techniques for online stochastic optimization. Kluwer Academic Publishers, Dordrecht (2004)

[Tuy06] Tuyls, K., Hoen, P.J., Vanschoenwinkel, B.: An Evolutionary Dynamical Analysis of Multi-Agent Learning in Iterated Games. JAAMAS 12(1), 115–153 (2006)

Recursive Adaptation of Stepsize Parameter for Non-stationary Environments

Itsuki Noda

ITRI, National Institute of Advanced Industrial Science and Technology
i.noda@aist.go.jp

Abstract. In this article, we propose a method to adapt stepsize parameters used in reinforcement learning for non-stationary environments. In general reinforcement learning situations, a stepsize parameter is decreased to zero during learning, because the environment is generally supposed to be noisy but stationary, such that the true expected rewards are fixed. On the other hand, we assume that in the real world, the true expected reward changes over time and hence, the learning agent must adapt the change through continuous learning. We derive the higher-order derivatives of exponential moving average (which is used to estimate the expected values of states or actions in major reinforcement learning methods) using stepsize parameters. We also illustrate a mechanism to calculate these derivatives in a recursive manner. Using the mechanism, we construct a precise and flexible adaptation method for the stepsize parameter in order to optimize a certain criterion, for example, to minimize square errors. The proposed method is validated both theoretically and experimentally.

1 Introduction

In most of the works on reinforcement learning that are used in agent learning, it is supposed that the environment for agents is stationary during and after learning. In other words, while the environment may react to agents' action and provide rewards dynamically, the rules of the change and the mechanisms of rewarding are supposed to be stationary forever. In such a case, it is reasonable that a stepsize parameter α is monotonically decreased to 0 through learning in the following temporal difference(TD) learning algorithm in order to estimate the expected values of the states or actions (Q-value) [1].

$$Q_{t+1}(state_t, act_t) = (1-\alpha)Q_t(state_t, act_t) + \alpha(r_t + \gamma \max_{act'} Q_t(state_{t+1}, act')) \quad (1)$$

By decreasing α sufficiently, we can reduce the noisy factors included in state transitions and rewarding errors. After the Q-values seem to be sufficiently near the true expected values, the agents generally stop learning and behave on the basis of the fixed Q-value. An important assumption here is that the true expected values are constant during and after learning [2].

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 74–90, 2010.

On the other hand, in common real world problems, especially the problems on open and multiagent systems, the environment may change gradually or rapidly. For example, market systems such as the stock and foreign exchange market can be affected by both agents' behavior and various other fundamental conditions. Therefore, it is difficult to suppose that the true expected rewards of states or actions are stationary. Instead, agents in such an environment should continue learning to adapt to changes in the environments. In this case, since we cannot decrease the stepsize parameter α monotonically down to 0, we control it such that it is capable of meeting the changes in the environment.

In order to adapt to such dynamic and non-stationary environments, George and Powell[3] proposed a method, called optimal stepsize algorithm (OSA), to control stepsize parameters in order to minimize noise factors on the basis of the relationships among the stepsize parameter, noise variance, and changes in learning values. Sato and et. al.[4] also proposed a framework to accumulate error variance to find out the suitable learning parameters. In both works, the focus is only on minimizing estimation errors, which the effect of the changes in the stepsize parameter on the learning processes is ignored.

For this issue, we focus on the effects of the changes in the stepsize parameter on the learning process, and extend the learning process to estimate the effects. On the basis of the estimation, we can construct a method to adjust the stepsize parameter in order to optimize a certain criteria, for example, minimizing an error.

2 Exponential Moving Average and Stepsize Parameter

2.1 Exponential Moving Average

In reinforcement learning, for example, TD learning, an agent learns to estimate the expected value of each state or action that is used in decision making according to the rewards that the agent receives as results of his/her action in the unknown environment. Generally, the estimation is done by the following *exponential moving average* (EMA) equation.

$$\tilde{x}_{t+1} = (1 - \alpha)\tilde{x}_t + \alpha x_t, \tag{2}$$

where x_t and $\tilde{x}_t$ are the actual observed value (for example, received reward r_t) and the corresponding expected value, respectively, that are updated through discrete time line t. α is a *stepsize parameter*, which indicates whether the agent regards recent observed values x_t as important, or the agent should take a long-term average so as to calculate the true expected value ($\tilde{x}_t$). Generally, $\tilde{x}_t$ can be interpreted to be an approximation of a moving average of x_t in the following time-window:

$$T = \frac{2}{\alpha} - 1. \tag{3}$$

2.2 Best Follow-Up to Random Walk

Suppose that an observation sequence $\{x_t\}$ consists of a true value sequence $\{s_t\}$ and noise sequence $\{\epsilon_t\}$ as described in the following equation.

$$x_t = s_t + \epsilon_t, \tag{4}$$

where ϵ_t is a random noise with average 0 and standard deviation σ_ϵ, and is independent from s_t. Furthermore, suppose that the true-value sequence $\{s_t\}$ is a random walk sequence as defined by the equation

$$s_{t+1} = s_t + v_t, \tag{5}$$

where v_t is a random value with average 0 and standard deviation σ_v.

In this case, we can derive the following lemma and theorem.

Lemma 1
The mean square error $\mathbf{E}(\delta_t^2) = \mathbf{E}((\tilde{x}_t - x_t)^2)$ *of expected value* $\tilde{x}_t$ *acquired by eq. (2) using observation* x_t *that follows eq. (4) is given by the following equation.*

$$\mathbf{E}(\delta_t^2) = \frac{1}{2-\alpha}(2\sigma_\epsilon^2 + \frac{1}{\alpha}\sigma_v^2). \tag{6}$$

(See section A for the proof.) □

Theorem 1
The stepsize parameter α *that minimizes the mean square error* $\mathbf{E}(\delta_t^2)$ *is given by the following equation.*

$$\alpha = \frac{-\gamma^2 + \sqrt{\gamma^4 + 4\gamma^2}}{2}, \tag{7}$$

where $\gamma = \frac{\sigma_v}{\sigma_\epsilon}$.
(See section B for the proof.) □

The theorem says that, if observed values consist of random walk values and independent random noise, the best stepsize parameter to balance the follow-up to the random walk and smoothening so as to remove the noise factor can be determined by eq. (7).

2.3 Recursive Exponential Moving Average and Higher-Order Partial Derivatives

In order to determine the stepsize parameter using eq. (7), the agent needs to know the standard deviations of the random walk and noise factor. In general, however, in real learning applications, these values are not known or change over time. Therefore, we try to extract the derivatives of the expected value $\tilde{x}_t$ using the stepsize parameter α, and construct a method to adapt α according to a given sequence of observation $\{x_t\}$.

First, we introduce the following *recursive exponential moving average* (REMA) $\xi_t^{\langle k \rangle}$ by applying eq. (2) recursively:

$$\begin{aligned}
\xi_t^{\langle 0 \rangle} &= x_t \\
\xi_{t+1}^{\langle 1 \rangle} &= \tilde{x}_{t+1} = (1-\alpha)\tilde{x}_t + \alpha x_t \\
\xi_{t+1}^{\langle k \rangle} &= \xi_t^{\langle k \rangle} + \alpha(\xi_t^{\langle k-1 \rangle} - \xi_t^{\langle k \rangle}) \\
&= (1-\alpha)\xi_t^{\langle k \rangle} + \alpha \xi_t^{\langle k-1 \rangle} \\
&= \alpha \sum_{\tau=0}^{\infty} (1-\alpha)^{\tau} \xi_{t-\tau}^{\langle k-1 \rangle}. \qquad (8)
\end{aligned}$$

With regard to REMA, we can state the following lemma and theorem.

Lemma 2
The first partial derivative of REMA $\xi_t^{\langle k \rangle}$ by α is given by the following equation:

$$\frac{\partial \xi_t^{\langle k \rangle}}{\partial \alpha} = \frac{k}{\alpha}(\xi_t^{\langle k \rangle} - \xi_t^{\langle k+1 \rangle}). \qquad (9)$$

(See section C for the proof.) □

Theorem 2
The k-th partial derivative of EMA $\tilde{x}_t$ $(= \xi_t^{\langle 1 \rangle})$ is given by the following equation:

$$\frac{\partial^k \tilde{x}_t}{\partial \alpha^k} = (-\alpha)^{-k} k! (\xi_t^{\langle k+1 \rangle} - \xi_t^{\langle k \rangle}). \qquad (10)$$

(See section D for the proof.) □

2.4 Gradient Descent Adaptation of Stepsize Parameter Using Higher-Order Derivatives and REMA

Because theorem 2 provides the derivatives of $\tilde{x}_t$ by α, we can construct algorithms to optimize a certain criterion, for example, mean square errors, by gradient descent/ascent methods. An important aspect of theorem 2 is that it can provide derivatives of any order. Therefore, we can form more precise gradient descent/ascent methods. We refer to such methods that use higher-order derivatives given by REMA as *recursive adaptation of stepsize parameters* (RASP).

Suppose that $\Delta\tilde{x}_t$ is the change in $\tilde{x}_t$ when α changes by $\Delta\alpha$. In this case, $\Delta\tilde{x}_t$ can be represented by Taylor expansion and theorem 2 as follows:

$$\begin{aligned}
\Delta\tilde{x}_t &= \sum_{k=1}^{\infty} \frac{1}{k!} \frac{\partial^k \tilde{x}_t}{\partial \alpha^k} \Delta\alpha^k \\
&= \sum_{k=1}^{\infty} (-1)^k \left(\frac{\Delta\alpha}{\alpha} \right)^k (\xi_t^{\langle k+1 \rangle} - \xi_t^{\langle k \rangle}). \qquad (11)
\end{aligned}$$

Further, generally, $\Delta\xi_t^{\langle k\rangle}$ for any k can be estimated by the first Taylor expansion and lemma 2 as follows:[1]

$$\Delta\xi_t^{\langle k\rangle} = \Delta\alpha \frac{\partial \xi_t^{\langle k\rangle}}{\partial \alpha} \tag{12}$$

$$\simeq k \left(\frac{\Delta\alpha}{\alpha}\right)(\xi_t^{\langle k\rangle} - \xi_t^{\langle k+1\rangle}). \tag{13}$$

These expansions indicate that RASP exhibits the following features.

1. We can approximate the precise changes in the estimation value $\tilde{x}_t$ even for a large $\Delta\alpha$, using higher-order derivatives calculated by REMA. Therefore, we can change α rapidly.
2. We can also calculate $\Delta\xi_t^{\langle k\rangle}$ by a modification of α, using the derivatives of $\xi_t^{\langle k\rangle}$. Therefore, the values of the variables that are affected by the changes in α are kept precise.

Of course, it is impossible to calculate infinite higher-order derivatives. Instead, we can set upper limit of k large enough to achieve the required precision. Because the calculation of REMA itself is very simple, the cost to calculate higher-order derivatives is small.

The following procedure details the use of RASP to minimize the square error between the expected value $\tilde{x}_t$ and the actual observation x_t. (We call this procedure RASP-MSE.)

Initialize: $\forall k \in \{0 \ldots k_{\max} - 1\} : \xi^{\langle k\rangle} \leftarrow x_0$
while forever **do**
  Let x be an observation.
  for $k = k_{\max} - 1$ to 1 **do**
    $\xi^{\langle k\rangle} \leftarrow (1-\alpha)\xi^{\langle k\rangle} + \alpha\xi^{\langle k-1\rangle}$
  end for
  $\xi^{\langle 0\rangle} \leftarrow x$
  $\delta \leftarrow \xi^{\langle 1\rangle} - x$
  Calculate $\frac{\partial \xi^{\langle 1\rangle}}{\partial \alpha}$ by eq. (10).
  for $k = 1$ to $k_{\max} - 1$ **do**
    Calculate $\Delta\xi^{\langle k\rangle}$ by eq. (11) and eq. (13).
    $\xi^{\langle k\rangle} \leftarrow \xi^{\langle k\rangle} + \Delta\xi^{\langle k\rangle}$
  end for
  calculate a new α according to δ and $\frac{\partial \xi^{\langle 1\rangle}}{\partial \alpha}$.
end while

In this procedure, there are several possible ways to decide the value of $\Delta\alpha$. As in a general gradient descent method, in this case, the only restriction is that $\Delta\alpha < 0$ when $\delta\frac{\partial \xi^{\langle 1\rangle}}{\partial \alpha} > 0$, and $\Delta\alpha > 0$ otherwise. In addition, because of the nature of EMA, we should keep the following points in mind.

[1] We can also use a higher-order Taylor expansion to utilize higher-order derivatives as shown in the appendix.

– α should be a real number in $[0, 1]$.
– α should not get too close to 0 because eq. (10) has a singular point at $\alpha = 0$.

Therefore, in the experiments described below, we use the following procedure to decide $\Delta\alpha$.

$$\begin{aligned}
\gamma'_{\text{old}} &\leftarrow \sqrt{\frac{\alpha^2}{1-\alpha}} \\
\lambda &\leftarrow -\bar{\lambda} \cdot \text{sign}(\delta \frac{\partial \xi^{\langle 1 \rangle}}{\partial \alpha}) \\
\gamma'_{\text{new}} &\leftarrow \exp(\log(\gamma'_{\text{old}}) + \lambda) \\
\alpha_{\text{new}} &\leftarrow \frac{-\gamma'^2_{\text{new}} + \sqrt{\gamma'^4_{\text{new}} + 4\gamma'^2_{\text{new}}}}{2} \\
\Delta\alpha &\leftarrow \alpha_{\text{new}} - \alpha
\end{aligned}$$

In this procedure, α is modified according to the uniformed-step changes in the logarithmic value of γ in eq. (7). Therefore, the changes in α are large when α is around 0.5, and small when α is close to 0 or 1.

3 Experiments

3.1 Exp.1: Learning Best α for Noise Reduction

In the first experiment, we show that the above procedure to adapt α yields the best stepsize parameter value for noise reduction that is determined by eq. (7).

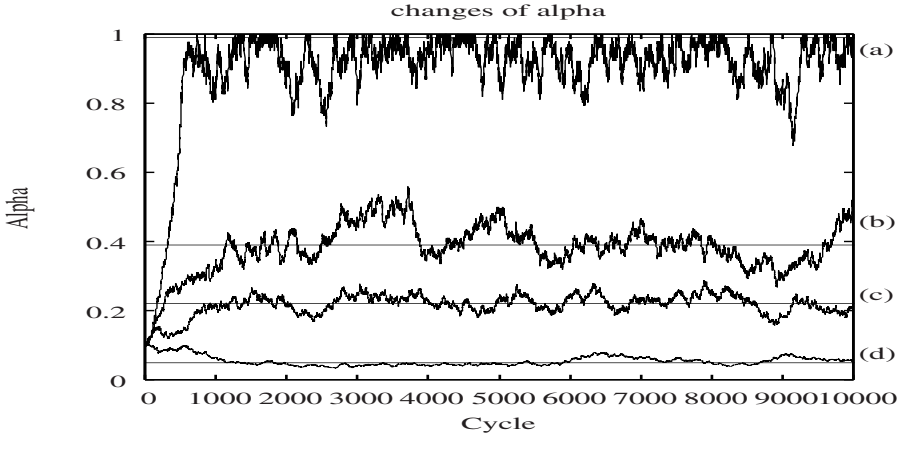


Fig. 1. Exp.1: Changes in α through the Learning of Observed Value Using the Various Ratios of Standard Deviations of Random Walk and Noise (γ)

Figure 1 shows the results of the adaptation of α through the learning of observation sequences $\{x_t\}$ with various γ (the ratio of standard deviations of random walk and noise). In each case of this experiment, we use the following standard deviations of random walk and noise.

	σ_s	σ_ϵ	(γ)	(α_{best})
(a)	0.01	0.001	10.00	0.990
(b)	0.01	0.020	0.50	0.390
(c)	0.01	0.040	0.25	0.221
(d)	0.01	0.200	0.05	0.048

Each curve in the graph of Figure 1 shows the changes in α through the learning of expected value $\tilde{x}_t$ by eq. (2) and adaptation of α by RASP-MSE. The horizontal axis in the graphs indicates the learning (and adaptation) cycle, while the vertical axis represents the value of α. Further, the horizontal line in each graph indicates the best stepsize parameter (α_{best}) as calculated by eq. (7). As shown in these graphs, α approaches the best value and is then consistent through learning. Note that α does not converge to the best value because of the noise factors added in the observed value. Fortunately, the perturbation is large only when α is relatively large; in this case, the effect of α changes slowly, so that the behavior of the learning does not change drastically even α changes with a large step.

Figure 2 shows the changes in the expected value $\tilde{x}_t$ as calculated by EMA. In the figure, (a) and (b) are the cases where the parameter γ is almost equal to 1.0 and 0.1, respectively. In other words, (a) is the case where the standard deviation of the true random walk value s_t exceeds the standard deviation of noise ϵ_t sufficiently, and (b) is the case where the noise factor is larger than the random walk. Graphs (a-1) and (a-2) show detailed close-ups of the changes in (a) at the early and mature stages[2] of learning, respectively. Similarly, (b-1) shows the detailed changes taking place at the mature stage of learning in (b) in detail. In these graphs, (a-1) shows that the expected value $\tilde{x}_t$ can not follow the quick changes in the true value s_t but does smoothen the changes. This is so as α is still too small at the early stage of learning. On the other hand, in (a-2), α is adapted to be suitable, and consequently, $\tilde{x}_t$ can follow s_t with minimum delay. In (b) and (b-1), α is kept small enough to reduce the large noise factor and allow $\tilde{x}_t$ to yield the best estimate of s_t.

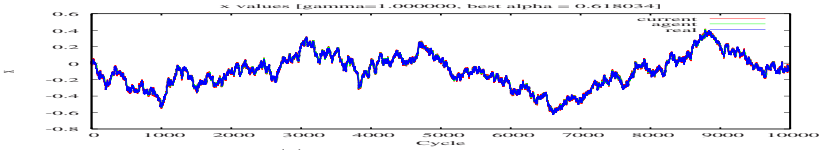
As shown in these results, RASP-MSE can acquire the suitable stepsize parameter α for a given sequence.

3.2 Exp.2: The Case of Square-Waved γ

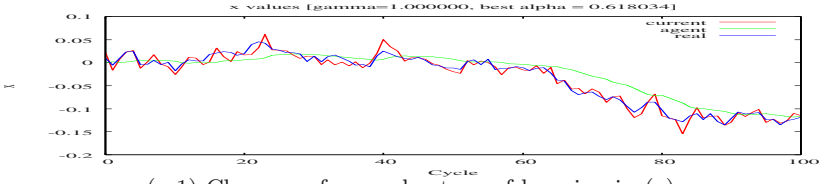
In order to show how RASP-MSE can follow the changes in the environment, we conducted an experiment in which γ changes along a square wave over time.

Figure 3 shows the result of an experiment to adapt α by RASP-MSE in the EMA learning of $\tilde{x}_t$ when γ alternates between 0.5 and 0.0005 every 1000

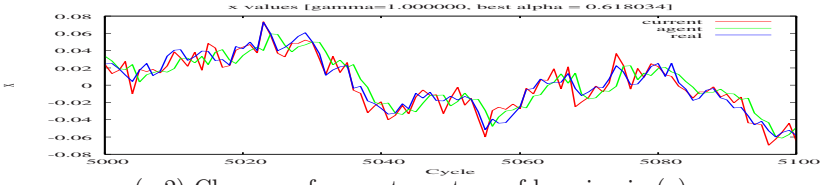
[2] Here, "mature" stage implies a phase when the learning is almost complete and α is close enough to the optimal value.



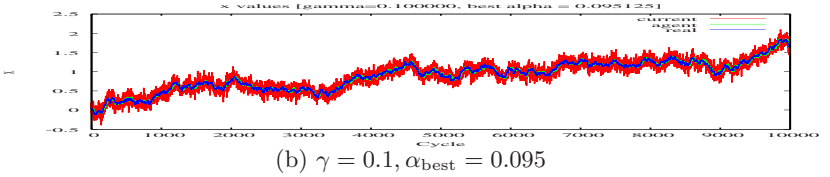
(a) $\gamma = 1.0, \alpha_{\text{best}} = 0.618$



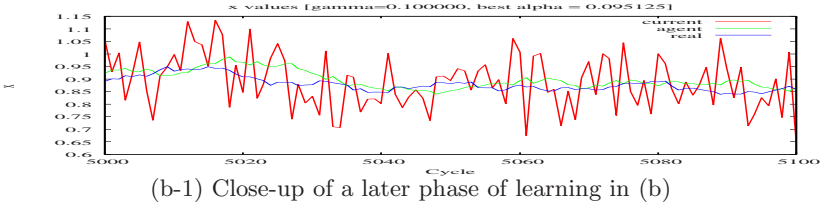
(a-1) Close-up of an early stage of learning in (a)
alpha has not adapted yet.



(a-2) Close-up of an mature stage of learning in (a)
alpha has already adapted.



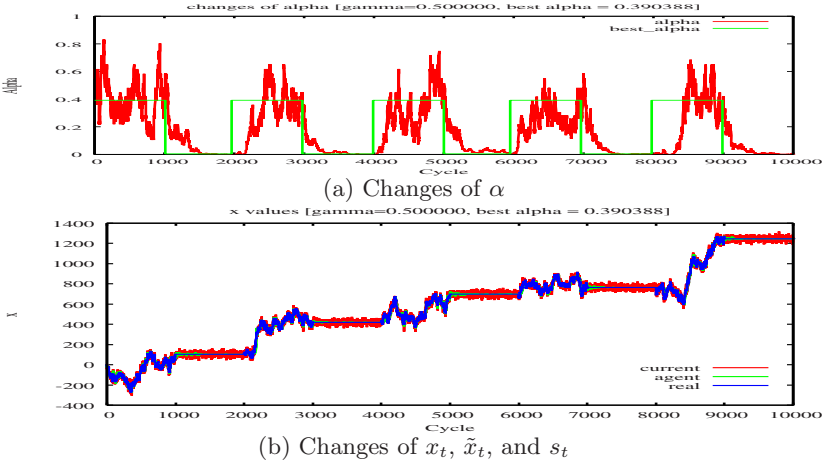
(b) $\gamma = 0.1, \alpha_{\text{best}} = 0.095$



(b-1) Close-up of a later phase of learning in (b)

Fig. 2. Exp.1: Changes in the Expected Value $\tilde{x}_t$

steps. The graph in (a) shows the changes in α through learning (a curve) along with the ideal changes expected according to γ (a square wave). The top and the bottom of the square wave are 0.39 and 0.0005, respectively. As shown in this graph, α tries to follow the changes in γ. Graph (b) shows the changes in the expected value $\tilde{x}_t$, observed value x_t, and true value s_t. During the period when γ is small (0.0005, where α's ideal value is 0.0005), $\tilde{x}_t$ becomes a type of long-term moving average so as to reduce the large noise factor: On the other hand, $\tilde{x}_t$ follows x_t tightly during the period when γ is large (0.5, where α's ideal value is 0.39). This result shows that RASP-MSE can follow the changes in the environment and determine a suitable stepsize parameters.



(a) Changes of α

(b) Changes of x_t, $\tilde{x}_t$, and s_t

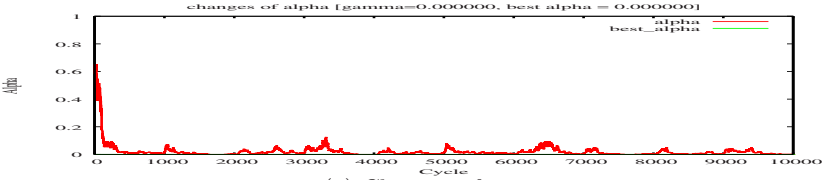
Fig. 3. Exp.2: Square-waved γ

3.3 Exp.3: Square-Waved True Value

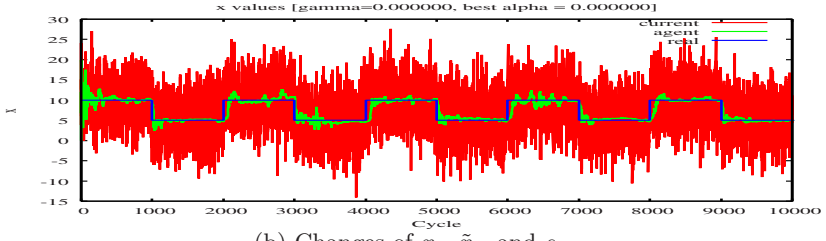
EMA is used in general reinforcement learning, for example, eq. (1), because it can reduce noise and yield a value that approaches the stationary true value. In the second experiment, we suppose that the true value is almost stationary but does change occasionally. In such a case, the learning mechanism needs to detect the changes in the true value. In the actual experiment, we use a sequence of true values $\{s_t\}$ that follows a square wave over time.

Figure 4 shows the result of an experiment to adapt α by RASP-MSE in the EMA learning of $\tilde{x}_t$ when the true value s_t alternates between 0.0 and 0.5 every 1000 steps. In this experiment the standard deviation of noise ϵ_t is 5.0. (a) shows the changes in α, and (b), in x_t, $\tilde{x}_t$, and s_t through learning. (c) shows a result of the case that we apply OSA [3] to the same problem for the comparison.

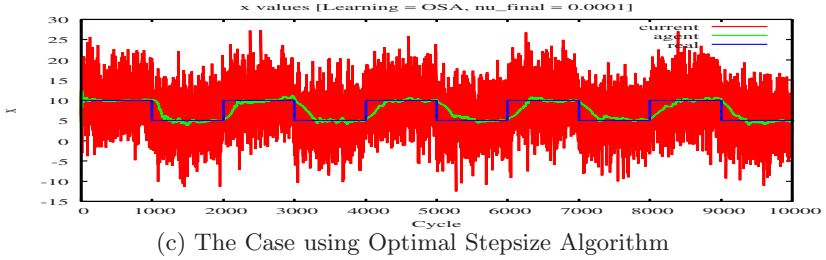
(b) indicates that RASP-EMA reduces the large noise factor and at the same time can follow the changes in the true value. Compared with (c), we found that following the true value is more precise by RASP-EMA than by OSA. Actually, the average square error of $\tilde{x}_t$ from x_t in (b) is 1.192, while the error in (c) is 2.496. Corresponding changes in α in (a) shows that α approaches zero almost all the times but is relatively large at the time when the true value s_t changes ($t = 1000, 2000, \ldots$). From the meaning of α in EMA ($\tilde{x}_t$ follows the previous observed value x_t when α is large, and $\tilde{x}_t$ becomes a long-term moving average of x_t when α is small), the change in α shown in (a) indicates that RASP-MSE detects the timing of changes in s_t and lets an agent regard the recent observation as plausible: On the other hand, RASP-MSE lets the agent use the long-term smoothed value when the environment is stationary. In other words, RASP-MSE can control the features of learning by EMA in accordance with the changes in the environment.



(a) Changes of α



(b) Changes of x_t, $\tilde{x}_t$, and s_t.



(c) The Case using Optimal Stepsize Algorithm

Fig. 4. Exp.3: Learning Square-waved True Value s_t

Limitations of RASP-MSE with Regard to Square-Waved True Value Sequences. Exp.3, described in section 3.3, shows the ability of RASP-MSE to follow a square-waved true value sequence. However, the proposed procedure is not able to follow all square waves. For example, if the observed value x_t includes noise with standard deviation 30.0 instead of 5.0, the RASP-MSE does not suitably follow the change in the true value s_t but regards the change as noise (figure 5). In the graph, $\alpha \cong 0$ during steps 3000–7000 steps. This implies that $\tilde{x}_t$ becomes a long-term moving average of the observation, where the term of the average is longer than the cycles of changes in the true value.

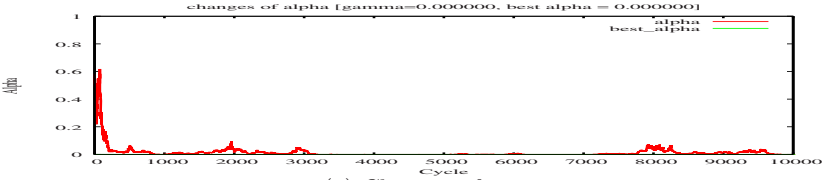
We can derive the theoretical upper-limit of the adaptation to the changes in the small square-waved true value as follows:

Suppose that the true value s_t changes according to the following formula:

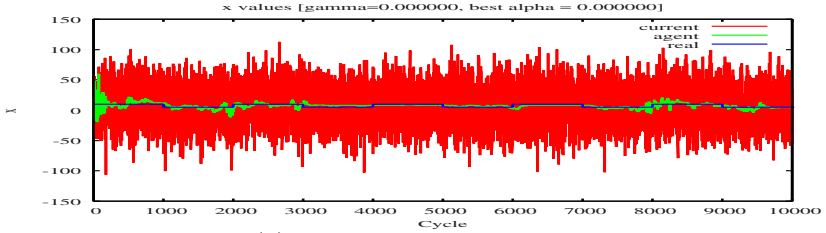
$$s_t = \begin{cases} -\delta & : \quad (2n-1)T \leq t < 2nT \\ \delta & : \quad 2nT \leq t < (2n+1)T \end{cases},$$

where $2T$ is a cycle of square-waved changes in the true value.

If α is almost zero such that $\tilde{x}_t$ represents a long-term moving average of x_t, the mean square error E_0 of the expectation is as follows:



(a) Changes of α



(b) Changes of x_t, $\tilde{x}_t$, and s_t.

Fig. 5. Exp.3-2: Learning in the Case of Small Square-waved True Value s_t

$$\begin{aligned} E_0 &= \mathbf{E}((x_t - \mathbf{E}(x_t))^2) \\ &= \delta^2 + \sigma_\epsilon^2. \end{aligned}$$

On the other hand, suppose that we can control α optimally, that is, α is raised to 1 at $t = nT$, and is decayed to be the value $1/(1 + t - nT)$ otherwise. In this case, $\tilde{x}_t$ becomes an average of x_t during each half cycle. Therefore, the mean square error E_{opt} is as follows:

$$\begin{aligned} E_{opt} &= \mathbf{E}((x_t - \tilde{x}_t)^2) \\ &= \frac{1}{T}\left[4\delta^2 + \sigma_\epsilon^2 + \sum_{\tau=1}^{T}(\sigma_\epsilon^2 + \frac{\sigma_\epsilon^2}{\tau})\right] \\ &= \frac{1}{T}\left[4\delta^2 + T\sigma_\epsilon^2 + \sigma_\epsilon^2 \sum_{\tau=1}^{T}\frac{1}{\tau}\right] \\ &= \frac{1}{T}\left[4\delta^2 + T\sigma_\epsilon^2 + \sigma_\epsilon^2 \mathcal{H}_T, \right], \end{aligned}$$

where $\mathcal{H}_T = \sum_{\tau=1}^{T} \frac{1}{\tau}$ is a harmonic series.

If $E_0 < E_{opt}$, we obtain the following inequality:

$$(T-4)\delta^2 < \mathcal{H}_T \sigma_\epsilon^2. \tag{14}$$

This is satisfied when $T \leq 4$. This implies that it is impossible to follow this quick changes ($T \leq 4$) by the proposed procedure, because the long-term average (the case of $\alpha \sim 0$) provides better estimation than the expectation by EMA with adaptive α.

In the case of $T > 4$, eq. (14) can be written as follows:

$$\frac{\delta^2}{\sigma_\epsilon^2} < \frac{\mathcal{H}_T}{T-4}. \tag{15}$$

This inequality shows the limitation of EMA with adaptive stepsize parameters: When the changes in the true value (δ) is small, the noise (σ_ϵ) is large, and/or the interval time (T) is short as shown in eq. (15), then it is impossible to follow the true value by EMA.

Consider the case of the experiment shown in figure 5, where $\delta = 5/2$, $\sigma_\epsilon = 30.0$, and $T = 1000$. Therefore, the left and right hand sides of eq. (15) are 0.0833 and 0.0867, respectively. This means that the condition of this experiment is beyond the scope of the EMA shown by eq. (15). This is the reason why RASP-MSE failed to adapt α in this experiment. As shown by the actual values on both sides of the inequality, however, the condition is very close to the boundary. Therefore, RASP-MSE sometimes detects the changes in the true value as shown in graph (a) of figure 5.

4 Discussion and Summary

In this article, we derived the relations between stepsize parameter α and expected value $\tilde{x}_t$ acquired by EMA, and provided a method called RASP that calculates the higher-order derivatives of $\tilde{x}_t$ by α. We also proposed a procedure called RASP-MSE that adjusts α suitably for given observed data both to reduce noise factors in the observation and to follow the changes in the environment. Experiments illustrated the functionality and performance of RASP-MSE for adjusting the stepsize parameters as shown in theorems and lemmas.

The main feature of RASP is that we can obtain derivatives $\partial\tilde{x}_t/\partial\alpha$. Therefore, we can apply it to various optimization applications that require EMA. For example, it can not only be applied to situations where the minimization of estimation error is desired, but also to the learning of decision making directly, for example, back-propagations in neural networks. Thus, it can be said that RASP has more potential than the other adaptation mechanisms of stepsize parameters such as OSA [3].

The stochastic gradient adaptive (SGA) stepsize method [5,6] is identical to RASP-MSE if we use only the first-order derivative. As we can calculate higher-order derivatives, the adaptation based on RASP can be more quick and precise. There are many other works on speed-up of reinforcement learning. Ahmadi et. al. tried to apply domain knowledge to selection of feature set to speed up the learning [7]. Abstracting feature and state spaces is also a major method to speed up and scale up the learning [8,9,10]. RASP can be combine to these works to increase adaptability to the changes of environment.

There still several open issues that include:

– To apply RASP-MSE to TD learning and multiagent learning, which may not follow the assumption of random walk.

– To utilize higher-order derivatives to calculate the best stepsize instead to change it gradually.
– To investigate the relation to several variable learning ratio technique [11,12,13], which speed-up and stabilize the learning.

Acknowledgments

This work was supported by JSPS KAKENHI 21500153.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
2. Even-dar, E., Mansour, Y.: Learning rates for q-learning. Journal of Machine Learning Research 5, 2003 (2003)
3. George, A.P., Powell, W.B.: Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. Machine Learning 65(1), 167–198 (2006)
4. Sato, M., Kimura, H., Kobayashi, S.: TD algorithm for the variance of return and mean-variance reinforcement learning (in Japanese). Transactions of the Japanese Society for Artificial Intelligence 16(3F), 353–362 (2001)
5. Benveniste, A., Metivier, M., Priouret, P.: Adaptive Algorithms and Stockastic Approximations. Springer, Heidelberg (1990)
6. Douglas, S.C., Mathews, V.J.: Stochastic gradient adaptive step size algorithms for adaptive filtering. In: Proc. International Conference on Digital Signal Processing, pp. 142–147 (1995)
7. Ahmadi, M., Taylor, M.E., Stone, P.: IFSA: Incremental feature-set augmentation for reinforcement learning tasks. In: The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (May 2007)
8. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112, 181–211 (1999)
9. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research 13, 227–303 (2000)
10. Schoknecht, R., Riedmiller, M.: Speeding-up reinforcement learning with multi-step actions. In: Dorronsoro, J.R. (ed.) ICANN 2002. LNCS, vol. 2415, pp. 813–818. Springer, Heidelberg (2002)
11. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. Artificial Intelligence 136, 215–250 (2002)
12. Abdallah, S., Lesser, V.: Learning the task allocation game. In: Proc. of AAMAS 2006, IFAAMAS, May 2006, pp. 850–857 (2006)
13. Marden, J.R., Arslan, G., Shamma, J.S.: Regret based dynamics: Convergence in weakly acyclic games. In: Proc. of AAMAS 2007, IFAAMAS, May 2007, pp. 194–201 (2007)

A Proof of Lemma 1

Suppose that the expected value $\tilde{x}_t$ calculated by eq. (2). Then $\tilde{x}_t$ can be acquired by the following summation formula:

$$\begin{aligned}
\tilde{x}_{t+1} &= (1-\alpha)\tilde{x}_t + \alpha x_t \\
&= \alpha x_t + (1-\alpha)\tilde{x}_t \\
&= \alpha x_t + (1-\alpha)(\alpha x_{t-1} + (1-\alpha)\tilde{x}_{t-1}) \\
&= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2(\alpha x_{t-2} + (1-\alpha)\tilde{x}_{t-2}) \\
&= \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + (1-\alpha)^3(\dots \\
&= \alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau} x_{t-\tau} \qquad (16)
\end{aligned}$$

When the true value s_t is generated by eq. (5), it can be re-written as follows:

$$\begin{aligned}
s_{t+1} = s_t + v_t \ &= \sum_{\tau=1}^{\infty} v_{t-\tau} \\
s_{t-\tau} = s_t - \sum_{\tau'=1}^{\tau} v_{t-\tau'}&. \qquad (17)
\end{aligned}$$

From eq. (4), eq. (16) and eq. (17), we can obtain the following equation.

$$\tilde{x}_{t+1} = \alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau} s_t - \alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau} \sum_{\tau'=1}^{\tau} v_{t-\tau'} + \alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau}\epsilon_{t-\tau}.$$

Here, we can rearrange the second term according to τ' as follows:

$$\begin{array}{r|l}
\tau \setminus \tau' & \quad 1 \quad 2 \quad 3 \quad \cdots \\
\hline
0 & (1-\alpha)^0 \ (\ ) \\
1 & +(1-\alpha)^1 \ (\ v_{t-1}\ ) \\
2 & +(1-\alpha)^2 \ (\ v_{t-1} + v_{t-2}\ ) \\
3 & +(1-\alpha)^3 \ (\ v_{t-1} + v_{t-2} + v_{t-3}\ ) \\
\vdots & \quad \vdots\ (\ \vdots \quad \vdots \quad \vdots \quad \vdots\ )
\end{array} \qquad (18)$$

Therefore,

$$\alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau} \sum_{\tau'=1}^{\tau} v_{t-\tau'} = \sum_{\tau'=1}^{\infty}(1-\alpha)^{\tau'} v_{t-\tau'}$$

Finally, we get the following equation.

$$\tilde{x}_{t+1} = s_t - \sum_{\tau'=1}^{\infty}(1-\alpha)^{\tau'} v_{t-\tau'} + \alpha \sum_{\tau=0}^{\infty}(1-\alpha)^{\tau}\epsilon_{t-\tau}.$$

Therefore, the estimation error δ_t $(= \tilde{x}_t - s_t)$ becomes

$$\delta_t = -\sum_{\tau'=1}^{\infty} (1-\alpha)^{\tau'} v_{t-1-\tau'} \epsilon_{t-1-\tau} - v_{t-2}.$$

Because ϵ_t and v_t are independent random numbers with means 0 and standard deviations σ_ϵ^2 and σ_v^2, respectively, the mean square of the above error $\mathbf{E}(\delta_t^2)$ can be calculated as follows:

$$\mathbf{E}(\delta_t^2) = \frac{1}{2-\alpha}(2\sigma_\epsilon^2 + \frac{1}{\alpha}\sigma_v^2).$$

■

B Proof of Theorem 1

The derivative of mean square error $\mathbf{E}(\delta_t^2)$ by α is as follows:

$$\begin{aligned}\frac{\partial \mathbf{E}(\delta_t^2)}{\partial \alpha} &= \frac{1}{(2-\alpha)^2}(2\sigma_\epsilon^2 + \frac{1}{\alpha}\sigma_v^2) + \frac{1}{2-\alpha}(-\frac{1}{\alpha^2}\sigma_v^2) \\ &= \frac{2(\alpha^2\sigma_\epsilon^2 + (\alpha-1)\sigma_v^2)}{\alpha^2(2-\alpha)^2}.\end{aligned}$$

Suppose that the above derivative is equal to 0. Then, we can obtain a solution of α in the range $(0, 1)$ as follows:

$$\alpha = \frac{-\sigma_v^2 + \sqrt{\sigma_v^4 + 4\sigma_\epsilon^2\sigma_v^2}}{2\sigma_\epsilon^2} = \frac{-\gamma^2 + \sqrt{\gamma^4 + 4\gamma^2}}{2}.$$

■

C Proof of Lemma 2

First, we show the following lemma.

Lemma 3

$$\xi_{t+1}^{\langle k \rangle} = \alpha^2 \sum_{\tau=0}^{\infty} \tau(1-\alpha)^{\tau-1} \xi_{t-\tau}^{\langle k-2 \rangle} \tag{19}$$

□

Proof
Suppose that

$$\begin{aligned}\eta_{t+1} &= \alpha^2 \sum_{\tau=0}^{\infty} \tau(1-\alpha)^{\tau-1} \xi_{t-\tau}^{\langle k-2 \rangle} \\ &= \alpha^2 \left[1(1-\alpha)^0 \xi_{t-1}^{\langle k-2 \rangle} + 2(1-\alpha)^1 \xi_{t-2}^{\langle k-2 \rangle} + 3(1-\alpha)^2 \xi_{t-3}^{\langle k-2 \rangle} + \ldots\right].\end{aligned}$$

Then, we can obtain the following equation:

$$(1-\alpha)\eta_t = \alpha^2 \left[1(1-\alpha)^1 \xi_{t-2}^{\langle k-2 \rangle} + 2(1-\alpha)^2 \xi_{t-3}^{\langle k-2 \rangle} + 3(1-\alpha)^3 \xi_{t-4}^{\langle k-2 \rangle} + \ldots\right].$$

This can be rewritten as follows:

$$\begin{aligned}\eta_{t+1} - (1-\alpha)\eta_t &= \alpha^2 \left[(1-\alpha)^0 \xi_{t-1}^{\langle k-2\rangle} + (1-\alpha)^1 \xi_{t-2}^{\langle k-2\rangle} + (1-\alpha)^2 \xi_{t-3}^{\langle k-2\rangle} + \ldots\right] \\ &= \alpha^2 \sum_{\tau=0}^{\infty} (1-\alpha)^\tau \xi_{t-1-\tau}^{\langle k-2\rangle} \\ &= \alpha \xi_t^{\langle k-1\rangle}.\end{aligned}$$

Finally, we can obtain the recurrence formula:

$$\eta_{t+1} = (1-\alpha)\eta_t + \alpha \xi_t^{\langle k-1\rangle}.$$

This formula is the same as the definition of $\xi_t^{\langle k\rangle}$ shown in eq. (8). Therefore, if $\eta_0 = \xi_0^{\langle k\rangle}$, η_t is identical to $\xi_t^{\langle k\rangle}$ for all t. Therefore, we can obtain eq. (19). ■

Using this lemma, we can prove Lemma 2 as follows:

In the case of $k = 1$, we can obtain the following equation:

$$\begin{aligned}\frac{\partial \xi_t^{\langle 1\rangle}}{\partial \alpha} = \frac{\partial \tilde{x}_t}{\partial \alpha} &= \frac{\partial}{\partial \alpha}\left[\alpha \sum_{\tau=0}^{\infty} (1-\alpha)^\tau x_{t-\tau-1}\right] \\ &= \sum_{\tau=0}^{\infty} (1-\alpha)^\tau x_{t-\tau-1} + \alpha \sum_{\tau=0}^{\infty} (-1)\tau (1-\alpha)^{\tau-1} x_{t-\tau-1} \\ &= \frac{1}{\alpha}\xi_t^{\langle 1\rangle} - \frac{1}{\alpha}\xi_t^{\langle 2\rangle} \\ &= \frac{1}{\alpha}(\xi_t^{\langle 1\rangle} - \xi_t^{\langle 2\rangle}).\end{aligned}$$

Therefore, eq. (9) is satisfied when $k = 1$.

Suppose that eq. (9) is satisfied for any $k < k'$. Then, we can calculate the derivative of $\xi^{\langle k'\rangle}$ as follows:

$$\begin{aligned}\frac{\partial \xi_t^{\langle k'\rangle}}{\partial \alpha} &= \frac{\partial}{\partial \alpha}\left[\alpha \sum_{\tau=0}^{\infty} (1-\alpha)^\tau \xi_{t-\tau-1}^{\langle k'-1\rangle}\right] \\ &= \sum_{\tau=0}^{\infty} (1-\alpha)^\tau \xi_{t-\tau-1}^{\langle k'-1\rangle} - \alpha \sum_{\tau=0}^{\infty} \tau (1-\alpha)^{\tau-1} \xi_{t-\tau-1}^{\langle k'-1\rangle} + \alpha \sum_{\tau=0}^{\infty} (1-\alpha)^\tau \frac{\partial}{\partial \alpha} \xi_{t-\tau-1}^{\langle k'-1\rangle} \\ &= \frac{1}{\alpha}\xi_t^{\langle k'\rangle} - \frac{1}{\alpha}\xi_t^{\langle k'+1\rangle} + \alpha \sum_{\tau=0}^{\infty} (1-\alpha)^\tau \frac{k'-1}{\alpha}(\xi_{t-\tau-1}^{\langle k'-1\rangle} - \xi_{t-\tau-1}^{\langle k'\rangle}) \\ &= \frac{1}{\alpha}\xi_t^{\langle k'\rangle} - \frac{1}{\alpha}\xi_t^{\langle k'+1\rangle} + (k'-1)\sum_{\tau=0}^{\infty} (1-\alpha)^\tau \xi_{t-\tau-1}^{\langle k'-1\rangle} - (k'-1)\sum_{\tau=0}^{\infty} (1-\alpha)^\tau \xi_{t-\tau-1}^{\langle k'\rangle} \\ &= \frac{1}{\alpha}\xi_t^{\langle k'\rangle} - \frac{1}{\alpha}\xi_t^{\langle k'+1\rangle} + (k'-1)\frac{1}{\alpha}\xi_t^{\langle k'\rangle} - (k'-1)\frac{1}{\alpha}\xi_t^{\langle k'+1\rangle} \\ &= \frac{k'}{\alpha}(\xi_t^{\langle k'\rangle} - \xi_t^{\langle k'+1\rangle}).\end{aligned}$$

As a result, eq. (9) holds for any $k > 0$. ■

D Proof of Theorem 2

In the case of $k = 1$, we can obtain the following equation:

$$\begin{aligned}\frac{\partial \tilde{x}_t}{\partial \alpha} &= \frac{\partial}{\partial \alpha}\xi_t^{\langle 1\rangle} \\ &= \frac{1}{\alpha}(\xi_t^{\langle 1\rangle} - \xi_t^{\langle 2\rangle}) \\ &= (-\alpha)^{-1}(\xi_t^{\langle 1\rangle} - \xi_t^{\langle 2\rangle}).\end{aligned}$$

Therefore, eq. (10) is satisfied when $k = 1$.

Suppose that eq. (10) is satisfied for any $k < k'$. Then, we can calculate the k'-th derivative as follows:

$$\begin{aligned}\frac{\partial^k \tilde{x}_t}{\partial \alpha^k} &= \frac{\partial}{\partial \alpha}\frac{\partial^{k-1}\tilde{x}_t}{\partial \alpha^{k-1}} \\ &= \frac{\partial}{\partial \alpha}\left[(-\alpha)^{-(k-1)}(k-1)!(\xi_t^{\langle k\rangle} - \xi_t^{\langle k-1\rangle})\right] \\ &= -(k-1)(-1)^{-(k-1)}\alpha^{-k}(k-1)!(\xi_t^{\langle k\rangle} - \xi_t^{\langle k-1\rangle}) \\ &\quad +(-1)^{-(k-1)}\alpha^{-(k-1)}\left[\frac{\partial}{\partial \alpha}\xi_t^{\langle k\rangle} - \frac{\partial}{\partial \alpha}\xi_t^{\langle k-1\rangle}\right]\end{aligned}$$

The first and second terms inside the brackets in the right hand side of this equation are $\frac{k}{\alpha}(\xi_t^{\langle k\rangle} - \xi k + 1_t)$ and $\frac{k-1}{\alpha}(\xi_t^{\langle k-1\rangle} - \xi k_t)$, respectively. Therefore,

$$\begin{aligned}\frac{\partial^k \tilde{x}_t}{\partial \alpha^k} &= (-1)^{-(k-1)}\alpha^{-k}(k-1)! \times \left[-k\xi_t^{\langle k+1\rangle} + (k + (k-1) - (k-1))\xi_t^{\langle k\rangle}\right. \\ &\qquad \left. +((k-1) - (k-1))\xi_t^{\langle k-1\rangle}\right] \\ &= (-1)^{-(k-1)}\alpha^{-k}(k-1)!\left[-k\xi_t^{\langle k+1\rangle} + k\xi_t^{\langle k\rangle}\right] \\ &= (-1)^{-k}\alpha^{-k}k!(\xi_t^{\langle k+1\rangle} - \xi_t^{\langle k\rangle}) \\ &= (-\alpha)^{-k}k!(\xi_t^{\langle k+1\rangle} - \xi_t^{\langle k\rangle}).\end{aligned}$$

As a result, eq. (10) holds for any $k > 0$. ■

Multiagent Reinforcement Learning Model for the Emergence of Common Property and Transhumance in Sub-Saharan Africa

Balázs Pintér, Ákos Bontovics, and András Lőrincz

Eötvös Loránd University, Pázmány Péter s. 1/C, Budapest, Hungary
{bli,bontovic,andras.lorincz}@elte.hu

Abstract. We consider social phenomena as challenges and measures for learning in multi-agent scenarios for the following reasons: (i) social phenomena emerge through complex learning processes of groups of people, (ii) a model of a phenomenon sheds light onto the strengths and weaknesses of the learning algorithm *in the context* of the model environment. In this paper we use tabular reinforcement learning to model the emergence of common property and transhumance in Sub-Saharan Africa. We find that the Markovian assumption is *sufficient* for the emergence of *property sharing*, when (a) the availability of resources fluctuates (b) the agents try to maximize their resource intake independently and (c) all agents learn simultaneously.

1 Introduction

The NewTies EU FP6 project[1] [4] started from two constraints: NewTies defined a series of challenges from social phenomena and wanted to model those phenomena through emergences. NewTies ended with the following conclusion: modeling through emergences provides information about the efficiency of individual and social learning *together* with the constraints about the environment. The reason is that in every model one tries to satisfy Occam's razor principle: "entities should not be multiplied unnecessarily" and tries to build a minimal model, but simplicity of the model may constrain potential emergences.

One particular aspect of NewTies was that individual learning was taken seriously: for each agent, sequential decision making was treated within the framework of reinforcement learning (RL) and the Markov decision process (MDP) model of RL [18] motivated by psychology and neuroscience [15]. Special constraints were (re-)discovered during this endeavor, e.g., (i) agents should build a model about the mind of the other agent [11] and (ii) factored reinforcement learning is necessary to counteract combinatorial explosion in complex scenarios [19,6].

The so called 'herders challenge' is relevant, because it shows an example where neither of the above conditions is necessary to emerge joint social learning.

[1] New and Emergent World models Through Individual, Evolutionary, and Social Learning, http://www.new-ties.eu

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 91–106, 2010.

The relevant aspect of this learning scenario is that the fluctuation of rainfall, and, because of that, the spatial and temporal fluctuation of resources is large.

We begin our paper with a brief introduction to the herders challenge, multi-agent systems, reinforcement learning and the NewTies framework we used (Sect. 2). Then, in Sect. 3 we describe our agent architecture: how our agents store information in their memory (in maps), how they perceive the world (through features), and how they act in this world (with macros). We provide the details about the model of the environment, the agents, and the interactions between the agents in Sect. 4. We continue with our results and their discussion (Sect. 5). Conclusions are drawn in Sect. 6.

2 Preliminaries

2.1 The Herders Challenge

Hardin, in his noted paper about the tragedy of the commons [7] described how, if left unchecked, herders would keep increasing the size of their stocks grazing on a common pasture until the pasture is overgrazed to the point that it can no longer sustain them. If a herder decides to add one more animal to his herd, he gains all the benefit, but the community as a whole bears the cost. A rational herdsman will add more and more animals to his stock, because that way he gets all the benefits and bears only a fraction of the cost.

In the last century the preeminent problem concerning African pastoralists was thought to be the degradation of rangelands because of excessive livestock numbers, based on the same argument [2]. The scientific basis for this view has been the concept of rangeland carrying capacity. This notion is also the basis of Hardin's paper: the increase in animal numbers decreases the availability of forage, a finite resource. In the end, there will be too many animals for the land to carry, and the land will no longer be able to sustain them. Hardin concludes that freedom in commons brings ruin to all.

One of the argument's premises is that the herders operate in a closed system. The scenario does not take into account any outside influences such as weather. But weather is the most powerful force in Africa, for example 83 percent of variation in the areal extent of the Sahara between 1980 and 1989 was explained by variations in annual rainfall [8]. And it changes everything: the equilibrium of animals and forage on which Hardin's argument rests ceases to exist.

Fluctuation is a significant risk that pastoralists have to cope with. According to [12], the most prominent livelihood strategy of pastoralists is the movement of their herds in reaction to anticipated seasonal and annual changes in pasture availability. Transhumance and common property are their means of averaging out the fluctuations. They can not change the environment, so they always move to the territories with more favorable weather.

Based on the arguments above, we decided on the following model. We start with a population of ranchers. Each of them owns a territory independently of the others. Rainfall is distinct and fluctuates independently in each territory. Ranchers can initiate the combining of territories; to do so, they only have to tell

another to 'share': to give one another usufruct (i.e. the right to use it) to each other's territory. If many of these reciprocal agreements are established, groups of solitary ranchers will evolve into communities of herders, where everyone is free to graze on another's land.

In the next sections we review the three pillars our paper is based on: multi-agent simulation (MAS), related work, and reinforcement learning (RL). We also introduce NewTies, the framework our multi-agent model is realized in.

2.2 Multi-agent Simulation

In this section we briefly introduce multi-agent simulation. Good introductions to multi-agent simulation are [20] and [21].

The basic unit in multi-agent modeling is the agent. An agent is anything that can perceive its environment through sensors and can act upon it through actuators. The agents interact with their environment, and each other. They are autonomous, have a local, limited view of the system and operate in a decentralized way. An agent that always tries to optimize an appropriate performance measure is called a rational agent.

Multi-agent simulation has several advantages compared to other, more traditional modeling methods such as dynamical systems. For one, the agent population can be heterogeneous. We can create any number of different agents, and put them into the same model. The simulation usually progresses in discrete time steps. Another advantage of MAS is that the environment can change as the simulation advances, and so can the agents. Agents can adapt to the environment and to each other using learning algorithms.

MAS connects the micro and the macro level through emergences. Other methods typically only model either the macro level with aggregate data, or the micro level. In multi-agent models, the agents act on their individual, micro level, but their behavior can produce phenomena on the macro level. Collective behavior can emerge that can not be trivially explained on the level of individual agents. The goal of multi-agent modeling is to gain insight into real world problems or conduct thought experiments through emergence.

2.3 Related Work

Our model is based on three main strains of research. It is a topographical model, it is concerned with cooperation among self-interested agents, and we apply reinforcement learning (RL).

The foundational topographical agent-based model was the model of Schelling [14]. Schelling studied the interactive dynamics of individual discriminatory choices. In his model, there are two kinds of agents, say red and blue. They are situated on a chessboard. There can be at most one agent in square. An agent moves into the nearest satisfactory square if in its neighbourhood (defined as the eight surrounding squares) less than half the agents have the same color (the threshold could be set to other values). Usually, after several time steps the agents stop moving, and red and blue clusters are formed.

The seminal Sugarscape model was developed by Axtell and Epstein [3]. Our model has some resemblance to the Sugarscape world, where agents live in a grid world, and have to harvest resources (sugar and spice) that grow in this world in order to survive.

König et al. [10] extend the Sugarscape model with memory for the agents. Their memory system is very similar to our maps. Agents store supplies in cells and the positions of other agents in their memories. The authors show that sustainability may be more easily achieved in a society with subordination and coordination, then in a society with isolated agents.

Rouchier et al. [13] model the regular relationships of herdsmen and farmers. Kohler et al. [9] model the settlement dynamics of the Mesa Verde Region, taking households as agents. Their model of the landscape is very detailed. It includes an annual model of paleoproductivity, soils, vegetation, elevation, and water resource type and location.

In cooperation, the fundamental work is that of Axelrod [1]. For a good review of cooperation between self-interested agents, see [5].

Reinforcement learning has also been used in multi-agent simulations. For a review on this subject see, e.g., Chapter 7 of [20] and references therein.

2.4 Reinforcement Learning

Reinforcement learning [18] is a framework for training an agent for a given task based on positive or negative feedback called *immediate rewards* that the agent receives in response to its actions. Mathematically, the behavior of the agent is characterized by a *Markov decision process* (MDP), which involves the *states* the agent can be in, *actions* the agent can execute depending on the state, a *state transition model*, and the *rewards* the agent receives.

One popular method for solving MDPs is based on *value functions*, the expected cumulated rewards that can be collected starting from any given state. The agent's behavior, or *policy* assigns actions to states and can be stochastic. The exploration – exploitation dilemma can be solved, e.g., by ϵ-greedy policy. In this case, greedy policy is used with probability $(1 - \epsilon)$, whereas exploration takes place with probability ϵ. A policy is called greedy, if it selects actions that give the highest possible expected cumulated reward.

SARSA learning. We will use the state-action-reward-state-action (SARSA) form (see [16] and references therein), a sampled iterative assignment of the Bellman equation for the state-action value function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

where α is an update rate, $r_{t+1} \in \mathbb{R}$ is the immediate reward received upon using action $a_t \in A$ and arriving to state $s_{t+1} \in S$, and $\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ is the difference between the currently approximated value of the state s and its approximation based on the next state and the immediate reward received. This is one version of the so called temporal difference (TD) learning

methods [18]. SARSA has the advantage that it implicitly learns model parameters. Model parameters are naturally sampled as the agent interacts with the world.

We take the state space as the Cartesian product of m variables or features: $S = S_1 \times S_2 \times \ldots \times S_m$. We discretize the full feature space and use tabulated temporal difference learning.

2.5 The NewTies Framework

NewTies designed an architecture to run multi-agent simulations. An earlier version of the NewTies architecture is described in [4]. The architecture has changed during the project, a concise summary of the current architecture follows. We detail only the parts used in our experiments.

There is a virtual clock set to 1 at the beginning of the simulation. Time passes in discrete amounts, and is measured by the clock in 'time steps'. When one unit of time elapses the world transfers into a new state, the agents act, and the current time step is incremented by one.

The agents are located on a flat, 2 dimensional surface (Fig. 1) divided into same-sized square regions in a grid pattern called locations. Each agent fits into exactly one location. Locations are referenced with discrete, integer coordinates. The position of each agent is defined by its coordinates and its facing. It can face in any of the eight directions. The agents live in a finite enclosed part of this surface, that can have any shape. Agents can move between adjacent locations.

There can be a number of other objects besides agents on this surface, we used plants and places.

Plants are food sources the agents can eat. They also fit into exactly one location. The most important property of a plant is its energy: the amount of energy it can grant to the agent that eats it. Plants can reproduce in a number of ways depending on the requirements of the concrete simulation, we detail our model in Sect. 4.1.

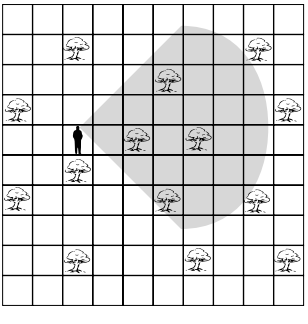


Fig. 1. The NewTies environment. The surface is divided into a grid that contains various objects. This example contains a few plants and a single agent. The agent is facing east, its field of view is 90 degrees in that direction. The agent could turn to face any of the eight neighboring positions and then proceed forward.

Places are large interconnected areas grouping individual locations. They can be of any shape and they are invisible to the agents. We modeled the rainfall patterns with same-sized square-shaped places: the amount of rainfall may differ on each place.

Agents have several attributes, the most important is energy. It is a real number that represents the current well-being of the agent. When it reaches zero the agent dies.

In every time step an agent first perceives its environment, processes the perceptions, then acts upon them. It receives a number of distinct perceptions, among them the most important: it sees every object in a 90 degree arc in front of it, up to a preset distance. It also knows its own energy level, and the messages sent to it by other agents in the previous time step.

The agent can perform various actions in each time step. We used the following: (1) `turn left/turn right`: change the direction the agent is facing, (2) `move`: move forward one location, (3) `eat`: if there is a plant in the location of the agent, eat it, and (4) `talk`: send a message to another agent.

3 The Agent Architecture

The agent architecture has four components. Maps, macros and features help the fourth component, the controller. Additionally, they are an integral part of the model: they determine what the agent can remember (maps), see (features) or do (macros). In the controller we use reinforcement learning.

Maps (Fig. 2) collect and store the observations of the agent over time, thus serve as a kind of memory. For example, if a plant gets into the field of view of an agent, and then the agent turns away, it can no longer see the plant. But it takes note on the map and remembers where the plant was, so the agent can decide to collect the plant later. Thus, maps help the agent cope with the problem of partial observability, which severely affects our multi-agent simulation. Agents can retain observations made before. But much of partial observability that springs from the nature of multi-agent systems still remains: when many agents interact, they can not predict the actions of the others. For example when our agent returns to the area it remembers to be full of plants, it might find it completely barren because the other agents have eaten all the plants in the meantime.

High level features map various low level information available to the agent to nonnegative integers to reduce the complexity of the learning problem. A feature is a function $\phi : map \times field\ of\ view \times ... \rightarrowtail \mathbb{Z}_0^+$. The state space is the Cartesian product of the codomains of these functions. The current state is the Cartesian product of their images in the actual time step. Features are an integral part of the model, as they determine what the agent can perceive, and so the controller can use. Features available to our agents are detailed in Sect. 4.2.

Macros are complex actions consisting of series of the simple actions described in Sect. 2.5. This way complex functions of the agent are automated. For example, the controller can choose between *find a plant and eat it* and *explore*, and

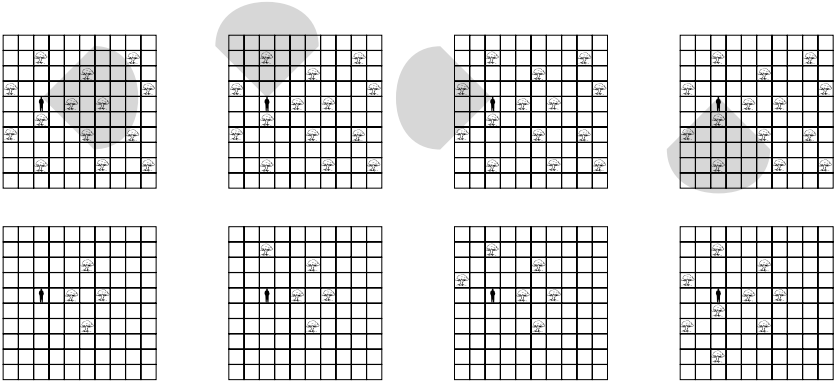


Fig. 2. Maps serve as a kind of memory. In the top row the agent is seen turning continually left (in the pictures we included only four of the eight directions). The plants it remembers meanwhile is seen in the second row. It remembers more and more plants in its map as it is turning, in the end of its turn it will know of all the plants within the distance it can perceive.

not between simple actions like *go forward* or *turn left*. Macros not only reduce computational complexity, but are an integral part of the model: they determine what the agents can and can not do.

These series of actions are generated by algorithms. For example there is a *go to a food and eat it* macro generator that goes through these steps: (1) look for a high energy food in one of the agent's map, (2) plan a route to that food that goes through shared territories, (3) generate the necessary `turn` and `move` actions to reach the food, and (4) generate an `eat` action to eat the food.

The agents use RL: maps are included into state descriptions through features, macros make the action set. The RL parameters are $\gamma = 0.95$, $\alpha = 0.05$ (Eq. 1), and $\epsilon = 0.1$.

4 The Model

In this section we detail our theoretical model and its implementation in the NewTies framework. We distinguish three main parts, each in its own section: the environment, the agents and the interactions between the agents. For the environment and the agents we provide the theoretical model first, and then the realization in NewTies follows. For the interactions, the theoretical model and the implementation are the same.

4.1 The Environment

Theoretical model. Agents live in a finite, square-shaped enclosed grid world. They can choose to graze their herds on any location, in that case their (herd's)

energy is increased, but the energy stored in the vegetation on that location is decreased. The energy that can be gained from the vegetation is sufficient that agents can never reach zero energy, that is, our agents can never die. However, we do model the need for a continuous supply of food. (For details see Sect. 4.2.)

The area is divided into a grid of same-sized square-shaped regions called territories. One territory can hold at most as many agents at once as there are locations in it. Rainfall periodically changes in each territory independently. The amount of rainfall is a uniform random number chosen from the interval $[0.5 - x, 0.5 + x]$, where x is the fluctuation. The vegetation of a territory regenerates at a constant rate that is proportional to the amount of rainfall on the territory.

Every agent has a home territory it can share and some usufructuary territories, territories it can use. Every territory has an owner. An agent can only move into its home and usufructuary territories, the other territories are closed to it. Details can be found in Sect. 4.3.

From now on, when we say usufructuary territories of an agent we also mean its home territory, as naturally the agent has usufruct over it.

Realization in NewTies. The agents live in a square-shaped area completely filled with plants: there is a plant on every location. They are the sole source of food for the agents. The plants do not disappear when an agent eats them, their energy decreases by a fixed amount instead. In every time step all of the plants replenish their energy by a little amount. The rate of one plant's replenishing is a linear function of the amount of rainfall on the territory the plant is on.

We used the already mentioned Places to model the weather. Territories were realized as square Places; 5 locations high and 5 locations wide (Fig. 3). The amount of rainfall was a uniform random number chosen from the interval $[0.5 - x, 0.5 + x]$, where x is the fluctuation. This number is generated separately for every territory in every 10,000th time step.

4.2 The Agents

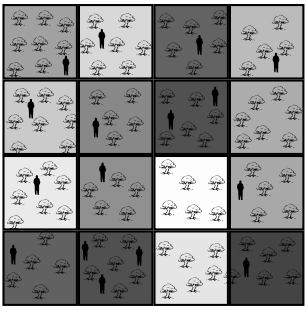


Fig. 3. The territories. A scenario with 16 territories. Every territory has a different amount of rainfall represented with different shades.

Theoretical model. We think of our agent as a herder who controls a group of animals. The most important statistics of an agent is its energy that measures how well the animals are. So the rational aim of every agent is to maximize its energy. In order to accomplish this, an agent is given a set of high level actions, or *macros*. The actions last for variable duration, for example if the agent goes to a location the duration of the action depends on the distance to the location. Every time an action is finished the agent has to choose another, but it can choose to do nothing (wait). The energy of the agents decreases even if they do nothing, and it decreases faster if they move. Note that the herder and his herd make one agent. So if the agent 'eats' then the herder grazes the animals.

The agents are informed about the outcome of each of their actions: they know how much energy they gained or lost using an action. They are given no information about the scenario in advance; they have to start exploring the possible effects of their actions by trial and error and form policies based on past experiences.

As mentioned in Sect. 4.1, our agents can not die, nor do they get hungry, as they always gain much more energy from grazing than they need for survival. However, we do model the need for a continuous supply of food with the help of reinforcement learning.

Reinforcement learning can look ahead to find sub-optimal actions that eventually lead to high rewards (in fact it *remembers* the previously experienced and so far optimal action sequence), but the fluctuation changes on a completely different timescale. So our agent can not keep track of the weather, or foresee that it will change, it is 'short-sighted'. It tries to optimize a policy that is short-term compared to the timescale of weather change. In other words it tries to consume as much energy as it can in the short term, so it needs a constant supply of food.

The agent was given the following actions:

1. go to a location with high energy vegetation and graze on it
2. propose a sharing agreement to another agent
3. break a sharing agreement
4. explore surroundings
5. wait (do nothing)

Because of the limitations of reinforcement learning we were constrained in the amount of information we could give to our agent. We tried to give it the minimum information we think a human would require to decide:

1. whether there is anyone to share with
2. whether there is anyone to break the share agreement with
3. the average energy of the vegetation in all the usufructuary territories (territories the agent can go into and graze in)
4. the average energy of the vegetation in all the territories whose owner the agent could establish a share agreement with

Realization in NewTies. The agent tries to accumulate the maximum amount of energy possible. This is realized by the reward: the reward after each action

is the difference in the energy of the agent before and after that action, that is, the energy gained or lost. Reinforcement learning is capable of finding action sequences where suboptimal actions at a given time instant (e.g., share) may lead to high rewards later (eat), so we suspected that even though share is an action that is not beneficial in itself, if the territory opened when it contains high energy vegetation, the agent will learn that it is a beneficial action.

In Sect. 3, we described the blueprint of our agents. Now we fill in the details, enumerate the particular features and macros used. These define what the agent can perceive and how it can act. Maps are not perceived directly by the agents; an agent has access to the features that can be derived from the map.

We used the following features:

1. '`share feature`': 0 or 1. It is 1 if and only if the execution of a share macro would most likely be successful in this time step. That is, if the agent can see one of its neighbors with whom it has not already shared its territory. A share action will only be successful if the partner agent has more than 60% of the maximum resource possible on its shared territories (see below). This information is not encompassed into this feature, the agent does not know it.
2. '`break feature`': 0 or 1. It is 1 if and only if the execution of a break macro would be successful in this time step. That is, if the agent can see another agent with whom it has an agreement and who is not on the agent's home territory.
3. '`average shared plants energy`': 0, ..., 9, the discretized average of the plants' energy on the usufructuary territories of the agent.
4. '`average neighbors' not shared plants energy`': 0, ..., 4, the discretized average of the plants' energy on the neighboring territories the agent can not currently enter.

We used the following macros:

1. '`go to the best food and eat it`': the agent goes to one of the foods with high energy on its usufructuary territories through its usufructuary territories and eats it.
2. '`share home territory`': if the agent can see a neighbor with whom it does not have a sharing agreement, then it initiates one to share their respective home territories with each other. They also tell each other which territory they own. After their first interaction they will know this for the length of the simulation.
3. '`break a share agreement`': if the agent can see another agent with whom it has a sharing agreement, then they break their agreement.
4. '`explore the surroundings`': the agent turns a few times in a random direction then moves forward through a few time steps
5. '`wait a time step`': the agent waits (does nothing) for a time step

4.3 The Agreements between Agents

We mentioned in the introduction that agents start as ranchers. Every rancher starts on a distinct territory, and they are confined to this territory, their home. They own it and can never lose it. In addition they can grant usufruct rights to other agents, if requested. They also gain usufruct rights to the home territory of the other agent in turn. This process is called sharing (Fig. 4), because agents share their home territories. Agents can walk and graze their herds on all their usufructuary territories, so if there are enough of them then there is the possibility of transhumance.

Agents only initiate sharing with their neighbors: that is, *Agent 1* only initiates sharing with *Agent 2* if *Agent 2*'s home territory and one of *Agent 1*'s *usufructuary* territories have a common border. Otherwise *Agent 1* would not gain anything because it would not have a route to *Agent 2*'s territory.

The other agent only accepts a share proposal if it has already enough food for itself: we chose 60% of the maximum amount of food possible, because 50% (the expected value) is just enough for the agent, so it sets a safety margin.

The procedure of sharing is the following:

1. *Agent 1* requests sharing
2. If *Agent 2* has more than 60% of the maximum possible plant energy on his shared territories, it answers with yes. If it has less, the answer is no, because it would endanger its own survival.
3. From now on they can both move to and eat from the home territory of the other.

Sharing agreements do not time out, they last forever. But they can be broken. In fact, it is easier to break an agreement than to establish one, because breaking is not constrained opposed to sharing. An agent can break an agreement any time,

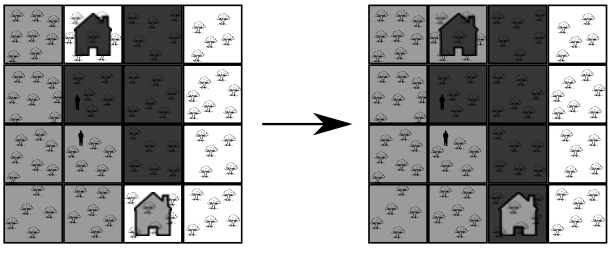


Fig. 4. The process of sharing territories. There are two agents in the process of establishing an agreement on the figure (the other agents are not shown). Their respective home territories are represented by the two houses in different shades of gray. An agent can enter only its home territory and the territories of the agents with whom it has a sharing agreement. This is represented with the two shades of gray: the agent whose home territory is colored dark (light) gray can only enter the dark (light) home and dark (light) colored areas (usufructuary territories). The home territory of the other agent can be entered only upon sharing.

the only constraints are that it must see the other agent it wants to talk to, and the other agent can not be on the home territory of the agent that breaks sharing at that time instant.

The procedure of breaking an agreement is the following:

1. *Agent 1* initiates the breaking of an agreement
2. *Agent 2* accepts it if it is not on *Agent 1*'s home territory.
3. From now on they can not step on the home territory of the other.

Note that as the consequence of breaking agreements, there can be territories that are shared but temporarily can not be reached by an agent.

5 Results and Discussion

We examined two scenarios. The first scenario consisted of 16 agents, each starting on its own home territory. Their home territories were placed onto a 4×4 chessboard (Fig. 3). In the second scenario there were 25 agents and 25 territories on a 5×5 chessboard. The length of the scenario was 50 000 time steps. This was long enough for the learning algorithm to stabilize, and short enough to make the time required to run one simulation feasible.

We have run the simulation 10 times for each value of the rainfall fluctuation from 0.00 to 0.50 with increments of 0.01, then computed the average and standard deviation for each value.[2]

We got the same results on both of the scenarios. Fig. 5 shows the average number of usufructuary territories per agent as a function of the fluctuation. If the fluctuation is low the agents do not share their territories. But as the fluctuation rises, the agents start to share. The number of usufructuary territories per agent is increasing, but so is the standard deviation: the system is unstable. As the fluctuation is above approximately 0.4, the number of usufructuary territories is constantly high, and the standard deviation is considerably smaller: the system becomes more stable. The agents gain usufructuary rights to 6-7 territories at most, out of the whole 16. For the scenario with 25 agents the usufructuary territories per agent rose a little bit, but not considerably: it is between 7 and 8. Although now there are 150% more territories, the number of usufructuary territories does not rise significantly. This may be because there is an optimal number of usufructuary territories per agent in the limit as the world grows.

It is also interesting to see the average plant energy: how much energy does the vegetation store at the end of the simulation, in other words how much energy do the agents conserve? In both scenarios (Fig. 6) the average plant energy rises as the fluctuation rises: the agents conserve more and more energy.

There is a surprising phenomenon in our model: the energy collected (Fig. 7) by our agents drops as the fluctuation rises. We think that there are three causes

[2] Other parameters can be found in the supplementary material `http://people.inf.elte.hu/lorincz/ALA_herders_params.pdf`

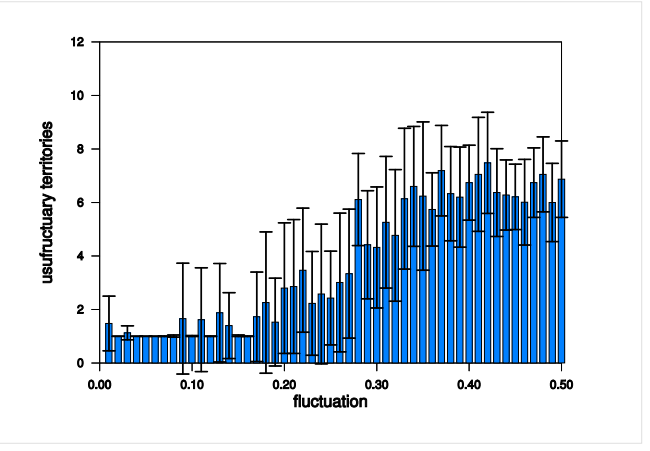


Fig. 5. Number of usufructuary territories per agent as a function of the fluctuation. It can be seen that the agents share with more partners and create larger common territories as the fluctuation increases (16 agent scenario).

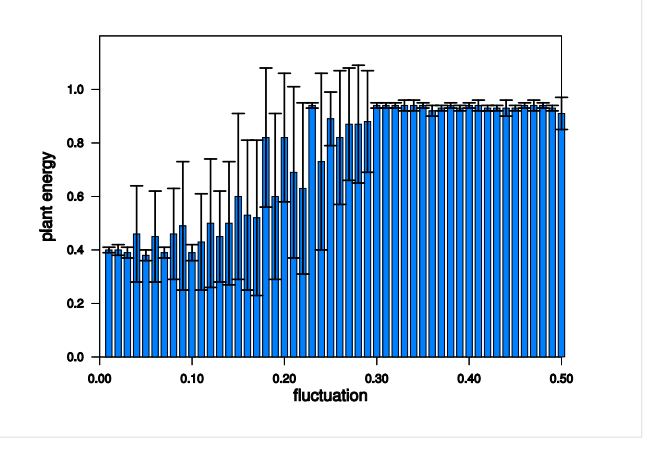


Fig. 6. Average plant energy as the function of the fluctuation. Agents conserve more and more as the fluctuation rises (25 agent scenario).

for this: first, scenarios with high fluctuations are much more difficult because of local variations, second, we did not model soil degradation, and third, the way the agents choose a location to graze on.

If the average fluctuation is 0.0, then agents basically only have to eat, and do nothing else. If it is 0.5, then there are local difficulties that the agents have to face. Even though the average plant energy is the same as in the former case, it happens a lot that the bulk of that energy is not accessible to one agent. For example, if the agent faces a severe, long lasting drought (e.g. the rainfall coefficient is below 0.1 in the agent's home territory) then it could suffer despite

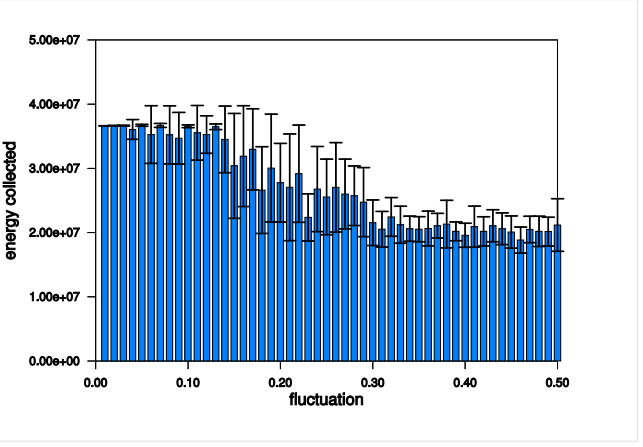


Fig. 7. Energy collected by the agents as the function of the fluctuation (25 agent scenario)

the fact that on average there is a lot of food around. Because the weather changes in every 10,000 steps and the agents optimize only for about 100 steps with $\gamma = 0.95$ (Sect. 4.2), it has to wander more than in the case with small fluctuation. At the same time, if an agent eats only on its home territory, then it has more time to eat than an agent that has to move to the territory with more food.

The second cause is that we did not model soil degradation. If perpetual grazing would degrade the soil as in real life it does, then agents doing nothing but eating would eventually gain much less energy than the agents that wander from territory to territory, and eat only high-energy plants.

The third cause is that when agents graze they always choose one location they remember to have high energy, and they move to that location. Clearly the average distance they travel grows as the number of their usufructuary territories grows. So even if we would create a controller that always chooses the *go to a high energy food and eat it* macro, it would occur that the agent who can not leave its home territory consumes more energy than the agent with several territories, because the former would travel less between *eat* actions.

When one agent has many usufructuary territories we talk about common ownership, because if one agent can use 7-8 territories on average then obviously one territory is used by 7-8 agents on average because there are the same number of agents as there are territories.

The agents established common territories, and with the help of these they managed to overcome the fluctuation. The expected value of rainfall is the same on all of the territories, the fluctuation only creates local variations. So the more territories an agent has access to, the closer the amount of rainfall averaged on its territories is to the expected value, and the less the fluctuation affects the agent.

There are two points to be mentioned here. One is risk management or insurance. Basically when agents establish sharing agreements, they insure themselves: they agree that they share their territories so if rainfall is low on either of them both can survive. The more sharing agreements an agent has, the better insured it is. They cope with local fluctuation by trying to have enough territories so that the effect of fluctuation is diminished.

The other point is Adam Smith's invisible hand [17]: each agent '*intends only its own gain and promotes an end that was not part of its intention*', but is good for the community as a whole. Every agent tries to maximize its own energy, but in doing so they insure themselves and the other agents, so the whole agent 'community' is insured against the fluctuation of the rainfall: if rainfall is low on a territory, they simply move somewhere else. In other words, the agents are selfish and despite that achieve an outcome that is good for all of them.

6 Conclusions

We have demonstrated conditions where adaptive agents established common property even though they maximized their own gains, and did not consider the effect of their actions on the other agents.

We described why Hardin's *tragedy of the commons* is not applicable to the conditions in Sub-Saharan Africa. Modeling real-world conditions we constructed a learning scenario where the 'tragedy' did not occur, although all of our agents were autonomous rational agents (they considered only their own benefit), just like Hardin's herdsmen. The most important characteristic of this scenario was the fluctuation of the regeneration rate of resources. The fluctuation was present in both space and time, and as an agent needed the resource in the short term, they could only choose to cope with the fluctuation in space. They established areas where they could freely move and graze, so they could always wander to a territory with high resources.

Acknowledgments

Thanks are due to Nigel Gilbert for helpful discussions. This research has been supported by the EC FET 'NewTies' Grant FP6-502386. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of other members of the consortium or the European Commission.

References

1. Axelrod, R.: The Evolution of Cooperation Basic Books (1984)
2. Behnke, R., Scoones, I.: Rethinking range ecology: Implications for rangeland management in Africa. Int. Inst. for Envir. and Development Paper No. 33 (1992)
3. Epstein, J.M., Axtell, R.L.: Growing Artificial Societies: Social Science from the Bottom Up (Complex Adaptive Systems). The MIT Press, Cambridge (1996)

4. Gilbert, N., den Besten, M., Bontovics, A., Craenen, B.G.W., Divina, F., Eiben, A.E., Griffioen, R., Hévízi, G., Lõrincz, A., Paechter, B., Schuster, S., Schut, M., Tzolov, C., Vogt, P., Yang, L.: Emerging artificial societies through learning. J. of Artificial Societies and Social Simulation 9(2), 9 (2006)
5. Gintis, H.: Modeling cooperation among self-interested agents: a critique. The Journal of Socio-Economics 33, 695–714 (2004)
6. Gyenes, V., Bontovics, Á., Lőrincz, A.: Factored temporal difference learning in the New Ties environment. Acta Cybernetica 18, 651–668 (2008)
7. Hardin, G.: The tragedy of the commons. Science 162(3859), 1243–1248 (1968)
8. Hulme, M., Kelly, M.: Exploring the links between desertification and climate change. Environment 76, 4–45 (1993)
9. Kohler, T.A., Kresl, J., van West, C., Carr, E., Wilshusen, R.H.: Be there then: a modeling approach to settlement determinants and spatial efficiency among late ancestral pueblo populations of the Mesa Verde region, pp. 145–178. U.S. southwest Oxford University Press (2000)
10. König, A., Möhring, M., Troitzsch, K.G.: Agents, Hierarchies and Sustainability Agent Based Computational Demography. Physica, 197–210 (2002)
11. Lőrincz, A., Gyenes, V., Kiszlinger, M., Szita, I.: Mind model seems necessary for the emergence of communication. Neural Inf. Proc. Lett. Rev. 11, 109–121 (2007)
12. Rass, N.: Policies and strategies to adress the vulnerability of pastoralists in Sub-Saharan Africa. PPLPI Working Paper No. 37, FAO (2006)
13. Rouchier, J., Bousquet, F., Requier-Desjardins, M., Antona, M.: A multi-agent model for describing transhumance in North Cameroon: Comparison of different rationality to develop a routine. Journal of Economic Dynamics and Control 25, 527–559 (2001)
14. Schelling, T.C.: Dynamic models of segregation. Journal of Mathematical Sociology 1, 143–186 (1971)
15. Schultz, W.: Getting formal with dopamine and reward. Neuron 36, 241–263 (2002)
16. Singh, S., Jaakkola, T., Littman, M., Szepesvári, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. Machine Learning 38, 287–303 (2000)
17. Smith, A.: The Wealth of Nations. Bantam Classics (March 2003)
18. Sutton, R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
19. Szita, I., Lőrincz, A.: Factored value iteration converges. Acta Cybernetica 18, 615–635 (2008)
20. Vlassis, N.: A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Morgan and Claypool Publishers (2007)
21. Wooldridge, M.: Introduction to MultiAgent Systems. John Wiley & Sons, Chichester (2002)

Learning to Locate Trading Partners in Agent Networks

John Porter, Kuheli Chakraborty, and Sandip Sen*

Department of Computer Science
University of Tulsa
sandip@utulsa.edu

Abstract. This paper is motivated by some recent, intriguing research results involving agent-organized networks (AONs). In AONs agents have a limited number of collaboration partners at any time, represented by edges in a network of agent nodes, and can rewire edges, i.e., change partners, to improve performance. The common underlying research issue in these domains is the search for desirable interaction or collaboration partners in a relatively large population. Agents have to learn to estimate the utility of current trading partners and adapt connections to improve profitability. A previous study found that random selection of partners in each time period produced better performance but incurred larger search costs compared to gradual rewiring of edges in the network in a production and exchange economy. We propose an exponentially decaying exploration scheme that produces similar utilities to random rewiring but with much less rewiring costs. We evaluate the effects of the number of trading partners on the utilities obtained by the agents. We hypothesize on the cause for the observed performance differences and verify that by showing that the observed performance differences with more realistic model of the economy that incorporate minimum trade volumes and storage capacities.

Keywords: Agent oriented networks, partner selection, learning.

1 Introduction

Given the significant interest and popularity of peer and social networking applications, recent work in multiagent systems have increasingly studied distributed formation and maintenance of social networks [1]. A number of such multiagent systems consist of self-interested agents interacting in open environments where the resources, goals, and requirements of agents change over time. In such domains, a rational agent can often benefit by forming mutually beneficial partnerships with other agents with complementary resources and capabilities. Given the dynamic and open nature of the environments, the local conditions for agents may change as existing agents may leave the environment and new

* This work is supported in part by a DOD-Army Research Office Grant #W911NF-05-1-0285.

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 107–122, 2010.

agents may enter the society. The search for effective collaborators, therefore, is a life-long process whereby agents continually seek to locate and harness beneficial relationships. We are interested in studying the dynamics of agent relationships in such decentralized environments where both parties must agree to enter into a collaboration. Examples of such domains abound in society and industry, including players joining teams or finding other players to partner with, coalitions of groups forming to pursue common cause, organizations forming supply chains that transform raw materials to finished products, etc.

As social networks and peer-to-peer (P2P) networks have received widespread use, various forms of network topologies and their associated properties have been studied in the literature [2]. In this paper, we focus on relationships between producer and consumer agents in a distributed environment. Agents in such an Agent Oriented Network (AON) are connected, at any point in time, with a limited number of other agents but can change their connections over time [10]. To obtain utility, agents need to trade with other agents producing complementary goods. A critical decision problem affecting the viability and success of agents in such an economy is their ability to identify beneficial trading partners. The trading partner selection problem poses interesting learning and adaptation questions including how to estimate the worth of trading partners, how to locate potentially beneficial trading partners, and how to balance exploration for new partners with reaping benefits from current relationships.

Gaston and desJardins observed that randomly connecting to other agents produced more profitable trades than using more stable wiring patterns [4]. This is an intriguing and counter-intuitive result, as in real economies we observe more stable and healthy partnerships between organizations, e.g., organizations in supply chains [6]. We wanted to explain this intriguing phenomena by a careful analysis of the experimental results. We also sought to evaluate relative merits of more inertial rewiring schemes which reduce exploration over time. For a fair evaluation of these schemes we developed a more realistic economy that incorporated features like minimum trade volumes and storage capacities. We observe the effects of the number of trading partners on the profitability of the trading agents when using different partner selection strategies. Our goal is to analyze the results from the simulation to both explain the observed phenomena and design more effective partner selection mechanisms to produce efficient agent networks.

2 Models

In this section, we first review a basic production and exchange model in an agent oriented network [4,9,10]. We then present an extension of the model that captures more realistic constraints.

2.1 Production and Exchange Model

Gaston and desJardins have used a simple production and exchange model to study strategies in AON exchange economies [4]. An AON is a network of agents

in which the agents self-organize and can rewire their own connections to other agents [9,10].

We present the trading and production model below and as described by Gaston and desJardins [4]. Every agent starts with some supply of two goods and a capacity to produce a fixed amount of only one of them. At each iteration agents choose whether to produce or exchange goods and thereby gain utility. Agents are greedy and attempt to maximize the utility they gain at each time step. They are also truthful and always provide correct information when proposing a trade. We present the trading and production model below and as described by Gaston and desJardins [4]

Trading Model. In this model there are n agents and two goods g_1 and g_2. g_1 is only traded in whole units while g_2 is infinitely divisible. g_k^i is the amount of good k that agent i currently possesses. The utility of agent i is given by the product of its stock of the two goods:

$$U^i = g_1^i g_2^i.$$

In each round the agents are chosen in random order and allowed to trade or produce. First, they have to calculate how much utility they would gain by trading. Each agent is linked to m other agents with whom it can trade. The chosen agent checks its *marginal rate of substitution* (mrs) against the mrs of each of the agents it is linked with. This value is calculated as follows and truthfully revealed:

$$mrs^i = \frac{\frac{\delta U^i}{\delta g_1^i}}{\frac{\delta U^i}{\delta g_2^i}} = \frac{g_2^i}{g_1^i}.$$

The agents may be able to gain by trading if their mrs's differ. When agent i considers trading with agent j, the exchange price, p_{ij}, is computed as

$$p_{ij} = \frac{g_2^i + g_2^j}{g_1^i + g_1^j}.$$

Next, a trade is simulated to evaluate corresponding benefits, though no actual goods are exchanged until agent i chooses one trading partner. A tax τ is applied to every transaction. If agent i is trading one unit of g_1 for every p_{ij} units of g_2 with agent j and δg_k^i is the amount of good k traded by agent i, then

$$\delta g_1^i = -\delta g_1^j = -(1+\tau)$$

$$\delta g_2^i = -\delta g_2^j = (1+\tau)p_{ij}.$$

Such exchanges between these two agents are repeatedly simulated until the utility of neither agent increases from further trading. The corresponding utility gain is recorded. Once such simulated trades have been executed for every agent connected to agent i, the most profitable partner, i.e., the agent with whom trading provides maximum utility gain is selected i as the best possible trading partner in this time step.

Production model. Every agent has a production capacity Δg_i uniformly distributed in the range $[1, q]$ for one of the goods g_1 or g_2. Thus, if i produces g_1 its change in utility after production is

$$\Delta U^i = \Delta g_1^i g_2^i.$$

Once an agent knows how much utility it can gain by producing, it can choose whether to produce or trade with its best partner. Once it has made this decision and carried out the corresponding action, the agent can choose to rewire its trading connections for the next iteration.

2.2 Enhanced Production and Exchange Model

In many real world examples, agents gain utility through consuming goods or manufacturing new products from raw materials rather than by just possessing them. These agents also often have a limited space available for storage which can be expensive. Motivated by these and other considerations, in this section we propose some extensions to the model presented in the previous section to represent and reason with more realistic scenarios.

To model the production of new products using existing or procured raw materials, we propose a system of clearing. Whenever an agent has both types of goods, it combines them to manufacture a product for which the goods serve as raw materials. Thus, no excess goods are stored. Some agents are more efficient and can manufacture products while consuming lesser amounts of the good that they do not produce. For manufacturing a product, an agent must use some multiple, G, units of the good it does not produce for every unit of good that it can produce. Thus, if agent i is a producer of good 1 its new utility gain function from manufacturing would be:

$$\Delta U^i = \zeta \min(g_1^i, \frac{g_2^i}{G^i}).$$

Agent i loses the corresponding amounts of goods 1 and 2 and gains utility, based on the parameter ζ, by selling the manufactured product.

The agents have limited storage capacity for raw materials, and the maximum amount of the produced good, e.g., g_1, that an agent can store is $\bar{S}$ times its production rate. Agents also have a lower bound on the amount of good they have to have before they can try trading; this limit corresponds to a minimum trade volume. If, at the beginning of their turn, an agent has less than $\underline{S}$ times its production rate then it does not try to trade. An agent may still end up trading in this situation, but only if another agent initiates a trade with it.

Trading can be an expensive operation and it is counterproductive to perform a number of small trades. $\underline{S}$ and $\bar{S}$ comprise a trading window for an agent. It cannot trade if its stock of the good it produces is below $\underline{S}$. On the other hand, if the agent cannot find a trading partner before its stock of produced good reaches $\bar{S}$, then it will start losing production opportunities as there is no space to store additional produced good.

As a final change to the production and exchange model we allowed continuous production. Agents could produce every turn and even in turns where they are trading.

3 Rewiring Strategies

The goal of rational agents within such production and exchange economies will be to locate most beneficial trading partners. Agents, therefore strategically rewire connections in an effort to locate more fruitful partnerships. We evaluate three rewiring strategies **random mixture**, **random selection**, and **rewiring with exploration**. The first two were used by Gaston and desJardins [4]. **Random mixture** (RM) is the simplest strategy. At each iteration agents randomly reinitialize every connection.

When using **random selection** (RS), an agent first decides whether it should rewire. It keeps an exponential weighted moving average, V, of the utility gained in each iteration. The utility agent i expects to gain in the next iteration, t, is

$$V_t^i = V_{t-1}^i + \alpha(\Delta U_{t-1}^i - V_{t-1}^i).$$

If $V_t^i < \Theta$ then the agent chooses to rewire. $\alpha \in [0, 1]$ is a learning parameter and Θ a threshold.

If it chooses to rewire, it still must choose which connections to rewire. This decision is also based on an exponentially weighted moving average of connection strengths represented by connection weights. If ΔU_{t-1}^{ij} is the change in utility that agent i could have received by trading with agent j on iteration t, agent i updates its connection weight W_t^{ij} for the connection to agent j as follows:

$$W_t^{ij} = W_{t-1}^{ij} + \beta(\Delta U_{t-1}^{ij} - W_{t-1}^{ij}),$$

where $\beta \in [0, 1]$ is a learning parameter. The agent rewires every connection for which $W_t^{ij} < \Phi$, where Φ is a threshold parameter. New connection weights are initialized to the average of the current connection weights.

We now introduce a third rewiring strategy to reduce search and exploration over time: when using the **rewiring with exploration** or, more concisely, **exploration** (RE) strategy, each agent has an initial exploration rate $x_0 \in (0, 1]$, and this exploration rate is exponentially reduced at a rate η, i.e., $x_t = \eta x_{t-1}$. The rewiring rate is based on this x_t as well as V_t^i as described above and the base expected utility, V_0^i. In the RE strategy the probability of an agent rewiring a connection is given by

$$p_t^i = x_t * \max(0, (1 - \frac{V_t^i}{V_0^i})).$$

The base expected utility is initialized as the average expected utilities for other agents this agent connects to.

As in the RS strategy, agents keep track of the weight for each connection, W_t^{ij}. However, while the RS strategy can rewire multiple connections in one

time step, an agent using the *RE* strategy is more cautious and rewires only the connection with the lowest weight, and only if the corresponding weight satisfies the condition $W_t^{ij} < \Phi$.

4 Experimental Results

We now discuss our experimental results. We begin with the results from the Production and Exchange model used in [4] which is followed by results from the Enhanced Production and Exchange Model. The parameters used in the model are as follows: n=300, q=30, τ=0.05, and m was varied from 2 to 10 in steps of 2. The agent's learning parameters were set at $\alpha = \beta = \theta = \phi = 0.1$ and both the initial expected utility, V_0^i, and at the beginning of each run, the initial valuation of every connection, W_0^{ij}, were set to 1 following Gaston and desJardins [4]. The exploration strategy began with an exploration rate of $x_0 = 0.3$ and uses a decay rate of $\eta = 0.996$. All results are based on randomly generated initial network structures.

4.1 Production and Exchange Model Results

Experiments in this section uses the basic Production and Exchange model used by [4].

Homogeneous Populations. The first set of experiments were run with homogeneous agent populations where all agents use the same rewiring strategy.

Non-continuous Production. We note that the basic model precludes production when trading. For this model we present, in Table 1, the utilities obtained, the number of trades per agent per round and the number of rewirings per agent per round for both $m = 2$ and $m = 10$.

For small number of connections ($m = 2$), the *RM* Strategy provided slightly higher utility than the other two strategies while the *RS* and *RE* strategies generated very similar utilities. If an *RM* agent was connected to only poor trading partners in a round then it is more likely to produce than trade. So long as it trades often enough, this extra production will yield it a greater overall utility. The RM agents accrued enough goods and could eventually trade sufficiently to outperform the other wiring strategies. *RS* and *RE* agents quickly found trading partners more often and hence were more likely to trade. This led to too frequent trading and so an overall lower utility as trades were often not of very high quality with some partners.

The increased utility obtained by *RM* agents, however, comes at a considerable cost. These agents rewired all their connections every round. This strategy incurs enormous overhead in situations where finding a new agent and setting up trade with them is an expensive process. Preparing to trade with another agent can be costly if trust relations are important or if contractual terms need to be negotiated. *RS* and *RE* agents only rewired those connections which did not produce enough utility. We observe that the nature of rewiring patterns are

Table 1. Experiments with 2 and 10 connections per agent for a population of 300 agents where agents do not produce when trading

Non-continuous Production & Exchange						
	m=2			m=10		
	Utility	Trades	Rewirings	Utility	Trades	Rewirings
Random Mixture	31070414	0.0175	2	31078338	0.019333	10
Random Selection	30566272	0.0185	.005	31044192	0.0145	0.005167
Exploration	30471054	0.0225	0.000333	31040828	0.018833	0.0005

cyclic for both *RS* and *RE* agents. Whenever a connection is rewired, its weight is reset and it takes a few rounds to relearn that a new connection is not useful. A connection that was at some point useful tends to remain useful enough to stay above the cutoff threshold for rewiring.

RS rewires every connection below the threshold all at once while *RE* spreads the rewirings out. Thus the *RE* strategy keeps the rewiring cost from spiking. The sudden spikes in rewiring cost from the *RS* strategy could be problematic in some settings, and particularly when real-time performance guarantees are required. The two strategies are equally effective in finding good trading partners.

When the agents had more connections, e.g., $m = 10$, *RS* and *RE* agents were able to consistently select good trading partners and the utility advantage of the *RM* agent all but vanished. From Table 1 we notice the distinct advantage of reduced wiring cost of the other strategies over the *RM* strategies. To better understand the effect of the number of connections on the utilities returned by the different wiring strategies we plot those values in Figure 1. We see that with more partners, *RS* and *RE* strategies return higher utilities that reaches close to that of the *RM* agents whose performance is not significantly affected by the value of m.

Continuous Production. We observed that though the *RM* agents obtained more utility, they actually traded less often. While this anomaly could have been explained by the fact that the *RM* agents made better trades, we did not find any evidence to corroborate that. An alternative conjecture that surfaced at this point was whether it was trading more often that was costing the *RS* and *RE* agents. The intuition was that as agents could either trade or produce, but not both, trading in a given period would preclude an agent from producing in that period and this could have an adverse effect on cumulative utility. To further investigate this conjecture we altered the production and exchange model to allow agents to produce even when it is trading (we call this the *continuous production environment*). In this environment, therefore, agents could produce every turn, even if they had traded.

We present the corresponding results for homogeneous groups of agents with 2 and 10 connections in Table 2. As to be expected, with continuous production the utilities of all rewiring strategies improve. More importantly, we observe that for $m = 10$ the *RE* and *RS* strategies now slightly outperform the *RM*

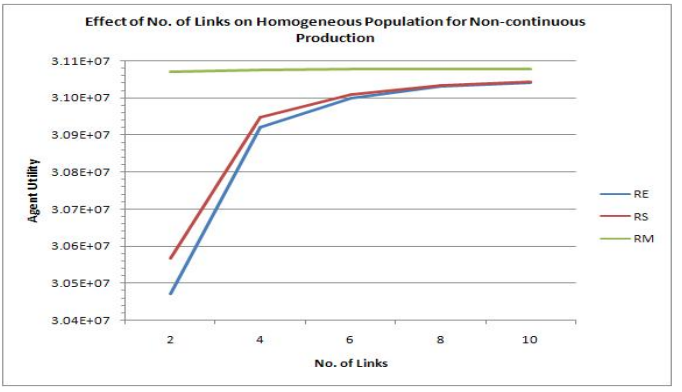


Fig. 1. The effect of number of connections per agent, m, on the utilities returned by the three wiring strategies in homogeneous populations of 300 agents when agents do not produce when trading (RS - random selection, RM - random mixture, RE - Exploration)

Table 2. Experiments with 2 and 10 connections per agent for a population of 300 agents where agents produce when trading

Continuous Production & Exchange						
	m=2			m=10		
	Utility	Trades	Rewirings	Utility	Trades	Rewirings
Random Mixture	32990700	0.9975	2	32998468	1	10
Random Selection	32985861	0.998167	0.009667	32998614	1	0.153667
Exploration	32986554	0.999	0.0005	32998481	1	0.002333

strategy. This confirms our conjecture that allowing production while trading, which also corresponds to realistic scenarios, can make more patient rewiring strategies more competitive. The effects of number of connections on the agent utilities for the continuous production environments are presented in Figure 2. We note that the strategies perform almost at the same level starting at as few as 4 trading partners.

To illustrate the effects of rewiring strategies on the number of rewirings, we plot, in Figure 3, the number of rewirings over the course of a run by homogeneous groups of RS and RE agents. We do not plot the rewirings of RM agents as each RM agent rewires each connection every round. Note the periodic, spiked nature of the plot for RS agents and the gradually decreasing rewiring trends for the RE agents as discussed in the previous section.

Heterogeneous Populations. In the next set of experiments we experimented with heterogeneous agent populations in the basic production and exchange model. We included equal proportions of RM, RS, and RE strategies in a population of 300 agents. Figure 4 shows the effect of varying m on agent utilities

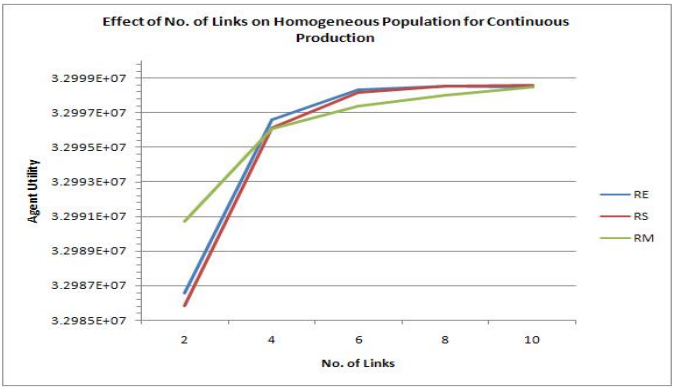


Fig. 2. The effect of number of connections per agent, m, on the utilities returned by the three wiring strategies in homogeneous populations of 300 agents when agents do produce when trading (RS - random selection, RM - random mixture, RE - Exploration)

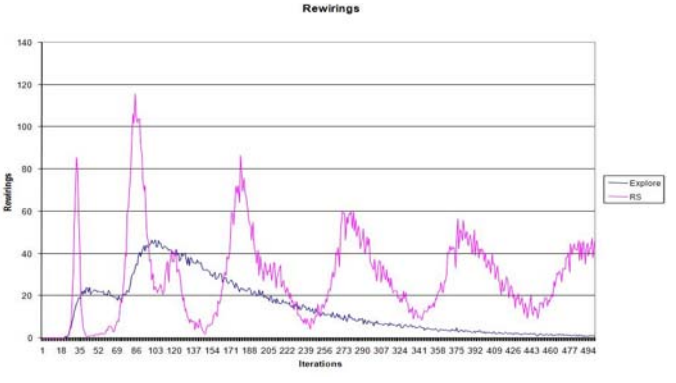


Fig. 3. The number of rewirings by homogeneous populations of 300 agents using RM (random mixture) and RE (Explore) wiring strategies when agents do produce when trading ($m = 10$)

when each rewiring strategy is used by 100 agents in the environments where agents do not produce while trading. The first striking result is that the RM agents noticeably outperform the RS and RE agents. This is true even for higher values of m. Interestingly, there is a drop in performance of the RS agents when m increases from 2. By comparing the plots in Figures 1 and 4 we find that the RM agents in heterogeneous groups actually perform better than when they did in a homogeneous group for corresponding values of m. The RS and RE agents, on the other hand, perform worse in heterogeneous groups. This means that the RM agents actually benefit at the expense of the RS and RE agents. This can be explained by the fact that after a good trade RS and RE agents may need

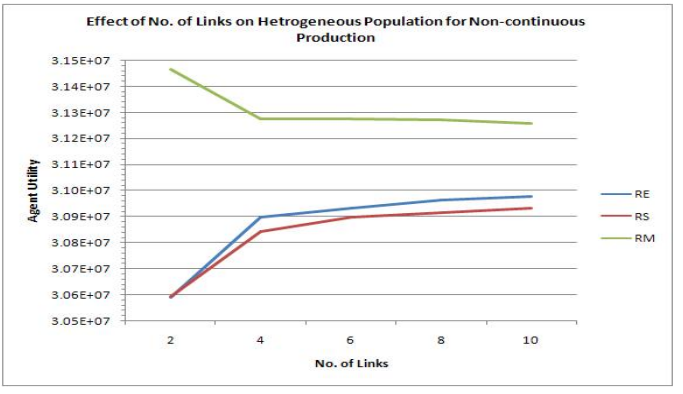


Fig. 4. The effect of number of connections per agent, m, on the utilities returned by the three wiring strategies in a heterogeneous population with each rewiring strategy used by 100 agents when agents do not produce when trading (RS - random selection, RM - random mixture, RE - Exploration)

time to trade again with their partners whereas RM agents can randomly locate partners that are ready to trade.

We further observe, from plots in Figure 5, that the superior performance of the RM agents in heterogeneous populations is sustained even in the continuous production environment and for high values of m. This is particularly interesting as the RM agents lost their performance advantage in homogeneous groups for the continuous production environment for high values of m (see Figure 2).

4.2 Enhanced Production and Exchange Model

Experiments in this section uses the Enhanced Production and Exchange Model that we have introduced in section 2.2. For this model we use Minimum Trade Volume($\underline{S}$)=3 and Storage Capacity($\bar{S}$)=4.

Homogeneous Populations. The first set of experiments was run with homogeneous agent populations. In a homogeneous population, there is significant and interesting effect on the performance of RE, RS and RM strategies when varying other domain characteristics like continuous and non continuous production, $\underline{S}$, $\bar{S}$, number of links, etc.

Non-Continuous Production. The effect of number of connections on the agent utilities for the non-continuous production environment (see Figure 6) shows the advantage of the judicious exploration scheme. In contrast to the basic production and exchange model, the order of performance is RE followed by RS followed by RM. In this model all agents have to accumulate sufficient stock and accrue minimum trade volume before trading. Hence the agents are making less trades,

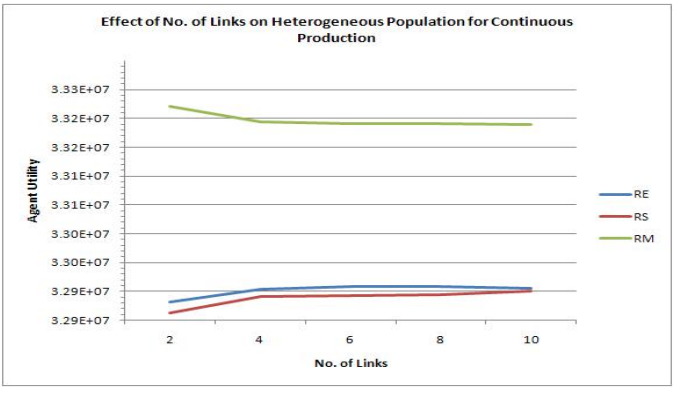


Fig. 5. The effect of number of connections per agent, m, on the utilities returned by the three wiring strategies in a heterogeneous population with each rewiring strategy used by 100 agents when agents do produce when trading (RS - random selection, RM - random mixture, RE - Exploration)

which lower their overall utility somewhat, but this decline is more pronounced for RS and particularly RM agents compared to RE agents. RM suffers more because, in contrast to the basic model, randomly selected agents are less likely to be available for trading at each time instant (in the basic model there are no stock constraints on trading and hence all agents can trade at each turn). Since RE identifies better trading partners and repeatedly uses the same trading partners unless required to change, RE outperforms RS and RM. Similarly RS also out performs RM because it identifies some good partners but not to the extent RE is able to do. When m was increased from 2 to 10 in steps of 2 the performance of each of the RE, RM and RS strategies improve but their performance difference is maintained throughout.

Continuous Production. For continuous production model where agents are allowed to produce even when they are trading, there are significant increases in the performance of RE, RS and RM strategies over the non-continuous production scenario. This is because the agents could produce every turn, even if they had traded, and hence gain higher utility from these additional stocks. The relative performance of the three strategies follow trends similar to the non-continuous production case. With increase in the value of m the difference between RE and RS increases. On the other hand the difference between RS and RM reduces and stabilizes for 6 or more connections.

Effect of change in Minimum Trade Volume and Storage Capacity. We next observe the effect of change in the Storage Capacity, $\bar{S}$, and Minimum Trade Volume, $\underline{S}$, on the performance of the rewiring strategies.

We hold $\bar{S}$ constant at 6 and increase the value of $\underline{S}$ from 2 to 6 in steps of 1. This variation significantly affects agent utilities (see Figure 8). With increase in

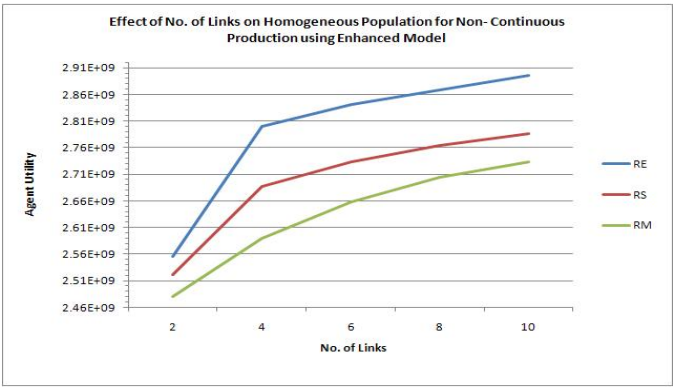


Fig. 6. Effect of m in homogeneous populations using the enhanced production and exchange model for non-continuous production (RS - random selection, RM - random mixture, RE - Exploration)

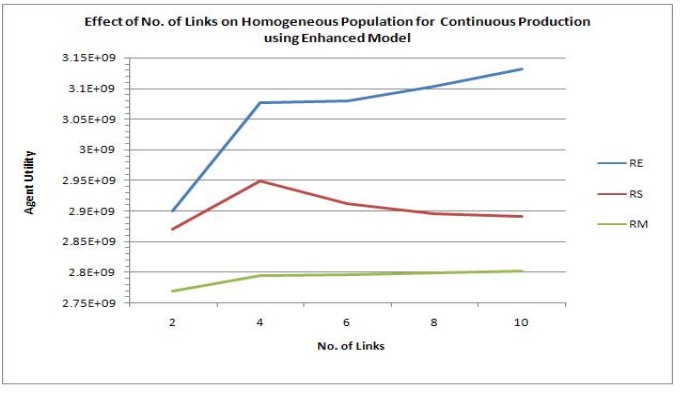


Fig. 7. Effect of m in homogeneous populations using the enhanced production and exchange model with continuous production (RS - random selection, RM - random mixture, RE - Exploration)

value of $\underline{S}$, the overall performance of agents gradually decreases. When $\underline{S}$=2 and $\bar{S}$=6, agents can produce until they find good trading partners to trade with. With increase in value of $\underline{S}$, the trading window computed as the difference between $\bar{S}$ and $\underline{S}$ reduces. If an agent cannot find good trading partners within the trading window, it loses production opportunity as maximum storage limit is reached. This results in a corresponding drop in total agent utilities.

We performed additional experiments to compare the effects of different $\underline{S}$ and $\bar{S}$ while keeping the trading window, i.e., $\underline{S}-\bar{S}$, the same. We used two configurations: C1 with $\underline{S}$= 3 and $\bar{S}$=5, and C2 with $\underline{S}$= 4 and $\bar{S}$=6. In both cases the trading window is 2. We found that going from C1 to C2 increases the

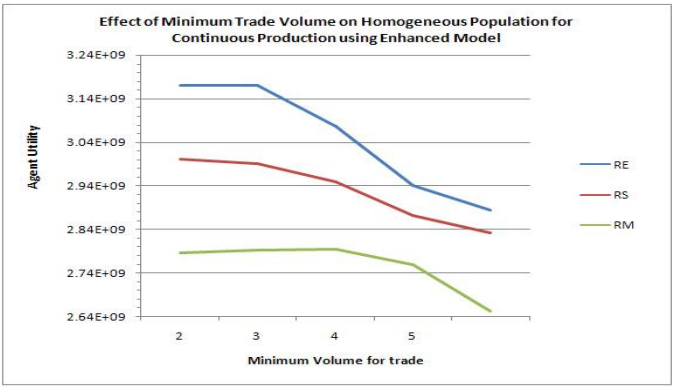


Fig. 8. Effect of increasing $\underline{S}$ while holding $\bar{S}$ constant in homogeneous populations using the enhanced production and exchange model with continuous production (RS - random selection, RM - random mixture, RE - Exploration)

performance advantage of RE over RS and that of RS over RM. Also, with the increase in the number of connections, m, the performance of RE improves further compared to that of the performance of RS and RM.

Heterogeneous Populations. In the next set of experiments we experimented with heterogeneous agent populations in the enhanced production and exchange model. We placed equal proportions of RM, RS, and RE strategies in a population of 300 agents.

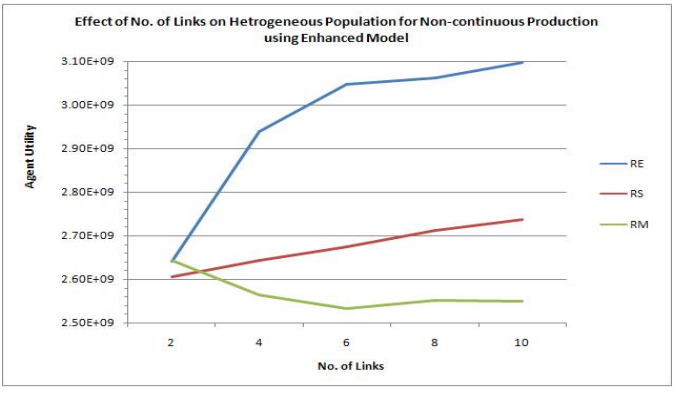


Fig. 9. Effect of m in heterogeneous populations using the enhanced production and exchange model for non-continuous production (RS - random selection, RM - random mixture, RE - Exploration)

Non-Continuous Production. In this configuration, when $m = 2$, the utilities produced by the RE, RS and RM strategies are almost equal (see Figure 9). But with increase in the value of m, e.g., when $m = 4$, agents have more trading partners per iteration and are able to locate desirable partners with less exploration. For sufficiently high m values, therefore, the utilities of RE agents increase significantly over RS and RM. When we compare the results of the heterogeneous population for the corresponding number of trading partners in the homogeneous population results (see Figure 6), we find that the RE strategy actually benefits at the expense of RM and RS strategy.

Continuous Production. We also performed experiments with continuous production for heterogeneous populations. The trends are similar to the case of non-continuous production. The primary difference is that the agent utilities are higher as they have more stock to trade with.

5 Related Work

The problem of finding suitable collaborators is an active area of research in multiagent systems. One solution is to use referrals [3,7,8,11]. In this solution agents provide both services and referrals to other agents. Agents which provide high quality service are likely to be recommended by many agents. Agents must, however, learn the trustworthiness and expertise of other agents in order to gauge the value of a recommendation.

Another possible solution to this problem is to use a matchmaker. Agents reveal information to a trusted third party who arranges the connections. Assuming agents truthfully reveal to the matchmaker, optimal matches can be found, computing these optimal matches, however, can be expensive. Also, agents using a centralized matchmaker are vulnerable to a failure in the matchmaker. Distributed matchmaking [5] reduces the scalability problem and improves fault tolerance.

Trustworthy referral and matchmaking services can be more efficient and eliminate the need for time-consuming individual learning by members of the agent network. Such services can also be designed for better scale-up. Nonetheless, distributed learning based mechanisms, as proposed in this paper, can be more resilient and robust and better handle local preferences and utility functions without revealing individual preferences to centralized agencies.

6 Discussions

We investigated the effects of introducing exploration into a rewiring strategy for locating effective trading partners within networks in production and exchange economies. Though random rewirings in each round can produce more utility, it incurs significant cost for changing connections. The proposed decaying exploration rewiring strategy and a more patient random selection strategy incurs significantly lesser rewiring costs. Additionally, the exploration strategy provides

certain benefits over random selection: it smooths out the rewirings over time and decreases the number of rewirings required. The performance advantage of the random rewiring strategy diminishes with higher number of connections per agent and when agents are allowed to produce while trading. Interestingly, however, the performance advantage is regained by the random rewiring strategy when all agent types are present in a heterogeneous society.

We believe that the basic production and exchange economy model is oversimplified and does not adequately represent real-life scenarios. We therefore evaluate the performance of the three rewiring strategies in an enhanced production and trade model that includes constraints on minimum trade volumes and storage capacities. In contrast to the basic model, the decaying exploration mechanism outperforms the more random rewiring strategies in this more realistic environments. This performance advantage also suggests the need for investigating smarter learning mechanisms for identifying preferred trading partners.

The rewiring with exploration strategy is cautious in the sense that in one iteration it rewires at most one connection, the connection with the lowest weight if that falls below a threshold. We can experiment with a stochastic decision mechanism that chooses rewiring candidates with a probability proportional to their deviation from the average connection weight.

We plan to study further enhancements to capture more realistic domain constraints. We would like to further study the effects of referrals in these models as well as bi-directional, mutually-accepted connections in place of the unilateral connections used here.

References

1. Airiau, S., Sen, S., Dasgupta, P.: Effect of joining decisions on peer clusters. In: AAMAS 2006: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, pp. 609–615. ACM Press, New York (2006)
2. Amaral, L., Scala, A., Barthelemy, M., Stanley, H.: Classes of small-world networks. Proceedings of the National Academy of Sciences 97(21), 11149–11152 (2000)
3. Candale, T., Sen, S.: Effect of referrals on convergence to satisficing distributions. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M. (eds.) Proc. 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), pp. 347–354. ACM, New York (2005)
4. Gaston, M.E., des Jardins, M.: Agent-organized networks for multi-agent production and exchange. In: Veloso, M.M., Kambhampati, S. (eds.) AAAI, pp. 77–82. AAAI Press / The MIT Press (2005)
5. Iamnitchi, A., Foster, I.: A peer-to-peer approach to resource location in grid environments. In: Weglarz, J., Nabrzyski, J., Schopf, J., Stroinski, M. (eds.) Grid Resource Managenment. Kluwer Publishing, Dordrecht (2003)
6. Lederer-Antonucci, Y.L., Greenberg, P.S., zur Muehlen, M., Ralph, G.: Establishing trust in a business-to-business collaboration: Results from an international simulation. In: Proc. of the IRMA 2003 Conference, pp. 922–924 (2003)
7. Sen, S., Sajja, N.: Robustness of reputation-based trust: Boolean case. In: Proceedings of the First Intenational Joint Conference on Autonomous Agents and Multiagent Systems, pp. 288–293. ACM Press, New York (2002)

8. Singh, M.P., Yu, B., Venkatraman, M.: Community-based service location. Commun. ACM 44(4), 49–54 (2001)
9. Wilhite, A.: Bilateral trade and 'small-world' networks. Computational Economics 18(1), 49–64 (2001)
10. Wilhite, A.: Self-organizing production and exchange. Computational Economics 21(1-2), 107–123 (2003)
11. Yolum, P., Singh, M.P.: Engineering self-organizing referral networks for trustworthy service selection. IEEE Transactions on Systems, Man and Cybernetics- Part A: Systems and humans 35(3) (May 2005)

Coordinating Learning Agents for Multiple Resource Job Scheduling

Kagan Tumer[1] and John Lawson[2]

[1] Oregon State University
Corvallis, OR 97330
kagan.tumer@oregonstate.edu
[2] NASA Ames Research Center
Mail Stop 269-4
Moffet Field, CA 94035
lawson@email.arc.nasa.gov

Abstract. Efficient management of large-scale job processing systems is a challenging problem, particularly in the presence of multi-users and dynamically changing system conditions. In addition, many real world systems require the processing of multi-resource jobs where centralized coordination may be difficult. Most conventional algorithms, such as load balancing, are designed for centralized, single resource problems. Indeed, in such a case, load balancing is known to provide optimal solutions. However, load balancing is not well suited to the more general, distributed, multi-resource allocation problem across heterogeneous networks that is frequently encountered in real world applications. Approaches based on heuristics can be designed to handle multi-resource allocation, but such approaches do not necessarily attempt to optimize *directly* a system-wide objective function. In this paper, we investigate a multiagent coordination approach to distributed, multi-resource job scheduling across heterogeneous servers. In this approach, agents at servers make local decisions to optimize an agent specific objective. The agent objectives though, are derived so that they are aligned with the overall efficiency of the system. We demonstrate that such a system outperforms (sometimes dramatically) more crudely constructed multiagent systems as well as a multi-resource version of load balancing.

1 Introduction

With the ever increasing connectivity between servers, networked or grid computing is becoming a natural alternative to either dedicated homogeneous server grids or supercomputers for processing large numbers of jobs with varying priorities and resource requirements. However, managing a large, distributed data and job processing system capable of handling multiple resource requirements is a challenging problem, in that many difficulties need to be simultaneously addressed. In the presence of heterogeneous servers (e.g., processor speed, memory), jobs with multiple resource requirements (e.g., data access, memory) dynamic environments (e.g., job arrivals do not follow a static distribution) and the

M.E. Taylor and K. Tuyls (Eds.): ALA 2009, LNAI 5924, pp. 123–140, 2010.

presence of disturbances in the system (e.g., failing servers or links) most algorithms designed for a single resource allocation algorithm either do not apply or fail to provide good solutions.

Indeed, though the single-resource case has been extensively, studied [36], the multi-resource job scheduling across a network of heterogeneous servers has received much less attention [29].[1] In addition, because of the natural distributed nature of such system, approaches based on centralized control are often inappropriate. Such methods, provide rigid, inefficient solutions, and in most cases have communication and synchronization requirements that offset any of the benefits of using a grid based system.

Load balancing is a centralized algorithm that has been successfully applied to single resource scheduling problems. In fact, for single resource optimization problems, there are theoretical results showing that load balancing does provide optimal solutions [36]. Generalizing load balancing to the multi-resource case, though, is far from straightforward. In its simplest form, multi-resource load balancing aims at ensuring that the level of activity on each server stays the same, i.e., the load on the system is balanced across all the servers. This approach to load balancing *assumes* that the load being distributed across the servers is a de-facto desirable solution, i.e. that it optimizes some pre-specified global objective. In the multi-resource case, this assumption is no longer valid, and one needs to determine which resource (or which combination of resources) needs to be "balanced". In fact, different extensions of load balancing to the multi-resource case leads to the optimization of different functions [29], and thus there are no guarantees that balancing a particular combination of the resources will lead to the optimization of the global objective. A further limiting feature of load balancing is that it requires centralized control, and though heuristics exist to overcome limitation for the single-resource case, the performance of such algorithms suffers greatly in the multi-resource case [29].

Multiagent learning methods are ideally suited to handle the challenges presented by such problems. In particular, agents based on reinforcement learning [5,7,25,37,40] offer adaptive and flexible solutions that sidestep the potential mismatch between balancing a "load" across the network and optimizing the global objective function, and have been successfully applied to data routing problems [5,21,27,32,38,45]. Indeed, the agent based approach we propose aims to optimize the global objective without directly aiming to balance the load across the servers. It is entirely possible that good solutions to a multi-resource job scheduling across a heterogeneous grid problem reside in states where some servers are idle while others are operating at full capacity and have full queues. As long as that system behavior is considered good in terms of the global objective, no consideration should be made to "split", or balance the load.

[1] Throughout this paper, we refer to servers with different resource configurations as "heterogeneous" servers. We assume that there are no compatibility issues related to compilation of the jobs, and that any job can be executed at any server, assuming the server has the necessary resources. In some articles [29], this type of network is referred to as a "near-homogeneous" computational grid.

Because of its direct aim at optimizing an objective function, agent-based methods address the limitations of load balancing. As such methods based on creating a currency [34], bio-inspired swarms [14,49], mechanism design [9] and coordination [4] have been proposed, as have other innovative methods [11,18,22,28,19,15]. However, they introduce a new difficulty: how to ensure that the actions of multiple agents lead to a good global solution. To address this coordination issue, we need to ensure that the objective function of each agent is designed in a manner that promotes two properties. First, an action that improves the agent's objective function should also improve the global objective. Second an agent needs to clearly see the impact of its actions on its own objective function [1,2,3,47]. An agent based solution to grid computing where the agent objectives are set according to these two criteria offers the best compromise between a rigid centralized solution and a distributed solution where the interaction among the agents can have deleterious side effects on system behavior.

In this paper, we present an agent-based solution to the multi-resource optimization problem in heterogeneous network that outperforms both the multi-resource version of load balancing (by up to four times), and a "naive" multiagent system in which all the agents attempt to directly optimize the system objective. *The key contribution of this paper is in providing local objective functions for the agents (components of a server) in a manner that allows them to adapt locally, while ensuring that their achieving their local objectives improves global performance.* In Section 2 we present the system model and discuss the system dynamics of the multi-resource job scheduling across a heterogeneous grid. In Section 3, we derive the agent based algorithm and present a multi-resource load balancing algorithm, along with a simple performance bound. In Section 5, we show simulation results where the multiagent system approach significantly outperforms multi-resource load balancing. Finally, in Section 6 we discuss these results and highlight future directions of research.

2 Multi-resource Optimization

With demand for computing resources increasing as both the number of users and the complexity of the applications increase, the ability of a system to efficiently schedule and process jobs is becoming increasingly important. As such, heterogeneous computational grids where jobs can enter the network from any point and be processed at any point are becoming increasingly popular. Below, we describe a model for such a computational grid and show how an agent-based approach can be implemented.

2.1 System Model

The computational grid model we use consists of a network of N servers each with K resources $(r_1, ...r_k)$. Each server has a specified capacity for each resource assigned to be an integer ranging from $[1, M]$. Thus, M measures the

heterogeneity of the resources. For example, the first resource r_1 can correspond to the processing speed of the server. In our configurations, on average, each server has 2-4 neighbors with which it has a direct connection.

Each job entering the system is also specified by K resource requirements ranging from $[1, M]$. For example, the first job resource r_1 is an indication of the number of cycles the job requires to be processed. In this formulation, for each resource $r_i, i > 1$, the server resource capacity must be equal or greater than the job's requirement in order for a job to run on a particular server. Intuitively, this corresponds to the requirement that a server must have enough memory to accommodate a given job.

2.2 System Dynamics

In this model, each server has its own wait queue for jobs. For simplicity, we allow only one job to run on a server at a time; the other jobs remain in the queue until the processor becomes available. Jobs enter the local queues either externally (to the system) or are shipped from other servers. Jobs entering externally are sent to the back of the queue while jobs received from other queues go to the front. There are two reasons why shipped jobs go to the front: First, it provides a measure of "fairness" as those jobs already had to wait in the queue of the server in which they were originally placed. Second it provides efficiency, as it forces the system to deal with "difficult" jobs (either run them or ship them if they could not be run). This approach prevents these jobs from being endlessly shuffled. At each server, the first job in the queue is activated if the processor is available, and the resource requirements are met. If the processor is available, but the server does not have the resource capacity to run the job, the server remains idle until the problem job is sent to another server.

The dynamics of our simulations thus proceed as follows. At each time step τ, a random number of new jobs are added to the wait queue of each server. In particular, each server has a probability of receiving a new job at each time. If a given processor is idle, and the first job in the queue meets the resource requirements, that job is activated. If not, the server remains idle. In addition, for each τ, the server makes a decision about the first job in the queue, deciding whether to keep the job or send it to a neighboring server. These decisions are made based on the agents' probability vectors which in turn are set using a basic learning algorithm (discussed in more detail in Section 3.3).

Thus, there are two main sources of inefficiency in the system. The first are the bottlenecks created by jobs whose requirements exceed the capacity of their server. When such a job get to the front of the queue, the server remains idle until the job is shipped to a neighbor. The second source of inefficiency arises from mismatches between a processor's speed and a job's cycle requirement.

3 Multiagent Architecture

There are many possible ways to map the multi-job scheduling problem onto a multiagent system, including simply assigning an agent to each server and letting

those agents' actions be determining where to send a particular job. Instead, in this work, we explore the mapping where there are multiple agents at each server. This results in a system with more agents with a relatively easier learning problem, rather than fewer agents with a more difficult learning problem. In fact, this choice shifts the burden from a pure learning problem where the details of the agents' algorithms are the key to the coordination problem to *how the agents interact* with one another. In particular, to each agent, we assign a vector $\boldsymbol{p}$ whose components give the probability of routing a job to its various neighbors. In this scenario, the agents are given the task of setting their own probability vector. The design question consists of determining what objective function each agent should attempt to optimize so that they set the probability vectors that also optimize the overall job processing efficiency of the full system.

The resource specifications of a job determined which agent at the server is responsible for the shipping decision. In this work, we focus on the job partitioning where for jobs with K resources, 2^K agents can be assigned per server where agent 1 deals with jobs such that $r_1 \in [1, M/2], ..., r_k \in [1, M/2]$, agent 2 deals with jobs $r_1 \in [M/2+1, M], r_2 \in [1, M/2]$, $\cdots$, $r_k \in [1, M/2]$, etc. This approach can be directly applied in systems with a small number of resources (e.g., three for processing speed, memory requirement and disk access), and this is the method we use in this paper. If the number of resources to manage becomes large, then resources can be clustered together as appropriate. This can be achieved either by direct design or by having agents form teams based on the correlations of those resources.

For the dynamics governing the system evolution, we will distinguish between two time scales : τ gives the time steps at which the system operates (e.g., jobs enter the system, move between queues, and are processed) whereas t gives the time steps at which the agents operate (e.g., observe their objectives, change their actions). This distinction is important because it is the only way by which an agent can get a "signal" from the system that reflects the impact of its decision, i.e, the system has to settle down before an objective can be matched to an action. Therefore, an agent i changes its probability vector at each time t. Within a "single agent time step" t though, many jobs enter the system, are executed, routed etc. each of which occurs at time interval τ ($t >> \tau$).

3.1 State Space and Global Objective

Let us define the state of each agent i at time t as by

$$z_{i,t} = \{(0, w_0, I_0^{i,t}, e_0^{i,t}), \cdots, (j, w_j, I_j^{i,t}, e_j^{i,t}), \cdots\} \tag{1}$$

where j is a job number identifying a job, w_j is the weight of the jth job which gives the importance of that job in the system, $I_j^{i,t}$ is the "job indicator" function and is equal to 1 if job j was handled (received, shipped or executed) by agent i at time step t, and 0 otherwise, and $e_j^{i,t}$ determines whether job j was executed at agent i at time step t.

Now, the state of the full system, z_t at time t, is given by:

$$z_t = \{(0, w_0, 1, e_0^t), \cdots, (j, w_j, 1, e_j^t), \cdots\} \tag{2}$$

where e_j^t determines whether job j was executed at time step t. Note that the job indicator function I_j^t is always set at 1 for the full system, since by definition, if the job is in the system, it must have been handled by at least one agent. Nevertheless, we keep the notation, both for ensuring consistency between the state vector of an agent and that of the full system, and because its presence in the global objective will facilitate the derivation of the agents' objectives.

Based on this, the global objective at time t is given by:

$$G(z_t) = \frac{\sum_j w_j.e_j^t}{\sum_j w_j} \tag{3}$$

Intuitively, G gives the weighted ratio of all the jobs that were processed at time step t to all jobs that entered the system at that time step (recall that "time step t" is a window of time, not a single time step from the point of view of the jobs that operate at interval $\tau << t$.)

3.2 Agent Objectives

In this work we investigated three different types of agent objectives. Each was used exactly in the same manner with the same learning algorithms. Hence, the only difference in system performance is based on the objective the agents were trying to optimize.

- The first agent objective was the global objective given in Equation 3. This objective allowed each agent to directly attempt to optimize the full system objective directly. By definition, this objective guarantees that if all agents succeed in optimizing their own objectives, the system objective will also be optimized. However, because in large systems each agents objective will depend on the actions of other agents, in practice this objective function only provides good solutions for very small systems [2,13,44].
- The second agent objective was the difference objective discussed which aims to isolate the impact of an agent on the system [2,3,43,44,47]. This is achieved by computing the difference between the system objective and the system objective that would result if agent i were removed from the system. An agent can be "removed" from the system by setting $I_j^{i,t}$ to 0 for all jobs j for which it was set to 1 at time step t. This results in the state $z_{-i,t}$, which is used to obtain the difference objective (D_i) for agent i:

$$\begin{aligned} D_i(z_t) &= G(z_t) - G(z_{-i,t}) \\ &= \frac{\sum_j w_j.e_j^t}{\sum_j w_j} - \frac{\sum_j w_j.e_j^t.\bar{I}_j^{i,t}}{\sum_j w_j} \\ &= \frac{\sum_j w_j.e_j^t.I_j^{i,t}}{\sum_j w_j} \end{aligned} \tag{4}$$

  where $\bar{I}_j^{i,t}$ is the complement of $I_j^{i,t}$ and equals 1 when $I_j^{i,t}$ equals 0 and 0 when $I_j^{i,t}$ equals 1. Intuitively, D_i represents the weighted fraction of jobs that were handled by agent i to the jobs that entered the system.

– The third agent objective was the "Selfish" objective, where the agents were only concerned with processing jobs that were assigned to them. The selfish objective (S) for agent i is given by:

$$S_i(z_t) = G(z_{i,t}) = \frac{\sum_j w_j.e_j^t.I_j^{i,t}}{\sum_j w_j.I_j^{i,t}} \tag{5}$$

Intuitively, S gives the ratio of the jobs processed by the system at time step t, to the total jobs that passed through that agent, hence the indicator function in the denominator.

Notice that both D and S are specifically tuned to the performance of a particular agent, their form is significantly different. D attempts to measure the impact of agent i on the system, whereas S attempts to measure the efficiency of agent i directly, without attempting to measures its effect on the full system. Systems using both D and S are highly sensitive to the actions of the agent, and D is much more aligned with the system objective than S is. Similarly, though both G and D are aligned with the system objective (tautologically for G), an agent using D will have an easier time seeing the impact of their actions on their objective functions. Note that regardless of which objective function the agents use, the system performance is always measured by the global objective given in Eq 3.

3.3 Agent Learning

As discussed above, the agent learning takes place at a higher time scale than the system operates. For a given time step t, each agent follows a fixed policy (e.g., the probability vectors ($\boldsymbol{p}_t$) that determine how the jobs will be sent out are fixed). During that time step t, the system operates at τ intervals (for these experiments $t = 400\tau$). At the end of time step t, the objective functions values ($V(\boldsymbol{p}_t)$) are calculated and recorded in the agents' training sets. In order to be able to compare the performance individual probability vectors, we clear the system (i.e. the queues) after each t. During the initial phase, $0 \leq t \leq 100$, the probability vectors are set at random. After this "data collection" phase, $t > 100$, the agents use a basic learning algorithm to set their probability vectors as described below.

The learning algorithm first generates R candidate probability vectors ($R = 10$ here) with a Gaussian distribution about the current probability vector ($\boldsymbol{p}_t$). Expected objective function values ($\hat{V}(\boldsymbol{p}_t)$) are estimated by performing a weighted average over objective function values from the agents' training set. The objective values are weighted by both how long ago the value was recorded (data aging) and the distance between the candidate and the previous probability vector:

$$\hat{V}(\boldsymbol{p}_t) = \frac{\sum_q V_q(\boldsymbol{p}_q)\ e^{-\alpha_t(t-q)}\ e^{-\alpha_p||\boldsymbol{p}_t-\boldsymbol{p}_q||}}{\sum_q\ e^{-\alpha_t(t-q)}\ e^{-\alpha_p||\boldsymbol{p}_t-\boldsymbol{p}_q||}}. \tag{6}$$

Here, t is the current learning period, q is the period which resulted in objective value V_q, $\boldsymbol{p}_t$ is the current probability vector, $\boldsymbol{p}_q$ is the vector that resulted in objective value V_q, and α_p and α_t are system parameters that tradeoff the impact of how recent an objective value was (α_t) and how close that probability vector was to the current one (α_p). Depending on the agent objective chosen, V_q is given by G, D_i, or S_i as discussed in Section 3.2. The new probability vector is then chosen by sampling a Boltzmann probability distribution over the estimated values $\hat{V}(\boldsymbol{p}_t)$. This process, summarized in Figure 1, allows for good exploration of the probability space, while ensuring that the most recent probability vectors have more relevance in that they are more likely to provide solutions tuned to the current conditions.

For each agent i:

For $0 \leq \mathrm{t} \leq 100$

1. Generate random a probability vector, $\boldsymbol{p}_t$
2. Use $\boldsymbol{p}_t$ for τ steps
3. Compute objective function value, $V(\boldsymbol{p}_t)$, resulting from $\boldsymbol{p}_t$
4. Store the pair $\{\boldsymbol{p}_t\ ;\ V(\boldsymbol{p}_t)\}$ in the agent's training set
5. Clear queues
6. $\mathrm{t} \leftarrow \mathrm{t} + 1$

For $100 < \mathrm{t} \leq T_{final}$

1. Generate R candidate probability vectors $\boldsymbol{p}_t^r$ by perturbing $\boldsymbol{p}_{t-1}$
2. Compute estimated objective values $\hat{V}(\boldsymbol{p}_t^r)$ using Equation 6
3. Select a new probability vector, $\boldsymbol{p}_t$ using a Boltzmann Distribution over $\hat{V}(\boldsymbol{p}_t^r)$
4. Use $\boldsymbol{p}_t$ for τ steps
5. Compute objective function value, $V(\boldsymbol{p}_t)$, resulting from $\boldsymbol{p}_t$
6. Store the pair $\{\boldsymbol{p}_t\ ;\ V(\boldsymbol{p}_t)\}$ in the agent's training set
7. Clear queues
8. $\mathrm{t} \leftarrow \mathrm{t} + 1$

Fig. 1. Agent learning algorithm. Agents take random actions for 100 time steps. After that each agent perturbs their current probability vector, estimates the objective functions that may result from those new probability vectors and selects one based on the objective values (probability vectors leading to high objective values are more likely to be selected).

4 Multi-resource Load Balancing Algorithm

In addition to the agent-based methods introduced in this article, we also investigated the feasibility of a distributed, deterministic, multi-resource load balancing algorithm. For each server, we calculated a load for each of the k resources,

$l_k = \Sigma_n^N (s_k^n / c_k)$ where s_k^n is the need of resource k of job n and c_k is the the capacity of resource k of the server. Thus, the resource load has been normalized to the resource capacity of the server. We assign a load to a particular server i as the average of its individual resource loads $L_i = Avg(l_k)$. We, then, calculate the system load as the average over the servers $L_{avg} = Avg(L_i)$.

The load balancing algorithm proceeds as follows. At each time step τ, each server calculates its own load and compares it with the global load L_{avg}. If the server's load is greater than the global, modulo some tolerance, the server looks to get rid of its highest load job. Each server has access to global information about the loads on all the other servers. Using this information, the server determines which of the other servers has the lowest load. It then ships its high load job to the low load server via the one of its neighbors that lies on the shortest path between the sending and the receiving servers.

5 Experimental Results

We ran extensive simulations that tested the performance of the algorithms in a variety of settings. All the results reported here were on networks of $N = 50$ servers having $K = 2$ and $K = 4$ resources. The 50 servers had 4 or 16 agents respectively, making for 200 to 800 total agents in the system. The servers were connected into a network having a ring configuration with random connections added in the spirit of "small world" networks [35,46]. In general, each server had 2-4 neighbors with which it had a direct connection.

We examined the performance for different number of resources K, job arrival probabilities r and different resource ranges M. We tabulated the performance for the multiagent approach with learning agents, a load balancing algorithm generalized for the multi-resource case, and a random shipping algorithm RAND. In the RAND algorithm, the proportion vectors for shipping/holding the first job in the queue was set randomly. This is the situation when the agents are in the training phase of their learning algorithm. For scenarios involving learning agents, we performed experiments using agent objectives based on global (G), selfish (S), and the difference (D) objectives.

The experiments can be grouped into three categories:[2]

- Low Difficulty: There are two resources, each having two types and the jobs enter the system at a slow pace. All three key parameters have "low" settings: ($K = 2$, $r = .2$, $M = 2$).
- Medium Difficulty: One of the parameters is set to a "high" setting. For example, there are four resources, **or** there are eight settings for each resource,

[2] The case where all three parameters are set to "high" results in an impossible problem in that the jobs entering the system have a high probability of not running on most of the machines. Because jobs are coming into the system at a high rate, the system never has a chance to move those jobs before they "clog" the system, leading to a situation where jobs are entering the system faster than the system can process them. This situation leads to poor performance by all the algorithms as the problem is in essence unsolvable.

or jobs have a high arrival rate. This covers the following parameter combinations: ($K = 4$, $r = .2$, $M = 2$) ; ($K = 2$, $r = .8$, $M = 2$) ; ($K = 2$, $r = .2$, $M = 8$) .

– High Difficulty: Two of the parameters are set to "high" values. For example, there may be both four resources **and** eight types for each, or four resources **and** a high rate of job arrival). This covers the following parameter combinations: ($K = 4$, $r = .2$, $M = 8$) ; ($K = 4$, $r = .8$, $M = 2$) ; ($K = 2$, $r = .8$, $M = 8$) .

In the following subsections, we present the results of all the algorithms for the three cases outlined above. The results show the algorithm performance at the end of the runs (t=400) and are averaged over 50 different randomly generated network configurations. The best performance in each case is noted in bold when the differences in the mean ($\frac{\sigma}{\sqrt{N}}$ for N runs with standard deviation σ) are statistically significant.

5.1 Low Difficulty Parameters

Table 1 shows the results (and the standard deviations σ) for the setting where the system is not overloaded and where there are only two resource types. Load balancing performs best in this setting. This is an interesting, though expected result. All the algorithms perform well in this case and there are two reasons for the success of load balancing. First, this is a situation that is closest to the single resource allocation problem where load balancing excels. Second, because the problem is easy, there is no need for the agent based algorithms to explore alternative solutions. However, because of the nature of such algorithms, they occasionally try a suboptimal solution to determine whether a better alternative exists. This "exploration" is a desirable trait. In this case however, because the "greedy" solution is good, any exploration causes a minor drop in performance. Figure 2 shows the convergence characteristics of the agent based algorithms, demonstrating the quick learning capability of agent using S and D objectives.

Table 1. System Processing Efficiency for Easy Setting

K	r	M	Algorithm	Global Objective	σ
			RAND	0.9318	-
			G	0.947	0.0052
2	.2	2	S	0.976	0.0039
			D	0.979	0.0023
			LB	**0.997**	0.00083

5.2 Medium Difficulty Parameters

Table 2 shows the results (and the standard deviations σ) for moderately difficult settings. All three cases have one form of difficulty (too high an arrival rate, too

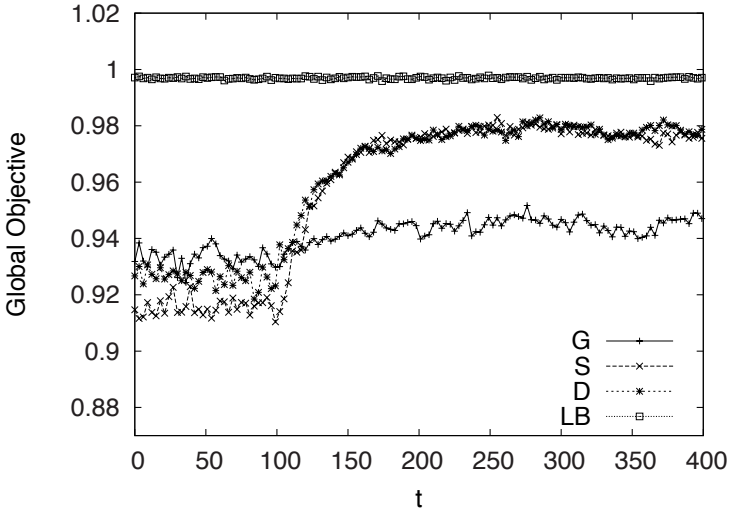


Fig. 2. Simulations results for 50 servers with 4 agents each with parameter values: (K=2,r=0.2,M=2). Each t represents a "run" of 400 τ time steps with each agent having a fixed probability vector $\boldsymbol{p}$ during the run. At the end of each run, objectives are calculated, the queues cleared, and the agents reset/modify their $\boldsymbol{p}$ based on their learning algorithms. Results are averages over 50 different systems configurations, and error bars (differences in the mean) are less than .01 in all figures.

many resources or too many types of jobs). In this setting, the agent based algorithms significantly outperform load balancing. The performance of load balancing degrades markedly for high K, and especially for high M. In fact, even setting the probability vectors at random (RAND) outperforms load balancing for $M = 8$.

This can be understood by the fact that the agent based approaches make decisions about only the first job in the queue. But it is this first job that can create serious bottlenecks in the system; if the first job needs more resources than the server can provide, the job cannot run and remains in queue, blocking other jobs from being processed as well. Load balancing, on the other hand, is attempting only to equalize the load across on the entire queue and does nothing to deal with such potential bottlenecks. For large M, the potential for bottlenecks increases markedly. Random probability vectors have the advantage over load balancing that they operate directly on the location where a bottleneck can occur. In cases for which there are many resources but low arrival rates ($r = 0.2, M = 8$) this provides an advantage for the random probability algorithm, whereas for cases with few resources buy high arrival rates, it does not.

More interestingly, in this setting, both S and D outperform G, showing the need for providing local and agent specific objectives. This result is explained by the need of the agents to extract the signal from the noise in order to learn

Table 2. System Processing Efficiency for Moderate Settings

K r M	Algorithm	Global Objective	σ
2 .2 8	RAND	0.644	-
	G	0.670	0.012
	S	**0.793**	0.011
	D	**0.793**	0.012
	LB	0.225	0.013
2 .8 2	RAND	0.626	-
	G	0.629	0.0095
	S	0.654	0.010
	D	**0.691**	0.0088
	LB	0.645	0.014
4 .2 2	RAND	0.530	-
	G	0.549	0.012
	S	**0.749**	0.011
	D	0.687	0.0098
	LB	0.474	0.020

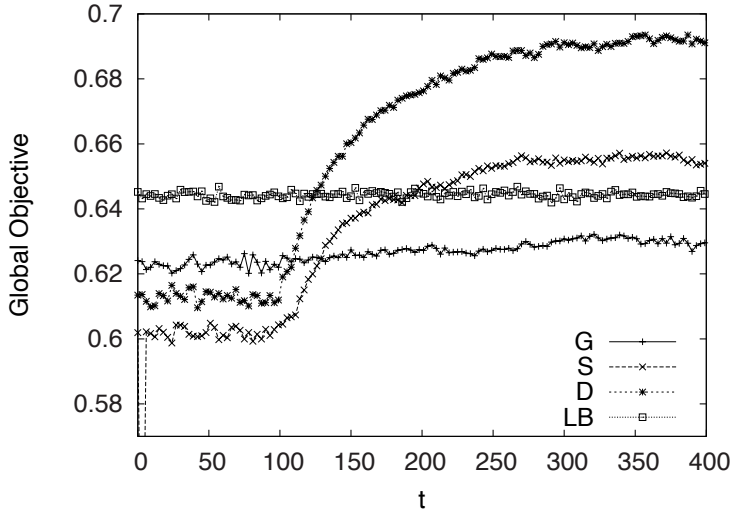


Fig. 3. Simulations results for 50 servers with 4 agents each with parameter values (r=0.8,M=2). For this medium difficulty problem, agents using D and G as objectives outperform load balancing.

the right actions. The close dependence of these agent specific objectives to the actions of the agents allows these algorithms to learn in settings where agents using G do not. Figure 3 shows the convergence characteristics of the agent based algorithms, demonstrating that the rapid learning capability of S and D. In this

setting, both S and D outperform load balancing shortly after their learners are turned on.

5.3 High Difficulty Parameters

Table 3 shows the results (and the standard deviations σ) for the difficult problem settings. All three cases have two forms of difficulty (e.g., high arrival rates *and* too many job types). In this setting, not only do agent based algorithms significantly outperform load balancing, but the D agent objective function starts to outperform the other objective functions. These results also show the importance of setting the agents' objectives to be functions that are both aligned with the system objective and impacted by the agents' actions. The team game (G) objective has poor learning properties for the individual agents since it includes information from the full system. The selfish (S) objective is not aligned with the global objective, and therefore leads to the agent learning the wrong actions. The difference objective consistently outperforms G and S for the difficult parameter settings, because it depends more closely on the action of the agents and is aligned with the global objective.

Table 3. System Processing Efficiency for Difficult Setting

K r M	Algorithm	Global Objective	σ
2 .8 8	RAND	0.194	-
	G	0.198	0.0077
	S	0.241	0.0093
	D	**0.249**	0.013
	LB	0.0974	0.0045
4 .2 8	RAND	0.130	-
	G	0.137	0.0050
	S	0.178	0.0072
	D	**0.238**	0.014
	LB	0.0092	0.0022
4 .8 2	RAND	0.176	-
	G	0.189	0.0069
	S	**0.195**	0.0080
	D	**0.195**	0.0073
	LB	0.139	0.0099

Note that for $K = 2$, $r = .8$, and $M = 8$, the differences in the mean are significant since the results are based on 50 runs and in this case $\frac{\sigma}{\sqrt{N}} = 0.0018$, well below the difference between the performance of S and D. Figure 4 provide the convergence results for that setting, showing that because they depend on the actions of the agents more closely S and D outperform G and that because it is aligned with the system objective, D outperforms S.

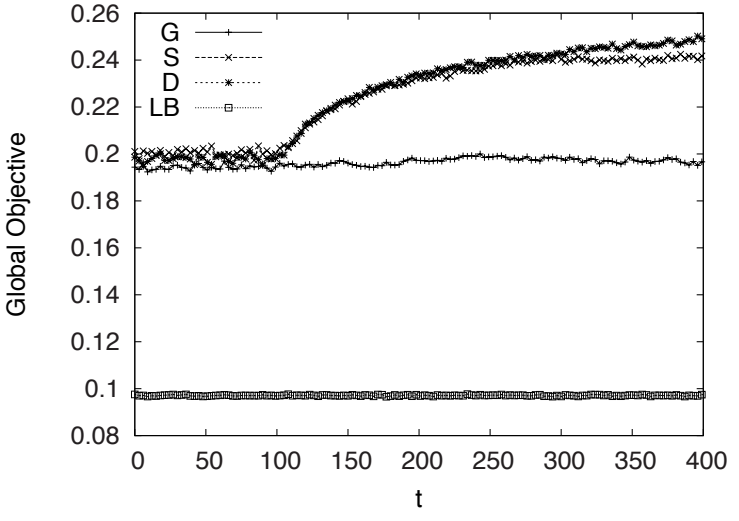


Fig. 4. Simulations results for 50 servers with 4 agents each with parameter values (r=0.8,M=8). For this hard problem, load balancing performs very poorly, and agents using tailored objectives S and D perform significantly better than agents using G.

6 Discussion

In this work we investigated how agent based algorithms can learn to effectively solve a multi-resource optimization problem involving networks of heterogeneous servers. Conventional approaches to this problems (e.g., as load balancing) work well when there is instantaneous, centralized control. For all but very few applications, this is an unreasonable assumption on the system's capabilities. Practical, heuristics based approaches on the other hand provide good solutions for the resource problems, but often break down in the more general, multi-resource optimization case.

The agent based solution we propose is based on assigning agents to each server whose actions are to determine whether a job requiring specific resources should be processed at that server or shipped to another server, and if so, to which other server. After a job is shipped, a new agent (residing at the new server) becomes responsible for that job. These decisions are based on agent objective functions (i.e., local goals) which are constructed to be aligned with the global objective and be directly impacted by the actions of an agent.

The results demonstrate that for particularly easy configurations, the agent-based methods do not outperform (and in fact underperform by a slight margin) a multi-resource version of load balancing. For moderately difficult problems, the agent based approaches start to outperform load balancing. In those cases, a multiagent system in which all the agents attempt to optimize the same global objective function only provide marginal improvements over conventional load

balancing. However, those marginal improvements are obtained without requiring a centralized controller (only requirement is for the global objective to be broadcast at regular intervals). Finally, for difficult problems, agents using the difference objective (D) outperform both team games (G), selfish agents (S) and load balancing (up to four times).

In this study we explored only cases where the number of resources is small (K=2 and K=4)allowing for an agent to be responsible for each permutation of resources (split into high-low for each resource). When the number of resources rises to preclude each agent being responsible for a particular permutation, job "types" need to be selected. This process can either be done using prior knowledge or by using correlations among the jobs. The key factor in achieving good results is in having both an appropriate number of agents in the system and in ensuring each agent has an action space that is appropriate for the task. Exploring how agents can be grouped or "teamed" at each server to provide good solutions is an intriguing avenue for future research.

Acknowledgments

The authors would like to thank David Wolpert for helpful discussions.

References

1. Agogino, A.K., Tumer, K.: Handling communication restrictions and team formation in congestion games. Journal of Autonomous Agents and Multi Agent Systems 13(1), 97–115 (2006)
2. Agogino, A.K., Tumer, K.: Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. Journal of Autonomous Agents and Multi Agent Systems 17(2), 320–338 (2008)
3. Agogino, A.K., Tumer, K.: Efficient evaluation functions for evolving coordination. Evolutionary Computation 16(2), 257–288 (2008)
4. Akkiraju, R., Keskinocak, P., Murthy, S., Wu, F.: An agent-based approach for scheduling multiple machines. Applied Intelligence 14(2), 867–872 (2005)
5. Boyan, J.A., Littman, M.: Packet routing in dynamically changing networks: A reinforcement learning approach. In: Advances in Neural Information Processing Systems, vol. 6, pp. 671–678. Morgan Kaufmann, San Francisco (1994)
6. Bredin, J., Maheswaran, R.T., Imer, C., Başar, T., Kotz, D., Rus, D.: Computational markets to regulate mobile-agent systems. Autonomous Agents and Multi-Agent Systems 6(3), 235–263 (2003)
7. Bredin, J., Maheswaran, R.T., Imer, C., Basar, T., Kotz, D., Rus, D.: A game-theoretic formulation of multi-agent resource allocation. In: Proceedings of the fourth International Conference of Autonomous Agents, pp. 349–356 (2000)
8. Buhler, P., Vidal, J.M.: Towards adaptive workflow enactment using multiagent systems. Information Technology and Management Journal 6(1), 61–87 (2005)
9. Byde, A.: A comparison between mechanisms for sequential compute resource auctions. In: AAMAS 2006: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, pp. 1199–1201. ACM Press, New York (2006)

10. Camille, B., Pierrick, P., Chaib-draa, B.: R-FRTDP: a real-time DP algorithm with tight bounds for a stochastic resource allocation problem. In: Proceedings of the 20th Canadian Conference on Artificial Intelligence, Montreal, Canada (May 2007)
11. Cheriton, D.R., Harty, K.: A market approach to operating system memory allocation. In: Clearwater, S.E. (ed.) Market-Based Control: A Paradigm for Distributed Resource Allocation. World Scientific, Singapore (1995)
12. Claus, C., Boutilier, C.: The dynamics of reinforcement learning cooperative multiagent systems. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, June 1998, pp. 746–752 (1998)
13. Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems, vol. 8, pp. 1017–1023. MIT Press, Cambridge (1996)
14. de Oliveira, D., Ferreira Jr., P.R., Bazzan, A.L.C.: A swarm based approach for task allocation in dynamic agents organizations. In: AAMAS 2004: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, pp. 1252–1253. IEEE Computer Society, Los Alamitos (2004)
15. Dorigo, M., Gambardella, L.M.: Ant colony systems: A cooperative learning approach to the travelling salesman problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
16. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure, 2nd edn. Morgan Kaufmann, San Francisco (2004)
17. Georgousopoulos, C., Rana, O.F.: Choosing a load balancing scheme for agent-based digital libraries. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330, pp. 51–62. Springer, Heidelberg (2006)
18. Ghosh, B., Muthukrishnan, S.: Dynamic load balancing in parallel and distributed networks by random matchings (extended abstract). In: SPAA 1994: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures, pp. 226–235. ACM Press, New York (1994)
19. Globus, A., Crawford, J., Lohn, J., Pryor, A.: Scheduling earth observing satellites with evolutionary algorithms. In: Proc. of International Conference on Space Mission Challenges for Information Technology, SMC-IT (2003)
20. Greenwald, A., Friedman, E., Shenker, S.: Learning in network contexts: Experimental results from simulations. Journal of Games and Economic Behavior: Special Issue on Economics and Artificial Intelligence 35(1/2), 80–123 (2001)
21. Heusse, M., Snyers, D., Guerin, S., Kuntz, P.: Adaptive agent-driven routing and load balancing in communication networks. Advances in Complex Systems 1, 237–254 (1998)
22. Hsiao, M.-T.T., Lazar, A.A.: Optimal flow control of multi-class queueing networks with decentralized information. In: IEEE Infocom 1989, pp. 652–661 (1987)
23. Huai, J., Wo, T., Liu, Y.: Resource management and organization in crown grid. In: InfoScale 2006: Proceedings of the 1st international conference on Scalable information systems, p. 10. ACM Press, New York (2006)
24. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275, 51–54 (1997)
25. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)

26. Kotz, D., Nieuwejaar, N.: Dynamic file-access characteristics of a production parallel scientific workload. In: Proceedings of Supercomputing 1994, Washington, DC, pp. 640–649. IEEE Computer Society Press, Los Alamitos (1994)
27. Kumar, S., Miikkulainen, R.: Dual reinforcement Q-routing: An on-line adaptive routing algorithm. In: Artificial Neural Networks in Engineering, vol. 7, pp. 231–238. ASME Press (1997)
28. Kurose, J.F., Simha, R.: A microeconomic approach to optimail resource allocation in distributed computer systems. IEEE Transactions on Computers 35(5), 705–717 (1989)
29. Leinberger, W., Karypis, G., Kumar, V., Biswas, R.: Load balancing across near-homogeneous multi-resource servers. In: Proceedings of the ninth heterogeneous Computing Workshop, Cancun, Mexico, pp. 61–70 (2000)
30. Li, Z., Parashar, M.: An infrastructure for dynamic composition of grid service. In: Proceedings of the 7th IEEE International Conference on Grid Computing, Barcelona, Spain, pp. 315–316. IEEE Computer Society Press, Los Alamitos (2006)
31. Lynden, S., Rana, O.F.: Coordinated learning to support resource management in computational grids. In: 2nd IEEE International Conference on Peer-2-Peer Computing, Linkoping, Sweden. IEEE Computer Society Press, Los Alamitos (2002)
32. Marbach, P., Mihatsch, O., Schulte, M., Tsisiklis, J.: Reinforcement learning for call admission control and routing in integrated service networks. In: Advances in Neural Information Processing Systems, vol. 10, pp. 922–928. MIT Press, Cambridge (1998)
33. Di Martino, B., Rana, O.F.: Grid performance and resource management using mobile agents. In: Getov, V., Gerndt, M., Hoisie, A., Malony, A., Miller, B. (eds.) Performance Analysis and Grid Computing. Kluwer, Dordrecht (2003)
34. Mishra, D., Rangarajan, B.: Cost sharing in a job scheduling problem using the shapley value. In: EC 2005: Proceedings of the 6th ACM conference on Electronic commerce, pp. 232–239. ACM Press, New York (2005)
35. Newman, M.E.J.: Models of the small world (a review). Journal of Statistical Physics 101, 819–841 (1987)
36. Shirazi, B.A., Hurson, A.R., Kavi, K.M.: Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos (1995)
37. Stone, P.: Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge (2000)
38. Stone, P.: TPOT-RL applied to network routing. In: Proceedings of the Seventeenth International Machine Learning Conference, pp. 935–942. Morgan Kauffman, San Francisco (2000)
39. Stone, P., Veloso, M.: Multiagent systems: A survey from a machine learning perspective. Autonomous Robots 8(3), 345–383 (2000)
40. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
41. Tang, H., Tianfield, H.: Self-organizing networks of communications and computing. International Transactions on Systems Science and Applications 1(4), 421–431 (2006)
42. Tianfield, H., Unland, R.: Towards self-organization in multi-agent systems and grid computing. Multiagent and Grid Systems 1(2), 89–95 (2005)
43. Tumer, K., Agogino, A.: Distributed agent-based air traffic flow management. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems, Honolulu, HI, May 2007, pp. 330–337 (2007)

44. Tumer, K., Agogino, A., Wolpert, D.: Learning sequences of actions in collectives of autonomous agents. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, July 2002, pp. 378–385 (2002)
45. Tumer, K., Wolpert, D.H.: Collective intelligence and Braess' paradox. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence, Austin, TX, pp. 104–109 (2000)
46. Watts, D.: Small Worlds. Princeton University Press, Princeton (1999)
47. Wolpert, D.H., Tumer, K.: Optimal payoff functions for members of collectives. Advances in Complex Systems 4(2/3), 265–279 (2001)
48. Xuan, P., Lesser, V., Zilberstein, S.: Communication decisions in multi-agent cooperation: Model and experiments. In: Proceedings of the Fifth International Conference on Autonomous Agents, pp. 616–623. ACM Press, New York (2001)
49. Yu, X., Ram, B.: Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups. International Journal of Production Research 44(22), 4793–4813 (2006)

Author Index