Workshop 19: Adaptive and Learning Agents and Multi-Agent Systems

Franziska Klügl
Karl Tuylos
Sandip Sen
(Editors)
Preface

As agent-based systems get larger and more complex, there is a compelling need for agents to learn and adapt to their dynamic environments. Indeed, how to adaptively control, coordinate and optimize adaptive multiagent systems is one of the emerging multi-disciplinary research areas today. Such systems are often deployed in real-world situations with stochastic environments where agents have limited perception and communication capabilities. Furthermore, in a number of distributed domains without centralized control, different agents will have different behaviours, capabilities, learning strategies, etc. There is a pressing need, then, to both, study and develop the convergence of multiple learners using the same learning scheme as well as understand the emergent dynamics of multiple learners with varying learning schemes.

This workshop will explore all agent learning approaches, with particular emphasis on multiagent settings where the scale and complexity of the environment require novel learning techniques. The goal of this workshop is to bring together not only scientists from different areas of computer science, e.g., agent architectures, reinforcement learning, evolutionary algorithms but also from different fields studying similar concepts, e.g., game theory, bio-inspired control, mechanism design.

Whereas research in machine learning involving single agents is as old as the field of computational intelligence itself, interest in studying the techniques for and dynamics of multiple concurrent learners began around mid 1990’s. To encourage discussion and research on these issues, a workshop on “Adaptation and Learning in Multiagent Systems” was organized in association with IJCAI-95 in Montreal, Canada. Since then a number of workshops and symposia have been held on the topic and several journal special issues have also been published. It is particularly heartening for us to note that research papers on agent and multiagent learning are a regular feature in most major AI and machine learning conferences including AAAI, IJCAI, ICML, ECML, AAMAS, NIPS, etc. Several satellite workshops focusing on this topic have also been organized in conjunction with these premier international conferences, with ALAg 2007 as a most recent highlight at an AAMAS conference. ALAg 2007 has been organized with the idea of bringing these workshops into some explicitly organized form.

Concurrently, the Adaptive Learning Agents and Multiagent Systems (ALAMAS) workshop has established as a series in Europe with yearly editions starting from 2001. The focus here was also on different facets of learning and adaptation in the multiagent world. In early April 2007, the seventh edition of ALAMAS was organized in Maastricht (NL) as a stand alone event. Mid of 2007, it was decided to organize an event merging these two different workshop series, at least when AAMAS is being organized in Europe. Our goal was strengthening the agent learning community by combining these parallel efforts into one workshop and giving them a unique platform for presentation and exchange of ideas. The somehow cumbersome title of this workshop ALAMAS&ALAg indicates this embracing idea. We hope that this will trigger also discussions about how to consolidate the organization of events devoted to learning and adaptation in agent and multiagent systems in the future. We believe that it is an opportune time to put together a yearly forum that would bring together all active researchers in this area and foster lively discussions, debate, and cross-fertilization of ideas.
We thank all authors who responded to our call-for-papers with interesting contributions. We look forward to a lively workshop with informative discussions and constructive exchange of ideas. We are thankful to the members of the program committee for the quality and sincerity of their efforts and service. We also thank the AAMAS conference for providing us a platform for holding this event. We look forward to the participation of all the attendees to the workshop to make this ALAMAS&ALAg event a fruitful and memorable experience.

Franziska Klügl, Karl Tuyls and Sandip Sen
ALAMAS&ALAg 2008 Co-Chairs
Organization

Franziska Klügl  
Department of Artificial Intelligence and Applied Computer Science  
Julius-Maximilians-Universität Würzburg  
kluegl@informatik.uni-wuerzburg.de

Karl Tuyls  
Department of Industrial Design  
Eindhoven University of Technology  
k.p.tuyls@tue.nl

Sandip Sen  
Department of Mathematical & Computer Sciences  
The University of Tulsa  
sandip-sen@utulsa.edu

Program Committee

Bikramjit Banerjee, The University of Southern Mississippi, USA  
Ana L.C. Bazzan, UFRGS, Porto Alegre, BR  
Zahia Ghuessoum, University of Paris 6, FR  
Amy Greenwald, Brown University, USA  
Daniel Kudenko, University of York, UK  
Akira Namatame, National Defense Academy, Japan  
Ann Nowe, Vrije Universiteit Brussels, BE  
Liviu Panait, Google Inc Santa Monica, USA  
Lynne Parker, University of Tennessee, USA  
Enric Plaza, Institut d'Investigació en Intel·ligència Artificial, Spain  
Jeffrey Rosenschein, The Hebrew University of Jerusalem  
Michael Rovatsos, Centre for Intelligent Systems and their Applications, UK  
Matt Taylor, The University of Texas at Austin, USA  
Goran Trajkovski, South University, Savannah, GA, USA  
Kagan Tumer, Oregon State University, USA  
Katja Verbreek, KaHo Sint-Lieven, Belgium
# Table of Contents

**An Intelligent USD/JPY Trading Agent**  
*R. P. Barbosa, O. Belo*  
1

**Plan-based Reward Shaping for Reinforcement Learning**  
*M. Grzes, D. Kudenko*  
9

**Learning Potential for Reward Shaping in Reinforcement Learning with Tile Coding**  
*M. Grzes, D. Kudenko*  
17

**A multiagent approach to hyperredundant manipulators**  
*D. Hennes, K. Tumer, K. Tuyls*  
25

**Learning to Control the Emergent Behaviour of a Multi-agent System**  
*U. Richter, M. Mnif*  
33

**Teamwork decision-making with MDP and BDI**  
*P. Trigo, H. Coelho*  
41

**Incorporating Learning in BDI Agents**  
*S. Airiau, L. Padham, S. Sardina, S. Sen*  
49

**Criteria for Consciousness in Artificial Intelligent Agents**  
*R. Arrabales, A. Ledezma, A. Sanchis*  
57

**A Reinforcement Learning Approach to Setting Multi-Objective Goals for Energy Demand Management**  
*Y. Guo, A. Zeman, R. Li*  
65

**Learning When to Take Advice: A Statistical Test for Achieving A Correlated Equilibrium**  
*G. Hines, K. Larson*  
73

**Norm emergence with biased agents**  
*P. Mukherjee, S. Sen, S. Airiau*  
81

**Bio-inspired Cognitive Architecture for Adaptive Agents based on an Evolutionary Approach**  
*O. J. Romero López, A. de Antonio Jiménez*  
89

**Multi-Agent Reinforcement Learning for Intrusion Detection: A case study and evaluation**  
*A. Servin, D. Kudenko*  
97

**Multi-Agent Learning with a Distributed Genetic Algorithm: Exploring Innovation Diffusion on Networks**  
*F. Stonedahl, W. Rand, U. Wilensky*  
105

**Transferring Instances for Model-Based Reinforcement Learning**  
*M. E. Taylor, N. K. Jong, P. Stone*  
113

**Author Index**  
119
An Intelligent USD/JPY Trading Agent

Rui Pedro Barbosa
Department of Informatics
University of Minho
4710-057 Braga, Portugal
+351 913854080
ru.barbosa@di.uminho.pt

Orlando Belo
Department of Informatics
University of Minho
4710-057 Braga, Portugal
+351 253604476
obelo@di.uminho.pt

ABSTRACT
In this paper we describe the implementation of an intelligent agent capable of autonomously trading the USD/JPY currency pair using a 6 hours time frame. The agent has 3 major components: an Ensemble Model, a Case-Based Reasoning System and a Rule-Based Expert System. Each of these components carries out a different task in the agent’s trading decision process. The Ensemble Model is responsible for performing pattern recognition and predicting the direction of the exchange rate. The Case-Based Reasoning System enables the agent to learn from empirical experience, and is responsible for suggesting how much to invest in each trade. Finally, the Rule-Based Expert System enables the agent to incorporate non-experiential knowledge in its trading decisions. We used 12 months of out-of-sample data to verify the profitability of the agent. Over this period, it performed 826 simulated trades and obtained an average profit per trade of 6.88 pips. It accurately predicted the direction of the USD/JPY price in 54.72% of the trades, 65.74% of which were profitable. The agent was integrated with an Electronic Communication Network and has been trading live for the past several months. So far its live trading results are consistent with the simulated results, which lead us to believe this research might be of practical interest to the trading community.

Categories and Subject Descriptors

General Terms
Algorithms, Economics, Experimentation.

Keywords
Forex trading, hybrid agent, autonomy.

1. INTRODUCTION
Trading in financial markets is undergoing a radical transformation, one in which quantitative methods are continuously becoming more important. This transformation is particularly noticeable in the Forex Market, where the adoption of algorithmic trading is expected to grow from 7% by the end of 2006 to 25% by 2010 [1]. The development of intelligent agents that can act as autonomous traders seems like a logical step forward in this move away from traditional methods, often referred to as the “algorithms arms race”. With this in mind, in this paper we will describe the development of an autonomous trading agent that makes extensive use of artificial intelligence models. The idea of using artificial intelligence models in trading is not really new, as there are already plenty of studies in this field. A special emphasis has been given to the use of neural networks to perform financial time series prediction [4][7][11]. Several studies have shown that neural networks can model financial time series better than traditional mathematical methods [6][8]. Lately, researchers have displayed a growing interest in the development of hybrid intelligent systems for financial prediction [5][9][12]. These studies have shown that, in general, hybrid systems can outperform non-hybrid systems.

Even though most studies seem to show that artificial intelligence models can produce reasonably accurate financial predictions, that in itself will not impress most traditional traders. These studies usually measure a model’s performance based on its accuracy (for classification) or the mean squared error (for regression). The problem with this approach, from a trader’s point of view, is that higher accuracy does not necessarily translate into higher profit. A single losing trade can wipe out the profit of several accurately predicted trades. A low mean squared error is also far from being a guarantee that a model can produce profitable predictions [3]. Some studies try to tackle this problem by using model predictions on out-of-sample data to simulate trades. This might make for a better study from a traders’ point of view, but it is still not a perfect solution. Simulated trades do not account for problems that frequently occur while trading live, such as slippage and partial fills. The effect of these problems on the overall profitability of a trading strategy is not negligible.

In the end, profit and drawdown are the only performance gauges that really matter to the trading community. Any performance claims are also expected to be backed up by a meaningful track record of live trading. Our research will be exclusively directed at the expectations and requirements imposed by the trading community. We will describe the development of a USD/JPY trading agent whose main goal is to maximize the profit and to minimize the drawdown while trading live. Unlike most studies in this field, which describe tools that can be used to aid the traders, we will be looking at a solution that can actually replace the traders. This means the agent should be able to operate autonomously, placing trades and handling money management without requiring human intervention. The agent’s structure is loosely based in the decision process of a traditional trader: it can intuitively recognize patterns in financial time series and predict the direction of the price, it can remember previous trades and use that empirical knowledge to decide when and how much to invest, and it can incorporate knowledge from trading books and trading experts into its trading decisions.

2. FOREX MARKET
The Forex Market is the largest financial market in the world. In this market currencies are traded against each other, and each pair
of currencies is a product that can be traded. For instance, USD/JPY is the price of the United States Dollar expressed in Japanese Yen. At the time of writing of this paper the USD/JPY price is 106.25, meaning we need 106.25 JPY to buy 1 USD. Trading this pair in the Forex Market is pretty straightforward: if a trader believes the USD will become more valuable compared to the JPY he buys USD/JPY lots (goes long), and if he thinks the JPY will become more valuable compared to the USD he sells USD/JPY lots (goes short). The profit/loss of each trade can be expressed in pips. A pip is the smallest change in the price of a currency pair. For the USD/JPY pair a pip corresponds to a price movement of 0.01. The actual value of each pip depends on the amount invested. For example, if we buy/sell 100,000 USD/JPY each pip is worth 1,000 JPY (100,000 times 0.01), or 9.41 USD (1,000 divided by 106.25).

3. USD/JPY TRADING AGENT

We will describe the implementation of an agent with the ability to place a trade every 6 hours, from Sunday 18:00 GMT to Saturday 00:00 GMT, using the USD/JPY currency pair. The agent’s structure is represented in Figure 1. It has two percepts (price changes over a period of time and result of previous trades) and is capable of a single action (placement of new trades). Its structure is composed of three interconnected components:

- **Ensemble Model** – it consists of several classification and regression models that try to find hidden patterns in price data, and is responsible for predicting if the price of the USD/JPY currency pair will go up or down.
- **Case-Based Reasoning System** – each case in this system corresponds to a previously executed trade and its final result (profit or loss in pips). This empirical information is used to suggest when and how much to invest in each trade.
- **Rule-Based Expert System** – contains several rules regarding when to invest and when to stop a trade. This system is responsible for making the final trade decisions, using the predictions from the Ensemble Model and the suggestions from the Case-Based Reasoning System. Its rules need to be provided to the agent by trading experts because the agent would not be able to learn them by itself while trading.

3.1 Agent’s Intuition

To be able to know when to buy or sell the USD/JPY currency pair, the agent will need to intuitively guess if the pair’s price is going up or down in the near future. A common definition for intuition is “knowing without reasoning”. It is hard to explain how this mental process works and even harder to try to implement its software equivalent. In a loose way we can look at intuition as a complex pattern recognition process [2]. Even if we are oversimplifying a complex concept, that definition perfectly suits our needs. We can easily base our trading agent’s intuition in an Ensemble of classification and regression models capable of finding hidden patterns in nonlinear financial data.

Before deciding which models will be part of the Ensemble Model, we need to obtain historical price data that can be used to train them. This type of data is freely available on the Internet. ¹

¹ We used price data downloaded from www.dukascopy.com and www.oanda.com.
model in the Ensemble. All the models were trained and tested using the Weka data mining software. 2

Table 1. Attributes used to train each model in the Ensemble.

<table>
<thead>
<tr>
<th>Model</th>
<th>Attributes</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance-Based K*</td>
<td>hour (nominal) day of week (nominal) last 6 returns moving average current class</td>
<td>next class</td>
</tr>
<tr>
<td>C4.5 Decision Tree</td>
<td>hour (nominal) day of week (nominal) last 6 returns moving average current class</td>
<td>next class</td>
</tr>
<tr>
<td>RIPPER Rule Learner</td>
<td>hour (nominal) day of week (nominal) last 6 returns moving average current class</td>
<td>next class</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>hour (nominal) day of week (nominal) current return</td>
<td>next class</td>
</tr>
<tr>
<td>Logistic Decision Tree</td>
<td>hour (nominal) day of week (nominal) last 6 returns moving average current class</td>
<td>next class</td>
</tr>
<tr>
<td>Instance-Based K*</td>
<td>hour (nominal) day of week (nominal) last 6 returns moving average current class</td>
<td>next return</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>hour (numeric) day of week (numeric) last 10 returns moving average current return</td>
<td>next return</td>
</tr>
</tbody>
</table>

The models were trained with attributes such as the hour, the day of the week and the current class or return. We also tried several attributes regularly used in technical analysis by traditional traders, such as moving averages, the Relative Strength Index, the Williams %R and the Average Directional Index, amongst others. Of these, only the moving averages added predictive power to the models. The usefulness of the moving averages was not unexpected, as it had already been demonstrated by several studies in the past [10].

In order to make the agent autonomous, the models in its Ensemble need to be periodically retrained with more data. To accomplish this, before each prediction the available instances are divided into two datasets: the test set consisting of the most recent 100 instances, and the training set consisting of all the instances left. Using these two sets of data the following sequence of steps is applied to each model in the Ensemble:

1. The model is retrained using the training set and tested using the test set.
2. For each instance in the test set a trade is simulated (if the model predicts “the price will go up in the next 6 hours” we simulate a buy, otherwise we simulate a short sell). The results from the simulation are used to calculate the overall profit factor, long profit factor and short profit factor of the retrained model:

   \[
   \text{Overall PF} = \frac{\sum \text{pips won}}{\sum \text{pips lost}} - 1 \tag{1}
   \]

   \[
   \text{Long PF} = \frac{\sum \text{pips won when predicting up}}{\sum \text{pips lost when predicting up}} - 1 \tag{2}
   \]

3. If the overall profit factor of the retrained model is higher or equal to the overall profit factor of the original model, then the retrained model replaces the original model in the Ensemble. Otherwise, the original is kept and the retrained model is discarded.

4. The selected model makes its prediction: if it predicts “the price will go up in the next 6 hours” the weight of its vote is its long profit factor; otherwise, if it predicts “the price will go down in the next 6 hours”, the weight of its vote is its short profit factor. If the weight is a negative number then it is replaced with zero, which effectively means the model’s prediction is ignored.

After all individual models have made their predictions, the ensemble prediction is calculated by adding the votes of all the models that predicted “the price will go up in the next 6 hours” and then subtracting the votes of all the models that predicted “the price will go down in the next 6 hours”. If the ensemble prediction is greater than zero then the final class prediction is “the price will go up in the next 6 hours”, otherwise if it is lower than zero the final prediction is “the price will go down in the next 6 hours”.

There are several reasons why we decided to perform the predictions using an Ensemble Model and the previously described algorithm:

- Some models are more profitable under certain market conditions than others. An Ensemble Model can be more profitable than any of its individual models because it can adapt to the market conditions. That is accomplished by continuously updating the weight of the vote of each of the individual models: as a model becomes more profitable its vote becomes more important.

- Some models are better at predicting when the market will go up and others are better at predicting when the market will go down. By using an Ensemble Model we can combine the qualities of the best models at predicting long trades and the best models at predicting short trades. That is accomplished by using the models’ long profit factor and short profit factor as their votes’ weight.

- An Ensemble Model makes our trading strategy resilient to changes in market dynamics. If a single classification or regression model is used for prediction and it starts turning unprofitable, the trading strategy will soon become a disaster. On the other hand, if that model is a part of our Ensemble Model, as it becomes unprofitable its vote continuously loses weight up to a point where its predictions are simply ignored. And since our strategy tries to improve the models by retraining them with more data as it becomes available, it is very likely that the unprofitable model will end up being replaced with a more profitable retrained version of itself.

- Our algorithm optimizes profitability instead of accuracy. Obviously the learning algorithms used to retrain the models still optimize their accuracy, but the decision to actually make the retrained models a part of the Ensemble Model is based entirely on their profitability.

- Retraining the models before each prediction is the key to our agent’s autonomy. The agent can keep learning even

\[\text{Short PF} = \frac{\sum \text{pips won when predicting down}}{\sum \text{pips lost when predicting down}} - 1 \tag{3}\]

2 Weka is an open source data mining software available at www.cs.waikato.ac.nz/ml/weka.
while trading, because new unseen data will eventually become a part of the training set.

Our strategy is not without faults though. The decision to replace an original model with a version of itself trained with more data is based on the simulated profitability displayed with the test data. This means we are selecting models based on their test predictions, which might lead to selecting models that overfit the test data. However, this ends up not being a very serious problem, because our algorithm eventually replaces unprofitable models with more profitable retrained versions of themselves (that might or might not overfit a different set of test data).

The decision to use only 100 instances for testing the models in the Ensemble might seem a bit odd, as most literature regarding supervised learning would recommend the use of at least 30% of the available data. However, there are several reasons why we made our agent use such a small set of test data:

- Usually we would need a lot of test data to make sure a model did not overfit the training data. Our agent does not need that because its predictions are not based in a single model. So even if one of its models overfits the training data, that is not necessarily a problem. Over time the agent is able to ignore models that overfit the data (i.e., models that are unprofitable in out-of-sample trading) and eventually replaces them with retrained versions of themselves. That is the reason why we can save much needed data for training, which would otherwise be required for testing.

- Heteroskedasticity is a key feature of most economic time series. This means that volatility is clustered: usually a long period of low volatility is followed by a short period of high volatility and this pattern is repeated ad eternum. Since the weights of the models’ votes are based in their simulated profitability using the test instances, we need to keep the test set small enough that the weights can adapt quickly when the market enters a period of high volatility. In other words, the shorter the test set, the faster the agent can adapt to changes in market dynamics.

- A new instance is available after each trade. This instance becomes a test instance, and the oldest instance in the test set becomes a training instance. This means that, as time goes by, the training set grows while the test set remains the same size and moves like a sliding window. What this implies is that the shorter the test set, the faster the new instances can be used for training. In other words, the shorter the test set, the faster the agent can learn new patterns.

We used the predictions from the Ensemble Model component to simulate trades using out-of-sample data corresponding to the period from February 2007 to January 2008. The accumulated profit in pips over this period is displayed in Figure 2. After an initial period of unprofitable trading, where the weights of the models’ votes in the Ensemble were adapting to the market conditions, this component was able to recover and ended up with a profit of 4,238 pips after 1,038 trades. Although these results are pretty good, this trading strategy still needs some improvement because its drawdown is a little high (355 pips).

Figure 2. Ensemble Model accumulated profit.

The chart in Figure 3 casts some light into the way the Ensemble Model component is able to adapt to changing market conditions. It shows the average long and average short weights of the votes of the 7 models in the Ensemble, and the USD/JPY price change over the out-of-sample period.

Figure 3. Average weight of the models’ votes.

As the price trends up the long votes’ average weight increases, while the short votes’ average weight shrinks, and vice-versa. The grayed out periods in the chart are particularly interesting. Over these periods, the average weight for long votes is very close to zero. What this means is that models predicting that “the price will go up in the next 6 hours” are being ignored. So if a single model with short profit factor greater than zero predicts “the price will go down in the next 6 hours”, then the final ensemble class prediction will automatically be the same, even if the other 6 models predict a price increase. It is this mechanism of selecting the best models according to the market conditions that allows the Ensemble Model component to quickly adapt to changes in the price trend and volatility.

3.2 Agent’s Empirical Knowledge

Deciding when to buy or short sell a financial instrument is a very important part of successful trading. But there is another equally important decision: how much to invest in each trade. If we have a model that consistently produces profitable predictions we might feel tempted to double our investment per trade. That will in fact double the profit, but will also double the exposure and the drawdown (loosely defined as the maximum loss an investor should expect from a series of trades). Keeping the drawdown low is of vital importance to traditional traders because, no matter how profitable a trading strategy is, a large drawdown can cause a margin call and pretty much remove the trader from the market. So doubling the investment per trade is not the best money management strategy for our trading agent. A better way to increase the profitability without a proportional increase in the
risk would be to double the investment in trades with high expected profitability, use the normal investment amount for trades with average expected profitability and skipping trades with low expected profitability.

In order to determine the expected profitability of a trade we will be looking at the individual predictions of the models that are part of the Ensemble Model. Intuitively, we might expect that the probability of a trade being successful will be higher if all the individual models make the same prediction (all predict “the price will go up in the next 6 hours” or all predict “the price will go down in the next 6 hours”), compared to a trade where the models’ predictions are mixed (some predict “the price will go up in the next 6 hours” and some predict “the price will go down in the next 6 hours”). Empirical evidence demonstrates that those expectations are well founded. Certain combinations of individual predictions really are more profitable than others. Our agent’s money management strategy is based on that empirical observation.

The agent uses a Case-Based Reasoning System to decide how much to invest in a trade. Each case in this system represents a trade previously executed by the agent, and contains the following information: the predicted class, the trade result (profit or loss in pips) and the individual predictions from the models in the Ensemble Model. The agent uses the empirical information it gathers from these cases to calculate the expected profitability of a trade before it is placed. It then decides if a trade is worth opening, and if so how much should be invested. To accomplish this, the following sequence of steps is executed before each trade is placed:

1. The Ensemble Model makes the ensemble prediction and sends the sequence of individual predictions from its models to the Case-Based Reasoning System. This system retrieves from its database all the cases with the same class prediction and the same sequence of individual predictions.
2. If it can’t retrieve at least 7 cases from the its database, the Case-Based Reasoning System removes the last prediction in the sequence of individual predictions and retrieves the cases again. This process is repeated until enough cases are retrieved.
3. The Case-Based Reasoning System calculates the overall profit factor of the retrieved cases using Equation (1). That is the expected profitability of the trade.
4. If the overall profit factor of the retrieved cases is greater or equal to 1 the agent doubles the investment; if it is lower or equal to 0 the agent skips the trade; otherwise, the regular investment amount is used.

After a trade is executed and closed, a new case is inserted in the Case-Based Reasoning System database. The agent uses the overall profit factor of the matching cases in the database to make the money management decision, which is yet another way in which it tries to optimize the profit.

Figure 4 shows the result of combining the predictions of the Ensemble Model component and the money management strategy of the Case-Based Reasoning System to simulate trades using the out-of-sample data.

This combination of the two components performed 826 trades, with a final profit of 5,336 pips and a drawdown of 650 pips. There was a very interesting decrease of 25.6% in the number of trades, if we compare these results with the ones obtained by the Ensemble Model component alone. However, even though there was an increase in the profit, this strategy still needs improvement because its drawdown is too high.

3.3 Agent’s Expert Knowledge

No matter how “smart” our USD/JPY agent is, there is still some trading knowledge it will not be able to pick up from its empirical trading experience. For this reason, the final trading decisions are taken by a Rule-Based Expert System, where important rules can be defined by trading experts.

Some of these rules can be quite simple. For example, we may want the agents to skip trades in low liquidity days, such as those around Christmas or New Year’s Day, when the already naturally high volatility of the Forex Market is exacerbated. Or we may want them to skip trades whenever major economic reports are about to be released, to avoid the characteristic chaotic price movements that happen right after the release. The primary example of such a report is the United States Nonfarm Payrolls, or NFP, released on the first Friday of every month.

Other more important rules are those where the settings for take profit orders and stop loss orders are defined. These are necessary so that the agent knows when to exit each trade. A take profit order is used to close a trade when it reaches a certain number of pips in profit, to guarantee that profit. A stop loss order is used to close a trade when it reaches a certain number of pips of loss, to prevent the loss from widening. Considering the historical volatility of the USD/JPY pair, we defined a rule that will certainly have a significant impact in the overall profitability of the agent: each trade is accompanied by a take profit order of 20 pips. This means that whenever a trade reaches a profit of 20 pips it is automatically closed. In other words, we are capping our maximum profit per trade to 20 pips (40 pips when the investment is doubled). A trade that is not closed with the take profit order will only be closed when the 6 hours period ends and a new trade is opened.

Before each trade the Rule-Based Expert System component receives the prediction from the Ensemble Model and the suggested investment amount from the Case-Based Reasoning System. It then uses the rules defined by expert traders to make the final decision regarding the trade direction, investment amount and exit conditions. The agent can be made completely
autonomous by using a broker’s proprietary API to send the final trade decisions directly into the market.

Figure 5 shows the results of combining the Ensemble Model and the Rule-Based Expert System to simulate trades using the out-of-sample data.

![Figure 5](image)

Figure 5. Ensemble Model and Rule-Based Expert System accumulated profit.

This strategy netted 4,287 pips of profit, with a drawdown of 261 pips. Compared to using the Ensemble Model component alone there was a 1.2% increase in the profit and a 36.0% decrease in the drawdown. The lower drawdown is exactly what we needed, but unfortunately there is also a big profit reduction if we compare these results with the ones obtained with the combination between the Ensemble Model and the Case-Based Reasoning System.

3.4 Results

Through simulation, we have shown that each of the agent’s components makes a different contribution to the trading profit and the drawdown. The actual agent consists of all the three components working together. Figure 6 shows the agent’s simulated trading results. The agent was able to reconcile the Case-Based Reasoning System capacity to increase the profits and the Rule-Based Expert System capacity to reduce the drawdown. It obtained a final profit of 5,687 pips with a drawdown of 421 pips. This is, by any standards, an excellent performance.

![Figure 6](image)

Figure 6. Trading agent’s accumulated profit.

The chart in Figure 7 shows the USD/JPY price movement during the simulation period, and the comparison between the agent’s performance and the performance of each combination of its components. It is easy to see that not only is the agent more profitable than any combination of its components, its profit curve is also the smoothest. We can also verify that the agent is not directionally biased: it is profitable no matter if the USD/JPY price is going up or down. It is also important to note that the agent performed acceptably in periods of high volatility (such as the month of August 2007).

![Figure 7](image)

Figure 7. Performance comparison.

Table 2 resumes the trading statistics of both the agent and the components combinations. The first interesting statistic in this table is the fact that the Ensemble Model can only predict if the price will go up or down with 54.14% accuracy. This percentage might seem too low, but it makes sense when we consider that this component optimizes profitability instead of accuracy. Therefore, even though the component is not very accurate, the profit it obtains from the accurately predicted trades is a lot higher than the losses it suffers from incorrectly predicted trades. Its success rate, i.e., the percentage of trades that are closed in profit, is equal to its accuracy because all the trades are closed at the end of the 6 hours period, when a new trade is entered.

The 54.72% accuracy of the agent is higher than the accuracy of its Ensemble Model because both its Case-Based Reasoning System and its Rule-Based Expert System can make it skip trades that are expected to be unprofitable. That explains why the agent did only 826 trades, against the 1,038 trades that would have been performed by the Ensemble Model component alone. The agent has a 65.74% success rate, which is considerably higher than its accuracy. That is due to the take profit rule in the Rule-Based Expert System. This rule allows the agent to be profitable even if it makes a wrong prediction, just as long as the price moves at least 20 pips in the predicted direction.

<table>
<thead>
<tr>
<th>Components</th>
<th>Accuracy</th>
<th>Success</th>
<th>Profit</th>
<th>Drawdown</th>
<th>Trades</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensemble</td>
<td>54.14%</td>
<td>54.14%</td>
<td>4,238</td>
<td>355</td>
<td>1,038</td>
</tr>
<tr>
<td>Ensemble + CBR System</td>
<td>54.72%</td>
<td>54.72%</td>
<td>5,336</td>
<td>650</td>
<td>826</td>
</tr>
<tr>
<td>Ensemble + Expert system</td>
<td>54.14%</td>
<td>65.13%</td>
<td>4,287</td>
<td>261</td>
<td>1,038</td>
</tr>
<tr>
<td>Agent</td>
<td>54.72%</td>
<td>65.74%</td>
<td>5,687</td>
<td>421</td>
<td>826</td>
</tr>
</tbody>
</table>

While pips are a good way to measure the performance of our Forex trading agent, it might be interesting to see how that performance translates into actual money won or lost. Forex investments are usually leveraged (which means they are done with borrowed funds), so the total profit obtained by the agent will always depend on the size of its trades. Let us assume we have a starting capital of $100,000, and we want our agent to use a low risk trading strategy, with trades of 100,000 USD/JPY. As long as the agent has more than $100,000 in its account its trades will not be leveraged, except when it doubles the investment for trades with high expected profitability. As previously seen, for a USD/JPY price of 106.25, the pip value for a 100,000 USD/JPY
trade will be $9.41. Since our agent obtained a total profit of 5,687 pips, its profit in dollars after 12 months of trading is $53,515, or 53.5%. This is a really good performance, but things get even more interesting if we consider the agent could have used a higher initial leverage. Figure 8 displays the equity curves for a $100,000 account, using different trade sizes.

Amazingly, if the agent used a standard trade size of 2,000,000 USD/JPY, its $100,000 account would have grown to $1,170,293 in 12 months. However, it is easy to see why using such high leverage would be too risky in live trading. From July 8th to July 11th the agent suffered its maximum drawdown of 421 pips. A trade size of 2,000,000 USD/JPY corresponds to $188.2 per pip, so there was a drawdown of $79,232. This loss is barely noticeable in the equity curve displayed in Figure 8, because it happened at a time when the agent had already a really high account balance. But let us imagine the agent placed its first trade on July 8th. Its initial balance of $100,000 would then drop $79,232 in 3 days, and depending on the broker and the market fluctuations, the agent might suffer a margin call and end up with a loss of more than 80%, and be unable to trade again. If the agent was using a more reasonable trade size of 500,000 USD/JPY, the maximum drawdown would have been only $19,808, and it would have turned $100,000 into $367,573 in 12 months.

As we mentioned before, simulated results can give us a general idea regarding an agent’s ability to be profitable while trading live, but cannot provide any guarantees. There are many details concerning live trading that can have a tremendous impact in the final net profit. The only way to prove that an agent can be profitable is to allow it to create an extensive track record of live trading. In order to accomplish this we integrated our agent with an Electronic Communication Network, where it has been trading autonomously since October 2007. As expected, the agent’s actual live trading results are not as good as the simulated results, with a decrease of around 20% in the total profit. This difference is due to commissions, slippage, partial fills and interest payments, amongst other things. But the agent’s results are still very good, with an average profit of 5.5 pips per live trade, which compares with an average profit of 6.9 pips per simulated trade over the same period of time.

4. FINAL REMARKS

In this paper we described the implementation of an agent with the ability to autonomously trade the USD/JPY currency pair. The agent’s structure is loosely based in traditional trading, i.e., it enables the agent to:

- learn from previous trades,
- incorporate knowledge obtained from non-experiential sources into its trading strategy.

Each of these capabilities corresponds to a component in the agent’s structure. These components are, respectively, an Ensemble Model, a Case-Based Reasoning System and a Rule-Based Expert System.

Using simulated trading we were able to demonstrate the positive impact of each of the agent’s components in the trading profit. Live trading results seem to suggest that the agent is indeed capable of being profitable while trading without supervision. However, only after a couple of years will we be able to make any claims regarding the agent’s ability to survive and thrive in all market conditions.

A common way to reduce the risk inherent to trading is through diversification. In our case, investment diversification can be easily achieved by implementing a basket of agents trading different uncorrelated currency pairs and using different time frames. We are currently looking into this multi-agent investment strategy. Given the growing interest in algorithmic and quantitative trading, it is our belief that it will be of much interest to the traditional trading community.

5. REFERENCES

Plan-based Reward Shaping for Reinforcement Learning

Marek Grzes, Daniel Kudenko
Department of Computer Science
University of York
York, YO10 5DD, UK
{grzes,kudenko}@cs.york.ac.uk

ABSTRACT
Reinforcement learning, while being a highly popular learning technique for agents and multi-agent systems, has so far encountered difficulties when applying it to more complex domains due to scaling-up problems. This paper focuses on the use of domain knowledge to improve the convergence speed and optimality of various RL techniques. Specifically, we propose the use of high-level STRIPS operator knowledge in reward shaping to focus the search for the optimal policy. Empirical results show that the plan-based reward shaping approach outperforms other RL techniques, including alternative manual and MDP-based reward shaping when it is used in its basic form. We show that MDP-based reward shaping may fail and successful experiments with STRIPS-based shaping suggest modifications which can overcome encountered problems. The STRIPS-based method we propose allows expressing the same domain knowledge in a different way and the domain expert can choose whether to define an MDP or STRIPS planning task. We also evaluate the robustness of the proposed STRIPS-based technique to errors in the plan knowledge.

1. INTRODUCTION
Reinforcement learning (RL) is a popular method to design autonomous agents that learn from interactions with the environment. In contrast to supervised learning, RL methods do not rely on instructive feedback, i.e., the agent is not informed what the best action in a given situation is. Instead, the agent is guided by the numerical reward which defines the optimal behaviour for solving the task. The problem with this kind of numeric guidance in goal-based tasks is that the reward from the environment is given only upon reaching the goal state. Non-goal states are not rewarded which leads to two kinds of problems:

1. The temporal credit assignment problem, i.e., the problem of determining which part of the behaviour deserves the reward.

2. Slower convergence: conventional RL algorithms employ a delayed approach propagating the final goal reward in a discounted way or assigning a cost to non-goal states. However the back-propagation of the goal reward over the state space is time consuming.

To speed up the learning process, and to tackle the temporal credit assignment problem, the concept of shaping reward has been considered in the field [6, 7]. The idea of reward shaping is to give an additional (numerical) feedback to the agent in some intermediate states that helps to guide it towards the goal state in a more controlled fashion.

Even though reward shaping has been powerful in many experiments it quickly turned out that, used improperly, it can be also misleading [7]. To deal with such problems Ng et al. [6] proposed potential-based reward shaping \( F(s, s') \) as the difference of some potential function \( \Phi \) defined over a source \( s \) and a destination state \( s' \):

\[
F(s, s') = \gamma \Phi(s) - \Phi(s').
\]

They proved that reward shaping defined in this way is necessary and sufficient to learn a policy which is equivalent to the one learned without reward shaping.

One problem with reward shaping is that often detailed knowledge of the potential of states is not available, or very difficult to represent directly in the form of a shaped reward. Rather, some high level knowledge of the problem domain exists, that does not lend itself easily to explicit reward shaping.

In this paper we focus on the use of high-level STRIPS operators to automatically create a potential-based reward function, that improves the ability and speed of the agent to converge towards the optimal policy. The only interface between the basic RL algorithm and the planner is the shaping reward and information about the current state. In related works where planning operators were also used [4, 9] a RL agent learns an explicit policy for these operators. In our approach symbolic planning provides additional knowledge to a classical RL agent in a principled way through reward shaping. As a result, our approach does not require frequent re-planning as is for example the case in [4].

We evaluate the proposed method in a flag-collection domain, where there is a goal state (necessary for applying STRIPS) and a number of locally optimal ways to reach the
goal. Specifically, we demonstrate the success of our method by comparing it to RL without any reward shaping, RL with manual reward shaping, and an alternative technique for automatic reward shaping based on abstract MDPs [5] when it is used in its basic form. Thus, the contribution of the paper is the following: 1) we propose and evaluate a novel method to use the STRIPS-based planning as an alternative to MDP-based planning for reward shaping; 2) we show that MDP-based reward shaping may fail and successful experiments with STRIPS-based shaping suggest modifications which can overcome encountered problems. The STRIPS-based method we propose allows expressing the same domain knowledge in a different way and the domain expert can choose whether to define an MDP or STRIPS planning task. The STRIPS-based approach brings new merits to reward shaping from abstract/high level planning in domains with the intensional representation [2] which allows for symbolic reasoning.

High-level domain knowledge is often of a heuristic nature and may contain errors. We address this issue by evaluating the robustness of plan-based reward shaping when faced with incorrect high-level state definitions or plans.

The remainder of this paper is organised as follows. Section 2 introduces reinforcement learning. The proposed method to define the potential function for reward shaping is introduced in section 3. The experimental domain is described in section 4 and the chosen RL algorithms are presented in section 5. Section 6 shows how the proposed method can be used in the experimental domain, and a range of empirical experiments and results are presented in section 7. Section 8 concludes the paper with plans for further research.

2. MARKOV DECISION PROCESSES AND REINFORCEMENT LEARNING

A Markov Decision Process (MDP) is a tuple \((S, A, T, R)\), where \(S\) is the state space, \(A\) is the action space, \(T(s, a, s') = Pr(s_{t+1} = s'|s_t = s, a_t = a)\) is the probability that action \(a\) in state \(s\) at time \(t\) will lead to state \(s'\) at time \(t+1\), \(R(s, a, s')\) is the immediate reward received when action \(a\) taken in state \(s\) results in a transition to state \(s'\). The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and a reward function) are available, this task becomes a planning problem which can be solved using iterative approaches like policy and value iteration [12]. Value iteration which is used in this work applies the following update rule:

\[
V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}(R_{ss'} + \gamma V_k(s')), \tag{2}
\]

The value of state \(s\) is updated according to the best action after one sweep of policy evaluation.

MDPs represent a modelling framework for RL agents whose goal is to learn an optimal policy when the environment dynamics are not available. Thus value iteration in the form presented in Equation 2 cannot be used. However the concept of an iterative approach in itself is the backbone of the majority of RL algorithms. These algorithms apply so-called temporal-difference updates to propagate information about values of states \(V(s)\) or state-action \((Q(s, a))\) pairs.

These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. Model-free SARSA is such a method [12]. It updates state-action values by the formula:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \tag{3}
\]

It modifies the value of taking action \(a\) in state \(s\), when after executing this action the environment returned reward \(r\), moved to a new state \(s'\), and action \(a'\) was chosen in state \(s'\). Model-based RL algorithms (e.g., DynaQ) learn additionally how the world responds to its actions (transition probabilities) and what reward is given (reward function) and use this model for simulated backups made in addition to real experience.

Immediate reward \(r\) which is in the update rule given by Equation 3 represents the feedback from the environment. The idea of reward shaping is to provide an additional reward which will improve the performance of the agent. This concept can be represented by the following formula for the SARSA algorithm:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, a, s') + \gamma Q(s', a') - Q(s, a)],
\]

where \(F(s, a, s')\) is the general form of the shaping reward which in our analysis is a function \(F : S \times S \rightarrow \mathbb{R}\), with \(F(s, s')\). The main focus of this paper is how to compute this value in the particular case when it is defined as the difference of potentials of consecutive states \(s\) and \(s'\) (see Equation 1). This reduces to the problem of how to compute the potential \(\Phi(s)\).

3. PLAN-BASED REWARD SHAPING

The class of RL problems is investigated in which background knowledge allows defining state and temporal abstractions using intensional representation [2]. Abstract states are defined in terms of propositions and first order predicates, and temporally extended actions (or options [13]) can be treated as primitive actions at the abstract level. The function \(f_{abs}(S) = Z\) maps states \(s \in S\) onto their corresponding abstract states \(z \in Z\).

3.1 Potential Based on STRIPS Plan

The intensional representation allows for symbolic reasoning at an abstract level when options can be defined in terms of changes to the symbolic representation of the state space, e.g., they can be expressed as STRIPS operators. For such problems STRIPS planning can be used to reason at this abstract level. When the RL problem is to learn a policy which moves the agent from start state \(s_0\) to goal state \(s_g\) it can be translated to the high level problem of moving from state \(z_0 = f_{abs}(s_0)\) to state \(z_g = f_{abs}(s_g)\). Because of the intensional representation at the abstract level, symbolic reasoning can be used to solve the planning problem of moving form state \(z_0\) to goal state \(z_g\). It is a classical planning task which can be solved using standard STRIPS planners (Graphplan [1] is used in our experiments). The trajectory \(\omega = (z_0, z_1, ..., z_g)\) of abstract states (obtained from plan execution at the abstract level) can be used to define the potential for low level states as:

\[
\Phi(s) = step(f_{abs}(s)),
\]

where the function \(step(z)\) returns the time step at which given abstract state \(z\) appears during plan execution. In
other words, the potential is incremented after the RL agent has successfully completed an abstract action in the plan, and reached a (low-level) state that is subsumed by the corresponding abstract state in the trajectory.

The question remains what potential to assign to those abstract states that do not occur in the plan. One option is to ignore such states and assign a default value of zero. This approach can strongly bias the agent to follow the given path. The agent would be discouraged from moving away from the plan. As it will be discussed later, this leads to problems when the plan is wrong and in particular when there is no transition from state \( z_i \) to state \( z_{i+1} \) in the environment. The agent may not be able to get out of state \( z_i \) because of the negative reward for going to any state other than \( z_{i+1} \).

We propose a more flexible approach that will allow the agent to abandon the plan and look for a better solution when the plan is wrong. Figure 1 shows the algorithm. States which are in trajectory \( \omega \) (plan states) have their potential set to the time step of their occurrence in the plan. Non-plan states that are reachable from any state \( z \in \omega \) have their potential set to the potential of the last visited plan state (variable \( last \)). In this way the agent is not discouraged from diverging from the plan (it is also not rewarded for doing so).

A problem with this approach is that some non-plan states can be reached from different levels of potential. For this reason, for each non-plan state the highest value of the last potential is stored in the array \( \text{Max} \). The main aim of using this array is to prevent continuous changes in the potential of non-plan states which may be disadvantageous for the convergence of the value function.

\[
\text{initialise } last \leftarrow 0 \\
\text{if } f_{abs}(s) \in \omega \text{ then} \\
\text{ last } = \text{step}(f_{abs}(s)) \\
\text{ return } \text{step}(f_{abs}(s)) \\
\text{ else if } last > \text{Max}(f_{abs}(s)) \\
\text{ Max}(f_{abs}(s)) = last \\
\text{ return } last \\
\text{ else } \\
\text{ last } = \text{Max}(f_{abs}(s)) \\
\text{ return } \text{Max}(f_{abs}(s)) \\
\text{ end if} \\
\text{ end if}
\]

**Figure 1:** Assigning potential \( \Phi(s) \) to low level states through corresponding abstract states.

The abstract goal state in the considered class of RL tasks needs to be defined as a conjunction of propositions. The most straightforward way to define potential for such goals manually is to raise it with each goal proposition which appears in a given state. This kind of potential, even though it gives some hints to the agent which propositions bring it closer to the goal, does not take into account how the environment is regulated (there may be a certain sequence of achieving goal conditions, that leads to higher rewards). One example is the travelling salesmen problem. Potential raised just for each visited town will strongly bias the nearest neighbourhood strategy. An admissible heuristic based on, e.g., minimum spanning trees can be used to give correct (optimistic) potential [8]. In our approach instead of encouraging the agent to obtain just goal propositions, a more informed solution is proposed that takes into account how the environment behaves.

### 3.2 Potential Based on Abstract MDP

Marthi [5] proposed a general framework to learn the potential function by solving an abstract MDP. In this section we show how this idea can be applied with the same kind of knowledge that is given to the STRIPS-based approach. The automatic shaping algorithm obtains potential by firstly learning dynamics for options (i.e., actions at the abstract level) and secondly solving an abstract MDP. Options can be defined as policies over low-level actions. Because in our class of problems options are assumed to be primitive and deterministic actions at an abstract level, computation of their dynamics can be omitted. An abstract MDP (e.g., value iteration from Equation 2 can be applied) can be solved before target RL learning and the obtained value function is used directly as the potential. The following equation describes this fact:

\[
\Phi(s) = \hat{V}(f_{abs}(s)),
\]

where \( \hat{V}(z) \) is the value function over state space \( Z \) and it represents an optimal solution to the corresponding MDP-based planning problem. Because the high-level model is deterministic and options make transitions between abstract states, this planning task can be solved using the following formula:

\[
V_{k+1}(z) = \max_z [R_{z,z'} + \gamma V_k(z')].
\]

Knowledge equivalent to STRIPS operators can be used to determine the next possible states \( z' \) for given state \( z \). The reward given upon entering the abstract goal state and discount factor \( \gamma \) can be chosen to make the difference in the value function between neighbouring states equal to one, thus enabling us to perform easier comparisons with the STRIPS-based reward shaping approach.

### 4. EXPERIMENTAL DOMAIN

The proposed algorithms are evaluated on an extended version of the navigation maze problem. This problem has been used in many RL investigations, and is representative of RL problems with the following properties:

- There exists an abstract goal state. This can stand for a number of actual states. A well-defined goal state is necessary for applying STRIPS planning.
- There are many ways to reach the goal, with varying associated rewards. In other words, there are local policy optima that the RL agent can get stuck in.

We use the artificial domain to evaluate our algorithm, therefore, it will be suitable for any real-world problem with these properties (approach adopted also in [9]).

In the basic navigation maze problem an agent moves in a maze and has to learn how to navigate to a given goal position. In the extended version of this problem domain, the agent additionally has to collect flags (i.e., visit certain points in the maze) and bring them to the goal position. The
reward at the goal is proportional to the number of flags collected. In order to introduce abstraction and demonstrate the use of high-level planning, the maze is additionally partitioned into areas (rooms).

Because an episode ends when the agent reaches the goal position regardless of the number of collected flags, this problem has been used in the past to evaluate sophisticated exploration strategies (e.g., [3, 11]). The learning agent can easily get stuck in a local optimum, bringing only a reduced number of flags to the goal position.

An example maze is shown in Figure 2. The agent starts in state S and has to reach goal position G after collecting as many flags (labelled A, B, C, D, E, F) as possible. The episode ends when the goal position has been reached and the reward proportional to the number of collected flags is given. Thus the reward is zero in all states except the goal state. The agent can choose from eight movement actions which deterministically lead to one of eight adjacent cells when there are no walls. The move action has no effect when the target cell is separated by a wall.

5. EVALUATED ALGORITHMS AND PARAMETERS

To conduct the evaluation two RL algorithms are used: SARSA and DynaQ. The usage of SARSA aims at investigating the influence of potential-based reward shaping on model-free reinforcement learning. Model-based methods are represented by DynaQ. All these RL algorithms were used in its basic form as they are presented in [12]. The following common values for parameters were used: \( \alpha = 0.1, \gamma = 0.99 \), the number of episodes per experiment \( 10^5 \). In all experiments an \( \epsilon \)-greedy exploration strategy was used where epsilon was decreased linearly from 0.3 in the first episode to 0.01 in the last episode.

Reward shaping was applied to all the above RL algorithms. Plan-based reward shaping was compared with a non-shaping approach and with three other shaping solutions. This results in five reward shaping options: 1) no reward shaping, 2) STRIPS-based reward shaping, 3) abstract MDP-based reward shaping, 4) flag-based reward shaping, 5) composed reward shaping. STRIPS-based and abstract MDP-based reward shaping appear in the form as they were introduced. In the above no-shaping case no shaping reward is given. The flag-based shaping reward is determined by the number of collected flags, and the potential is the function \( \Phi(s) = \text{flags}(s) \), where \( \text{flags}(s) \) is the number of collected flags in state \( s \). It is an instance of the manual shaping approach (discussed in Section 3.1) which raises the potential for each goal proposition achieved in the current state. This kind of reward shaping thus represents the "nearest flag" heuristic. In composed reward shaping the potential is a sum \( \Phi(s) = \text{plan}(s) + \text{flags}(s) \) of STRIPS-based potential \( \text{plan}(s) \) and the number of collected flags in state \( s \) \( \text{flags}(s) \). Flag-based reward shaping when combined in this way with STRIPS-based shaping may hurt the performance of pure STRIPS-based approach. However the "nearest flag" bias added by flag-based information can help in the case of incorrect planning knowledge. For this reason such composition of flag- and STRIPS-based shaping named composed is also evaluated.

If not explicitly mentioned otherwise, all experiments were repeated ten times and the average performance is shown in the result graphs.

6. POTENTIAL FOR EXPERIMENTAL DOMAIN

This section shows how the proposed RL and reward shaping approaches were applied to the flag collection domain.

6.1 Low Level Model

In our experiments, reinforcement learning is carried out at the low level which is defined by the target MDP \( (S, A, R, T) \), where \( S \) is the state space defined by the position of the agent in the 13x18 maze and by the collected flags, and \( A \) is the set of eight primitive actions corresponding to eight movement directions. The reward function \( R \) and transition probabilities \( T \) are not known to the agent in advance.

6.2 High Level Knowledge

Plan-based reward shaping assumes that there exists a high level structure in the modelled world. The access to two types of knowledge is required:

1. State mapping The mapping from low level to abstract states. The function which maps low level states into abstract states identifies each abstract state as the area in the maze in which the given low level position is located. Hence, the abstract state is determined by the room location of the agent and the collection of collected flags. Such a state can be symbolically expressed as:

   \( \text{robot_in(roomB)} \land \text{taken(flagE)} \land \text{taken(flagF)} \).

2. Transitions Possible transitions between high level states. In this case there are two types of knowledge which allow defining transitions at the abstract level:

   (a) Possible transitions between areas in the maze (i.e., which adjacent rooms have doors between them).

   (b) Location of flags: in which room a given flag is located.

Figure 2: The map of the maze problem. S is the start position and G the goal position. Capital letters represent flags which need to be collected.
6.3 High Level Planning Problems

Knowledge about the high level structure of the world is used to define high level planning problems. The abstract state representation attributes are used to define state representation for both classical and MDP-based planners. In the case of the STRIPS representation the location of the robot and symbolic names of collected flags are used for an intensional description of the world. For the MDP-based planner the state space is enumerated and all possible states are collected in the tabular representation which has 448 entries. In both cases the state encoding preserves the Markov property.

Both investigated planning approaches require action models. In this case knowledge about transitions and intensional state representation is used to define high level actions. The following STRIPS operators were used:

\[
\text{(TAKE } ((\text{flag} \cdot \text{FLAG}) \cdot \text{(area) AREA})) \\
\text{(preconds (flag-in flag <area> (robot-in <area>))} \\
\text{(effects (del flag-in flag <area>) (taken <flag>)))} \\
\text{(MOVE } ((\text{from} \cdot \text{AREA}) \cdot \text{<to> AREA})) \\
\text{(preconds (robot-in <from>) (next-to <from> <to>))} \\
\text{(effects (del robot-in <from>) (robot-in <to>))}
\]

These operators together with the knowledge about the possible transitions between areas and the location of flags allow reasoning about the changes in the environment. The same knowledge is used to define possible transitions between abstract states in the abstract MDP, strictly to find for each state the set of reachable states. According to the description of the algorithm, deterministic options are assumed which allow for deterministic transitions between abstract states.

Introduced STRIPS actions allow reasoning symbolically about the changes in the world. The planner has to find a sequence of MOVE and TAKE actions which can transform the system from the start state in which robot-in(hallA) to the goal state:

\[
\text{robot-in(roomD) } \land \text{taken(flagA) } \land \ldots \land \text{taken(flagF)}.
\]

Because of the closed-world assumption (everything not mentioned explicitly in the description of the state is assumed to be false) the start state has to define all initial facts, like locations of flags (e.g., flag-in(flagA, roomA) and connections between rooms (e.g., next – to(hallB, roomC)). The last group of facts is called rigid facts because they do not change over time (the fact whether rooms are connected or not remains unchanged).

Both the MDP and STRIPS planning problems can be solved in advance before the learning takes place. Once these problems have been solved they can be used to assign potential to high level states directly and to low level RL states indirectly via the mapping function, which translates low level states to high level abstract states. The potential is assigned to abstract states in the manner presented earlier.

7. EMPIRICAL RESULTS

In this section the empirical results are presented and discussed.

Even though the high level plans used for reward shaping are optimal according to the provided high level knowledge, this knowledge may contain errors. Therefore, the plan may not be optimal at the low level where the RL agent operates. For this reason the presentation of experimental results is divided into two sections. First, the results on different RL algorithms are analysed when the high level plan is optimal. Afterwards, various possible plan deficiencies are defined and their impact is empirically evaluated.

7.1 Results with Optimal Plan

Results presented in this section are for the test domain as shown in Figure 2. High level plans generated by STRIPS planning and the abstract MDP are both optimal at the lower RL level. The STRIPS plan is shown in Figure 5. The MDP-based plan leads to the same sequence of visited abstract states as in the STRIPS plan when the policy determined by the value function is followed from the start to the goal state.

The discussion of experimental results is done separately for model-free and model-based RL algorithms.

7.1.1 Model-Free Methods

The first set of experiments looks at the performance of the different reward shaping approaches when used with model-free RL. In Figure 3 results with SARSA are presented. They show the difficulty of the investigated maze problem in terms of exploration. In all 10 runs the no-shaping RL version was not able to learn to collect more than one flag. It quickly converged to a sub-optimal solution which takes only flag D and directly moves to position G (the goal position). The only approaches that were able to learn to collect all flags (though not in all runs) are using STRIPS-based and composed reward shaping.

The experimental results show that this problem poses a challenge to model-free methods and is difficult to solve without properly used background knowledge.

In the above results the MDP-based reward shaping displayed a particularly worse performance than not only STRIPS-based but also less informed methods. A more detailed analysis was undertaken to look for the reason of this low performance.
Some conclusions can be drawn from the analysis of the histogram (see Figure 4) which shows how many times each abstract state was entered in both STRIPS-based and MDP-based approaches. The presented graph is for a single run of SARSA.

The first observation from this experiment is that the algorithm with MDP-based plan tried many different paths, especially in the first episodes of learning. In the STRIPS-based case there is only one path along which potential increases. In the MDP-based case many different paths can be tried because the potential increases along many paths when moving towards the goal. When the agent moves away from the plan it can still find a rewarded path to the goal because the MDP-based policy defines an optimal path to the goal, not only from the start but from all states. This led to a rather "undecided" behaviour of the algorithm in the early stages of learning. The agent tries many different and advantageous paths, but because different paths are tried, they do not converge quickly enough (compare the number of steps made by SARSA with MDP-based shaping shown in Figure 6).

In effect, short and sub-optimal paths, like, e.g., the one that goes from start state $S$ directly to goal $G$ after taking flag $D$, quickly dominate because they lead to better performance than very long paths that collect more flags, because they have not converged yet. The histogram shown in Figure 4 provides more evidence for this hypothesis. First of all, it can be noticed that the number of visited abstract states is almost twice bigger in the MDP-based case. The agent considers a higher number of paths to be "interesting" in this case. In this particular run the number of visited abstract states was 106 in the STRIPS-based and 202 in the MDP-based case (there were 3141 and 6876 visited low level states respectively). Specifically, in the STRIPS-based case, the states that are visited when the optimal plan is followed, are those with the highest number of visits in the histogram. Other abstract states which also have high values in the histogram are adjacent to those which follow the optimal path. It is worth noting that states which follow the optimal path are not visited very often in the MDP-based case.

The main conclusion from this empirical analysis is that in the case of model-free RL algorithms and a difficult problem domain (in terms of exploration), it may be better to assign potential according to one particular path which can converge quickly rather than to supply many paths with increasing potential. The latter raises the probability of converging to a sub-optimal solution.

This observation suggests one potential improvement to MDP-based reward shaping when problems discussed here may arise. Instead of using the value function for the entire state space as potential, the best path which corresponds to the STRIPS plan can be extracted. When this path (in the same way as the STRIPS plan) is used with our algorithm to define potential, it can direct the agent in a more focused way toward the goal when it can be easily misled by a suboptimal result.

7.1.2 Model-based Methods
In our experiments, DynaQ represents the category of model-based reinforcement learning algorithms. Figure 7 shows the results with the different reward shaping techniques. The first observation is that DynaQ can deal much better with the problem domain than model-free SARSA. Even in the no-shaping case on average 5.4 flags were collected. The informed reward shaping methods (composed, MDP-based, STRIPS-based) performed better, showing the fastest increase in the obtained reward during almost the entire period of learning. With STRIPS-based and composed schemas to assign potential to abstract states, all six flags were collected in all ten test runs.

Results with model-free SARSA showed that reward shaping is essential in solving problems where it is easy to get
stuck in a sub-optimal solution. Model-based methods like DynaQ make better use of what has been experienced during the learning process and make additional simulated backups over the state space. In this way it is possible to propagate information about highly rewarded areas even without visiting these areas many times (in a deterministic environment it is enough to make each transition once).

7.2 Results with Sub-optimal Plans

In this section we take a closer look at various errors in the STRIPS-plans, that may be caused by incomplete or imprecise knowledge. Due to space restrictions, we do not show graphs for all experiments, but rather summarise the results in the text.

7.2.1 Plan Too Long

Incomplete knowledge about the environment can lead to the situation when the planner computes a longer plan than necessary. In the actual environment direct transitions from state $z_i$ to state $z_{i+k}$ where $k > 1$ may be possible, even though the planning knowledge did not include this fact.

In our experiments we created an additional transition from room E to room B, that has not been taken into account in the computation of the STRIPS-plan. After collecting two flags in room E the agent wants to collect flag B in room B. According to the plan it has to go through room C and hall B.

Empirical tests show that this kind of plan deficiency does not seem to cause problems for the RL agent (Figure 8 shows the results for SARSA). The transition from E to B when discovered is well rewarded because it has a higher difference in potential (6 in room E and 9 in room B).

The results for the other RL algorithms are similar and show the same trend.

7.2.2 Plan Assumes Impossible Transition

Incorrect knowledge can cause also the opposite effect: connections between two states that are assumed by the plan knowledge may not exist in the actual environment. In our experiments we created such a situation where the plan was computed assuming a connection between rooms E and B. The lack of this connection during learning is destructive for SARSA. However, model-based DynaQ finds a solution that is close to optimum and plan-based shaping performs better than no-shaping and flag-based shaping.

7.2.3 Missing Goal Conditions

This experiment evaluates the RL approaches when the plan was computed with a missing goal condition, thus potentially missing required actions, or including actions that are undoing part of the goal.

In our experiments we assumed that the information about flag B has not been given to the planner. The question is whether the learning agent is able to find the missing element through exploration. This is principally possible because the proposed schema to assign potential to non-plan states does not penalise moving away from the plan. The evaluation results show that the only configuration in our experiments that was able to perform better than the given (sub-optimal) plan was DynaQ with STRIPS-based shaping. Simulated backups led to the required propagation of the information about the discovered flag B.

7.2.4 Wrong Sequence

Even when high level knowledge about the domain is complete and the problem is specified correctly there is one more factor which may lead to sub-optimal policy at the low RL level. The main goal of classical planning algorithms is to find a plan which can transform the system from the start to the goal state. This achievement of the plan is usually satisfactory and the cost of actions is not taken into account in most STRIPS-based planners. In introduced in this paper application of classical planning this may lead to sub-optimal plans when high level actions can have different cost when implemented by low level primitive actions. To test our algorithm with these deficiency of plan, the experimental domain was modified in the following way. Halls A and B were merged into one hall and the high level plan was modified so flags were collected in the following order: B, A, C, E, F, D. This plan is clearly sub-optimal. Even though all flags are in the plan there is another plan that results in a shorter travelled distance. This setting was also difficult to tackle by most RL approaches. In this case, only model-based DynaQ with composed reward shaping was able to do better than the sub-optimal plan.

In summary, our results show that even when plans are not optimal or contain errors, RL algorithms are performing best when STRIPS-based reward shaping is used, but are not always able to converge to the optimum. Nevertheless, this can be satisfactory because the goal is often not to find the optimal solution but an acceptable policy in a reasonable amount of time.
8. CONCLUSIONS AND FUTURE WORK

In this paper we show a new method to define the potential function for potential-based reward shaping, using abstract plan knowledge represented in the form of STRIPS operators. We empirically compared the performance of our proposed approach to RL without any reward shaping, RL with a manually shaped reward, as well as a related automatic reward shaping approach based on abstract MDPs [5]. The results of the experiments demonstrate that the STRIPS-based reward shaping improves both the quality of the learned policy, and the speed of convergence over the alternative techniques.

Overall, the results can be summarised as follows:

1. RL problems that are difficult in terms of exploration can be successfully tackled with model-free methods with plan-based reward shaping.

2. Model-based methods can find solutions to these problems without reward shaping in some cases, but reward shaping always speeds up learning.

3. STRIPS-based shaping showed better results than the MDP-based approach, because the agent was strongly influenced by the plan that guides it towards a good policy. Thus, this observation suggests one potential improvement to MDP-based reward shaping. Instead of using the value function of the entire state space as potential the best path which corresponds to the STRIPS plan can be extracted and used with our algorithm to define the potential.

Additionally STRIPS-based approaches can deal with much bigger state spaces at an abstract level because states are not explicitly enumerated. Symbolic planners can solve large problems (with huge state spaces) through their compact and highly abstract representations of states. Such planning together with model-free RL (with which STRIPS-based planning works well) can therefore be used with large state spaces and with function approximation in particular. It is worth noting that function approximation has been up to now used mainly with model-free RL algorithms and SARSA in particular [10].

STRIPS-based reward shaping is easier to scale up than, e.g., MDP-based reward shaping. For MDP-based abstract planning the state space has to be enumerated, which may require stronger abstraction or function approximation when applied to RL domains with very large state spaces. However, a positive feature of MDP-based planning is that it can deal in a natural way with different costs of high level actions (something that is more difficult to achieve with STRIPS).

Overall, STRIPS-based reward shaping can be seen as an alternative to MDP-based reward shaping with the proposed extension as both these techniques are planning methods. It is up to the domain expert which method to choose, depending on the form of available knowledge.

In future research we intend to investigate the ability of RL to explicitly correct errors in high-level plan knowledge and revise it based on the learning experience. Another future challenge is to apply plan-based reward shaping to multi-agent learning, using techniques from multi-agent planning [8].

Acknowledgment

This research was sponsored by the United Kingdom Ministry of Defence Research Programme.

9. REFERENCES


ABSTRACT
Potential-based reward shaping has been shown to be a powerful and flexible method to incorporate background knowledge into reinforcement learning agents. However, the question remains how to compute the potential which is used to shape the reward. In this paper we propose a way to solve this problem in reinforcement learning with tile coding. Where the Q-function is represented with low-level tile coding, a V-function with coarser tile coding can be learned in parallel and used to approximate the potential for ground states. The novel algorithm is presented and experimentally evaluated.

1. INTRODUCTION
Reinforcement learning (RL) is a popular method to design autonomous agents that learn from interactions with the environment. In contrast to supervised learning, RL methods do not rely on instructive feedback, i.e., the agent is not informed what the best action in a given situation is. Instead, the agent is guided by the immediate numerical reward which defines the optimal behaviour for solving the task. This leads to two kinds of problems: 1) the temporal credit assignment problem, i.e., the problem of determining which part of the behaviour deserves the reward; 2) slower convergence: conventional RL algorithms employ a delayed approach propagating the final goal reward in a discounted way or assigning a cost to non-goal states. However the back-propagation of the reward over the state space is time consuming.

To speed up the learning process, and to tackle the temporal credit assignment problem, the concept of shaping reward has been considered in the field [10, 11]. The idea of reward shaping is to give additional (numerical) feedback to the agent in order to improve its convergence rate.

Even though reward shaping has been powerful in many experiments it quickly turned out that, used improperly, it can be also misleading [11]. To deal with such problems potential-based reward shaping $F(s, s')$ was proposed [10] as the difference of some potential function $\Phi$ defined over a source $s$ and a destination state $s'$:

$$F(s, s') = \gamma \Phi(s) - \Phi(s'),$$

where $\gamma$ is a discount factor. Ng et al. [10] proved that reward shaping defined in this way is necessary and sufficient to learn a policy which is equivalent to the one learned without reward shaping.

One problem with reward shaping is that often detailed knowledge of the potential of states is not available, or very difficult to represent directly in the form of a shaped reward.

In this paper we propose an approach to learn the shaping reward online and use it to enhance basic reinforcement learning when tile coding [7] is used for function approximation. The algorithm starts without any prior knowledge and learns a policy and a shaping reward at the same time. At each step the current approximation of the shaped reward is used to guide the learning process at the ground level. The algorithm applies two levels of tile coding: the first one to learn the value function for ground RL, and the coarse-coded second one to approximate the shaping reward.

When relating our approach to automating shaping [8], the contribution of this paper is two-fold: 1) our algorithm learns reward shaping online through off-model reinforcement learning when tile coding [7] is used for function approximation. The algorithm starts without any prior knowledge and learns a policy and a shaping reward at the same time. At each step the current approximation of the shaped reward is used to guide the learning process at the ground level. The algorithm applies two levels of tile coding: the first one to learn the value function for ground RL, and the coarse-coded second one to approximate the shaping reward.

A review of related research is collected in the latter part of the paper.

We also explicitly address and experimentally evaluate domain properties and the design of RL solutions under which the proposed enhancement to RL algorithms is most efficient and its application may be particularly beneficial. The principal advantage of our approach is a better convergence rate. Furthermore, overhead computational cost is low, the solution is of general applicability, and knowledge is easily acquired and incorporated. Knowledge which is necessary to design tile coding for ground RL is sufficient to apply our extension.
2. MARKOV DECISION PROCESSES AND REINFORCEMENT LEARNING

A Markov Decision Process (MDP) is a tuple \((S, A, T, R)\), where \(S\) is the state space, \(A\) is the action space, \(T(s, a, s')\) is the probability that action \(a\) when executed in state \(s\) will lead to state \(s'\), \(R(s, a, s')\) is the immediate reward received when action \(a\) taken in state \(s\) results in a transition to state \(s'\). The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and a reward function) are available, this task becomes a planning problem which can be solved using iterative approaches like policy and value iteration \([15]\). Value iteration applies the following update rule:

\[
V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}^a[R_{ss'} + \gamma V_k(s')].
\]  

(2)

The value of state \(s\) is updated according to the best action after one sweep of policy evaluation.

MDPs represent a modelling framework for RL agents whose goal is to learn an optimal policy when the environment dynamics are not available. Thus value iteration in the form presented in Equation 2 can not be used. However the concept of an iterative approach in itself is the backbone of the majority of RL algorithms. These algorithms apply so called temporal difference updates to propagate information about values of states \((V(s))\) or state-action \((Q(s, a))\) pairs. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. Model-free SARSA is such a method \([15]\). It updates state-action values by the formula:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)].
\]  

(3)

It modifies the value of taking action \(a\) in state \(s\), when after executing this action the environment returned reward \(r\), moved to a new state \(s'\), and action \(a'\) was chosen in state \(s'\).

Immediate reward \(r\) which is in the update rule given by Equation 3 represents the feedback from the environment. The idea of reward shaping is to provide an additional reward which will improve the performance of the agent. This concept can be represented by the following formula for the SARSA algorithm:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, a, s') + \gamma Q(s', a') - Q(s, a)],
\]

where \(F(s, a, s')\) is the general form of the shaping reward which in our analysis is a function \(F: S \times S \to \mathbb{R}\), with \(F(s, s')\). The main focus of this paper is how to approximate this value online in the particular case when it is defined as the difference of potentials of consecutive states \(s\) and \(s'\) (see Equation 1). This reduces to the problem of how to learn the potential \(\Phi(s)\).

3. VALUE FUNCTION APPROXIMATION WITH TILE CODING

To deal with huge or infinite state spaces (e.g., due to continuous variables), value function approximation has been successfully used \([14]\). It is a supervised learning approach which aims at approximating the value function across the entire state space. It takes advantage of the fact that states with similar values of state features have in most cases a similar value of the \(V\)-function. The idea is to represent the value function \(V\) as a vector of parameters \(\theta\) with the size of this vector smaller than the number of states. In this way the update of the value function according to one state is generalised across "similar" states.

Tile coding \([14]\) is a particular method to define basis function \(\phi_i(s)\) for states or state-action pairs. This method partitions the input space into several displaced layers (tilings) of overlapping tiles. Each state can be allocated to exactly one tile in each tiling. Thus \(\phi_i(s)\) takes value 1 for tiles it is allocated in and 0 otherwise. Figure 1 shows how it can be computed in a 2D space. Tiles allow for generalisation to neighbouring positions. For example, an update of the value function in position \(x\) has an impact on the value function in position \(y\) which may not be visited during the entire period of learning. One of the key motivations to propose the algorithm introduced in the next section is the fact that coarser generalisation (see Figure 1b) allows for a more rapid propagation of the value function. This coarser generalisation can be used to guide the learning of the more detailed value function.

4. LEARNING POTENTIAL FOR REWARD SHAPING

We propose a RL architecture with two levels of tile coding. The first one learns an approximation of the \(Q\)-function at the ground RL level. The second, coarser one learns an...
abstract V-function which is used as the potential to calculate the shaping reward (see Equation 1) for the lower level. The algorithm which is proposed here builds on two techniques existing in the field: 1) multigrid discretisation used with MDPs [2], and 2) automatic shaping which was recently proposed [8].

The multigrid discretisation in the MDP setting [2] was used to solve an MDP in a coarse-to-fine manner. While this technique is well suited to dynamic programming methods (a coarse problem at a high level can be solved and used at a more detailed level), there was no easy way to merge layers with a different resolution when applied to RL algorithms. First such attempts were made by [1] and this problem was evident there. The need for knowledge of the topology of the state space is necessary in their solution to define how multiple levels are related, but this fact made this approach infeasible for RL tasks. It used multigrid as a way of obtaining knowledge, but the mechanism to use this knowledge at a ground RL level was missing. We propose potential-based reward shaping as a solution to these problems. The ground RL algorithm does not have to be modified and knowledge can be given in a transparent way via an additional shaping reward.

In the automatic shaping approach [8] an abstract MDP is formulated and solved. In the initial phase of learning the model of an abstract MDP is built and after a defined number of episodes an abstract MDP is solved exactly and its value function used as the value of the potential for ground states. In this paper, we propose an algorithm which applies a multigrid strategy when tile coding is used for function approximation. Instead of defining an abstract task as dynamic programming for solving an abstract MDP, we use RL to solve the high level task online. RL with representation based on tile coding results in a natural translation between ground and abstract levels. Tile coding in itself can be easily applied in a multigrid fashion and because it has been mostly used with off-model RL and SARSA in particular\(^1\), it is sensible to apply RL for solving an abstract level problem. Because such an abstract RL does not need to learn the model, a shaping reward can be provided right from the start of learning. Additionally our method with tile coding does not require any more knowledge about the environment as to define tile coding at the ground level. We do not need methods to translate abstract to ground states or approximating environment dynamics (transition probabilities) at an abstract level.

Algorithm 1 summarises our approach. It follows the structure of SARS($\lambda$) with tile coding used in [13]. In our case learning at the ground level is the same as in the base-line except we explicitly show the fact that eligibility traces are optimised. The solution motivated by truncated temporal differences [3] is applied. Queue $\mathcal{E}$ (experience buffer) stores the trace, i.e., tiles for all state positions of the trace. The size $N$ of this queue is limited by the condition $(\gamma \lambda)^N \geq 10^{-9}$. When new states are placed at the front of the queue, vector $\epsilon$ can be computed according to Algorithm 2. The second modification which is crucial for our discussion is the point where the base-line algorithm is given shaping reward $F(s, s')$ in line 20 of Algorithm 1 where temporal difference is computed. The way in which $F(s, s')$ is evaluated defines our extension.

The shaping reward $F(s, s')$ is computed in line 11 as the difference of the value function of current and previous states visited by the agent. Thus $\Phi(s) = V(s)$ where $V$ is the value function of the abstract RL task. All parameters of this task have subscript $\nu$ and it is learned using temporal difference updates (lines 14 and 28) with tile coding. The mapping from state $s$ to the set of tiles $\mathcal{G}$ used at the abstract level is done in a straightforward way without any special knowledge. Basically, the lower resolution of tiles can be applied. However with optional, additional knowledge about the problem such a mapping can remove state variables and appropriately focus the high level learning.

High level RL is treated as a Semi-MDP since due to coarse tile coding an agent can be several time steps within one high level position. For this reason time $t$ is used when temporal difference in line 13 is evaluated.

The generic function $\text{reward}_1(r)$ shows that high level learning can receive an internally modified reward. According to our empirical evaluations $\frac{r}{10}$ gives good results on different domains where both the positive and negative reward is given. The division by factor 10 guarantees that the shaping reward extracted from an abstract V-function has smaller impact than the environment reward.

4.1 Properties of the Algorithm

Algorithm 1 was designed to learn online an admissible heuristic assessment of the distance to the goal (represented in the algorithm by the high level V-function) which is the backbone of the best-first search. This type of knowledge determines some properties of our algorithm.

Even though the shaping reward is learned with a separate tile coding and separate vector of parameters, its performance is strictly correlated with relations between the Q- and V-function in general and the design of both levels of tiles. The following factors can thus have influence on the performance of the algorithm proposed in the paper.

- V(s) values learned at the high level are a function of only states whereas ground RL learns Q(s,a) values in order to deal with unknown environment dynamics. This difference suggests that the positive influence of the potential extracted from V(s) should be bigger with a bigger number of actions $a \in A(s)$ because V(s) learns only values of states whereas Q(s,a) additionally distinguishes actions (there are more values to converge). Thus V(s) can converge faster and give positive guidance for learning Q(s,a) at the ground level.

- There can exist structural dependencies between features in the state space. Such structural dependencies can be used to define a reduced representation at an abstract level. For example, a reduced number of features can provide a high level guidance (e.g., goal

\(^1\)Empirical results in the literature [13] show that SARSA is generally better than Q-learning when tile coding is used. The explanation is justified in the literature by the fact that SARSA is an on-policy method.
Algorithm 1 SARSA(λ)-RS: Gradient-descent SARSA(λ) with tile coding, eligibility traces and potential-based reward shaping from temporal difference learning of an abstract level value function.

1: **RLstartEpisode**
   2: θ = 0, a = 0, t = 0 and c, cr, ← numbers of tilings
   3: G ← set of tiles for current state s
   4: V = ∑i∈G θ(i)
   5: a ← random action in state s
   6: F_a ← set of tiles for a and current state s
   7: E ← pushfront E a; Q_a = ∑i∈F_a θ(i)
   8: **RLstep**:
   9: G′ ← set of tiles for current state s′
   10: V′ = ∑i∈G′ θ(i); t = t + 1
   11: F(s, s′) = γ V′ − V; r_o = reward_o(r)
   12: if r_o ≠ 0 or G ≠ G′ then
   13: δ_e = r_o + γ V′ − V; t = 0
   14: e = trace(G); θ_o ← θ_o + α δ_e e
   15: end if
   16: G = G′; V = V′
   17: a ← best action in state s′
   18: with probability δ: a ← random action in state s′
   19: F_a ← set of tiles for a and current state s′
   20: Q'_a = ∑i∈F_a θ(i); δ = r + F(s, s′) + γ Q'_a − Q_a
   21: e = trace(E); θ ← θ + α δ e
   22: E ← pushfront E a; Q_a = Q'_a
   23: **RLEndEpisode**:
   24: G′ ← set of tiles for current state s′
   25: r_o = reward_o(r); t = t + 1
   26: if r_o ≠ 0 or G ≠ G′ then
   27: δ_e = r_o + γ V′ − V;
   28: e = trace(G); θ_o ← θ_o + α δ_e e
   29: end if
   30: δ = r − Q_a
   31: e = trace(E); θ ← θ + α δ e

Further sections evaluate Algorithm 1 and test some of the aforementioned hypotheses on a range of RL tasks.

5. **EXPERIMENTAL DESIGN**

A number of experiments have been performed to evaluate Algorithm 1. The following values of common RL parameters were used: α = 0.1, λ = 0.1, γ = 0.7, γ_r = 0.99 and γ_o = 0.99. For given values of parameters a maximum size of queue (|E|) is N = 56. In all experiments ε-greedy exploration strategy was used with ε decreasing linearly from 0.3 in the first episode to 0.01 in the last episode. All runs on all tasks were repeated 30 times and average results are presented in graphs. Following the evaluation process from recent RL competitions, the accumulated reward over all episodes was used as a measure to compare results in a readable way. Error bars of the standard error of the mean (SEM) are also presented.

6. **EXPERIMENTAL DOMAINS**

The following set of popular RL tasks were used as test domains in our experiments.

6.1 **Mountain Car**

The first experiments were performed on the Mountain Car task according to the description in [15]. The agent received a reward of 1 upon reaching the goal state on the right hill and -1 on the left hill. An experiment was terminated and the agent placed in a random position after reaching any of the two goal positions or after 10^3 episodes. Following [15] 10 tilings with 9x9 tiles were used for the Q-function and 6x6 for the V-function.

6.2 **Car Parking**

The Car Parking task was implemented according to [4]. The two runs are reported for two configurations of the task. The first one is with all parameters specified in [4], i.e., there were 6 tilings over one group of three state variable with 5x5x5 tiles per tiling. The same tilings were used also for the V-value. In the second run the size of the active area was doubled with x1 = 16.5 and y1 = 25. Because of the bigger size, the number of intervals was also doubled giving 10x10x10 tiles per tiling for the Q-function. Tiles for the V-function were not changed yielding higher generalisation.

6.3 **Boat**

The problem is to learn how to navigate a boat from the left bank to the quay on the right bank of the river. There is a strong non-linear current in the river. Our implementation follows the description in [5] with narrower quay (Z, width 0.2) and random starting positions used recently in [6] where it was shown to be challenging for classical RL algorithms. To deal with non-linear current, continuous or finely discretised actions are required [6]. This domain is used to check the influence of the number of actions on the performance of proposed reward shaping.

7. **RESULTS**

7.1 **Mountain Car**

For all interested in implementing this domain it is worth noting that in [4], Appendix A, condition r ≠ 0 and equation (c) the second sin should be cos.
This experiment aimed at investigating the performance of our algorithm with (Figure 3) and without (Figure 2) eligibility traces. It is important to emphasise that according to Algorithm 1 eligibility traces are used only at the ground level, i.e., there are no eligibility traces for updates of the high level V-function. The obtained results show that the proposed method to learn reward shaping leads to better results in both the presence and absence of eligibility traces. Shown errors bars indicate that learning with reward shaping is more stable across many runs.

Because eligibility traces are more challenging for our algorithm, further experiments compare it with SARSA(\(lambda\)). Furthermore, the length of eligibility traces was in its highest value according to the condition \((\lambda \gamma)^N \geq 10^{-9}\) making it more difficult for reward shaping to bring additional improvement. In comparisons without eligibility traces (i.e., when both compared algorithms were without eligibility traces) our reward shaping performed better in all cases like in the Mountain Car problem presented here.

### 7.2 Car Parking

In the Car Parking problem with the bigger size of the working area the type of knowledge which is learned at the high level starts playing more significant role. Figure 4 shows base-line results for the original task. Except giving more stable results, the reward shaping does not bring improvement in this setting. The picture of this configuration [3] shows relatively small distance to the goal from the fixed position of the car. In the second configuration in which the distance to the goal is bigger, goal-homing knowledge becomes more important. This is reflected in Figure 5. Overall, the advantage of our algorithm becomes more important on bigger instances of problems.

### 7.3 Boat

The agent controls the boat by the desired direction in the range \([-90^\circ, 90^\circ]\). Two experiments with discretisation into 5 and 40 values are reported. The same number of 5 tilings in both cases were used with 10x10x10 tiles for Q- and 8x8x8 tiles for the V-function. The ranges of positions x and y were scaled by factor 1.95 for tiles in the V-function representation yielding higher generalisation and more distinct separation of tiles between much different states.

Results for SARSA(\lambda) in Figures 6 and 7 show that the reward shaping has higher positive influence (statistically significant in all cases) when there are more actions available in each state. The V-function converges proportionally faster and provides positive feedback for learning Q-values. SARSA(\lambda)-RS with 40 actions converges faster in the initial phase of learning at a pace similar to SARSA(\lambda) with only 5 actions but obtains better results in the long run. The problem of slow convergence of SARSA(\lambda) with 4 ac-
functions (i.e., the number of actions desired for this domain) which was pointed out in [6] can thus be mitigated by our algorithm. Additional experiments with 20 actions yielded results where reward shaping performed better than with 5 actions and worse than with 40 actions showing coherence with our hypothesis.

8. RELATED WORK
The motivating literature [2, 1, 8] for our approach is discussed and referred in the description of our algorithm in Section 4.

Some work related to our method was presented in [17] where double CMAC was also applied. In this case Q- Instead of the V-function is used at an abstract level. The high level Q-values are used to guide the exploration in the initial learning phase. This approach lacks the reference to the potential-based reward shaping as results in [17] do not indicate a clear advantage of that method. Without the robust mechanism of potential-based reward shaping the Q-function needed to be used at an abstract level. The usage of the V-function would require for example approximating transition probabilities. In our case it is enough to learn only the V-function which can converge sufficiently faster to be useful for potential-based reward shaping.

The variable resolution discretisation/abstraction has been considered in the field [9]. The idea is to split some cells (states) and bring a higher resolution to same areas of the state space in order to represent a better policy. Our approach can be seen as orthogonal to this technique. We learn the shaping reward which can be used to guide ground learning with also a variable resolution discretisation. The interesting question arises whether a variable resolution would not improve the process of learning a potential function when applied at an abstract level and focused on fast propagation of guidance. When applied at the ground level it is intended to play the opposite role, i.e., to provide a higher resolution where it is necessary [9].

The relationship of the number of tilings and the interval size was studied in [12]. Their results show that a smaller number of tilings with wider intervals speeds up learning in initial episodes but hurts convergence at later stages. In contrast, narrower intervals (with preferably one tiling) slow down initial learning. Choosing in our algorithm a fine grained encoding with a small number of tilings at the ground level and coarse generalisation for reward learning can be seen as an easy way to have fast convergence at the beginning and good convergence at the end of learning.

9. CONCLUSION AND FUTURE WORK
We propose an algorithm to learn the potential function online at an abstract level, and experimentally evaluate it. The approach with tile coding function approximation shows that simultaneous learning at two levels can converge to a stable solution. The algorithm is based on the SARSA algorithm (on-policy temporal difference learning) which in contrast to Q-learning is considered to be better suited for function approximation [13]. The V-function at an abstract level is updated according to the same trajectories as ground learning.

Conditions and task properties which determine when the algorithm works better are discussed and evaluated experimentally. The application of this algorithm is especially beneficial when: 1) there are many actions in each state (e.g., the infinite number of continues actions); 2) a high resolution of the policy is required (due to details in the environment) with a wide range of values of state variables, i.e., on the bigger instance of the domain; 3) high level guidance can be distinguished from a subset of state variables; 4) the final reward is given only upon reaching goal states.

The strong points of the algorithm: 1) improved convergence speed in most domains, especially those that have the properties outlined in the previous paragraph; 2) without eligibility traces comparable convergence can be achieved at lower cost, because there is at most one backup of the V-function for each SARSA backup; eligibility traces require significantly more updates; 3) in contrast to eligibility traces, separate and external representation of knowledge is obtained; this high level knowledge may be useful for knowledge transfer [16]; 4) no need for explicit domain knowledge; in the basic form the high level learning can be defined using the same knowledge which is used to design tile coding at the ground level.

Reward shaping results in the biggest speedup during the initial phase of learning. The algorithm can be extended by conditions which determine when reward shaping or at least high level learning should be stopped. This could lead to
better results in domains such as the Boat task, as can be seen in Figures 6 and 7 where both curves get closer in the latter period of learning.

Acknowledgment
This research was sponsored by the United Kingdom Ministry of Defence Research Programme.

10. REFERENCES
ABSTRACT

The ability to reach into confined and unorganized environments, and grasp and manipulate various objects make hyper-redundant robotic manipulators the ideal tool for a variety of applications. Such applications include search and rescue, exploration of unknown environments, assembly and manufacturing as well as robotic-surgery. A key challenge that has limited the applicability of such robotic manipulators is the difficulty in controlling a robot with a very large number of interacting components. This paper aims to address this issue by using a new adaptive multiagent control approach for flexible shape changes of so called snake-arms and discusses extensions including a universal, more versatile, tree-like robotic manipulator. Our preliminary results show the feasibility of the approach and the potential benefits of a multiagent approach which includes scalability, fault tolerance, adaptivity and automated load balancing. A key finding of this study is the necessity of principled credit assignment for the agents, so that their collective actions optimize the performance of the full robot structure.

Keywords
Multi-agent systems, hyper-redundant robotic manipulators

1. INTRODUCTION

Hyper-redundant robotic manipulators (HRRMs) [7] are modular robot arms that are able to reach into confined and chaotic environments [6, 30] while using flexible shape changes to avoid obstacles. Form and capabilities are modeled to match examples from nature such as elephant trunks, tentacles or snakes. High maneuverability is reached by linking a large amount of kinematic modules on one another, consequently resulting in a manipulator with many degrees of freedom. This restrains the possibility to use traditional centralized control methods as for instance inverse kinematics algorithms. We propose to distribute the control problem using a neural-evolving multi-agent system in which each kinematic module is actuated by a separate agent.

Multiagent systems is an area of Artificial Intelligence that studies the decentralized behaviors of interacting autonomous artificial agents [15, 32]. Multiagent systems offer a novel solution paradigm for complex problems of today’s society. More and more contemporary technological challenges, such as automated traffic coordination and distributed sensing, require decentralized solutions. Multiagent systems represent an effective framework for investigating the behavior of distributed systems of interacting agents (robots, software agents and humans) and for developing solutions to control and improve such systems. The focus on agent interactions distinguishes multiagent systems from other disciplines like distributed control or distributed artificial intelligence.

Multiagent systems have build-in redundancy that allows for robustness: if one or a few components fail the system is still able to operate properly. In addition, different tasks can be assigned to independent agents pursuing these goals in parallel. A multi-agent system is modular per definition and components can be added easily, hence multi-agent system also scale better than centralized systems. These qualities make multi-agent systems desirable or even essential for a variety of applications. In particular, multi-agent control fits perfectly to the needs of HRRM control systems. Indeed, a robotic manipulator that is built from possibly hundreds of components will need to have the following properties:

- Adaptive to the changes within the environment, as determining how each component should operate will be an intractable task;
- Fault tolerant, as some components will fail or degrade during operation;
- Flexible in the number of components that are needed for a particular task and have links (agents) added or subtracted as needed; and
- Reconfigurable in its overall shape as multiple extensions may be needed for different tasks.

While a multiagent approach offers an elegant solution to such issues, it also introduces new problems. For example, the presence of multiple (learning) agents results in a highly dynamic and nondeterministic environment. In the case of
HRRM, due to the linking of segments, agents are highly dependent on each other and therefore mutually influence the learning performance. Evaluating the performance of a single agent becomes a critical design question as the decision has direct implication on the performance of the full system. In general, the development of a credit assignment structure that is both sensitive to a particular agent’s actions (e.g., an agent’s reward depends heavily on its own actions) and aligned with the system (e.g., what is good for the agent is also good for the system) is crucial to allow for efficient learning in such a multiagent system.

Though learning methods such as reinforcement learning have been applied to HRRM controllers in early work, such approaches were based on treating the full system as a single learning entity (for example, Q-learning for HRRM controllers [11]). In addition, approaches based on neural networks have also been investigated [14]. Such approaches do provide adaptivity and some degree of fault tolerance, but because they do not exploit the distributed nature of multiagent systems, they do not provide the flexibility and reconfigurability that is needed in many instances.

This paper presents an approach to HRRM control having an agent at each link and ensuring the coordinated behavior of the system through aligning the rewards of the agents. Section 2 provides background information about hyper-redundant robotic manipulators and outlines task and application domains. Section 3 discusses key challenges in multi-agent learning. A presentation of preliminary results follows in Section 4. Finally, we provide a brief discussion and highlight future directions for this work in Section 5.

2. HYPER-REdundant Manipulators
Hyper-redundant robotic manipulators (HRRM) embody a high degree of kinematic redundancy. "Hyper-redundant" meaning that the number of actuated degrees of freedom exceeds the minimal number required to perform a certain task (e.g., end-effector relocation in three dimensional space). In particular, kinematic modules are chained together to form a robotic limb that allows for versatile shape changes. These robotic manipulators are mechanically biomorph to elephant trunks, tentacles, worms or snakes. Therefore they are also referred to as snakearms. We distinguish snakearms from snakebots, the former depict robotic manipulators whereas the latter resemble full autonomous robots. Due to their flexibility, snakebots successfully cope with different terrains and are able to perform a variety of locomotion behaviors including but not limited to longitudinal locomotion, rolling, swimming and climbing. However, this work focuses on HRRMs rather than full autonomous robots. Figure 1 shows a HRRM with 22 kinematic modules. The fixed end of the robotic limb is called root or anchor point. The free end is referred to as tip or head.

2.1 Mechanical design challenges
The mechanical design challenges of HRRMs are twofold:

1. Minimize the cross-sectional area and length of actuated joints.
2. Optimize the torque-weight ratio of actuated joints.

Figure 1: Snake-arm grasping a round object without the use of an end-effector.

To build a highly flexible and versatile robotic manipulator the number of degrees of freedom must be large with respect to the length of the robotic limb. This directly translates to the first challenge. Each kinematic module (in the later context called segment) features one joint with one or more degrees of freedom. In the former case segments are chained together while rotating every other segment by 90 degrees along the longitudinal axis and hence resulting in a manipulator that can actuate in three dimensional space.

Despite a concise format, actuated joints for HRRMs also require an optimal torque-weight ratio. The drive torque for actuated joints near the root is much higher since they support the whole or a large part of the robotic limb. Electric motor powered joints deliver a limited torque-weight ratio and hence are only suitable for HRRMs with restricted segment quantity and length. Pneumatic as well as wire-driven joints use an actuator unit which is either mounted on the root segment or remotely connected via air or wire pipes respectively. Excluding heavy-weighted components like air compressors or electric motors for the control of drive wires reduces joint weight while delivering the same amount of torque. However, the actuator units are often bulky and therefore not suitable for every application (e.g. mobile robots). It should be noted that some HRRMs designs feature a pointing structure which means that limb segments decrease in cross-sectional area toward the tip. These designs mimic the biological examples more precisely (i.e. elephant trunks, tentacles) thus being superior in performing grasping tasks and furthermore reduce torque load by decreasing weight.

With the advancements in material sciences during the last decade a new option for joint actuators has opened up: ultra-compact and lightweight binary dielectric polymer ac-
tuators [29]. Binary actuators have two stable states and hence robotic manipulators based on such actuators can only reach a finite number of locations. However, chaining a large number of binary actuators together forms a HRRM that is comparable to manipulators featuring continuous actuators. Characteristics as light-weighted, high persistence and cost efficiency make binary actuators a competitive alternative to conventional ones [20].

2.2 Control challenges
The control process of robotic manipulators is closely related to the inverse kinematics problem. As opposed to the forward kinematic problem, the joint angles are unknown but the target shape is defined in terms of task constraints. For example: Consider a humanoid robot arm. The goal is to relocate the robot hand to a new location. The inverse kinematics problem is the problem of finding a shoulder and elbow joint angle configuration that moves the hand to the target position.

In the case of HRRMs the inverse kinematic problem is much more expensive due to the enormously increased number of degrees of freedom. Furthermore, a unique solution is very unlikely (depending on the constraints). Hence, an additional cost function and heuristic must be used to search for an efficient solution (e.g. minimizing the difference to the previous configuration). Traditional inverse kinematic algorithms are unsuitable for HRRMs due to computational inefficiency caused by expensive inverse matrix operations. The approach presented in [8] and [9] overcomes this problem by reducing the inverse kinematics problem to determining the backbone curve behavior. Where the backbone curve captures the robotic arm’s macroscopic geometric features using curve shape functions restricted to a modal form. Another approach that also disclaims expensive high-dimensional matrix operations is presented in [28].

2.3 Primitive HRRMs movement patterns
HRRMs are used within a diverse range of environments and applications. This section identifies three movement patterns that form the most recurrent tasks.

End-effector relocation
The most simple task a HRRM controller might face is the relocation of the end-effector mounted at its tip. It is for instance required for traditional robotic tasks like assembly work within some sort of manufacturing factory. Obviously in this case accuracy is the key factor directly followed by cost optimization (i.e. relocation speed). Considering the manufacturing environment, tasks are performed iteratively over and over again in the exact same way and therefore can be easily pre-calculated and optimized by traditional inverse kinematics algorithms (see Section 2.2).

Head following
Head following behavior is essential for applications where the HRRM must intrude a confined and chaotic environment as for instance present in search and rescue missions. The robotic arm is attached to some sort of feeder which first inserts the head and then consecutively the remaining segments. Since the environment is unknown in advance crucial obstacle data is missing to identify task constraints for the inverse kinematics methods. The head following approach solves this problem by restricting the movement of following segments to be as close to the path of the head as possible. Where the head features some sort of sensor equipment which helps to manually or automatically guide it around obstacles.

Grasping without end-effector
The third basic movement pattern can be best described by the biological inspiration. Tentacles grasp by curling the end of an arm around an object (see Figure 1). This method is superior to simple robotic grippers which can only grasp objects that are subjected to defined shape and size restrictions. However, it is not easy to define manually which shape configuration holds an object optimally thus making grasping an ideal application for adaptive control.

2.4 Applications
HRMMs find application in a variety of domains and different branches of industry. The versatility and flexibility of HRMMs allows to intrude confined areas which are inaccessible for conventional industrial robot arms. Different end-effectors (e.g. gripper, welding apparatus or observation sensors) can be mounted at the tip of the robotic limb to pursue required tasks. A list of application areas include nuclear power plants, aerospace and automotive industry, security, search and rescue tasks as well as robotic surgery [7, 6, 30].

3. MULTIAGENT LEARNING
Multiagent systems focus on interactions between agents which distinguishes them from other disciplines like distributed control or distributed AI. Unfortunately, it is not trivial or straight forward to identify what interactions cause the emergence of an intelligent system. Therefore learning in multiagent systems is not based on learning by supervision or examples, as for instance techniques from the machine learning area, but rather uses reinforcement signals from the environment - awarding good and punishing poor performances.

3.1 Reinforcement and neural-evolutionary learning
Reinforcement learning models the learning process of an adaptive agent [12, 21]. The agent iteratively interacts with its environment and observes a feedback signal from the environment in order to maximize its expected long term reward. As opposed to supervised learning the agent is not provided with the correct or optimal state-action pair at any time. Single agent reinforcement learning has been studied extensively in the past and recently multiagent reinforcement learning achieves great interest in the Artificial Intelligence research community [10, 13, 17, 18, 19, 22, 25, 26, 27].

Traditionally, reinforcement learning is strongly associated with the notion of value iteration. Value based learners estimate a state-action value function that determines the utility of performing a given action in a given state. Complex environments often implicate non-linear value functions. Thus they require reinforcement learning techniques based on general function approximation methods such as neural networks.
Neural-evolutionary algorithms are optimization techniques inspired by the biological concept of evolution. A population of solution concepts is subjected to the iterated application of evolutionary operators such as selection and mutation. The success potential of such candidate solutions is evaluated and measured by some fitness function. This technique can be transferred to the concept of reinforcement learning: The neural-evolutionary algorithm evaluates candidate solutions situated in an environment and the observed reward signal serves as a fitness measurement. Reinforcement learners attempt to maximize the expected reward in the same manner as evolutionary algorithms maximize the overall fitness [16].

3.2 Credit assignment problem

Learning in multiagent systems is inherently more challenging than single-agent learning. With the presence of multiple adaptive agents, the environment becomes highly dynamic and non-deterministic. Furthermore, the reinforcement signal is only a global evaluation of the entire system. The credit assignment problem addresses the question how to split credit among the agents in a fair manner.

Designing optimal credit assignments is a promising way to boost learning performance and speed. This paper builds upon work by Tumer et al. [1, 2, 3, 4, 23, 24, 31] and transfers results to the domain of HRRM. Designing personal agent reward structures that are both factored and measurable is a crucial point in order to increase learning performance and speed. Factoredness determines the degree of alignment between personal and global goals, whereas learnability measures the sensitivity of an agent’s reward function to its own actions.

3.3 Evaluation function properties

In this section we illustrate some important properties of evaluation functions based on work described in [31] and in the context of multi-rover control work described in [1, 2, 23]. Let the system evaluation function be given by $G(z)$, where $z$ is the state of the full system (e.g., the relevant internal parameters of all agents, in this case all joint angles, and the state of the environment). Let the private evaluation function for agent $i$ be given by $g_i(z)$. First we want the private evaluation functions of each agent to have high factoredness with respect to $G$, intuitively meaning that an action taken by an agent that improves its private evaluation function also improves the system evaluation function (i.e., $G$ and $g_i$ are aligned). Formally, the degree of factoredness between $g_i$ and $G$ is given by:

$$F_{g_i} = \frac{\int_i \int_i u[(g_i(z) - g_i(z')) (G(z) - G(z'))]dz' dz}{\int_i \int_i dz' dz},$$

where $z'$ is a state which only differs from $z$ in the state of agent $i$, and $u[x]$ is the unit step function, equal to 1 when $x > 0$. Intuitively, a high degree of factoredness between $g_i$ and $G$ means that an agent evolved to maximize $g_i$ will also maximizing $G$.

Second, the private evaluation function must be more sensitive to changes in that agent’s performance than to changes in the performance of all the other agents. Formally, we quantify the agent-sensitivity of evaluation function $g_i(z)$ as:

$$\lambda_{g_i}(z) = \mathbb{E}_{z^i} \left[ \frac{|g_i(z) - g_i(z - z_i + z_i')|}{|g_i(z) - g_i(z' - z_i' + z_i')|} \right],$$

where $\mathbb{E}_{z^i}[]$ provides the expected value over possible values of $z'$, and $(z - z_i + z_i')$ specifies the state vector where the components of agent $i$ have been removed from state $z$ and replaced by the components of agent $i$ from state $z'$. So at a given state $z$, the higher the agent-sensitivity, the more $g_i(z)$ depends on changes to the state of agent $i$, i.e., the better the associated signal-to-noise ratio for $i$. Intuitively then, higher agent-sensitivity means there is “cleaner” (e.g., less noisy) selective pressure on agent $i$. Ideally we want evaluation functions that are both factored and highly agent-sensitive.

3.4 Difference evaluation functions

Let us now focus on improving the agent-sensitivity of the evaluation functions. To that end, consider difference evaluation functions [31], which are of the form:

$$D_i \equiv G(z) - G(z - c_i),$$

where $z - c_i$ contains all the states on which agent $i$ has no effect, and $c_i$ is a fixed vector. In other words, all the components of $z$ that are affected by agent $i$ are replaced with the fixed vector $c_i$. Such difference evaluation functions are fully factored no matter what the choice of $c_i$, because the second term does not depend on $i$’s states [31] (e.g., $D$ and $G$ will have the same derivative with respect to $z_i$). Furthermore, these difference evaluation functions usually have far better agent-sensitivity than a system evaluation function; the second term of $D$ removes some of the effect of other agents (i.e., noise) from $i$’s evaluation function. In many situations it is possible to use a $c_i$ that is equivalent to taking agent $i$ out of the system. Intuitively this causes the second term of the difference evaluation function to evaluate the fitness of the system without $i$ and therefore $D$ evaluates the agent’s contribution to the system evaluation.

Deliberated design of evaluation functions is crucial to enable the evolution of multiagent systems. The design of inter-agent-interactions which eventually lead to the emergence of intelligent behavior is not straightforward and hence evolution is the most practicable option. In turn this makes the credit assignment problem one of the key challenges in multiagent research.

4. PRELIMINARY STUDY RESULTS

This section presents initial experiments and results of the multiagent approach to hyper-redundant robotic manipulators. The initial experiments directly build on the work described in the previous section. We use a set of agents subjected to a common goal: end-effector relocation. The system consists of a simple HRRM in planar space constructed from $N$ kinematic modules using continuous joint actuators where each agent is in control of one joint. The global evaluation function is given by:

$$G = -\delta(e, t) = -||e - t||,$$

where $e$ and $t$ are the planar positions of the end-effector and target respectively.

In our experiments the agents are situated in a dynamic and episodic environment. This means all agents can act within
a time frame of \( \tau \) steps, called episode. After the episode ends, \( G \) is used to evaluate the performance of the system. Hereafter the target position \( t \) is randomly relocated within a half circle around the anchor position of the manipulator.

The input (or perception) space of agents is in \( \mathbb{R}^4 \). We define \( s_i \) as a vector pointing to the center location of segment \( i \). Then the input vector for each segment \( i \) consists of the four components: distance between center of segment \( i \) and the target location \( ||s_i - t|| \), angle between segment \( i \)'s orientation and the vector \( s_i - t \), the local joint angle and the average of \( k \) surrounding joint angles. The first three components are local perceptions while the last component gives an indication of the global curving.

Agents can manipulate the joint angle value of a segment by superimposing an output signal \( o \in [-1, 1] \) which is scaled by a factor \( \alpha \). Hence the angle change per step is limited within the range \( [-\alpha, \alpha] \). This design is favored over agents which can set absolute joint angle values since it limits the dynamics and reduces the impact on other agents.

A Multi Layer Perceptron (MLP) performs the input to output mapping. Our approach uses an evolutionary algorithm to search through the solution space of MLPs. Each agent maintains a population of neural networks. At the begin of each episode one MLP controller is selected due to a \( \epsilon \)-greedy criteria. With a fixed probability of \( p \) mutation is performed by superimposing noise on the weight matrix of the selected neural network. After the episode ends the fitness value is assigned according to the evaluation function.

The distributed HRRM end-effector relocation control task is particularly difficult because agents are highly dependent on each other. Therefore, it is not possible to design a straightforward difference evaluation function as discussed in Section 3.4. To overcome this problem our initial experiments use a method that builds difference evaluation values in multiple steps [22]: First the entire system is evaluated for the period of one episode. This global evaluation based on (4) serves as a reference point. Then the environment is repeatedly reset to the initial state of the episode and for each agent \( i \) the episode is re-evaluated while removing its impact. In this particular case we neglect actions from agent \( i \) during the evaluation (meaning one segment is disabled). The difference between system evaluation and evaluation without impact of agent \( i \) is used as an estimate for the performance of agent \( i \). It should be noted that this increases the simulation time by a factor of \( N \).

Preliminary results show that the difference evaluation indeed outperforms a multiagent system trained with the global evaluation function. Figure 2 plots the average error rate over learning iterations of 15 independent runs using \( N = 7, k = 3 \) and \( \tau = 150 \) with \( \alpha = 5^\circ \). A successful end-effector relocation performed by a trained multiagent system is displayed in Figure 3.

![Figure 2: Results for learning in the end-effector relocation task. Comparison between global \( G \) and difference evaluation function \( D \).](image)

![Figure 3: Successful single HRRM end-effector relocation, to be read from left to right and top to bottom. The sequence shows joint configurations for consecutive steps where the target position is indicated by a circle.](image)

Furthermore, the results are extended to investigate the scaling performance of global and difference evaluation functions for the same relocation task. The number of segments was consecutively increased from \( N = 3 \ldots 10 \) while all other parameters remain fixed. Figure 4 plots the average final error after 800 episodes for 20 independent runs. Systems that are trained with \( G \) show learning achievements up to \( N = 6 \) segments. If the complexity is further increased the agent-sensitivity of \( G \) is too low to allow for learning; this results in a performance level only as good as a random controller. The difference utility function \( D \) remains learnable even if the number of segments is further increased.

These findings affirm the preceding statement that delib-
ered and principled credit assignment design is crucial to enable the evolution of complex multiagent systems (see Section 3.4).

Figure 4: Scaling results for learning in the end-effector relocation task. Comparison between global evaluation function $G$, difference evaluation function $D$ and a random controller.

5. DISCUSSION AND FUTURE DIRECTIONS

In this paper we have shown the feasibility of an adaptive multiagent control method for hyper-redundant manipulators. This new approach uses neural-evolutionary learning techniques in combination with intelligent credit allocation built on difference evaluation functions. We have transferred recent research developments to this robotic application and preliminary results indicate the potential of difference evaluation functions within this domain.

The long term research potential for this project is very promising, and the following topics require particular attention. First, the end-effector relocation task was used in this study to explore different input/output configurations for agents. This basic task is used to design and evaluate difference functions. Extending these results to object avoidance, realistic local sensing and grasping without end-effectors are logical next steps, before the results can be credibly extended to three dimensional problems and applied to a real life robot.

Second, the HRRM presented in this study was a single, snake-like structure. Either using multiple HRRMs or using a single forked HRRM for grasping is an appealing extension to snake-like HRRMs. A forked-HRRM as shown in Figure 5 can both hold and rotate an object which is a desirable property.

Third, the forked-HRRM concept can be pushed further, leading to a universal robotic manipulator resembling a "living"-tree like structure. Advantages of such a configuration include an increase in application and task versatility. In addition to holding and rotating objects, such an extended HRRM can use one subtree to hold and another to observe or manipulate the object. In addition, this structure allows transition from tasks requiring finesse to power by modifying the level at which the object is grasped. Figure 6 depicts such a tree-like HRRM concept.

In the case of a HRRM tree the mechanical design challenges as discussed in Section 2.1 become even more important. The extended amount of segments demands compact and light-weighted yet powerful actuators. This makes the investigation of binary actuators for the use in extended HRRMs worthwhile, not only from the mechanical but also for the control perspective: interactions between simple agents cause complex systems which are not trivial to control but often lead to the emergence of intelligent behavior [5].

Figure 5: Extended HRRM grasping an object with two branches (bottom). Either two HRRMs or a “forked” HRRM can be used for grasping objects.

Figure 6: Tree-like extended hyper-redundant robotic manipulator. Such an HRRM can be used for both tasks requiring finesse (using the tips) or power (using the stronger trunks).
Finally, if the binary actuators are combined with simple local sensing (i.e., discrete number of input states) the control task does no longer require general function approximation for the input-output mapping. This opens the possibility to use other learning techniques besides neural-evolutionary learning; reinforcement learning such as Q-learning and more advanced alternatives can be applied and investigated.

In all cases, the extensions to the basic HRRM structure allows for increased functionality and redundancy. As such the multiagent coordination aspects loom large for each of these steps as the performance will depend on how the agents coordinate (first and second extensions) or how the agents cope with their own learning problems (third extension). We are currently investigating the effectiveness of multiagent learning approaches to the control of HRRMs and the extensions that will allow HRRMs to be applicable to diverse real world problems.

6. REFERENCES


Learning to Control the Emergent Behaviour of a Multi-agent System

Urban Richter
Karlsruhe Institute of Technology
Institute AIFB
76128, Karlsruhe, Germany
urban.richter@kit.edu

Moez Mnif
Leibniz Universität Hannover
Institute of Systems Engineering
Appelstr. 4, 30167 Hannover, Germany
mnif@sra.uni-hannover.de

ABSTRACT
Organic Computing (OC) has the vision of addressing the challenges of complex distributed systems by making them more life-like (organic), i.e., endowing them with abilities such as self-organisation, self-configuration, self-repair, or adaptation. This can only be achieved by giving the system elements adequate degrees of freedom, which may result in an emergent behaviour, which can be positive (desired) as well as negative (undesired). In this context, the so-called observer/controller architecture has become widespread in the OC community, as a design paradigm to assure the fulfilment of system goals (given by the developer or user). An observer/controller loop enables adequate reactions to control the – sometimes completely unexpected – undesired emerging global behaviour resulting from local agents’ behaviour. Therefore, organic systems need the ability to quantify the system status as well as the ability to learn (because in most cases the developer cannot predict all system states).

In this paper, a nature-inspired multi-agent scenario is taken to validate a generic observer/controller architecture, which has been designed as part of an intended organic framework – providing generic toolbox mechanisms to observe, analyse, and control emergent behaviour in self-organising systems. In particular, we discuss the learning ability of the 2-level learning architecture of the controller by presenting first experimental results of different learning methods and strategies.

General Terms
learning, emergence, Organic Computing

Keywords
entropy, observer/controller, learning classifier systems

1. INTRODUCTION
Today’s technical systems are becoming increasingly complex. Future systems will consist of a multitude of complex soft- and hardware components (multi-agent systems), which interact with each other to satisfy global system functional requirements. This trend bears also the risk of more and more breakdowns and other unexpected behaviour.

Organic Computing (OC), as an emerging research field in Germany 1 and worldwide 2, is driven by the vision of addressing the challenges of complex distributed systems by making them more life-like (organic), i.e., endowing them with abilities such as self-organisation, self-configuration, self-repair, or adaptation. This can only be achieved by giving the system elements some kind of awareness of their current situation and the ability to provide appropriate responses to dynamically changing environmental conditions (by giving them adequate degrees of freedom). The principles of OC are strongly related to the objectives of Autonomic Computing [9].

Self-organising systems bear several advantages compared to classical, centrally controlled systems. Amongst others, the failure of a single component should not cause a global malfunction of the whole system. Such a system will be able to adapt to changing circumstances. As a result, self-organisation could be viewed to be a means of reducing the complexity of computer systems. Nevertheless, self-organising systems are complex systems themselves. However, the user does not have to manage this complexity as, to a far extent, the system manages itself, that means the externally visible complexity is reduced.

In such self-organising systems the local interaction of the system elements may result in an emergent global behaviour, which can have positive (desired) as well as negative (undesired) effects. In this context, the so-called observer/controller architecture has become widespread in the OC community as a design paradigm to assure the fulfilment of system goals (given by the developer or user). An observer/controller loop enables adequate reactions to control the – sometimes completely unexpected – undesired emerging global behaviour resulting from local agents’ behaviour.

Self-organisation and emergent phenomena also give rise to new problems unknown in the engineering of classical technical systems. A global emergent behaviour usually is a non-linear combination of local behaviours. Its design process with both potential design directions (top-down vs. bottom-up design) turns out to be a highly non-trivial task: For a top-down approach it is hard to deduce adequate local

1OC is also the name of a priority programme funded by the German Science Foundation (DFG) (2005–2011).
2There are many conferences and workshops addressing the topic of OC, like ARCS, ATC, CEC, etc.
rules from a desired global behaviour, and in the bottom-up direction it remains quite often unclear how local rules map to a global behaviour, see [9].

1.1 Controlled Self-organisation

Computer systems should use self-organisation to achieve a certain externally provided goal. Therefore, OC systems are assumed to support controlled self-organisation. This requires a range of methods for monitoring and analysing the system performance and for providing appropriate control actions whenever necessary. The generic observer/controller architecture as specified in [1, 13] promises to provide the necessary components for satisfying all these demands (cf. Fig. 1). Similar to the MAPE cycle (monitor, analyse, plan, and execute) of Autonomic Computing (cf. [9]), a closed control loop is defined to keep the properties of the system under observation and control (SuOC) within preferred boundaries. The observer observes certain (raw) attributes of the system and aggregates them to situation parameters, which characterise concisely the observed situation from a global point of view, and passes them to the controller. The controller acts according to an evaluation of the observation (which might include the prediction of future behaviour); if the current situation does not satisfy the requirements, it takes action(s) to direct the system back into its desired range, observes the effect of the intervention(s), and takes further actions, if necessary. Using this control loop an organic system will adapt over time to its changing environment. It is obvious that the controller should make use of a (real-time) learning capability to tackle these challenges. Although the observing and controlling process is executed in a continuous loop, the SuOC is assumed to run autonomously even if the observer and controller are not present – even though in a suboptimal way.

In this paper, a nature-inspired multi-agent scenario is taken to validate the generic observer/controller architecture, which has been designed as a part of an intended organic framework – providing generic toolbox mechanisms to observe, analyse, and control emergent behaviour in self-organising systems. In particular, we discuss the learning ability of the controller by presenting experimental results of different learning methods and strategies. The paper is structured as follows: In Sect. 2 we summarise shortly the main ideas of the generic observer/controller architecture. In Sect. 3 we give an overview of how we define emergence in technical scenarios and how we measure these phenomena. Learning as a special capability of the controller and a special focus of this paper is described in Sect. 4. The experimental simulation environment, which is used for investigations, is presented in Sect. 5. Experimental results of applying the observer/controller paradigm to our scenario are shown in Sect. 6. The paper concludes with a summary and a discussion of future work.

2. OBSERVER/CONTROLLER

Reducing the complexity of tomorrow’s technical systems along their whole product life cycle (from design to productive deployment and maintenance) OC systems are endowed with self-x-properties making them flexible and adaptive.

Examples motivating this goal are frequent in nature. In contrast to natural systems, which seem to have intrinsic goals, the technical context specifies the system goals explicitly and in many cases even requires the fulfilment of some constraints. Therefore, reaching the stage of endowing the system with intrinsic local goals can be seen as long-term objective. The design process of such an organic system will be neither a classical top-down approach, as practised in classical system engineering, neither a bottom-up approach, as suggested by the notion of self-organisation in general and as observed in nature. In OC technical systems are endowed with an observation and control layer called observer/controller architecture. This design paradigm is necessary to achieve the objectives given by the user or the developer.

As shown in Fig. 2, the SuOC accomplishes the productive work of the system and is endowed with the self-x-properties. The SuOC is similar to a multi-agent system composed of a multitude of agents communicating with each other to achieve a system-wide goal based on local rules. The central observer/controller layer – for a hierarchical or distributed version of this architecture see [4] – monitors all components and aggregates the results to system-wide indicators reflecting the overall situation of the system (called situation parameters or system fingerprint). This set of situation parameters is reported to the controller, which has the task to influence the system to satisfy the objective function as specified by the system developer or user. Therefore, the controller searches continuously its rule base for the best mapping of situation parameters to correspondent actions and has to adapt dynamically to changing environments (changes in the SuOC or changing goals). This changes require learning capabilities, as investigated in Sect. 4.

The observer collects the raw data coming from the SuOC and aggregates them to a global system-wide fingerprint. This process includes a pre-processing of the data (smoothing, extraction of derived attributes like velocity, etc.), an analysis to determine system-wide indicators, and a predictor to predict the next raw data as well as the next system wide indicators (by using specified or statistical methods like chart analysis methods). For this purpose, we need metrics and methods to quantify emergent states of the system. Finally, the aggregator collects all this aggregated information and passes it to the controller.
3. HOW TO DEFINE EMERGENCE IN A TECHNICAL CONTEXT

Defining emergence is a rather controversial issue. In literature, several definitions from different scientific fields are found. These definitions go from ones with a whiff of magic (non-explainability, non-predictability, etc.) to ones which try to give a formal mathematical definition [6]. In a technical context it is obvious that we have to dissociate from the former ones and have to search for a formal – and at the same time – practically usable definition of emergence. In the following we sum up an entropy-based definition of emergence, which we have elaborated on in former work [11].

For this paper we consider and discuss only the learning of emergent states of the system coming out as the result of a self-organisation process. In general, the observer selects adequate situation parameters allowing the controller to influence the system properly. However, in this paper we will concentrate on emergent phenomena as a very interesting effect in multi-agent systems.

Observations made in nature and in further emergent systems have shown that certain ingredients seem to be necessary to call an observed phenomenon emergent: A large population of interacting elements (agents) without (or with a minimum) of central control which leads to a macroscopic pattern which is perceived as structure or order. So we can say that emergence can be characterised as self-organised order.

A well-known metric to quantify order is the entropy used in thermodynamics and information theory. Low entropy is equivalent to a higher system order and the other way around. We propose the following entropy-based method to quantify emergence. This method produces a fingerprint characterising the whole system as a result of a transformation of the raw data to some higher abstract metrics. The main idea of computing the fingerprint of a system is firstly to identify the common attributes of its elements, secondly to build for each attribute a relative frequency of the occurrence of each value (which can be considered as a probability distribution), and thirdly to compute the entropy related to each attribute on basis of Shannon’s information theoretical definition of the entropy:

\[
H_A = \sum_{j=0}^{N-1} p_j \cdot \log \frac{1}{p_j}
\]

in which \(A\) represents a given attribute and \(N\) the number of the different attribute values (the unit of measurement is bit/attribute). Finally, one has to compute the degree of emergent order of each attribute according to:

\[
E_A = H_{A_{\max}} - H_A
\]

\(H_{A_{\max}}\) is the entropy of the attribute \(A\) in case of an equal distribution of attribute values (lowest level of order). Instead of the emergence \(E\), we can also compute a relative emergence \(e\) according to:

\[
e_A = \frac{H_{A_{\max}} - H_A}{H_{A_{\max}}} \quad (0 \leq e_A \leq 1)
\]

The list of all \(E_k\) or \(e_k\) (\(k\) denoting an attribute) values of the SuOC constitutes a vector, which is called system fingerprint (see also Fig. 3) and which evolves over time.

4. LEARNING

Learning is a key aspect of OC systems in general and of the generic observer/controller architecture in particular. In OC systems the agents of the SuOC and the controller itself could be endowed both with the capability of learning. For this paper we consider and discuss only the learning
capability of the controller as mentioned in Sect. 2, learning on agents’ level is the topic of future work.

4.1 General Thoughts

In general, a system has the capability to learn, if it can improve autonomously its response to input values from some set X. That means, take time values \( t_1 \) and \( t_2 > t_1 \) such that for any \( t > 0 \) the response to an input from the set \( X \) at time \( t_2 + t \) has a higher quality than the response to the same input at time \( t_1 \).

This learning capability requires some learning mechanism which may modify the behaviour of the system by (i) changing the values of some attributes of the system or of its environment, or (ii) changing the behavioural repertoire of the control mechanism (which corresponds to our understanding of observer/controller).

There is a broad range of possible learning mechanisms, which could make use of learning by experience, trial-and-error, reinforcement learning (RL), neural networks, or meta-heuristics like genetic algorithms, ant colony optimisation, or simulated annealing, to name a few.

In the case of the generic controller we decided to combine the advantages of an on-line learning on Level 1 (to tackle the real-time requirements of organic systems) with the advantages of an off-line planning instance on Level 2 (to minimise the possible mistakes that may arise during on-line learning).

4.2 On-line Learning and Planning Capabilities

As shown in Sect. 2, we have developed a multi-level observer/controller architecture to enable controlled self-organisation in technical applications, which has the capability to adapt to changes on two levels: At the on-line level (Level 1) the proposed architecture learns about the environment, and about the performance of its control strategies. It does so on-line and in real-time. Level 2 implements a planning capability based on a simulated model of the environment. At this level the agent can test and compare different alternative strategies off-line, and thus plan its next action without actually acting in the environment. This 2-level architecture has (theoretically) several important advantages: (i) The off-line planning allows to find appropriate actions without actually having to test different alternatives in the real world. The latter could be detrimental, as testing out potentially bad strategies in the real world can incur a tremendous cost and cause the system to fail permanently. (ii) Level 1 acts as a kind of memory, and allows to react quickly. If a situation close to a previously encountered situation re-appears, the system can respond immediately. (iii) A model-based planning as on Level 2 is always limited by the necessary simplifications made in the model. Thus, the best action with respect to the model is not necessarily also the best action with respect to the real world. In our architecture, Level 1 thus is allowed to slightly fine-tune the solutions from the planning module.

The proposed architecture has similarities with model predictive control [3], which basically only uses Level 2 planning and an on-line adaptation of the Level 2 simulation model. It also has similarities with pure on-line learning mechanisms like learning classifier systems (LCS) or RL, which only act on Level 1. And there is the idea of anytime learning by Grefenstette and Ramsey [7, 12], where an EA is equipped with a case-based memory, which can be seen as a 2-level structure, but no off-line planning takes place.

With the best of our knowledge, our proposed 2-level learning approach as part of our observer/controller architecture is novel in the area of collaborative multi-agent systems. Our architecture has the advantage of on-line adaptation in real-time and prevents the disadvantage of testing bad solutions in the real world by using a model of the reality for validation of promising new actions.

4.3 Learning Classifier Systems

As depicted in Sect. 2, the controller uses some kind of mapping to choose an appropriate parameter set to configure the action that prevents emergence as measured by the observer. To keep the problem manageable, the input space has to be partitioned so that situations which are sufficiently similar to allow usage of the same parameter set will be covered by the same mapping entry. Since it is hard to anticipate all situations that might occur at the controlled scenario and since it would need a tremendous effort for a human developer to develop good control strategies for all situations that can be imagined, the means for generating mapping entries have to be provided by the controller. These tasks combined are what LCS [8] are supposed to do: classify input, find appropriate action, and learn by gaining experience.

A standard LCS uses rules (called classifiers) to store its knowledge. These classifiers consist of a condition that is matched against the input, an action, and some kind of fitness or value. The value is used to decide which classifier should be chosen if more than one classifier matches the current input. The encoding of conditions is done such that different levels of “generality” are possible, hence the range of input values each classifier matches against may vary from a single point to the entire search space. In a step that is called “rule discovery” new classifiers are generated using genetic operators like crossover and mutation on existing ones, changing both condition and action. Furthermore, every time no classifier matching the current input is available, one or more classifiers with a matching condition and randomly chosen action are created (this is called “covering”).

After a classifier has been generated, the system has to determine its value. Every time a classifier’s action is chosen, the value of this classifier is updated using some objective function. Usually, the environment’s reaction on some action executed by the LCS is not instantaneous, therefore the current value of the objective function is used to update the value of the classifier that was active during the previous time step.

The simplest way to increase performance would be to try and find classifiers which maximise the objective function’s value. Another approach more suitable for complex problems is to use three distinct components as “value”: the prediction \( p \) of the objective function’s value after this classifier’s action has been executed, the average error of this prediction \( e \), and a fitness value \( F \) computed from this prediction error. The prediction is used to choose an action for execution: The classifier with best prediction, weighted by its fitness, is likely to result in best performance. Furthermore, the fitness is used to find classifiers suitable as input for the generation of new classifiers using genetic operators. The goal is to represent the entire search space with as few
classifiers as possible while keeping only those that are more accurate in their prediction. This approach is realised in XCS, the eXtended Classifier System [16].

For LCS a wide variety of different implementations has been proposed, most of which are based on work done by Wilson [15, 16]. While Wilson used a binary coding of the stimuli for these rather simple LCSs, different approaches to represent real-valued input have been examined [5, 14, 17].

5. EXPERIMENTAL ENVIRONMENT

The multi-agent system described below and used for the experimental validation of the observer/controller architecture in this paper is inspired by nature and shows clustering from a macroscopic point of view as an emergent behaviour of local interactions. The simulation reproduces the collective cannibalistic behaviour of densely packed chickens in cages (cooperation with the University of Veterinary Medicine Hannover) and tries to explain the unwanted behaviour of clustering, which is frequently observed when a chicken is wounded, and which leads to a major loss of animals (up to 50% of the animals). If chickens perceive a wounded chicken, they chase this chicken and pick on it until it dies [11]. Chasing and picking wounded chickens leads to the emergent building of chicken swarms (or clusters). A swarm disperses when the wounded chicken is killed. The emergent behaviour is spatial, but swarms move over time. This is a case of negative, i.e., undesired, emergence.

While simulating this behaviour, order patterns emerge as expected in form of chicken swarms. Currently, in agriculture these patterns are interpreted by human experts. But from the viewpoint of OC it is the goal to observe, classify, and control this behaviour automatically. To achieve this goal and reduce the chicken death rate, we use the observer/controller paradigm as introduced in detail in Sect. 2 to report a quantified context of the underlying system to the controller, which evaluates the situation and reacts with adequate control actions to disperse chicken swarms or to prevent their formation.

![Figure 4: Snapshots of the chicken simulation: Unwounded chickens are white, wounded chickens are dark (red), and feeding troughs are represented by circles.](image)

(a) No cluster (a chasing situation)  (b) A cluster situation

In the simulation every chicken can move to eight different directions, one movement per each simulation tick (north, north-east, east, south-east, south, south-west, west, and north-west). All chickens move at the same speed (one field per simulation tick).

The simulation provides also the possibility to apply a noise signal with variable intensity $i$ and duration $d$ at any arbitrary position in the cage. This noise signal frightens the chickens, scares them off, which leads to the dispersion of a possible existent cluster. Moreover, the noise signal has a negative effect, too. If noise is too loud and occurs too long then chickens are killed (eventually) through noise intervention. Therefore, it won’t make sense to control continuously the chickens with a noise signal, unless killing all chickens is the focus.

A chicken is characterised by the attributes heading, position ($x$-, $y$-coordinates), and energy. It is directed by a predefined fixed finite state machine and will be influenced by the behaviour of other chickens in its local neighbourhood or by changes in the environment, e.g., noise that frightens the chickens. Chickens are considered as autonomous robots or agents with simple rules and local goals, they aim for surviving as long as possible, and they are attracted by wounded conspecifics. We should mention that the notions of robot, agent, or chicken have the same meaning in our context. We use the item chicken in analogy to the nature-inspired paradigm and abstract from the animal to presume that the single chicken is an autonomous robot or a technical agent, which shows no life of its own and instead reacts as specified by its developer.

Our simulation environment is set up with the parameters as listed in [11]. We observe a scenario of 40 chickens that move randomly in a playground, which has a dimension of $30 \times 30$ fields (cf. Fig. 4). When a chicken is killed, a new chicken is generated and placed randomly in the cage.

The observer collects the raw data of the chickens and computes, additionally to the emergence values $e_x$ of the $x$-coordinates, $e_y$ of the $y$-coordinates, and $e_h$ of the heading $^4$, the centroid $(x_c, y_c)$ of the chicken population. The situation parameters given to the controller are consequently a vector $S_t = (e_x, e_y, e_h, (x_c, y_c))$. The actions taken by the controller have the form $A = (d, i, (x_c, y_c))$, which can be simplified to $A = (d, i)$.

6. LEARNING TO CONTROL

Without controlling the global chicken behaviour approximately 336 chickens are killed during a simulation with 10 000 ticks. Applying a static noise signal with fixed intensity and fixed duration around the computed cluster centroid to frighten the chickens and disperse the cluster works well and decreases the death rate (cf. [11]). But this static control loop does not satisfy the idea of adaptation in changing environments.

Learning the best action for changing situations in the context of the chicken scenario means the following: At every simulation tick the global behaviour of the 40 agents characterised by three emergence values, $e_x$ for the $x$-coordinates, $e_y$ for the $y$-coordinates, and $e_h$ for the heading of all chicken, is evaluated. Based on this system fingerprint we use an XCS to learn if and which noise signal should be applied to the chicken cage. Two parameters belong to a noise signal: the duration $d$ (how long the signal is applied) and the intensity $i$ (which loudness of the noise signal is

$^4$The emergence of the energy value is not taken into account, since it is an intrinsic property of the agents, which cannot be seen by an external observer.
applied – it corresponds to the radius of the signal).

First investigations have been made with an action space spanned by \( d \in \{1, 2, \ldots, 15\} \) and \( i \in \{0, 10, 20, \ldots, 80\} \) (which results in 135 possible actions). It is obvious that only a (small) subset of these possible actions is reasonable. Using the standard XCS configuration given by Butz in [2], the XCS did not converge. To make the XCS converge, we assumed that \( t_h = 0.3 \) seems to be a reasonable threshold for the critical heading emergence (cf. [10]). To simplify our investigations for presentation within this paper, the other values are restricted to values as shown in Tab. 1.

Each line of Tab. 1 corresponds to a parameter set of a group of 18 tests which should be read as follows: Taking the first line in the table, i.e., it corresponds to tests with fixed single rule controllers with a noise signal \((d, i) = (5, i)\) which act if the critical emergence values match the observed situation with the constraint \((e_x \geq t_x, e_y \geq t_y)\) OR \((e_x \geq 0.1)\) OR \((e_y \geq 0.3)\) varying \(t_x \in \{0.1, \ldots, 0.6\}\) and \(i \in \{0, 10, 20\}\). The resulting objective space has a dimension of \# \(t_x \times \#t_y \times \#h \times \#d \times \#i = 6 \times 6 \times 3 \times 1 \times 3 = 540\). We have varied these parameter combinations, simulated each of them for 10000 ticks with 20 different seed values (to get a statistically valid result), and computed the average number of killed chickens \# \(kC\) (e.g., see Tab. 2). Fig. 5 shows the average number of killed chickens for the example of a fixed combination \(d = 5, t_y = 0.2, t_h = 0.3\) while varying \(t_x\) and \(i\).

### Table 1: Combinations of fixed single rule controller parameters

<table>
<thead>
<tr>
<th>(t_x)</th>
<th>(t_y)</th>
<th>(t_h)</th>
<th>(d)</th>
<th>(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1, 0.2, \ldots, 0.6</td>
<td>0.1, 0.2, \ldots, 0.6</td>
<td>0.1, 0.2, \ldots, 0.6</td>
<td>0.1, 0.2, \ldots, 0.6</td>
<td>0.1, 0.2, \ldots, 0.6</td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>5</td>
<td>0, 10, 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>9</td>
<td>0, 10, 20</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1 Parameter Studies

Essentially, the number of killed chickens is a function that depends on the mapping of a situation to an action. As mentioned before, the goal of the investigations with the fixed single rule controller is to search the objective space step by step for a possibly existent global optimum. The results are used to compare the quality of the XCS investigations. (ii) On-line investigation with an XCS: As shown in Sect. 4.3, the idea of XCS fits well to the organic observer/controller architecture. We adopt the XCS – as implemented by Butz – to the chicken scenario and use learning to minimise the chicken death rate. Of course, several learning combinations of the two mentioned extreme approaches (i) and (ii) are possible and as mentioned in Sect. 4.2 the 2-level approach is a possible combination of searching on-line the objective space by using off-line generated knowledge. But these combinations are not investigated in detail and will be subject of future work.

### 6.2 Using an XCS for Learning

We have taken the XCSJava1.0 reference implementation by Butz [2], modified it to work with real-valued input, simulated the chicken scenario with 20 different seed values and a maximal population of 800 classifiers, and computed again the average number of killed chickens \# \(kC\). As reward – to
Table 2: Results of the fixed single rule controller experiments over 10 000 ticks with the parameter combination \(d = 5\), \(t_y = 0.2\), and \(t_h = 0.3\)

<table>
<thead>
<tr>
<th>(t_x)</th>
<th>(t_y)</th>
<th>(t_h)</th>
<th>(d)</th>
<th>(average) #(kC)</th>
<th>(standard) deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>6</td>
<td>20</td>
<td>172.90</td>
</tr>
<tr>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>9</td>
<td>20</td>
<td>176.10</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>7</td>
<td>20</td>
<td>176.80</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>7</td>
<td>20</td>
<td>177.60</td>
</tr>
</tbody>
</table>

... ... ... ... ...

compute the fitness of a classifier – we monitor (and the XCS minimises) the number of killed chickens \#\(kC\) that are killed during and after an action \(A_i = (d_x, i)\) before the next action \(A_{j} = (d_y, j)\) (with \(i < j\)) occurs.

Line 6 of Tab. 4 shows the averaged result of simulation runs with 10 000 ticks. The attained average of 178.30 is very close to the best result of the parameter study (approximately 173), which is a very promising result. Furthermore, Tab. 4 shows experiments with varying simulation times to verify if the XCS converges with more simulation time to a better optimum. For the investigated scenario with 800 classifiers and the standard parameter setting by Butz a small improvement could be demonstrated if we simulate longer (cf. Fig. 6). The XCS is able to converge to a steady result.

![Figure 6: Number of killed chickens per 1 000 simulation ticks](image)

Table 3: Results of the single fixed rules experiments over 10 000 ticks sorted for the average \#\(kC\)

<table>
<thead>
<tr>
<th>(t_x), (t_y), (t_h)</th>
<th>(d), (i)</th>
<th>average #(kC)</th>
<th>standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1, 0.2, 0.3</td>
<td>6, 20</td>
<td>172.90</td>
<td>5.51</td>
</tr>
<tr>
<td>0.1, 0.1, 0.3</td>
<td>6, 20</td>
<td>173.05</td>
<td>7.13</td>
</tr>
<tr>
<td>0.3, 0.3, 0.3</td>
<td>9, 20</td>
<td>176.10</td>
<td>5.38</td>
</tr>
<tr>
<td>0.2, 0.3, 0.3</td>
<td>7, 20</td>
<td>176.80</td>
<td>4.25</td>
</tr>
<tr>
<td>0.1, 0.1, 0.3</td>
<td>7, 20</td>
<td>177.60</td>
<td>7.68</td>
</tr>
</tbody>
</table>

6.3 XCS vs. Single Fixed Rule Controller

Fig. 6 depicts the learning behaviour over time of the XCS in comparison to the best single fixed rule controller found through the parameter studies. It plots the average number of killed chickens per 1 000 ticks with increasing simulation time and each plotted value is the average over 20 runs. As expected, the single fixed rule controller shows a constant progression. In comparison the XCS controller begins with a greater number of lost chickens, which decreases in the course of time until becoming constant after approximately 25 000 ticks. This time corresponds to the time that is needed by the XCS to learn the best control strategy. At the end the XCS shows better results than the best fixed single rule controller, even if it shows the higher standard deviation (cf. Tab. 4), which is interpreted as a sign of instability of the learning process.

Table 4: Results of the XCS vs. the best single fixed rules controller \((t_y, t_y, t_h, d, i) = (0.3, 0.2, 0.3, 6, 20)\) with varying the simulation time

<table>
<thead>
<tr>
<th>simulated ticks</th>
<th>#(kC)</th>
<th>standard deviation</th>
<th>#(kC)</th>
<th>standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>625</td>
<td>12</td>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>1250</td>
<td>27</td>
<td>0</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>2500</td>
<td>50.15</td>
<td>0.67</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>5000</td>
<td>102.45</td>
<td>2.46</td>
<td>86.8</td>
<td>1</td>
</tr>
<tr>
<td>10000</td>
<td>178.3</td>
<td>3.97</td>
<td>172.9</td>
<td>5.51</td>
</tr>
<tr>
<td>50000</td>
<td>784.85</td>
<td>32.01</td>
<td>881.4</td>
<td>12.77</td>
</tr>
<tr>
<td>100000</td>
<td>1584.45</td>
<td>92.55</td>
<td>1781.4</td>
<td>21.48</td>
</tr>
<tr>
<td>200000</td>
<td>3092.5</td>
<td>144.86</td>
<td>3579.7</td>
<td>29.45</td>
</tr>
<tr>
<td>400000</td>
<td>6213.8</td>
<td>222.33</td>
<td>7165.75</td>
<td>58.07</td>
</tr>
<tr>
<td>800000</td>
<td>12509.1</td>
<td>635.35</td>
<td>14333.6</td>
<td>60.32</td>
</tr>
</tbody>
</table>

6.4 Learning Summary

In OC scenarios we have no time for time consuming parameter studies. We need on-line learning techniques that adapt in real-time to changing environments. But observing and controlling technical systems means that we cannot admit to learn through bad experience. Learning techniques that use off-line generated knowledge, e.g., by time consuming and low efficient parameter studies, and have the ability to adapt on-line seem to be an adequate approach, but in this direction more investigations have to be done: (i) Is it possible to optimise the results of the XCS implementation by varying the standard parameter combination given by Butz? Will the XCS converge to an optimum if the objective space grows from 15 to 135 actions? (ii) An organic system should adapt over time to a changing environment. The changes in the environment might lead to changes of the objective functions, and it is an open research question, if a classifier system can work and adapt to these changes in real-time. (iii) The best result of the fixed single rule controller investigation has been achieved with the assumption that the noise signal should be applied if the constraint \((e_N \geq 0.1)\) AND \((e_y \geq 0.1)\) OR \((e_h \geq 0.3)\) is true. Other combinations of emergence values are possible and need further investigations.
7. CONCLUSION AND OUTLOOK

OC has developed the idea of controlled self-organisation. In this paper we presented a generic observer/controller architecture and an entropy-based method to quantify emergence as a result of a self-organisation process in multi-agent systems. We discussed the learning aspect of the observer/controller architecture as a key aspect to allow for adaptation to environmental changes as well as to changes in the objective function of the system and developed the idea of a 2-level learning architecture of the controller. This architecture combines on-line learning with the possibility of knowledge generation and testing by an off-line planning instance.

We investigated the potential of applying concepts of OC to a multi-agent system of a group of self-organising agents, showing a macroscopic emergent behaviour that depends only on local rules. Our experimental results validate—at least partially—the idea of using the generic observer/controller architecture to modify the environmental parameters of the SuOC without modifying the local rules of the agents directly. We use a classifier system to learn on-line on Level 1 the best control intervention and compare the on-line results with results done by off-line fixed rules parameter studies. On-line learning leads to significant improvements in the performance of the global group behaviour. To show the advances of Level 2 in combination with Level 1 we need further investigations. Thus, our future work will focus on: (i) Further evaluation of the proposed 2-level learning architecture of the controller, comparing it with other architectures. This will also include the investigation of the extension to collective learning aspects of distributed controllers in distributed observer/controller architectures. (ii) Investigation of different variants of distributed observer/controller architectures. So far, the concepts mentioned above have been realised in a central manner. A central observer/controller is installed on top of the SuOC and allows for controlled self-organisation. We will continue this investigation and specification of basic mechanisms and focus our research on different variants of distributed observer/controller architectures and on generic patterns for coordination, collaboration, and learning in highly distributed SuOC.

8. ACKNOWLEDGEMENT

We gratefully acknowledge the financial support by the German Research Foundation (DFG) within the priority programme 1183 Organic Computing. We are indebted to E. Çakar, J. Hähner, F. Rochner, C. Müller-Schloer, and S. Tomforde, all from Leibniz Universität Hannover, J. Branke, H. Prothmann, and H. Schmeck, all from Universität Karlsruhe (TH), for their valuable suggestions. Also, we thank M. Engelmann for his work on the chicken simulation.

9. REFERENCES


ABSTRACT

In the aftermath of a large-scale disaster, agents' decisions derive from self-interested (e.g., survival), common-good (e.g., victims' rescue) and teamwork (e.g., fire extinction) motivations. However, current decision-theoretic models are either purely individual or purely collective and find it difficult to deal with motivational attitudes; on the other hand, mental-state based models find it difficult to deal with uncertainty. We present a hybrid, CvI-JI, approach that combines: i) collective ‘versus’ individual (CvI) decisions, founded on the Markov decision process (MDP) quantitative evaluation of joint-actions, and ii) joint-intentions (JI) formulation of teamwork, founded on the belief-desire-intention (BDI) architecture of general mental-state based reasoning. The CvI-JI model is evaluated from an experimental, case study based, outlook that explores the CvI-JI performance's improvement during the process of learning a coordination policy in a partially observable stochastic domain.

1. INTRODUCTION

The agents that cooperate to mitigate the effects of a large-scale disaster, e.g., an earthquake or a terrorist incident, take decisions that follow two large behavioral classes: the individual (ground) activity and the collective (institutional) coordination of such activity. Additionally, agents are motivated to form teams and jointly commit to goals that supersede their individual capabilities. Despite such motivation, communication is usually insufficient to ensure that decision-making is supported by a single and coherent world perspective. In general, the search for a coordination policy that responds to a large-scale disaster is a process beyond individual skills where optimality is non-existent or too expensive to compute [10].

The communication constraint causes the decision-making process to evolve simultaneously, both at collective (common-good) and individual (self-interested) strata, sometimes in a conflicting manner. For instance, an ambulance searches for a policy to rescue a perceived civilian, while the ambulance command center, when faced with a global view of multiple injured civilians, searches for a policy to decide which ambulance should rescue which civilian.

However, despite the intuition on a 2-strata decision process, research on multi-agent coordination often proposes a single model that amalgamates those strata and searches for optimality within that model.

The approaches based on the multi-agent Markov decision process (MMDP) [1] are purely collective and centralized, thus too complex to coordinate while requiring unconstrained communication. The multi-agent semi-Markov decision process (MSMDP) [8], although decentralized, requires each individual agent to represent the whole decision space (states and actions) which may become very large, thus causing the individual policy learning to be slow and highly dependent on up-to-date information about the decisions of all other agents. The game-theoretic approaches usually require each agent to compute the utility of all combinations of actions executed by all other agents (payoff matrix), which is then used to search for Nash equilibria (where no agent increases his payoff by unilaterally changing his policy); thus, when several equilibrium exist, agents may adhere to purely individual policies never being pulled by a collective perspective.

The multi-agent collective ‘versus’ individual (CvI) decision model [17], which is founded on the semi-Markov decision process (SMDP) framework, is neither purely collective nor purely individual and explores the explicit separation of concerns between both (collective and individual) decision strata while aiming to conciliate their reciprocal influence. Despite that, the CvI misses the agents' intentional stance toward team activity. On the other hand, the joint-intentions (JI) formulation of teamwork [5], based on the belief-desire-intention (BDI) mental-state architecture [11, 18], captures the agents' intentional stance, but misses the MDP domain-independent support for sequential decision-making in stochastic environments.

Research on single-agent MDP-BDI hybrid approaches formulate the correspondence between the BDI plan and the MDP policy concepts [13] and empirically compares each model's performance [12]. Approaches to multi-agent MDP-BDI hybrid models often exploit BDI plans to improve MDP tractability, and use MDP to improve BDI plan selection [15].

In this paper we take a different perspective on the multi-agent MDP-BDI hybrid approach. Instead of exploring the plan-policy relationship, we focus on the relation between the BDI intention concept and the MDP temporally abstract action concept [14]. We envisage an intention as an action that executes for time variable periods and, when terminated, yields a reward to the agent. We extend this view to the joint-intentions concept and integrate the resulting formulation in the 2-strata multilevel hierarchical CvI decision model. Thus, the CvI-JI is a hybrid approach that...
combines the MDP temporally abstract action concept and the BDI mental-state architecture.

The motivation for the hybrid CvI-JI model is to utilize the JI as a heuristic constraint that reduces the space of admissible MDP joint-actions, thus enabling to escalate the problems’ dimension. The experimental results show the CvI-JI policy learning improvement in a partially observable environment.

The next section describes the CvI decision model and the section 3 outlines the JI concepts that are most relevant for our hybrid approach. Section 4 formulates the hybrid CvI-JI decision model, which is experimentally instantiated and evaluated in section 5. Finally, section 6 presents our conclusions and future research goals.

2. THE CvI DECISION MODEL

The premise of the CvI decision model is that the individual choice coexists with the collective choice and that coordinated behavior happens (is learned) from the prolonged relation (in time) of the choices exercised at both of those strata (individual and collective). Additionally, coordination is exercised on high level cooperation tasks, represented within an hierarchical task organization. The tasks’ hierarchy is founded on the framework of Options [14], which extends the MDP theory to include temporally abstract actions (variable time duration tasks, whose execution resorts to a subset of primitive actions).

2.1 The framework of Options

Formally, an MDP is a 4-tuple \( M = (S, A, \Psi, P, R) \) model of stochastic sequential decision processes, where \( S \) is a set of states, \( A \) is a set of actions, \( \Psi \subseteq S \times A \) is the set of admissible state-action pairs, \( R(s, a) \) is the expected reward when action \( a \) is executed at \( s \), and \( P(s' | s, a) \) is the probability of being at state \( s' \) after executing \( a \) at state \( s \).

Given an MDP, an option \( o = (I, \pi, \beta) \), consists of a set of states, \( I \subseteq S \) from which the option can be initiated, a policy, \( \pi \), for the choice of actions and a termination condition, \( \beta \), which, for each state, gives the probability that the option terminates when that state is reached. The computation of optimal value functions and optimal policies, \( \pi^* \), resorts to the relation between options and actions in a semi-Markov decision process (SMDP). The relation is that “any MDP with a fixed set of options is a SMDP” [14]. Thus, all the SMDP learning methods can be applied to the case where temporally extended options are used in an MDP.

The option is an element of a multilevel hierarchy in which the policy of each option chooses among other lower level options. Thus, at each time step, the agent’s decision is entirely among options, some of which persist for a single time step (primitive action or one-step option), and others are temporarily extended (multi-step option).

2.2 The CvI collective and individual strata

The individual stratum is simply a set of agents, \( \Upsilon \), each agent, \( j \in \Upsilon \), having its particular capabilities described as an hierarchy of options. The CvI model admits agent heterogeneity (diverse option hierarchies), as long as all hierarchies have the same number of levels (depth), i.e., a similar temporal abstraction is used to design all hierarchies.

The collective stratum consists of a single agent (e.g., an institutional agent) that represents the whole set of individual stratum agents. The collective stratum agent cannot act on its own; its actions must be materialized through the individual stratum agents. The purpose of the collective stratum is to coordinate the individual stratum. Formally, at the collective stratum, each action is defined as a collective option, \( o_{\beta} = (o_1^j, \pi_1^j, \beta_1^j) \), where \( \beta = (\beta_1^j, \ldots, \beta_T^j) \) represents the simultaneous execution of option \( o_1^j \equiv (I_1^j, \pi_1^j, \beta_1^j) \) by each agent \( j \in \Upsilon \). The set of agents, \( \Upsilon \), defines an option space, \( O \subseteq O_1^j \times \ldots \times O_T^j \), where \( O_i^j \) is the set of agent \( j \) options and each \( o_{\beta} \in O \) is a collective option. The \( O \) decomposes into \( O_j \) disjoint subsets, each containing only the collective options available at the, \( d \), hierarchical level, where \( 0 < d \leq D - 1 \) and level-0 is the hierarchy root, at which there are no options to choose from, and level-\( D \) is the hierarchy depth. A level \( d \) policy, \( \pi_d \), is implicitly defined by the SMDP \( M_d \) with state set \( S \) and action set \( O_d \). The \( M_d \) solution is the optimal way to choose the level \( d \) individual policies which, in the long run, gathers the highest collective reward.

The CvI structure. The figure 1 illustrates the CvI structure where the individual stratum (each agent) is a 3-level task hierarchy and thus the collective stratum (represented by two, \( \delta_1 \) and \( \delta_2 \), collective option instances) is a 2-level hierarchy; at each level, the set of diamond ended arcs, links the collective option to each of its individual policies.

![Figure 1: The structure of the CvI decision model and the links between strata (superscript \( j \) refers to agent \( j \); subscripts \( k \) and \( p-k \) refer to \( k \) hierarchical level and \( k \) tree path).](attachment:image.png)

The CvI approach is decentralized as it lets each agent decide whether to make a decision by itself or to ask the collective layer for a decision. A centralized approach would define the \( M_d \) meta-policies and decides which individual policy to follow (i.e., decision-making would occur exclusively in the collective stratum).

The CvI dynamics. The figure 2 illustrates the CvI dynamics; at each decision epoch, \( \text{agent}_j \) gets the partial perception, \( \omega_j \), and decide-who-decides (d-w-d), i.e., the \( \text{agent}_j \) either: i) chooses an option \( o^j \in O^j \), or ii) requests, the collective stratum, for a decision. The collective stratum always replies with an option, \( o^j \), decision.

The d-w-d process represents the importance that an agent credits to each stratum motivation, which is materialized as the ratio between, the maximum expected benefit in choosing a collective and an individual decision. The expected benefit is given, at each hierarchical level, by the value functions of the corresponding SMDP \( M_d \).

A threshold, \( \kappa \in [0,1] \), supports the focus-grade between collective and individual strata. Such regulatory mechanism enables the (human) designer to specify diverse social atti...
from getting lost in the confusion of lower level details. It is usually effective to achieve coordination skills as they escape one-step options. Also, the highest hierarchical level(s) are.

3. THE JOINT-INTENTIONS (JI) MODEL

Given a set of agents, $\Upsilon$, standing for the individual stratum, and an agent, $\nu$, that impersonates the collective stratum, the design of a CvI instance is a 3-step process:

i. For each $j \in \Upsilon$, specify $O^j$ — the set of options and its hierarchical organization.

ii. For each $j \in \Upsilon$, and from the agent $\nu$ perspective, identify the subset of cooperation tasks, $C^j \subseteq O^j$ — the most effective options to achieve coordination skills; the remaining options, $J^j = O^j - C^j$, represent purely individual tasks.

iii. For each $j \in \Upsilon$, assign $\kappa$ its regulatory value — where $\kappa = 0$ is a common-good motivated agent, $\kappa = 1$ is a self-interested attitude, and $\kappa \in [0,1]$ embraces the whole spectrum between those two extreme decision motivations.

A simple, domain-independent design defines $C^j$ (item ii above) as the set of multi-step options; hence $J^j$ as the one-step options. Also, the highest hierarchical level(s) are usually effective to achieve coordination skills as they escape from getting lost in the confusion of lower level details. The approach, at its current stage, requires a designer to specify domain-dependent collective and individual options (i.e., $C^j$ and $J^j$ sets).

3. THE JOINT-INTENTIONS (JI) MODEL

The precise semantics for the intention concept varies across the literature. An intention is often taken to represent an agent’s commitment to perform an action, where a commitment is specified as a goal that persists over time, and a goal (often named as desire) is a proposition that the agent wants to get satisfied [3], [4], [11], [18]. An intention can also be taken to represent a linear plan that an agent has adopted to reach a state that the agent is committed to bring about [7].

The framework of joint-intentions (JI) adopts the semantics of the “intention as a commitment to perform an action” and extends it to describe the concept of teamwork. A team is described as a set, of two or more agents, collectively committed to achieve a certain goal [5]. The teamwork agents (those acting within a team) are expected to perform future-directed joint-intentions to act, keep those joint-intentions over time, and then jointly act. Formally,

given a set of agents, $\Upsilon$, a team is described as a 2-tuple $T = \langle \alpha, g \rangle$, where the team members are represented by $\alpha \subseteq \Upsilon$, and the team goal is $g$. In a team all members, $\alpha$, are jointly committed to achieve the goal, $g$, while mutually believing that they are all acting towards that same goal. The teamwork terminates as soon as all members mutually believe that there exists at least one member that considers $g$ as finished (achieved, impossible to achieve or irrelevant).

4. THE HYBRID CvI-JI DECISION MODEL

Given the CvI (cf. section 2) decision-theoretic model we regard the JI approach as a way to reduce the collective space exponentially in the number of team members. For example, given $\Upsilon$ agents, all with the same cooperation tasks, $C$, there are at most $|C|^\kappa$ admissible options to choose; during $(\alpha, g)$ teamwork, that number reduces to $|C|^\kappa - |\alpha|$ and such reduction motivates the formulation of the hybrid CvI-JI decision model.

The formulation of the hybrid CvI-JI decision model addresses (in the next sections) two questions: i) how to specify, at design time, the JI using the CvI components, and ii) how to integrate, at execution time, the JI specification in the CvI decision process.

4.1 Specify JI using the CvI components

The teamwork goal. The JI describes teamwork in terms of goals which, in general, take multiple time periods until satisfaction. The CvI specifies decisions in terms of options which are temporally abstract actions. Therefore, (a team) goal corresponds to (a team) option.

Given a goal, $g$, described as a proposition, $\phi$, we formulate the corresponding option as $(I, \pi, \beta)$, where $I$ is the set of states such that $\phi$ is satisfied, $\beta(s) = 1$ if $s \in (S - I)$ or $\beta(s) = 0$ otherwise, and $\pi$ represents any policy to satisfy $\phi$ (i.e., to terminate the option).

The teamwork commitment. The JI only requires agents to “keep the joint-intentions commitment over time, and then jointly act”. It is up to the agent to decide when to terminate executing an ongoing task and effectively start acting to achieve the team goal. Thus, being jointly committed to a goal, $g$, does not imply immediate action toward that same goal, $g$. For example, two ambulances may jointly commit to the same disaster while one of them is executing an action (e.g., delivering an injured civilian); as soon as the ongoing task is terminated, the ambulance starts acting towards the team goal.

Therefore, our hybrid CvI-JI formulation assumes that, at each decision epoch, an agent may establish a JI while still acting to satisfy another intention (either individual or JI). Thus, at each instant, an agent may have an ongoing activity and also one (at most) established JI. Our approach enables teamwork decisions to be asynchronous; agents do not need to wait, for each others’ option termination, before committing to a JI.

Our hybrid CvI-JI option selection function distinguishes two teamwork states: i) the “ongoing task continue” when an agent decides to establish a JI (becomes a team member) even though the agent still executes some other task, and ii) the “team option startup” when a team member decides to start executing the team option.

Given a team member, $j$, a team option, $o$, and its initiation set, $I_j$, we define the ongoing states, $I_{\text{ongoing}, j} \subset I$, where $j$ is allowed to continue executing an ongoing task.
The possibility that an agent drops a previous commitment. The CvI is a stochastic model so we assume once an agent commits to a team goal he will fulfil that commitment probability) into a single “teamwork design component” (tdc):

\[
\text{tdc} \equiv \langle j, o, I_{\text{ongo}}, p_{\text{commit}, j} \rangle,
\]

which describes, for each agent, \( j \in \mathcal{Y} \), and team option, \( o \), the set of states, \( I_{\text{ongo}, j} \), where the agent may continue an ongoing task before starting executing \( o \), and the probability, \( p_{\text{commit}, j} \), of effectively committing to \( o \).

The design of the tdc structure assumes that:

- a team option is always represented in more than one agent,

- an agent specifies a tdc instance for each team option he may get committed, and

- each \( I_{\text{ongo}, j} \) set is specified taking only the agent \( j \) local view of the environment.

The hybrid CvI-JI model describes, through the tdc instances, the domain-dependent teamwork knowledge which contributes to reduce the collective option space. Thus, CvI integrates JI as a heuristic filter (at collective stratum) that reifies the (human) designer domain knowledge. The next section integrates the heuristic filter in the decision process.

4.2 Integrate JI in the CvI decision process

The integration of the JI in the CvI decision process is designed, at the collective stratum, by modifying the CvI option selection process, which chooses, at each decision epoch, a level \( d \) collective option, \( \hat{O}_d \) given perceived state, \( s \), and a set of agents, \( B \), that request for a collective stratum decision. The algorithm 1 shows the option selection function, \texttt{chooseOption}, and the inclusion of the two subroutines, \texttt{applyFilter-JI} (cf. line 3) and \texttt{updateFilter-JI} (cf. line 5), that implement the CvI-JI integration.

```
Algorithm 1 Choose option at the level \( d \) of the CvI collective stratum.

1 function \texttt{chooseOption}( s, \hat{O}_d, \pi_d, B )
2 \hat{O}_d \leftarrow \texttt{getAdmissibleOptions}( s, \hat{O}_d, B )
3 \hat{O}_d'' \leftarrow \texttt{applyFilter-JI}( s, \hat{O}_d, B )
4 \hat{O}_d'' \leftarrow \texttt{applyPolicy}( s, \hat{O}_d'', \pi_d )
5 \texttt{updateFilter-JI}( \hat{O}_d, B )
6 return \hat{O}_d''
7 end function
```

The \texttt{getAdmissibleOptions} function (cf. algorithm 1, line 2) is exactly the same as in CvI; it evaluates the initiation set, \( \mathcal{I}_o \), of each collective option, \( o \), and returns the set, \( \hat{O}_d'' \), of admissible options (given the perceived state, \( s \), and the set of agents, \( B \), that requested a level \( d \) collective stratum decision). The \texttt{applyPolicy} function (cf. algorithm 1, line 4) chooses the next collective option to execute; the policy, \( \pi_d \), is either predefined or follows some explore-and-exploit reinforcement learning method. We followed the learning approach and implemented a \( \epsilon \)-greedy policy, which picks: i) a random admissible collective option, \( o \in \hat{O}_d'' \), with probability \( \epsilon \), and ii) otherwise, picks the highest estimated action value collective option, at the current state, \( s \), across all combinations of \( \alpha \in \mathcal{Y} \) and \( \tilde{Q}(s, o \in \hat{O}_d'') \).

The algorithm 2, \texttt{applyFilter-JI} function, shows the integration of JI commitments throughout the manipulation of the tdc instances (cf. expression 1). The set of goals that call for teamwork effort are represented by the, globally manipulated, TDC set (cf. line 3) which is initially empty. The first part (cf. lines 2 to 10, algorithm 2) determines the set tdc instances, TDC', that are admissible at the currently perceived state, \( s \), and that simultaneously correspond to a subset of \( B \). The teamwork reconsideration concept (cf. section 4.1) is represented by the possibility of discarding a previously established and currently admissible JI (cf. algorithm 2, line 5). The second part (cf. lines 11 to 16, algorithm 2) restricts the collective options to those that are compatible (all \( o \in \hat{O}_d'' \) components match) with the team options of all tdc in TDC'; the remaining collective options are discarded.

```
Algorithm 2 Apply JI filter to reduce the set of admissible collective options.

1 function \texttt{applyFilter-JI}( s, \hat{O}_d', B )
2 TDC' \leftarrow \emptyset
3 for each tdc \in TDC do
4 \text{if } ( s \notin tdc.I_{\text{ongo}}, j ) \text{ and } tdc.j \in B \text{ then}
5 \text{if } \text{random} \leq \text{tdc.p}_{\text{commit}, j} \text{ then}
6 TDC' \leftarrow TDC' \cup \{ tdc \}
7 end if
8 TDC \leftarrow TDC - \{ tdc \}
9 end if
10 end for
11 \hat{O}_d'' \leftarrow \emptyset
12 for each \( o \in \hat{O}_d' \) do
13 \text{if } o \text{ is compatible with TDC' then}
14 \hat{O}_d'' \leftarrow \hat{O}_d'' \cup \{ o \}
15 end if
16 end for
17 return \hat{O}_d'' \text{ if } \hat{O}_d'' = \hat{O}_d' \text{ when TDC' = } \emptyset
18 end function
```

The algorithm 3, \texttt{updateFilter-JI} function, describes the strategy used, at each decision epoch, to select a team goal and to find the set of agents that are available to commit to that team goal (i.e., select a goal, \( g \), and find the set, \( \alpha = \mathcal{Y} \), of agents available to form a team \( T \equiv \langle \alpha, g \rangle \)).

The implemented strategy simply selects the first admissible team goal and assumes that each agent “is available to commit to a team goal as long as he is not already a team member”. The TDC set is updated (cf. algorithm 3) according to that strategy, for all agents, at each decision epoch.

5. EXPERIMENTS AND RESULTS

We implemented the CvI-JI decision model and tested it in a multi-agent taxi environment: “a maze like grid inhabited by taxis (agents), passengers and sites”.

44
The taxi problem was originally proposed by Dietterich as a single-agent problem to explore the properties of a hierarchical reinforcement learning method. The single-agent problem was stated as follows: “A passenger appears at an origin site and wants to get transported to a destination site; the taxi is able to pick up, transport and drop down the passenger” [6]. The original taxi problem was extended to the multiple taxi coordination problem that was described as follows: “the environment is inhabited by multiple taxis (the agents), multiple passengers and a set of sites; a site may be the origin of several passengers, each one with its own destination; each taxi is able to pick up and drop down several passengers; each taxi may simultaneously transport several passengers” [17].

We propose a teamwork taxi coordination problem that extends the previous multiple taxi coordination problem, in order to enforce the teamwork behavior, as follows: “there are some predefined sites where passengers only accept to be transported all together (as in a family); at those sites a taxi may not pick up more than one passenger”. Such sites are named teamwork sites as taxis must work as a team to transport all passengers at the same time. We defined 3 different CvI-JI configurations, where a configuration assigns all \( j \) the same \( p_{commit:\ j} \in \{0, \frac{1}{2}, 1\} \) value. Thus, we have:

- **never JI**, when \( p_{commit:\ j} = 0 \),
- **sometimes JI**, when \( p_{commit:\ j} = \frac{1}{2} \), and
- **always JI**, when \( p_{commit:\ j} = 1 \).

The setup used for all experiments is: 5 x 5 grid, 4 sites \( S_b = \{b_1, b_2, b_3, b_4\} \), 2 taxis \( S_t = \{t_1, t_2\} \), 3 passengers \( S_{psg} = \{psg_1, psg_2, psg_3\} \), and a single teamwork site \( b_{tw} \in S_b \). The primitive actions, available to each taxi, are pick, put, move(m), where \( m \in \{N, E, S, W\} \) are the cardinal directions, and the wait action (added to the original taxi problem) to support the agent’s synchronization (e.g. at teamwork sites).

The environment of the teamwork taxi coordination problem is individually partially observable as each taxi does not perceive the other taxis’ locations; also, the environment is collectively observable as the combination of all individual observations determines a sole world state.

The goal of the individual stratum is to learn how to execute tasks (e.g. how to navigate to a site and when to pick up a passenger). The goal of the collective stratum is to learn to coordinate those individual tasks as to minimize the resources (time) to satisfy the passengers’ needs.

The learning of the policy at the collective stratum occurs simultaneously with the learning of each agent’s policy at the individual stratum. The results of the experiments (cf. section 5.5) show the hybrid CvI-JI performance improvement of the collective stratum learning process, when compared with the pure CvI (i.e., **never JI**) approach.

### 5.1 JI Specification

The JI is specified as a set of predefined \( tdc \) instances. The \( tdc \) instance is defined, for each taxi (agent) \( t_j \in S_t \) as \( \langle t_j, b_{tw}, I_{ongo: t_j}, p_{commit: t_j} \rangle \). The \( I_{ongo: t_j} \) specifies the following ongoing state set: i) the \( t_j \), \( b_{tw} \) transports a passenger, or ii) there is a passenger to pick up at \( t_j \) current location. The \( p_{commit: t_j} \) is assigned the value 0, \( \frac{1}{2} \) or 1, respectively for the never JI, sometimes JI or always JI experiment configuration.

### 5.2 Individual stratum specification

The taxi observation, \( \omega = \{x, y, psg_1, psg_2, psg_3\} \), is its \((x, y)\)-position and passenger, \( psg = \{loc, dest, orig\} \), status where \( loc \in S_b \cup S_t \cup \{t_{1acc}, t_{2acc}\} \) \( t_{1acc} \) means that taxi \( j \) accomplished delivery, \( dest_i \in S_b \) and \( orig_i \in S_b \). Therefore, the state space, perceived by each taxi, is described by a total of \( 5 \times 5 \times (8 \times 4 \times 4)^3 = 52,428,800 \) states.

The options’ hierarchy design considers the navigation between sites as (multi-step) options that choose among move(m) (one-step) options. Thus, the taxi capability is specified as a 3-level hierarchy, cf. figure 3, where root is the level-zero option, navigate(b), pick, put and wait are the level-one options and move(m) are the level-two one-step options (defined for each navigate(b)); a total of 7 one-step actions and 5 multi-step options.

![Figure 3: The individual stratum taxi’s hierarchy of options (root and navigate(b) are multi-step options; the remaining are one-step options).](image)

The taxi is not equipped with any explicit specification of its goal; also, it does not hold any internal representation of the maze grid. The taxi \( j \) decision is based solely on the information available at each decision epoch: i) its perception, \( \omega_j \), and ii) the immediate reward provided by the last executed one-step action.

The immediate rewards provided to a taxi are: i) 20 for delivering a passenger, ii) -10 for illegal pick or put, iii) -12 for any illegal move action in a teamwork site, and iv) -1 for any other action, including moving into walls and picking more than one passenger in a teamwork site.

### 5.3 Collective stratum specification
The collective stratum agent perceives a 5-tuple structure \( s = (t_1, t_2, psg_1, psg_2, psg_3) \) that combines all (individual stratum) partial observations, where \( t_j \) is the \((x, y)\)-position of agent \( j \). Therefore, the collective stratum state space is describe by \((5 \times 5)^2 \times (8 \times 4 \times 4)^2 = 1,310,720,000\) states.

The collective stratum chooses mainly among multi-step options, so we specify:

- \( \mathcal{C} = \{ \text{navigate}(b) \} \) for all \( b \in S_b \) \( \cup \{ \text{wait} \} \cup \{ \text{indOp} \} \), and
- \( \mathcal{J} = \{ \text{pick}, \text{put} \} \).

where \text{indOp} is an implicit option representing \( \mathcal{J} \) at the collective stratum. The figure 4 shows the space of collective stratum options; a total of \( 6 \times 6 = 36 \) options.

\[
\begin{align*}
\{ \text{navigate}(b_1), \text{navigate}(b_1), \text{navigate}(b_1), \text{navigate}(b_1), \text{wait}^1, \text{indOp}^1 \} \\
\times \\
\{ \text{navigate}(b_1), \text{navigate}(b_1), \text{navigate}(b_1), \text{navigate}(b_1), \text{wait}^1, \text{indOp}^1 \}
\end{align*}
\]

Figure 4: The collective stratum option space, built from the \( \mathcal{C} \) set (\text{indOp} represents, at the collective stratum, the purely individual tasks; superscript \( j \) refers to taxi \( t_j \)).

The \text{indOp} option gives place to a ping-pong decision scenario between strata, whenever an agent chooses to “request for a collective stratum decision” and the collective stratum replies: “decide yourself but consider only your purely individual tasks”. The decision forwards back to the agent (via \text{indOp}) raising a second opportunity for the agent “to choose an option in \( \mathcal{J} \)”. The ping-pong effect, while giving the agent a second opportunity to decide, does not increase the communication between strata and reduces (to \( \mathcal{J} \)) the context for an individual decision.

Our approach to the reward is to consider that all agents equivalently contribute to the current world state. Thus, the collective reward is defined as the sum of rewards provided to each agent; our purpose is to maximize the long run collective reward.

### 5.4 The experiment’s dimension

Within this experimental toy world, an individual agent perceives 52,428,800 states, and the collective stratum contains 1,310,720,000 states. Each individual decision considers 6 options, while for the collective stratum there are 36 collective options.

The decision-making of the collective stratum resorts, at each state, to the expected future value of each admissible collective option, whereas such evaluation is only acquired (learned) after the evidence (reward) gathered via the materialization (execution) of each decision.

Hence, the experiments capture some of the complexity of the decision-making process that aims to achieve coordinated behavior in a disaster response environment.

### 5.5 The Cvi-JI experimental evaluation

The purpose of our experiments is to measure the influence of the JI integration in the Cvi model. The performance of the learning process is used as the evaluation criterion and it is measured as the cumulative reward, gathered at the collective stratum, during an whole experiment. Each experiment executes for 700 episodes. An episode always starts with 2 passengers in the teamwork site and the third passenger in another site. Each episode terminates as soon as all passengers reach their destination. Policy learning uses a temporal difference approach (SMDP Q-learning [2], [14]) with the \( \epsilon \)-greedy strategy previously described (cf. section 4.2). Each experiment starts with \( \epsilon = 1.0 \) and, after the first 100 episodes, \( \epsilon \) decays 0.004 every 50 episodes.

We ran 3 experiments, one for each Cvi-JI configuration. The figure 5 shows that the never JI configuration exhibits the worst performance; it is about 6.5% worse than always JI and about 12% worse than sometimes JI; such difference remains almost uniform throughout the whole experiment. The sometimes JI reveals an unexpected behavior while, around episode 300, it starts to outperforms always JI.

An insight on these results is that the JI teamwork specific heuristic knowledge is exploited by the collective stratum, without compromising the exploration (search for novelty) that is required by the learning process.

The unexpected result is that the capability of not fulfilling a previous teamwork commitment (cf. sometimes JI) enables to find improvements over the fully reliable commitment attitude (cf. always JI). The Cvi-JI enables an agent to commit to a joint-intention while still pursuing its current goal.

Our main purpose is to promote a continuous (non interrupted) flow of decision-making and task execution activities. Such asynchronous decision-making process opens a time space between the time instant the agent establishes a JI and the time instant the agent actually begins acting to achieve the JI.

The overall problem includes teamwork activities as well as individual tasks. The possibility to reconsider a commitment, just before actually start acting, explores alternatives to teamwork. The ability to drop a pre-established JI enables to explore the state space and find individual policies in state points where the heuristic approach (JI) would suggest a teamwork approach.

Our results (cf. figure 5) show that the exploration of the individual policies combined with the heuristic teamwork approach enables to improve the process of learning a coordination policy.

### 6. CONCLUSIONS AND FUTURE WORK

In this paper, we have identified a series of relations between the 2-strata decision-theoretic Cvi approach and the joint-intentions (JI) mental-state based reasoning. We have extended Cvi by exploring the algorithmic aspects of the Cvi-JI integration. Such integration represents our novel contribution to a multi-agent hybrid decision model within a reinforcement learning framework. The initial experimental results, of the Cvi-JI model, sustain the hypothesis that the JI heuristic reduction of the action space improves the process of learning a policy to coordinate multiple agents. An interesting conclusion is that, taking into account our preliminary results, the teamwork reconsideration concept suggests investigating the hypothesis that not fulfilling a commitment (at a specific state) is an opportunity to find an alternative path that, in the long run, is globally better than teamwork after the JI.

This work represents the ongoing steps in a line of research.
that aims to develop agents that participate in the decision-making process that occurs in the response to a large-scale disaster. Future work will apply the CvI-JI in a simulated disaster response environment [9] and will explore teamwork (re)formation strategies [16] at the collective stratum.

Acknowledgment. Research at LabMAG FCT R&D unit.

7. REFERENCES


Incorporating Learning in BDI Agents

Stéphane Airiau  
RMIT University  
Melbourne, Australia  
stephane@utulsa.edu

Lin Padgham  
RMIT University  
Melbourne, Australia  
lin.padgham@rmit.edu.au  

Sebastian Sardina  
RMIT University  
Melbourne, Australia  
sebastian.sardina@rmit.edu.au

Sandip Sen  
University of Tulsa  
USA  
sandip@utulsa.edu

ABSTRACT
Belief, Desire, and Intentions (BDI) agents are well suited for complex applications with (soft) real-time reasoning and control requirements. BDI agents are adaptive in the sense that they can quickly reason and react to asynchronous events and act accordingly. However, BDI agents lack learning capabilities to modify their behavior when failures occur frequently. We discuss the use of past experience to improve the BDI agent’s behavior. More precisely, we use past experience to improve the context conditions of the plans contained in the plan library, initially set by a BDI programmer. First, we consider a deterministic and fully observable environment and we discuss how to modify the BDI agent to prevent re-occurrence of failures, which is not a trivial task. Then, we discuss how we can use decision trees to improve the agent’s behavior in a non-deterministic environment.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Intelligent agents; I.2.6 [Artificial Intelligence]: Learning

General Terms
Algorithms, Theory, Experimentation

Keywords
BDI, learning

1. INTRODUCTION
It is widely believed that learning is a key aspect of intelligence as it enables adaptation to complex and changing environments. Agents developed using the Belief-Desire-Intention (BDI) framework are capable of simple adaptations to their design-time behavior. First, such agents use hierarchical plans, where plan choices at each level are made in response to the current situation. Second, if a plan fails, often because the environment has changed since the plan was selected, agents backtrack and choose a different strategy for a particular level. However, BDI agents are unable to learn new behaviors from their experiences, and hence cannot significantly alter their behavior from the one specified during their initial deployment. In this work, we analyze BDI-based agent designs and identify opportunities and mechanisms for introducing behavior learning into BDI agents, so as to allow them to better adapt to their current environment on the basis of analysis of experiences.

Research in machine learning can be broadly categorized into knowledge-rich and knowledge-lean techniques. Whereas some researchers have proposed and investigated learning mechanisms that incorporate and utilize significant amount of domain knowledge [8, 10, 18], the large majority of popular learning techniques assume very little domain knowledge and are largely data, rather than model, driven [1, 3, 17, 19, 24, 28]. Research in multiagent learning [2, 22, 32] has also followed this trend. This is particularly unfortunate as practical multiagent systems should be designed to leverage existing domain knowledge in order to facilitate scalability, flexibility, and robustness. For most such online, real-time multiagent systems, individual agents need to quickly and effectively respond to unforeseen events as well as to gradual changes in working and environmental conditions. The amount of experience and adaptation time available will be orders of magnitude less than what is assumed in offline, knowledge-lean learning algorithms. Mechanisms that leverage domain knowledge that aids and guides the learning and adaptation process will be key to the development of successful agent learning approaches.

In the context of BDI agents, we can foresee significant synergistic possibilities for combining learning and reasoning mechanisms. While on the one hand encoded domain knowledge of BDI agents can inform and direct embedded learning modules, the latter can incrementally adapt and update components of the reasoning module to tune agent’s behaviors. Another possibility is for the learning module to use experience to refine coarse, approximate decision heuristics provided by the agent designer to produce improved behavior over time.

BDI agents are very effective practical reasoning agents, able to quickly reason and react to asynchronous events [12, 27, 26]. Roughly speaking, a BDI agent senses events from the environment it is situated in, and through deliberation and means-end reasoning, selects appropriate courses of actions to be carried on. BDI agents are well suited for complex applications with (soft) real-time reasoning and control requirements [13, 15, 16, 20]. If the environment is changing over time, however, and design-time knowledge is no longer appropriately tailored to the situation, then performance can
A critical issue for learning agents is how to adequately solve the exploration versus exploitation dilemma. Without appropriate prior knowledge, exploration is often blind, e.g., exploring agents choose actions from a uniform prior. Such uninformed exploration often compromises performance and can be arbitrarily costly. Available domain knowledge may help limit such losses by constraining action choices and focusing exploration on potentially useful regions of the action space. The information about useful areas for exploration could be encoded in the BDI plan library.

In the following section, we first provide an overview of the relevant aspects of typical BDI-style agents. We then discuss modifications to the BDI framework so as to include mechanisms for agents to improve their knowledge of which plans may be used in different situations. We do so by relying on a number of simplifying assumptions which make the scenario an "ideal" one. In Section 4, then, we outline ways in which these assumptions may be lifted.

2. BDI AGENTS

The BDI (Belief-Desire-Intention) model is based originally on the philosophical work of Dennet [9] and Bratman [5]. There is a large body of work in computer science that explores logics for such systems (e.g. [7, 25, 26]) as well as a substantial number of implemented systems (e.g. PRS [11], JAM [15], JACK [6], 3APL [14], Jason[4]). Basically BDI agents have a set of beliefs that represent the agent’s knowledge about the environment and about its own internal state; a set of desires or more commonly goals (non conflicting desires which the agent has decided to work towards achieving); and intentions regarding how the agent has decided to achieve its goals. In systems the intentions are generally a set of partially instantiated plans whose templates are part of the agent’s plan library.

An important aspect of BDI systems is that they do not plan from first principles. They have a library of hierarchical pre-defined plans or recipes which are selected based on suitability, and expanded as needed. They continually receive events from the environment, which can allow them to update their beliefs. Plans are then expanded based on current beliefs, making them extremely flexible and responsive to the environment. As a result they are well suited for complex applications with (soft) real-time reasoning and control requirements.

The architecture of a PRS agent is provided in Figure 1.

There are various versions of the basic BDI execution algorithm (e.g. [27, 34, 26]) a simple version of which is shown below:

While event queue and intention structure not nil do:

1. Select event (if any) this may be an external event from the environment, or an event generated by the agent itself.
2. Modify beliefs, goals, intentions. New information may cause the agent to update its beliefs and modify its goals and intentions.
3. Determine a set of “applicable plans” to respond to the new event. Applicable plans are those tagged as relevant to the event type, whose context condition match the current beliefs, and which have not yet been tried for response to this event.
4. Select one applicable plan and add it to the intention structure. This may be a new intention or it may expand the details of a current intention.
5. Execute the next step of a selected intention. This may execute an action or generate a new event.

The plan library contains plans of the form $e : \psi \leftarrow P$ where $P$ is the body of the plan, $e$ is an event that triggers the plan, $\psi$ is the context for which the plan can be applied (which corresponds to the preconditions of the plan). A plan body typically contains both actions, (which change the state of the environment), and subgoals which are events which result in expanding the details of the plan, by selecting a new plan for that event. By grouping together plans which respond to the same event type, the plan library can be seen as a set of goal-plan tree templates where goal (or event) nodes have children that are the alternative plans for achieving the goal, and plan nodes have children nodes that are the subgoals of the plan (or actions). These structures (see e.g. figure 2) can be seen as “AND”/“OR” trees: for a plan to succeed all the subgoals and actions of the plan must be successful (“AND”); for a subgoal to succeed one of the plans to achieve it must succeed (“OR”).

When a plan step (an action or subgoal) fails for some reason, this causes the plan to fail, and an alternative applicable plan for its parent goal is tried. If there is no alternative applicable plan, the parent goal fails, cascading the failure and search for alternative plans one level up the goal-plan tree. The search for alternative applicable plans on failure enables these systems to robustly recover from many problems, particularly problems where something has changed in the environment, motivating a different selection of plan.

The structured information contained in the goal plan tree can also provide guidance to the learning module. In particular, consider the context condition of plans, which are critical for guiding the execution of the agent program. A plan will not be used in the current

![Figure 1: Architecture of the Procedural Reasoning System (PRS).](image-url)
state if its context condition is not satisfied. Incorrect or inadequate context conditions can lead to two types of problems. If the context condition of a plan is over-constrained and evaluates to false in some situations where the plan could succeed then this plan will simply never be tried in those situations, resulting in possible utility loss to the agent. On the other hand, if the context condition is under-specified, it may evaluate to true in some situations where the plan will be ineffective. Such “false triggers” will result in unnecessary failures, and although the agent may recover by choosing alternative plans, it may lose valuable time or waste resources unnecessarily thereby losing utility. Hence, it would be preferable to learn from experience to avoid using plans that are unlikely to succeed at particular environmental states.

The rest of this paper explores the issue of learning to improve the context conditions specified by the programmer.

3. PLAN SELECTION REFINEMENT

In order to set the scene for our discussion of approaches to learning improvements in the context condition, we explore in what ways plan selection could be refined based on experience, in an idealised and highly constrained setting. This setting allows us to describe and understand the ideal situation, noting that non-trivial reasoning is involved if we are to eventually exclude all unnecessary failures. The most straightforward way to refine plan selection is by gradually modifying context conditions of plans to make them more and more specific. However we also explore a more subtle refinement that could be done by a smart reasoner, regarding selection of plans sequences in specific situations. In section 4 we will relax the idealised constrained setting and show how we can apply learning techniques to refinement of context conditions.

Firstly we assume the the environment of the agent is deterministic. That is, if an action fails in a particular world state once, it will always fail when tried in this state. This allows us conceptually to have the possibility of making updates to context conditions only when we know that update is correct. Secondly we assume that the context condition as initially specified is definitely a necessary condition (i.e., a minimum condition for the plan to succeed). We will now discuss the situations in which we can with certainty update the context condition, or refine some more complex selection function, with the aim of eventually improving the selection to avoid any failure based on poor plan choice. We note that failure can also occur when the environment changes between the time of plan selection, and the time of executing some aspect of that plan. It is exactly this kind of failure which BDI systems are good at avoiding (by doing plan decomposition only when needed), and recovering from (by trying an alternative plan if one fails). The latter mechanism also assists with managing the type of failure we are interested in learning to avoid.

In the following discussion we assume that the beliefs and the context conditions of plans are described using propositional logic. We also assume that all variables \( v \in V \) that affect the functioning of the system are observable and known. In addition we assume that all changes to any variable \( v \) are explicit: that is we know if a particular action may (under some circumstances) change \( v \). This is necessary for us to reason about changes caused by the agent, as opposed to those happening independently in the environment. Additionally we assume that the subset \( R_0 \subseteq V \) that is relevant for the success or failure of a particular plan \( p \) is defined. A likely candidate for \( R_0 \) can be identified as the set of variables that are accessed by some plan below the parent goal of \( p \) in the goal-plan tree. Although only a subset of \( R_0 \) may be used in the context condition of \( p \), other variables in this set may play a role in the success or failure of \( p \). Within plan bodies we allow only sequences of actions and subgoals, and also we impose the constraint that there is no interleaving of plans which are achieving concurrent goals. This removes the need to monitor for interactions between goals that could have an effect on success.

Let \( r_p(t) \) be the tuple containing the values of \( R_0 \) at time \( t \). If we have a plan, \( p \), with a single action \( a \) in its plan body, and this plan fails at time \( t \), then definitely we should be able to update the context condition of \( p \) to exclude the partial state represented in \( r_p(t) \). However, if \( p \) is a more complex plan than may be things (under the agent’s control) between \( t_1 \) when \( p \) is selected, and \( t_f \) when \( p \) fails, that could have prevented that failure. In such a case modification of the context condition is not justified. The challenge is to know when it is impossible that failure could have been prevented (given the state \( r_p(t) \) when \( p \) was selected), and therefore to know that we should modify the context condition of \( p \) to exclude \( r_p(t) \).

Let us consider a plan \( p \) that is a sequence of actions, \( a_1...a_n \) and \( p \) has failed at step \( a_i \). In this case, if none of the actions \( a_1...a_{i-1} \) affect variables in \( R_0 \), we can be assured that the failure of \( a_i \) (and therefore \( p \)) was inevitable given \( r_p(t_f) \) when \( p \) was selected. In this case modification of the context condition of \( p \to exclude r_p(t) \) is justified (as long as there have been no environmental changes to variables in \( R_0 \), which can be checked by whether \( r_p(t_f) = r_p(t_f) \)). In fact, as we have assumed that actions are fully deterministic, it does not really matter if \( a_1...a_{i-1} \) affects variables in \( R_0 \) as they will always lead to the same state, which eventually caused failure. However, if we generalise actions to be sub-goals, then choices are potentially available, and we must consider this in our assessment of whether the failure was inevitable.

Consider the example situation shown by the plan-goal tree in figure 3. Assume that plans \( p_{11}, p_{12}, p_{21}, p_{22} \) and \( p_{31} \) each consist of a sequence of one or more atomic actions (and are therefore deterministic). If \( p_{11} \) fails its context condition will be updated as discussed above. However, its failure will cause the failure of SG1, and therefore of \( p_{0} \). The question is when can we update the context condition of \( p_{0} \), to exclude \( r_p(t) \), the state when it was chosen and eventually led to failure. We cannot straightforwardly do this when \( p_{0} \) fails, as possibly the choice of \( p_{12} \) to achieve SG1 would have led to success. However if we have recorded that previously in a state equal to \( r_p(t) \) we started executing \( p_{0} \), subsequently chose \( p_{12} \) to accomplish SG1, and this also failed, then we would be justified in updating the context condition of \( p_{0} \), to disallow \( r_p(t) \).
This is because all possible execution paths from $p_0$, chosen in state $r_{p_0}(t_2)$, have led to failure.

We now modify the situation slightly and assume the failure occurs initially at $p_{31}$, thus propagating to $p_0$. In this case we can only know that the context condition of $p_0$ should be modified if all combinations of choices at $SG1$ and $SG2$ have been tried following selection of $p_0$ in state $r_{p_0}(t_2)$. However, there is some additional selection information that we could potentially use. If in the situation where we end up failing at $p_{31}$, we have used $p_{11}$ to achieve $SG1$ and $p_{21}$ to achieve $SG2$, then we know that this is a bad combination for the case where the state is $r_{p_0}(t_2)$ when we select $p_0$. In future cases where we select $p_0$ in this state, we should avoid this combination of choices for achieving $SG1$ and $SG2$. However this is not information that can be represented as part of the context condition of a particular plan. Rather it is a complex metalevel selection function for choosing (or excluding) a path through the goal plan tree of a particular plan. It is not clear where or how such information should be represented in the BDI hierarchy, although it is clear that in an idealised world such information is potentially available.

One approach to determining whether a context condition should be updated following failure in a child node is to use planning. Once we have detected a failure we would like to propagate modifications as far up the Goal-Plan tree as is justified. Each time we have modified the context condition of a plan, we can check whether any plans one level up in the Goal-Plan tree should also be modified, using Hierarchical Task Network (HTN) plan decomposition, based on the BDI Goal-Plan tree as described in [29]. In our example, if after modifying the context condition of $p_{31}$ we attempt plan decomposition on $p_0$ with initial state equal to $r_{p_0}(t_2)$, and fail to find a satisfactory decomposition, then we should modify the context condition of $p_0$ to exclude $r_{p_0}(t_2)$, as the plan decomposition has shown that there is no valid decomposition from this start state.

As we can see from the above discussion, in order to reliably update our knowledge about plan selection, we need to maintain information about all paths tried, from every relevant belief state combination, for all plans. Although some smart pruning may be possible based on knowledge of which paths can affect which variables, there is still a large amount of information that must be kept. Presumably compact representations such as those used by model checkers [1] could also assist with practical feasibility.

However we have also made assumptions about a deterministic world and complete knowledge that are infeasible in practice. Therefore, based on the understandings gained from the discussion above we explore the use of learning to update context conditions, when, based on experience, it appears to be warranted.

4. LEARNING REFINEMENT OF CONTEXT CONDITIONS

The context in which we have discussed the refinement of plans’ context conditions and the agent’s plan selection function in the previous section can be seen as “ideal”: (i) the environment is deterministic and fully observable; (ii) the (initial) context conditions are always necessary ones; and (iii) the success and failure of plans are always observable. Once we relax the condition of determinism and allow for non-deterministic actions (or lack of a fully observable environment, which is equivalent), we can no longer update any context condition based on a single failure as was the basic case for the previous section. Nevertheless, we should, over time, be able to learn that certain states do tend to lead to failure, and to refine our plan selection accordingly.

Decision Trees [21] are a natural choice of learning mechanism for this situation as they can deal with noise (created by the non-determinism in our case) and they support disjunctive hypotheses which we require. Also, a decision tree is readily convertible to rules, which are in fact the context condition.

Others have also used decision trees as a tool in learning context condition details. In [23], inductive learning was used to build a decision tree to decide on which plan to use. The instances observed between the update were used to estimate the accuracy of the tree, and when the tree was successful enough, it was used as context condition. The problem faced by the learning agent is similar to ours when all initial context conditions are set to true. Their approach is tailored to a particular example, and it is not clear how to generalize it. Our goal is to provide learning capabilities to a generic BDI agent.

Typically, a decision tree is built off-line from a set of data. In our case though we wish to act as best we can, but accumulate experience to make improved decisions as we get more information. Some algorithms are designed to induce decision trees incrementally [30, 33], and we are planning to use these algorithms for our task.

We will associate a decision tree with each plan in our goal-plan tree. As we build up and record our experience, the decision tree will identify likely failure conditions, allowing us to modify our context condition in a similar way to our simplest example in the previous section. We will illustrate with an example the basic principles of our approach.

4.1 Example

Let assume that we have the goal-plan tree in Figure 5: the agent wants to satisfy the goal $G$ and can choose between two plans: $p_1$ and $p_2$. Let us assume that there are three relevant variables in $\mathcal{R}_G$ and $\mathcal{R}_P = \{a, b, c\}$. First, let us assume that the context condition of both $p_1$ and $p_2$ is set to true, i.e., the programmer does not put any constraints on the applicability of either plan. As data is collected during the execution of the agent, a decision tree as in Figure 4 can be built for each plan. Once patterns seem to be clear (i.e., when not changes in the decision tree occur after seeing new instances)
We can extract failures in the previous section. Taking the information from the plan, for example, let us now assume that the context conditions for the plan. For example, considering Figure 5 we may consider that the design of the BDI programmer is such that a single plan was applicable given a state of the environment. It is possible, however, that multiple plans are applicable, in which case there will be an overlap in the context conditions of the rules.

In a first phase, the agent uses its BDI decision mechanism, and in addition, it records the statistics in the table, and uses the instances to build the tree incrementally. The algorithm in [30] does not require to store all instances to grow the tree. Once ’enough’ information is collected, the agent can start to use the decision tree in conjunction with the context condition encoded by the programmer. Each time the original context condition of a plan is satisfied, the decision tree will occasionally cause a failure in a plan that generally works and is a good choice (for a given \( \psi(t_i) \)). However, another reason for plan failure (other than poor choice) is that there is a change to some variables in \( R_p \) due to some environment factors outside the control of the agent. Such causes of failure should not lead us to conclude that \( P \) is a bad choice in \( r_p(t_i) \). Consequently, we do not record either the success or failure of any plan executions where a variable in \( R_p \) is externally influenced after the selection of \( p \).

**4.2 Adaptive Context Conditions**

So, let us first consider the case of non-deterministic fully observable environments, or what is equivalent, (deterministic) environments which are partially observable. Let us also consider that the context condition given by the BDI programmer are necessary. Under such settings, one can no longer rely on the fact that a plan would have the same success-failure outcome under exact belief states—the environment may behave differently or there are some hidden variables in the domain. Nonetheless, by monitoring the execution of such plan over time, we may be able to learn some "patterns" on its success and failure and refine its context accordingly.

As we assume that the context condition \( \psi \) set by the BDI programmer are necessary, the agent will use \( \psi \) and the decision tree to decide whether a plan is applicable. In addition, only a part of the table recording the number of successes or failures of a plan will ever be filled. In our example, only the cells in white can be filled for \( p_1 \) and only the cells in grey can be filled for \( p_2 \). Consequently, the table for each plan can be reduced to include only states not already excluded by the context condition. Note that in our example, the context conditions of the two plans do not overlap, i.e., the design of the BDI programmer is such that a single plan was applicable given a state of the environment. It is possible, however, that multiple plans are applicable, in which case there will be an overlap in the context conditions of the rules.

In a first phase, the algorithm uses its BDI decision mechanism, and in addition, it records the statistics in the table, and uses the instances to build the tree incrementally. The algorithm in [30] does not require to store all instances to grow the tree. Once 'enough' information is collected, the agent can start to use the decision tree in conjunction with the context condition encoded by the programmer. Each time the original context condition of a plan is satisfied, the decision tree is used to confirm whether the plan is indeed applicable or not. We now discuss a few scenarios on our example.

Let assume that each time \( c \) is true, the plan \( p_2 \) fails. After collecting enough information, the agent uses the decision tree to determine whether plan \( p_2 \) is applicable. In our example, only the cells in white can be filled for \( p_1 \) and only the cells in grey can be filled for \( p_2 \). Consequently, the table for each plan can be reduced to include only states not already excluded by the context condition. Note that in our example, the context conditions of the two plans do not overlap, i.e., the design of the BDI programmer is such that a single plan was applicable given a state of the environment. It is possible, however, that multiple plans are applicable, in which case there will be an overlap in the context conditions of the rules.

For example, let us now assume that the context conditions for the plans \( p_1 \) and \( p_2 \) are the following: \( a \land b \) for \( p_1 \) and \( \lnot a \land \lnot b \) for \( p_2 \). If we record our experiences in a table similar to Table 1, only part of Table 1 will be populated as the context condition prevents selection of some plans.

To build this table the agent must recall the state at which the plan was chosen, and then record the relevant success or failure in the table. There will inevitably be some amount of noise in the data due to the non-determinism (or hidden variables) that we have introduced. This will occasionally cause a failure in a plan that generally works and is a good choice (for a given \( r_p(t_i) \)). However, another reason for plan failure (other than poor choice) is that there is a change to some variables in \( R_p \) due to some environment factors outside the control of the agent. Such causes of failure should not lead us to conclude that \( P \) is a bad choice in \( r_p(t_i) \). Consequently, we do not record either the success or failure of any plan executions where a variable in \( R_p \) is externally influenced after the selection of \( p \).

**Table 1: Table summarizing the failure and success of a plan \( p \).**

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>number of success</th>
<th>number of failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For now, we assume that the number of instances observed is above a threshold. This threshold depends on the size of the table that can be filled given the context condition.
ing ‘enough’ evidence (we will discuss this issue later), the decision tree will consist of a single decision node with $c$ and two leaf nodes to tell to use the plan when $c$ is false, and not use it otherwise. The use of the decision tree will prevent the agent from choosing $p_2$ when $c$ is true, and hence, this action should improve the behavior of the agent. As a side effect, if the plan $p_2$ was originally the only applicable plan when its context condition was satisfied, there is now no plan applicable for this state of the world. In our example, no plan is applicable when $\neg a \land \neg b \land c$.

Now, let us come back to the example of Figure 3, and let us assume that when the decision tree of plan $p_{31}$ is used, there is a state of the environment where no plan is applicable for $SG_3$, yielding the failure of plan $p_0$. In Section 3, we described the condition to correctly update the context condition of $p_0$ when the context condition of $p_{31}$ is changed, which requires careful examination. While learning the decision tree for $p_0$, the precise reason for the failure of $SG_3$ is not important: whether $SG_3$ failed because $p_{31}$ failed or because no plan was applicable for $SG_3$, the result is the same: the goal $SG_3$ was not satisfied, and hence, $p_0$ failed.

As the agent acts, a decision tree is learnt for $p_{31}$, but also for $p_0$. In particular, the latest decision tree will learn to avoid the contexts that lead to a context that cause $SG_3$ to fail. Actually, for the update of the decision tree of $p_0$, it does not matter which sub-goal failed, what matters is the success or failure of the plan. However, what does matter is whether the failure of the sub-goal could have been avoided. For example, let us assume that, at time $t_0$, $SG_3$ fails with the context condition set by the BDI programmer. Let us assume that at time $t_c$, ‘enough’ data has been collected and the use of decision trees for $p_{21}$ and $p_{22}$ prevents failures of $SG_2$. Between $t_0$ and $t_c$, the data collected for the plan $p_0$ will contain a set $E$ of negative instances due to failure of $SG_3$. The set $E$ should now be considered as false negative, i.e., erroneous instances: the same instance would now be classified as a success given the correct decision trees for $p_{21}$ and $p_{22}$.

A first solution to this problem is to start collecting data for the decision tree of $p_0$ only when the decision trees below are approximately correct. This is akin to the case where all paths had to be visited before making an update in Section 3. If we can use the analysis of Section 3 to propagate findings, it would speed up the learning process. For example, if the decision trees of the plans that satisfy $SG_1$ are accurate, as $SG_1$ is the first sub-goal, some knowledge could be propagated to the decision tree of $p_0$. This may require a large total number of instances, especially if the depth of the tree is large. A different solution is to learn the decision trees concurrently, but wait longer before using them: as more data is collected after $t_c$, the impact of the initial false negative instances in $E$ will decrease (as positive instances will be present) and the tree should be correct in the long run. This solution does not wait for bottom trees to be learnt before collecting data, so it may require a smaller total number of instances to perform well. However, there is a risk of the trees not to be correct when they are used too quickly.

### 4.3 Generalizing Context Conditions

We now consider the cases where it is not the case that the plan’s context conditions given by the BDI programmer are always necessary conditions. In such cases, the agent may want, sometimes, to try a plan even if its context does not hold. For example, this may be a reasonable behavior when no plan is applicable for resolving a goal—it may be worth exploring the feasibility of some plan that looks currently not applicable. A priori, a BDI programmer should avoid having the situation where no plan is applicable. However, if the agent uses the learning technique we proposed above to refine the context condition, the chances to have situations where no plan is applicable increases.

First, the table capturing the successes and failures of a plan cannot be restricted to the entries that match the context conditions: as the plan can be used even when the context condition is not satisfied, any cells can be updated. Now let us come back to the example of Figure 5, and let us assume that the context condition of $p_2$ is $\neg a \lor \neg b \lor c$ when we use the decision tree. When $\neg a \lor \neg b \lor c$ occurs, no plan is a priori applicable.

The experience of the agent contains evidences that $p_2$ is not applicable. The alternative plan to satisfies the goal $G$ would be to use the plan $p_1$, but this plan is a priori not applicable, not because the agent has observed failures of $p_1$, but because of the context condition set by the BDI programmer. Because of the particular environment, and because the programmer did not include the variable $c$ in the expression of context condition of $p_2$, it may be possible that the context of $p_1$ should be different and include the variable $c$.

Given the fact that no plan is applicable, we now have an incentive to try to use $p_1$ when $\neg a \lor \neg b \lor c$ occurs. If $p_1$ also fails in that context, then, from experience, no plan is applicable. This exploration incurred a cost from the BDI agent: when $\neg a \lor \neg b \lor c$ occurred, the plan $p_1$ failed as the agent did not follow the prior knowledge input by the BDI programmer. If, however, the plan $p_1$ succeeds, even though it was not supposed to, the agent will be able to satisfy $G$.

The exploration is triggered by the fact that, from its own experience and the context condition set by the BDI programmer, no plan is applicable to satisfy the goal $G$. In that case, the agent will consider all plans that satisfy goal $G$ and the statistics about success of the plan. If there exist some plans with no records of failure or success for that particular context, the agent will try one randomly. After having observed a ‘sufficient’ number of instances, a plan will be declared applicable or not. If the plan is declared applicable we must update the context condition by adding the conjunction, e.g., $\Psi \land \neg a \lor \neg b \lor c$ for our example. Note that in the case where multiple plans can be explored, we could wait for all plans to have been ‘sufficiently’ tried and only update the context condition of the plans with the highest rate of success. For example, we can run a t-test to determine whether plans have a significantly different success rate and pick the ones with the highest one.

Hence, when the BDI programmer is not certain that the context conditions are necessary, we propose to use decision trees to refine or generalize the context conditions. The generalization can occur when the agent finds a state of the world where no plan is applicable (because of the context condition set by the BDI programmer or as a consequence of the refinement of a context condition). In that case, the agent will explore alternative plans and will determine, from experience, whether these plans are applicable or not.

### 4.4 Discussion

The question that remains is when and how to use the decision tree to refine the decision regarding whether a plan is applicable or not. There are two issues to consider. The first is the number of instances that are required to make an informed decision. The second is about the proportion of success: if a leaf node of the decision tree
contains 60% of success, should the plan be considered as applicable?

First, the agent must have observed a sufficient number of instances before starting to use the decision tree. Depending on the number of relevant variables and on the size of the tree below the plan, we can estimate how many examples are needed to make a decision with some confidence. For example, if a plan \( p \) has some sub-goals, the decision tree of \( p \) is probably approximately correct if the decision tree of the plans satisfying the sub-goals are correct.

If the agent observes failures and successes for a particular state of the world, when should the agent consider that a plan becomes

- no longer applicable (when the context condition does not match) or
- applicable (when the context condition matches but we observe a high rate of failures)?

Without knowledge of a utility function for satisfying the goal \( G \), there is a priori no clear decision. If failure has a high cost, it may be preferable to do nothing. At this point, it is not possible to provide a generic argument. External feedback about the agent's performance could later be used to tailor these decisions. For now, we envision to use a threshold value of the success rate (for example, an 80% success rate is required for a plan to be considered applicable).

In the example of Figure 3, in Section 3, we discussed how dependencies between different branches could affect success/failure at a particular node. For example it is possible that in state \( r_{p0} \), \( p_{31} \) fails because \( p_{11} \) was chosen to satisfy \( SG_1 \). If \( p_{12} \) is used instead, \( p_{31} \) would succeed. We noted that while there is information that could potentially be reasoned about, to avoid future failure, this is part of a complex metal-level selection function related to the goal. These kind of dependencies between ways of achieving various sub-goals actually create problems for our decision tree learning beyond the lack of ability to learn the correct combination. They can cause the decision tree to learn an inappropriate applicability rule. For the decision tree of \( p_0 \), let us assume that, when \( r_{p0} \) occurs, \( p_{11} \) and \( p_{12} \) are chosen with equal probability. In that case, \( p_0 \) will obtain a 50% success rate. If \( p_{11} \) is chosen with a higher probability, \( p_0 \) could easily be considered not applicable in the context \( r_{p0} \). To avoid inappropriate assumptions in this kind of situation we need to at least be aware of dependencies between plans. We expect to be able to use a variation on the dependency summaries of [31] to address this problem.

5. DISCUSSION AND CONCLUSION

BDI agents are adaptive in the sense that they can quickly reason and react to asynchronous events and act accordingly. To implement a BDI agent, a programmer must provide a plan library and a condition telling when each plan is to be used. When the environment is complex, when the success and failures of plans depends on complex conditions, writing correct context condition for each plan may be a difficult task. As BDI agents lack capabilities to modify their behavior when failures occur frequently, we propose to analyze past experience to improve the BDI agent’s behavior. More precisely, we propose to use past experience to improve the context conditions of the plans contained in the plan library, initially set by a BDI programmer.

We discussed how to modify the BDI agent to prevent re-occurrence of failures in deterministic and fully observable environment when the BDI programmer set necessary conditions for the success of each plan. We showed that even with these simplifying assumptions, correctly refining the context condition is not a trivial task. Then, we relax the assumption of a deterministic environment and we discussed how to use decision trees to improve the agent’s behavior. Finally, we relaxed the assumption that the initial context conditions are necessary (in a specific environment, the applicability of a plan may be more general, the BDI programmer may have over constrained a the applicability of a plan). In that case, the agent may have to perform some guided exploration: when no plan is applicable, plans that are not applicable and that have not been tried under specific conditions can be executed.

The objective of the learning is for the agent to avoid re-occurrence of failures by using past experience to refine (and generalize) the context conditions of the plans. We are currently implementing an hybrid BDI-learning agent and a testbed scenario inspired from RoboRescue. We plan to first explore the case where the initial context conditions are necessary and decision trees are used to refine the context conditions. The issues we plan to consider include when can we start to collect data, when can we start to use the trees, and how many instances are needed to converge to a satisfying behavior.

6. REFERENCES


Criteria for Consciousness in Artificial Intelligent Agents

Raúl Arrabales
Universidad Carlos III de Madrid
Avda. Universidad, 30
28911 Leganés, Spain
+34 91 6249111
rarrabal@inf.uc3m.es

Agapito Ledezma
Universidad Carlos III de Madrid
Avda. Universidad, 30
28911 Leganés, Spain
+34 91 6249110
Ledeza@inf.uc3m.es

Araceli Sanchis
Universidad Carlos III de Madrid
Avda. Universidad, 30
28911 Leganés, Spain
+34 91 6249423
masm@inf.uc3m.es

ABSTRACT
Accurately testing for consciousness is still an unsolved problem when applied to humans and other mammals. The inherent subjective nature of conscious experience makes it virtually unreachable to classic empirical approaches. Therefore, alternative strategies based on behavior analysis and neurobiological studies are being developed in order to determine the level of consciousness of biological organisms. However, these methods cannot be directly applied to artificial systems. In this paper we propose both a taxonomy and some functional alternatives based on behavior analysis and embodied intelligence. Furthermore, a list of measurable levels of artificial consciousness, ConsScale, is defined as a tool to determine the potential level of consciousness of an agent. Both the mapping of consciousness to AI and the role of consciousness in cognition are controversial and unsolved questions, in this paper we aim to approach these issues with the notions of I-Consciousness and embodied intelligence.

Categories and Subject Descriptors
I.2 [Artificial Intelligence]: General – cognitive simulation, philosophical foundations.

General Terms
Design, Theory.

Keywords
Machine Consciousness, Artificial Consciousness, cognitive agents, Cognitive Modeling.

1. INTRODUCTION
Determining the level of consciousness of a living organism is a hard problem. One could think that some sort of Turing test might be a plausible solution [31]. It is indeed what we do everyday when we perceive other subjects as conscious beings. These kinds of test that we are used to perform unconsciously are based on verbal report and observed behavior. We perceive other humans acting as if they were conscious and thus we infer they actually are. However, we do not have any scientific proof that others experiment any subjective life because we cannot perceive it directly [15]. Therefore, from a pure scientific point of view, zombies (organisms behaving as conscious beings but without any inner feeling) are conceivable, although probably not possible [7]. Some approaches have been proposed in order to overcome the issue of scientific proof of consciousness. From a philosophical standpoint, Dennett has proposed the heterophenomenology method, which consists on the application of the scientific method to both third-person behavior analysis and first-person self report [10]. From the neuroscience perspective, Seth, Baars and Edelman propose a set of criteria for consciousness in humans and other mammals [29]. A number of these criteria are based on neurobiological aspects. If the neuronal structures and the associated activity pattern that gives place to consciousness are identified, then we can look for them in animals endowed with a central nervous system [5]. Analogously, if some behavior patterns are identified as uniquely produced by a conscious subject, we can design experiments where these behaviors are tested. However, when it comes to artificial agents, most of the assumptions mentioned above cannot be directly applied. The following are the main reasons why we think the former criteria should not be used for evaluating artificial agents:

- **Artificial agents have different underlying machinery.** At the biological level, behavior of mammals is controlled by the endocrine and nervous systems. Even though some artificial agents are inspired or try to simulate the biological nervous system, their design is quite far from a realistic emulation. Therefore, it does not make sense, for instance, to look for a strong connection between thalamus and cortex as a possible sign of an underlying mechanism for consciousness in an artificial implementation (given the case that the implementation under study is endowed with a simulated thalamocortical structure).

- **Artificial agent’s behavior produces different patterns.** Moving the observer’s point of view from the biological level to the behavioral level, human behavior can be seen as regulated by cultural rules. Human ontogeny gives place to different behavioral patterns as a subject develops situated in a cultural environment. Given that the development of artificial agents differs from that, their behavior should not be analyzed following the same criteria that are applied to humans.

- **Lack of verbal report.** This is one of the key differences between human’s behavior and artificial agents’ behavior. Accurate verbal report (AVR) is probably the main way we can find out about the inner life experienced by a human subject. Given the lack of this kind of communication skills in artificial systems, AVR as we know it cannot be used to evaluate artificial agents.
Taking into account the reasons mentioned above and the fact that human culture strongly determine the production of consciousness in humans [6], we argue that the kind of consciousness that could be potentially produced in artificial agents would be of a different nature (although we think it still could be called consciousness, i.e. machine consciousness or artificial consciousness). Consequently, we believe that criteria for machine consciousness should be studied from the perspective of a specifically defined taxonomy of artificial agents. Even though some of the classes of artificial agents defined in this taxonomy cannot be directly compared with a corresponding example of biological organisms, both biological phylogenetic and human ontogenetic analogies can often be used to better understand the level of consciousness that can be associated to a particular class of agents, e.g. [16].

In the next section we aim to provide a comprehensive description of the main aspects of consciousness and their basic roles in cognition. Additionally, we redefine the dimensions of consciousness in terms of artificial intelligent agents, and therefore we characterize machine consciousness by analyzing the fundamental building blocks required in an agent’s architecture in order to produce the functionality associated with consciousness. Subsequently, in section 3, we discuss key particular functions of consciousness and their interaction in agent’s cognitive processes. In section 4, we have taken into account both the key functions of consciousness and agent’s basic architectural features to propose a taxonomy for artificial agents, where a concrete level of machine consciousness is assigned to each agent category. Section 5 provides a framework for classifying agents under the light of the proposed taxonomy. Finally, we conclude in section 6 with a brief discussion of current state of the art in terms of our proposed taxonomy.

2. CHARACTERIZING MACHINE CONSCIOUSNESS

Setting aside the discussion about whether or not a categorical implementation of an artificial form of consciousness is possible, we have adopted an incremental approach in which we consider that certain aspects of consciousness can be successfully modeled in artificial agents; while other aspects might be still out of the reach given the current state of the art in the field of machine consciousness. In this scenario, we need to define which are the conceptual building blocks integrated in a possible machine consciousness implementation. Then we could test the presence of these functional components and their interrelation within a given system in order to assess its potential level of machine consciousness. However, the definition of these components would require a complete understanding of ‘natural’ consciousness, and given that the quest for consciousness has not yet come to a successful end, a more modest framework has to be established in the realm of artificial systems. But, what are the components of consciousness that we are not able to explain or concretely define so far? We need to decompose, or at least conceptually decouple consciousness dimensions in order to be able to answer this question.

2.1 The Dimensions of Consciousness

An extremely complex phenomenon like consciousness can be seen as a whole, or more conveniently, it can be analyzed as if it was composed of two interrelated dimensions. A conceptual division can be outlined when a distinction is made between phenomenology and access [6]. While the access dimension (A-Consciousness) refers to the accessibility of mind contents for conscious reasoning and volition, phenomenology (P-Consciousness) is related to the subjective experience or qualia, i.e. how does it feel to be thinking about something, or what is it like to be someone else, as Nagel would formulate it [20]. Understanding how P-Consciousness is produced by biological organisms is a controversial problem usually regarded as the explanatory gap [17], which still remains to be closed (if ever possible). While the access dimension of consciousness has an obvious function, namely guiding conscious reason and action; the phenomenal dimension lacks a generally accepted function (see [6] and [8] for a detailed discussion on the matter). Qualia could be just a side effect produced by access mechanisms, or it could play a key role in the integration of multimodal perception [27]. What is generally accepted is that rather than binary properties, both access and phenomenal aspects of consciousness come in various degrees. Therefore, we think it is possible to represent a range of degrees of consciousness in a bi-dimensional space defined by phenomenal and access dimensions (see Figure 1). The access dimension represents the informational aspect of consciousness, while the phenomenal dimension represents the associated ‘what-is-it-like-ness’.

Figure 1. Consciousness bi-dimensional space in biological phylogenics.

The questions of having A-Consciousness without P-Consciousness and vice versa are typically controversial issues in the study of consciousness. In the present work, we have adopted the assumption that machine consciousness and ‘biological’ consciousness are actually different phenomena. Therefore, different kinds of consciousness could be present in artificial agents, and these new versions of machine consciousness could follow different rules in terms of the conceptual link between A-Consciousness and P-Consciousness. While we assume that both A-Consciousness and P-Consciousness increase uniformly at the same rate in biological phylogeny (as depicted in Figure 1), we consider that all combinations are a priori possible in artificial agents. We believe that the evolutionary forces involved in the design of biological organisms have always produced functionally coherent machinery; hence zombies or P-Unconscious (A-Consciousness without P-Consciousness) and A-Unconscious (P-Consciousness without A-Consciousness) do not naturally exist. Nevertheless, there exist cases of individuals that after suffering cerebral vascular accidents or traumatic brain injury become P-Unconscious or A-Unconscious in some respects and degrees. For
instance, brain-injured patients who have developed prosopagnosia are unable to consciously recognize faces despite being able to recognize any other visual stimuli. Even though prosopagnosic patients are unable to experience any feeling of familiarity at the view of faces of their closest relatives (loss of P-Consciousness), other cognitive operations are still performed with the perceived faces – a covert face recognition takes place – but their output fails to reach consciousness (a disorder of A-Consciousness). However, some A-Consciousness capability remains in many cases as they are usually able to implicitly access to knowledge derived from ‘P-Unconsciously’ unrecognized faces [28].

It is also important to distinguish between consciousness as it is applied to creatures and consciousness as it is applied to mental states [19]. Essentially, a conscious subject can have conscious and unconscious mental states. In the prosopagnosia example discussed above, conscious individuals fail to have P-Consciousness of faces at view and their A-Consciousness is also impaired to that respect. However, these subjects can perfectly be A-Conscious and P-Conscious of the voice and speech of their relatives or any other person. In this paper, we generally refer to creature consciousness, hence evaluating the potential level of consciousness of individuals as per their ability to have P-Conscious and A-Conscious states. The particular contents of the mental states will be analyzed later as part of the method to establish a taxonomy for machine consciousness.

2.2 A Computational Approach to Consciousness in Intelligent Agents

The possible functionality of P-Consciousness and the possibility of effectively having one dimension of consciousness without another remain unanswered questions. Therefore, the interrelation between access and phenomenology remains highly unclear and controversial. Some authors even consider P-Consciousness as an epiphenomenal process, hence independent of behavior (see for instance [32], while others tend to identify a key functional role in qualia [21].

Following a pure computational approach we could consider both A-Consciousness and P-Consciousness as being the same functional process, thus neglecting the possibility of subjective experience in artificial agents. However, we think that a different dimensional decomposition is to be made in the realm of machine consciousness (see Figure 2). Although the nature and required underlying machinery for qualia are not known, we believe that some functional characterization of P-Consciousness can be made. Therefore, we have adopted a functional point of view, in which we introduce a redefined dimension of consciousness called Integrative Consciousness (I-Consciousness). In our conception of machine consciousness, we have taken the assumption that I-Consciousness represents the functional aspect of P-Consciousness that exists in conscious biological organisms.

In order to characterize consciousness as a property of agents we need to formally define the basic components of an artificial situated agent. Such an agent interacts with the environment by retrieving information both from its own body and from its surroundings, processing it, and acting accordingly. Following Wooldridge’s definition of abstract architectures for intelligent agents [33], and taking into account the embodiment aspect of situated agents, we have identified a set of essential architectural modules: sensors, sensorimotor coordination, internal state, and effectors. These modules implement the following processes: perception, reason, and action. Consequently, the following abstract architectural components can be identified:

- **Body (B).** Embodiment is a key feature of a situated agent [11]. Agent’s body can be physical or software simulated (as well as its environment). A boundary is established between agent’s body and its environment (E). The rest of components are usually located within this boundary. We believe that it is important to make a distinction between agent’s body (or plant if we take a control theory standpoint) and the environment, as the first is directly controlled while the latter is indirectly controlled. The definition of the body of an agent is important as it determines what sensors it can use, how its effectors work, and ultimately how its perception and behavior is affected by its physical embodiment. Owning an active body is essential for the acquisition of consciousness.

- **Sensory Machinery (S).** Agent’s sensors are in charge of retrieving information from the environment (exteroceptive sensors) or from the agent's own body (proprioceptive sensors).

- **Action Machinery (A).** In order to interact with the environment the agent uses its effectors. Agent’s behavior is composed of the actions ultimately performed by this machinery.

- **Sensorimotor Coordination Machinery (R).** From purely reactive agents to deliberative ones, the sensorimotor coordination module is in charge of producing a concrete behavior as a function of both external stimuli and internal agent’s state.

- **Memory (M).** Internal agent’s state is represented both by its own structure and stored information. Memory is the mean to store both perceived information and new generated knowledge. We consider that even agents that do not maintain state can be said to have a minimal state represented by its own structure, i.e. preprogrammed sensorimotor coordination rules.

As Wooldridge has pointed out [33], different classes of agents could be obtained depending on the concrete implementation of the abstract architecture. Following the notation that we have adopted, we could say that different sensorimotor coordination functions give place to different classes of agents. For instance, reactive agents or BDI agents [23]. While sensorimotor coordination of reactive agents is characterized by a direct mapping from situation to action, BDI agents decision making is based on internal state representing beliefs, desires, and intentions.

In computational terms, consciousness can be regarded as a unique sequential thread that integrates concurrent multimodal sensory information and coordinates voluntary action. Hence, consciousness is closely related with sensorimotor coordination. Our aim is to establish a classification of agents according to the realization of the functions of consciousness in the framework of agent’s sensorimotor coordination.
3. FUNCTIONS OF CONSCIOUSNESS

As mentioned above, the question of what do qualia do in biological organisms is a controversial one. In this paper we propose that a naturalistic approach on the origin of consciousness can be applied to machine consciousness, and therefore identify the functions that can render an agent conscious (in the sense of Artificial Consciousness). In a vast ocean of information where A-Consciousness provides access to virtually any content of the mind, I-Consciousness which permits the construction of coherent and adaptive complex percepts. The set of finally accepted percepts form a unique and coherent stream of consciousness, which the agent exploits to develop other higher level cognitive functions.

Out of the set of cognitive functions that an intelligent agent could potentially exhibit, the following list of functions specifically characterizes the behavior of a conscious agent: Theory of Mind (ToM) and Executive Function (EF). ToM is the ability to attribute mental states to oneself and others. From a human developmental standpoint, Lewis suggests four stages in the acquisition of ToM: (1) “I know”; (2) “I know I know”; (3) “I know you know”; and finally (4) “I know you know I know” [18]. The term EF includes all the processes responsible for higher level action control, in particular those that are necessary for maintaining a mentally specified goal and for implementing that goal in the face of distracting alternatives [22]. Attention is an essential feature of EF. It represents the ability of the agent to direct its perception and action, i.e. selecting the contents of the working memory out of the entire mind’s accessible content. Planning, coordination, and set shifting (the ability to move back and forth between tasks) are also key processes included in EF. We argue that the integration of all of these cognitive functions could build an artificial conscious mind. However, each of the mentioned functions could also be implemented independently or partly integrated with other cognitive functions, thus giving place to different levels of implementation of artificial consciousness as discussed in the next section.
4. LEVELS OF MACHINE CONSCIOUSNESS

Table 1 describes ConsScale, which is a list of potential levels of consciousness for artificial agents. This scale has been defined in terms of reference agent abstract architectures and characteristic behaviors. The characteristic behavior assigned to each level has been derived from the functionality of consciousness discussed above. As illustrative analogy, machine consciousness levels are assigned a comparable level of consciousness in biological phylogenics and human ontogeny.

The first level in the scale, Disembodied, refers to a ‘proto-agent’ and serves as an initial reference that remarks the importance of a defined body as a requirement for defining a situated agent. The rest of the scale comprises a set of twelve ranks, where lower levels are subsumed by higher ones. Therefore, each stage of the incremental development of an artificial agent could be identified by a concrete level. Levels 0 and 1, Isolated and Decontrolled respectively, are also conceptual references which help characterize situatedness in terms of the relation with the environment. Both classes represent inert bodies lacking any functionality or interaction with the medium except the inevitable derived from the physical properties of their inactive bodies.

Therefore, these classes cannot be defined as situated agents. Level 2, Reactive, defines a classical reactive agent which lacks any explicit memory or learning capabilities. From level 2 onwards the agents make use of the environment as the mean to close the feedback loop between action and perception. Hence, all agent types above level 1 can be regarded as situated agents.

Although we are explicitly focusing in individual agent evaluation, it is important to note that additional learning or adaptation processes could exist at an evolutionary plane (assuming that agents are able to replicate, mutate, and evolve). For instance, although reactive rules are fixed for a level 2 individual, adaptation of reactive responses in a population of agents could take place over the generations.

Level 3, Rational, can be identified as the simplest form of a classical deliberative agent. At this level, the agent’s internal state is maintained by a memory system and sensorimotor coordination is a function of both perceived and remembered information. Proprioceptive sensing can be present at this level; however, it is not producing any self-awareness. The next level, Attentional, is characterized by an attention mechanism, which allow the agent to select specific contents both from the sensed and stored state information.

<table>
<thead>
<tr>
<th>Level of Machine Consciousness</th>
<th>Agent Architecture</th>
<th>Short Description</th>
<th>Characteristic Behavior</th>
<th>Biological Phylogeny</th>
<th>Human Ontogeny</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level -1 Disembodied</td>
<td><img src="image" alt="E B S A" /></td>
<td>Boundaries of the agent are not well defined. It can be confused with the environment.</td>
<td>None. It is not a situated agent.</td>
<td>Amino acid as part of a protein.</td>
<td>n/a</td>
</tr>
<tr>
<td>Level 0 Isolated</td>
<td><img src="image" alt="E B" /></td>
<td>Obvious distinction between body and environment, but no autonomous processing.</td>
<td>None. It is not a situated agent.</td>
<td>Isolated chromosome.</td>
<td>n/a</td>
</tr>
<tr>
<td>Level 1 Decontrolled</td>
<td><img src="image" alt="E S B A" /></td>
<td>Presence of sensors and/or actuators, but no relation between them.</td>
<td>None. It is not a situated agent.</td>
<td>Dead bacteria</td>
<td>n/a</td>
</tr>
<tr>
<td>Level 2 Reactive</td>
<td><img src="image" alt="E S B R A" /></td>
<td>Fixed reactive responses. R establishes an output of A as a predetermined function of S.</td>
<td>No higher function. Primitive situatedness based on reflexes.</td>
<td>Virus</td>
<td>n/a</td>
</tr>
<tr>
<td>Level 3 Rational</td>
<td><img src="image" alt="E S R M A" /></td>
<td>Actions are a dynamic function of both memory and current information acquired by S.</td>
<td>Basic ability to learn and proprioceptive sensing allow orientation and positioning behavior.</td>
<td>Earthworm 1 Month.</td>
<td></td>
</tr>
<tr>
<td>Level 4 Attentional</td>
<td><img src="image" alt="E S R M A" /></td>
<td>Attention mechanism selects Ei contents from S and M. Primitive emotions.</td>
<td>Ability to direct attention toward selected Ei allows attack and escape behaviors.</td>
<td>Fish 5 Months.</td>
<td></td>
</tr>
<tr>
<td>Level 5 Executive</td>
<td><img src="image" alt="E S R M A" /></td>
<td>Multiple goals can be interleaved as they are explicitly represented in memory.</td>
<td>Set shifting capability allows multiple goal achievement. Basic emotional learning.</td>
<td>Quadruped mammal 9 Months.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Artificial Agents Consciousness Scale (ConsScale)
A level 5 agent, Executive, includes a more complex internal state representation, which provides set shifting capabilities. The achievement of multiple goals is sought thanks to a higher coordination mechanism that shifts attention from one task to another. Level 6, Emotional, is the first level in which an agent can be to certain extend regarded as conscious in the sense of self-awareness. The main characteristic of this level is the support for ToM stage 1, “I know”. Complex emotions are built as a combination of basic emotions and they are not only used to evaluate external objects but to assess the internal agent status. Level 7, Self-Conscious, corresponds to the emergence of self-consciousness. At this level the agent is able to develop higher order thoughts [26], i.e. thoughts about thoughts, and more specifically thoughts about itself. Consequently it presents support for ToM stage 2, “I know I know”. Progressing to the next level, Empathic, the internal representation of the agent is enriched by inter-subjectivity. In addition to the model of the self, others are also seen as selves; hence, they are consequently assigned a model of subjectivity. This is the seed for a complex social interaction. The next step is represented by level 9, Social, where ToM is fully supported. Level 10, Human-Like, represents the sort of agent that is endowed with the same level of consciousness as a healthy adult human has. Therefore, the formation of a complex culture is a feature of this level. Finally, level 11 or Super-Conscious, refers to a kind of agent able to internally manage several streams of consciousness, while coordinating a single body and physical attention. A mechanism for coordination between the streams and synchronized access to physical resources would be required at this level.

### 5. CLASSIFYING AGENTS USING ConsScale

The levels of artificial consciousness defined in ConsScale are characterized by abstract architectural components and agent’s behavior. The architecture components represent functional modules whose integration makes possible the emergence of a characteristic behavior. Therefore, at least one behavior-based test can be associated to each level in order to assess if a particular
agent fulfills the minimum required behavioral pattern for that level. In fact, an agent can only be assigned a concrete level if and only if it is able to show the behavioral pattern of that level as well as the behavioral patterns of all lower levels, e.g. even though an agent is able to pass ConsScale level 7 behavior test, it does not necessarily imply that it can be regarded as Self-Conscious in terms of ConsScale. It would also need to comply with all lower levels.

As discussed above, the three first reference levels (Disembodied, Isolated, and Decontrolled) are a special case as they do not actually describe situated agents. Therefore, there are no behavioral tests associated to any of these first three levels. A given agent could be assigned either of these initial reference levels just by analyzing its architectural components. In contrast, from level 2 onwards a characteristic behavior pattern is defined per ConsScale level. This characteristic pattern should be taken as the base of any behavior test that can be assigned to a particular level. Reference behavior patterns for levels 2 to 11 are discussed below.

The characteristic behavior of level 2, Reactive, is the reflex, hence an agent able to autonomously react to any given environment situation is said to comply with level 2. When the response to a given environment state is not fixed, but it is a function of both the information acquired by S and agent’s internal state, then the agent is said to comply with level 3, Rational (note that some proprioceptive sensing mechanism is required to make agent’s internal state available in R, so it can be an input of the sensorimotor coordination function). Most BDI-type agents ([23]) could be classified as level 3 in terms of ConsScale.

If the agent is able to direct attention to a selected subset of the environment state (E_i) while other environmental variables are also sensed but ignored in R, and the selected perception is evaluated in terms of agent’s goals so subsequent responses are adapted (primitive emotions), then the agent is said to comply with level 4, Attentional. Level 4 agents are able to show specific attack or escape behaviors and trial and error learning. The ability to pay attention toward specific objects or events gives place to the formation of directed behavior, i.e. agent can develop behaviors clearly related to specific targets, like following or running away. Additionally, level 4 agents can have primitive emotion mechanisms in the sense that the objects to which attention is paid are emotionally evaluated as positive or negative. A positive emotion triggers decrease of distance behavior or bonding to selected object, while negative emotion triggers increase of distance and reinforcement of boundaries toward selected object ([9]).

If an agent that can be successfully classified as Attentional in terms of ConsScale also exhibits set shifting and basic emotional learning capabilities, then it can be regarded as Executive (ConsScale level 5). In addition to advanced planning, emotional learning is another characteristic that can be observed in some degree at this level, as the most emotionally rewarding tasks are assigned more time and effort.

By basic emotional learning we mean that the agent is able to learn basic rules from one task and adapt its behavior consequently in the performance of that particular task. In contrast, Emotional (ConsScale level 6) agents are characterized by complex emotions and complex emotional learning. This means that the agent generalizes the learned lessons to its general behavior, furthermore, emotions are also assigned to the self and self-status monitoring and evaluation gives place to a sense of “I know” (support for ToM stage 1). Even though a representation of the self is considered as an input of the sensorimotor coordination function, this is an implicit symbol. However, level 7 (Self-Conscious) is described by an explicit symbol for the self, which enables self-recognition. The reference behavior test for this level would be the mirror test, which although originally applied to primates ([13], has also been adapted to other mammals and even artificial agents. Takeno et al. have proposed a specific experiment design to test whether a robot is able to recognize its own image reflected in a mirror ([30], Planning capabilities are extended as the self is integrated both in the current state representation and future state estimation. Behavior at this level is also illustrated by the ability to use tools (see for instance [1]).

ConsScale Level 8 (Empathic) is achieved by an agent when it shows that it maintains a model of others, and therefore it collaborates accordingly with other agents in the pursuit of a common goal. In fact, joint goals require this, and the need for socially aware plans in BDI agents has been considered some time ago ([24]).

In level 9, Social, the internal model of other selves is enhanced with a full support of ToM. This means that characteristic behavior of this level is defined by sophisticated Machiavellian strategies (or social intelligence) involving social behaviors like lying, cunning, and leadership. In other words, an agent A could be aware that another agent B could be aware of A’s beliefs, intentions, and desires. Advanced communication skills are the characterization of this level behavior, where, for the first time, an agent would be able to purposely tell lies. There exist mathematical models of the dynamics of Machiavellian intelligence that could be used to test these sorts of behaviors with artificial agents ([14]).

While, the obvious test for level 10, Human-Like, is the Turing test ([31], also accurate communications skills (language) and the creation of a culture would be a clear feature of level 10. Other key characteristics are that the agent is able to profoundly modify its environment and society. The fluidity between social and technical intelligence permits the extension of its own knowledge using external media (like written communication) and technological advances are also possible.

Finally, we cannot envisage any conclusive behavior test for level 11 due to the lack of known exemplifying references.

6. CONCLUSIONS

We have proposed ConsScale as a machine consciousness taxonomy for artificial agents, which can be used as a conceptual framework for evaluating the potential level of consciousness of a given agent. Most of current implementations of artificial agents fall between levels 2 and 4 inclusive. The classification of any current implementation as fully belonging to level 5 could be thoughtfully discussed elsewhere; nonetheless, we think these kinds of agents are within current technology possibilities.

Identifying consciousness by means of interpreting behavior remains an open problem that is being currently addressed primarily in mammals, cephalopods, and birds ([12, 29]. However, more effort should be put in the domain of artificial agents.
7. ACKNOWLEDGMENTS
This research has been supported by the Spanish Ministry of Education and Science under project TRA2007-67374-C02-02.

8. REFERENCES
A Reinforcement Learning Approach to Setting Multi-Objective Goals for Energy Demand Management

Ying Guo, Astrid Zeman and Rongxin Li

Autonomous Systems Laboratory, CSIRO ICT Centre
North Ryde, Australia 2113
+61 2 9325 3156
{firstname.lastname}@csiro.au

ABSTRACT
In order to cope with the unpredictability of the energy market and provide rapid response when supply is strained by demand, an emerging technology, called energy demand management, enables appliances to manage and defer their electricity consumption when price soars. Initial experiments with our multi-agent, power load management simulator, showed a marked reduction in energy consumption when price-based constraints were imposed on the system. However, these results also revealed an unforeseen, negative effect: that reducing consumption for a bounded time interval decreases system stability. The reason is that price-driven control synchronizes the energy consumption of individual agents. Hence price, alone, is an insufficient measure to define global goals in a power load management system.

In this paper we explore the effectiveness of a multi-objective, system-level goal which combines both price and system stability. We apply the commonly known reinforcement learning framework, enabling the energy distribution system to be both cost saving and stable. We test the robustness of our algorithm by applying it to two separate systems, one with indirect feedback and one with direct feedback from local load agents. Results show that our method is not only adaptive to multiple systems, but is also able to find the optimal balance between both system stability and energy cost.

Categories and Subject Descriptors
J.2 [Physical Sciences and Engineering]: Engineering.

General Terms
Algorithms, Design, Experimentation, Management.

Keywords
Energy, Reinforcement learning, multi-agent systems, stability.

1. INTRODUCTION
As technology advances, consumers become increasingly more power hungry. This causes many countries, including Australia, to suffer from an increasing gap between electricity supply and demand [1][17]. The traditional way of tackling such a problem is to increase supply by investing heavily in infrastructure and building more generators. Alternatively, power load management can reduce levels of power consumption on the demand side, and hence reduce the level of energy required to run appliances, saving money and reducing the risk of inadequate supply [16][18].

Electricity distribution is a complex system, consisting of loads, generators, and transmission and distribution networks [5]. To control this physical system, generators and retailers bid into a market that balances supply and demand while ensuring safe network operation. Demand and price fluctuate quickly and loads that are responsive in real time can have high value to retailers and networks. This scenario is ideal for the adoption of multi-agent technology [3][12]. A network of autonomous agents can be overlaid on the physical distribution network, controlling customer loads and, where available, local generators [2][6][14][21].

Australia’s Commonwealth Scientific and Industrial Research Organisation (CSIRO) is developing an energy management and control system that consists of an agent network to be installed at multiple levels of the electricity distribution network [7][9][10]. This system has a three-level architecture consisting of the following types of agents: 1) the top-level broker agent, 2) the middle-level group agent and 2) the bottom-level appliance agent.

Appliance agents are responsible for the low-level management of consumption for each end-use device. At the bottom level, appliances can be intelligently switched on or off based on customer preferences [5]. A cluster of such agents is then managed by a group agent on the middle level [11][13]. The group agents also receive an energy quota from an upper-level broker agent representing the needs of electricity market
participants and network operators. This energy quota is a limit on the total energy consumption for a group of appliances.

In this paper, we focus on the intelligent management of the broker agent -- that is, how to choose the optimal strategy to provide the real-time quota to the group agent. As the top level in the system, the broker agent needs to incorporate information from the market (such as the current local price of energy) as well as input from group agents [4]. The broker agent is required to manage the risk of exposure to volatile wholesale pool prices and reduce strain on the network.

Energy price can change dramatically when the demand is very high such as a hot summer afternoon. Because market energy prices and weather conditions are dynamic, we chose to use the reinforcement learning (RL) framework as an online learning approach. The two usual approaches to reinforcement learning are model-free and model-based. Model-free algorithms perform well for simple problems. Because the energy market is a dynamic system which exhibits unpredictable properties such as the price of energy, we cannot easily generate a model beforehand. Hence we use the model free RL algorithm – Q-learning. To cope with the dynamic nature of the energy market, we enable the reward matrix to update towards the optimum (which is different from conventional RL) while the agent learns the environment. We present this approach and its application for setting the system-level goal. The experimental results show that by using this method, the broker agent is not only adaptive to multiple systems, but is also able to find an optimal balance between both system stability and energy cost.

The remainder of this paper is organised as follows. Section 2 outlines our approach and Section 3 describes the RL learning process. In Section 4, we present experimental results in a simulated environment. Finally, conclusions based on these experiments are given in Section 5.

2. PROBLEM DESCRIPTION

2.1 Australian Energy Network Model

In Australia, the National Electricity Market Management Company Limited (NEMMCO) was established in 1996 to administer and manage the National Electricity Market (NEM), to develop the market and continually improve its efficiency [1]. Figure 1 (left) shows the typical Australian energy and financial flow under management of NEMMCO. Within such flow structure, wholesale trading in electricity is conducted under a spot market where supply and demand are instantaneously matched in real-time through a centrally-coordinated dispatch process. Generators offer to supply the market with specific amounts of electricity at particular prices. From all offers submitted, NEMMCO determines which generators are to produce electricity by meeting prevailing demand in the most cost-efficient way. NEMMCO then dispatches these generators into production. A dispatch price is determined every five minutes, and six dispatch prices are averaged every half-hour to determine the spot price for each trading interval for each of the regions of the NEM. NEMMCO also sets a maximum spot price of $10,000 per Megawatt hour. This is the maximum price at which generators can bid into the market. During most times of the year, the spot price is low (less than $20 per Megawatt hour), but every now and then, it can also be very high (near or at the maximum price), when there is a peak demand. This occurs when energy supply is under extreme pressure, such as when there is extreme high temperature on a hot summer day.

Electricity is not economically storable, and being a volatile commodity, production is subject to rigid, short-term capacity constraints. Since demand is highly variable, this means there will be times when there is plenty of capacity and the only incremental costs of producing electricity are fuel, operating and maintenance costs. At other times, the capacity constraint will be binding, causing the incremental cost to increase greatly and market prices to rise [2][17][19].

![Figure 1](image)

Figure 1: An energy network model. Left: Typical Australian energy and financial flow. Right: Enlarged three-level multi-agent structure for our system.

2.2 Multi-Agent Architecture Placement

For NEMMCO, the highest priority is power system reliability. To improve reliability of the power system and avoid the maximum spot price, one approach is to intelligently reduce the peak demand. To do so, we design an energy management architecture as shown in Figure 1 (right). This structure is comprised of two layers (bottom-level & middle level) of energy distribution agents, and one layer of consumption manager agents (top-level). This tri-level architecture can be easily outlined in a bottom-up fashion. On the consumption level, the energy usage of electrical devices or appliances may be controlled, to a varying degree, by the device-manager agents that switch on and off the power supply to the device. A cluster of such agents, grouped together according to physical or market driven factors, are managed by a group agent on the lower distribution level [11][13]. The group agent, in turn, receives a group quota from an upper-level distribution agent, referred to as the broker agent. The broker agent, which can view the spot price as well as the group demand requirements, defines the quota in order to achieve system-level goals, described in Section 2.4.

2.3 The Role of Broker Agent

The goal of the broker agent is to retrieve up-to-date information on the market and demand requirements, make dynamic, informed decisions on local energy consumption limits and propagate this information down to the group agent. The information passed to a
group agent is referred to as a *cap*, and is the upper limit of energy consumption allocated to a particular group.

The sequence of events for deciding a cap in the broker agent is as follows. Firstly, the broker agent communicates with external parties to obtain knowledge of current and historical data, such as the latest market price, regional demand, local weather, etc. Secondly, the broker agent communicates with the group agent to gain knowledge of local energy demand requirements. These local energy demand requirements are expressed as two sets of information: 1) the minimal achievable consumption of the group and 2) the unconstrained (default) consumption of the group. Here minimal is defined as the total minimal consumption over an entire future market cycle interval (e.g., five minute interval). The unconstrained (default) consumption of the group refers to the total consumption of the group when agents operate under normal conditions without external influence (see Figure 2). Lastly, the broker agent calculates the cap on the next market cycle interval for the group agent using all information available to it. The cap calculated by the broker lies inclusively between the values of the constrained and unconstrained plans sent from the group agent.

![Figure 2. Exchange of information between Broker and Group Agent.](image)

One would assume that an optimal cap function would be the minimal achievable consumption for the group; that is, restricting the agent energy consumption levels to the lowest possible value that the system can cope with. In fact, when imposing a minimal cap, electricity which is not consumed in one period will usually need to be consumed at a later time. The longer a minimal cap is imposed, the more devices are pushed to a stressful limit, so that when the cap is released, all appliance agents, having been energy-starved, choose to switch on at that point, causing demand to increase rapidly. This result is demonstrated in Figure 3, which shows the total power demand of a system of one hundred simulated refrigerators: when a strict cap is imposed in the first half hour, a peak level in consumption occurs when the cap is released. Such peak demand can cause negative effects such as shortage in energy provision or a peak price. System stability is then threatened. Hence the broker agent needs to define a cap which not only reduces energy costs, but also avoids instability within the system.

### 2.4 Cap Function Definition

We assume the capped value of collective energy consumption is imposed over a period $T$. The cap can be expressed as a constant that directly defines the limit or as a time varying, piece-wise function. The cap function, $L(t_j)$, where $t_0 \leq t_j < t_0 + T$, is represented in the following form:

$$L(t_j) = l_j \left[ \kappa_{\text{min}}(t_j) - \kappa_{\text{min}}(t_j) \right] + \kappa_{\text{min}}(t_j),$$

where $l_j \in [0, 1]$ is time invariant, $t_j = t_0 + j \delta$, $j \in Z$, $J = \{ j \in Z : 0 \leq j \delta < T \}$. $\kappa_{\text{min}}(t_j)$ is the unconstrained consumption of the group, and $\kappa_{\text{min}}(t_j)$ is the minimal consumption of the group. Here $\delta$ is the market cycle interval. We also assume that $\kappa_{\text{min}}(t_j)$ and $\kappa_{\text{min}}(t_j)$ are time invariant during each interval.

This function defines a sequence of percentages over $T$, where $l_j = 0$ means the cap is equivalent to the minimal consumption $\kappa_{\text{min}}(t_j)$, $l_j = 1$ means the cap is equivalent to the unconstrained consumption $\kappa_{\text{min}}(t_j)$. When $0 < l_j < 1$, the cap is chosen to be a value between $\kappa_{\text{min}}(t_j)$ and $\kappa_{\text{min}}(t_j)$, calculated by $L(t_j)$.

![Figure 3. Imposing a strict cap on consumption creates future oscillations in demand.](image)

### 3. The LEARNING METHOD for OPTIMAL CAP SELECTION

#### 3.1 Reinforcement Learning

The main goal now is to find an algorithm that the broker agent can use to set an near-optimal cap for the group agent. The judgment of the “optimal” cap can be measured based upon two important elements:

- **Cost**: the global cost of utilizing energy;
- **System stability**: the energy demand distribution along the time dimension.

In this optimization problem, there is insufficient information to apply a supervised learning methodology. In typical supervised learning algorithms, sample input-output pairs are required to be learnt from. The broker agents in our problem have no a-priori knowledge of “correct” cap settings. The dynamic nature of the energy market is yet another reason why the application of a supervised learning methodology is unsuitable. The market
changes its behavior due to many direct and indirect factors such as local temperature, weather condition, market spot price and predicted demand.

It is therefore necessary to use a learning methodology which allows the agent to learn its behavior online based on feedback from the environment. We choose one of the most popular reinforcement learning (RL) approaches, Q-learning, to allow the broker agent to learn from experience. Behavioral learning is adaptive with time, allowing RL to cope with control of a dynamic system, such as energy demand. Adopting behavioral learning provides the added advantage that there is little need for the broker agent to know about the system. The broker agent can learn how to set the cap through trial-and-error interactions with the environment.

For Q-learning, a matrix $R$ is used to model the environment reward system [20]. Then, another matrix, named $Q$, is put into the brain of the agent that will represent the memory of what the agent has learned through experience. In our problem, because of the non-stationary and dynamic nature of the energy system, the environment may be different for the same agent action. To cope with this problem, we enable the $R$ matrix to be updated in parallel with learning the $Q$ matrix. The pseudo code of this method is presented in Table 1.

### 3.2 State Space Definition

To initialize the learning algorithm, we first need to define the state space, the action space and the action reward / penalty function. In our problem, the energy demand, the electricity price, and the cap are real values. To simplify the algorithm, we first transform these values into discrete quantities.

- **Three energy demand** ($d$) states are {low, medium, high} using two thresholds $D_l$ and $D_H$:
  
  $d = \begin{cases} 
  \text{low} & \text{as } d < D_l \\
  \text{medium} & \text{as } D_l \leq d < D_H \\
  \text{high} & \text{as } D_H \leq d
  \end{cases}$

- **Two electricity price** ($P$) states are {normal, abnormal}:
  
  $P = \begin{cases} 
  \text{normal} & \text{when } P < $100/MWh \\
  \text{abnormal} & \text{when } P \geq $100/MWh
  \end{cases}$

- **Cap ratio** ($l$) states are {50%, 70%, 80%, 90%, 100%}.

The cap function $L(t)$ can then be calculated according to equation (1) as the action.

We define the combination of the energy demand $d$, the electricity price $P$, and the cap ratio $l$ as the state $s=(d, P, l)$. Hence the size of the state space $s$ is: $3 \times 2 \times 5 = 30$ in this case. The above threshold and discrete settings are predefined according to properties of the Australian market. We will show in the experimental section that such simple discretization and small state space still provides good results for the broker agent. Note that we can always enlarge the state space by defining finer discrete thresholds.

### 3.3 Learning Process for Cap Selection

The broker agent is first initialized with an allocation of group agents under its control. $\kappa_{un}(t_0)$ is the initial unconstrained consumption plan (request) sent from the group agent, along with the minimal consumption plan $\kappa_{min}(t_0)$ for the group. On initialization, the broker agent also receives environmental state information such as $P(t_0)$, the market electricity price at $t = t_0$.

Firstly, the group agent sends the plan information to the broker agent. The broker then calculates an initial cap $L(t_0)$ using equation (1), with a randomly chosen $l_1$. This cap is propagated down to group agents which then adjust the consumption profiles of their loads. The real energy consumed $d(t_1)$ is reported to the broker agent when the current market cycle is complete. The broker agent’s state at time $t_1$ is decided as $s(t_1) = (d(t_1), P(t_1), L(t_1))$. Here, the real energy consumed $d(t)$ is a function of the previous cap and previous plans $\kappa_{min}(t-1)$, $\kappa_{min}(t-1)$. It is defined as:

$$d(t) = \kappa_{min}(t) + \beta \times (1 - l_{t-1}) \times (\kappa_{min}(t-1) - \kappa_{min}(t-1)),$$

where $\beta$ is an effect-factor which describes the side effect of the cap from the previous time step (ref. to Figure 3).

At the $Nth$ time step, a reward is provided as:

$$w = -\left(\alpha \sum_{j=0}^{N} P_j \times d_j + \sigma^2\right),$$

The reward function has a two-fold purpose, as described in Section 3.1, which is reflected in the function as two separate terms. The first term gives a reward based on the cost of energy in the previous $N$ time units. The second term gives a reward based upon the variation $\sigma$ of the cap $L(t)$. The parameter $\alpha$ is a weight which reflects the importance between these two measurements.

The pseudo code of the algorithm is as in Table 1. During each training session (each episode), the broker agent explores the environment (represented by Matrix $R$) and retrieves the reward $w$ until the end of the learning process. The purpose of training is to enhance the “brain” of the broker agent, represented by the $Q$ matrix. More training will result in a finer-tuned $Q$ matrix that can be used by the agent to make optimal decisions.

The feedback mechanism within the system creates a flow-on effect of decisions. That is, a different cap which is chosen within one market cycle will result in different energy requirements for following cycles. This flow-on effect is represented in the algorithm as the discount-rate parameter $\gamma$, which has a range of 0 to 1 ($0 \leq \gamma \leq 1$). If $\gamma$ is closer to zero, the agent will tend to consider only immediate reward. If $\gamma$ is closer to one, the agent will consider future reward with greater weight, willing to delay the reward. The size of the $R$ matrix is decided by the number of states. That is, a 30x30 matrix. It is initialized to be a matrix with values of negative infinity because there is no knowledge about the environment, hence no reward. The $R$ matrix is then updated as a function of the immediate reward at each time step. We can perform learning of the $R$ matrix whilst also learning the $Q$ matrix simultaneously. The $R$ matrix dynamically adjusts to suit the non-stationary environment, including complex effects such as the market, group agent demand and weather.
Table 1. The Learning Approach

<table>
<thead>
<tr>
<th>Initialize environment reward matrix R: elements equal to (-\infty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize the discount-rate parameter (\gamma \ (0 \leq \gamma \leq 1))</td>
</tr>
<tr>
<td>Initialize the Q matrix as a zero matrix</td>
</tr>
<tr>
<td>Set episode counter to one</td>
</tr>
</tbody>
</table>

WHILE episode counter is less than or equal to total number of episode

- Decide the initial state – random select \(l_i\), read in \(\kappa_{\text{max}}(t_0)\), \(\kappa_{\text{min}}(t_0)\), \(P(t_0)\)
- DO while not read end of current episode
  - Select \(l_i\) among all possible actions for current state
  - Using \(l_i\) to calculate \(d(t)\) and go to the next state
  - Get next state based on all possible actions
  - Calculate immediate reward \(w\) using eq. (4)
  - Update \(R\) matrix: \(R(s, l) \leftarrow (R(s, l) + w)/2\)
  - Update \(Q\) matrix:
    \[ Q(s_i, l_i) \leftarrow R(s_i, l_i) + \gamma \max_{l} Q(s_{i+1}, l) \]
- ENDDO
ENDWHILE

4. EXPERIMENTS

In this section, we will show how the above learning approach is applied to the broker agent that can choose an optimal strategy to provide the real-time quota to the group agent. We will consider two different scenarios separately.

4.1 Scenario One – Without Direct Feedback From Group Agent

We started our first experiments with the scenario that the cap setting does not directly cause group agents to change their consumption. Hence the indirect effect is calculated by equation (3). For this scenario, the NEMMCO datasets were used.

4.1.1 Data Preparation

The price and demand values chosen fell in the January 2006 period, shown in Figure 4. There were several price peaks during this month for the NSW state of Australia, where the highest value was $529.95 on 23rd January 2006. In fact, the highest demand (12674.6 MWh) also occurred on the same day.

To generate the learning samples, we firstly cut the datasets into pieces (episodes) where each piece has 5 data points as \(E_n = \{(d_1, P_1), (d_2, P_2), (d_3, P_3), (d_4, P_4), (d_5, P_5)\}.

Figure 4 NEMMCO Demand and price data of January 2006 for the NSW region of Australia.

Figure 5 Cap setting where optimization is focused on minimizing demand and cost. Top: Cap chosen by broker agent; Bottom: Unconstrained consumption vs. capped energy consumed.
4.1.2 Experiments

With each episode $E_m$, we learn the $R$ matrix and $Q$ matrix using the following update rule. Firstly, we read in $(d_1, P_1)$, and randomly select a cap ratio $l_1$. The immediate reward can be calculated according to equations (3) and (4). Another random selected cap is defined as $l_2$. With the calculated $d_2$ (using eq. (3)), the next state is then fixed. Hence the corresponding value in the $R$ matrix is updated using the immediate reward, and so is the $Q(s, l)$. In our experiments, we set parameters $\gamma = 0.8$, and $\beta = 0.2$. Using data for the entire month, the first half of the episodes were used for learning the $R$ and $Q$ matrices and the latter half were used for testing.

Experimental results are shown in Figure 5, Figure 6, and Figure 7. Because the reward matrix $R$ is learnt online, the reward function (4) is crucial to the learning results. With the different choice of $\alpha$ in the reward function (4), different cap optimization results were achieved. When the optimization was focused on the cost of demand, the capped demand was kept low (below 10,000 Megawatts for the majority of the time in Figure 5). Smaller demand creates instability in the cap ratio (see top figure in Figure 5). On the other hand, if we put too much emphasis on cap stability, energy demand and cost were ignored, leading to the results in Figure 6.

The best results, illustrated in Figure 7, negotiated the balance between demand cost and system stability. Clearly, the low cap matches the high price very well. When low market prices occurred, the cap was released to 90% of demand requested, avoiding future demand oscillation (as shown in Figure 3).

4.2 Scenario Two – With Direct Feedback From The Group Agent

For this scenario, the recently developed CSIRO multi-agent platform [15] was used. In this platform, cap setting in one market cycle or time step directly affects the group agents’ demand consumption for the next time step. Hence, equation (3) is no longer required.

Within this architecture, when a group agent receives a consumption limit, and subsequently determines that it is below the normal collective requirement (unconstrained demand), it will then modify the devices’ plans to delay their electricity usage whilst also minimizing the impact on the appliances (level of interference) or their operations (e.g. average temperature in a refrigerator). Placing a constraint on the level of consumption for one market cycle will increase the level of required consumption for that agent in the following market cycle. Details of the system architecture are described in [10].

Datasets of demand were generated by the CSIRO demand management simulator. Price data used was retrieved from NEMMCO archives for January 2005 for the NSW region of
Australia. Experiments were simulated over a period of one month, with 5 minute time intervals for each market cycle. The unconstrained demand and price profiles are shown in Figure 8.

4.2.1 Data Preparation

Datasets of demand were generated by the CSIRO demand management simulator. In these experiments we simulate 100 appliances (refrigerators) with randomly selected cycle times between 30 and 50 mins. The temperature bounds of all appliances are between 1 and 10 degrees Celsius. All appliances are rated at 1000kW, consuming an average of 1000kW per hour. Price data used was retrieved from NEMMCO archives for January 2005 for the NSW region of Australia. Experiments were simulated over a period of one month, with 5 minute time intervals for each market cycle. The unconstrained demand and price profiles are shown in Figure 8.

4.2.2 Experiments

For this scenario, the $R$ and $Q$ matrices were learnt over the first half of the month, before the resultant $Q$ matrix was tested for the remainder of the month. When optimization focused on the cost of demand, the capped demand was kept low, producing similar results to Scenario One. When the cap was chosen as a compromise between system stability and cost, results were again similar to Scenario One, illustrated in Figure 9. The real energy consumed is lower than the unconstrained consumption plan from the group agent, while the system stability is achieved as well. These experiments demonstrate the ability for the algorithm to be applied to a real-time environment where direct feedback is implemented. Because comparable results were achieved for this scenario, the assumption (equation (3)) can be used when the cap optimization approach is used for other market systems where the closed loop in Figure 2 does not exist.

5. CONCLUSION

The electricity market is a dynamic system, where demand and price can fluctuate dramatically. Demand management is a recent attempt to control fluctuations in energy requirements at the appliance level, hence reducing the cost of energy provision. Using agent technology, devices can monitor and control their consumption, essentially delaying their energy usage when the demand for electricity is high. This architecture provides benefits to multiple stakeholders, including monetary relief to consumers, as well as relief to the energy network infrastructure.

Initial experiments on reducing energy consumption for short periods were conducted in order to quantify cost savings for high price periods. These experiments revealed an unpredictable effect of system instability when system constraints were released (Figure 3). In order to achieve lower cost without compromising system stability, we propose a learning method to set system-level goals. Using this learning approach, we are able to define a dual-purpose reward function to train the system on increasing system stability as well as reducing energy cost.

Figure 8: Demand data generated by the CSIRO demand management simulator and price data used from NEMMCO NSW, January 2005.

Figure 9: Cap setting where the optimization focus is on load demand and cost. Top: Cap chosen by the broker agent; Bottom: Unconstrained consumption vs. capped energy consumed.
Experiments using our method were conducted within a simulated feedback system as well as a closed-loop system. For our non-closed system (Section 4.1, Scenario One), we simulate feedback within the system using a function which allows us to parameterize the flow-on effect of reducing demand. For our closed-loop system (Section 4.2, Scenario Two), we include models of individual appliance agents, which are directly affected when constraints are placed on the system. For both scenarios, the experimental results show that the energy consumption cost is reduced with a stable system while the broker agent implements the control strategies learnt by our algorithm.

Conducting experiments in multiple environments illustrates the adaptability of our learning method for energy demand management. Results show a compromise can be met between system stability and cost. The method we introduce uses only a small state space for both the decision and reward matrix but still produces effective outcomes. For future work, we wish to explore alternative methods for setting system-level goals. A reduction in parameter space and/or in training time whilst also providing system stability and cost reduction would be ideal.

6. ACKNOWLEDGMENTS
Our thanks to Geoff James, Geoff Poulton, Mikhail Prokopenko and Peter Wang for valuable discussion.

7. REFERENCES
[9] Li, J., Poulton G., and James, G., Agent-based distributed energy management, The 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Queensland, Australia, 2-6 December, 2007.
Learning When to Take Advice:  
A Statistical Test for Achieving A Correlated Equilibrium

Greg Hines  
Cheriton School of Computer Science  
University of Waterloo  
Waterloo, Canada  
ggdhines@cs.uwaterloo.ca

Kate Larson  
Cheriton School of Computer Science  
University of Waterloo  
Waterloo, Canada  
klarson@cs.uwaterloo.ca

ABSTRACT
In this paper we study a multiagent learning problem where agents can either learn via repeated interactions, or can follow the advice of a mediator who suggests possible actions to take. We present an algorithm that each agent can use so that, with high probability, they can verify whether or not the mediator’s advice is useful. In particular, if the mediator’s advice is useful then agents will reach a correlated equilibrium, but if the mediator’s advice was not useful, then agents are not harmed by using our test, and can fall back to their original learning algorithm. We then generalize our algorithm and show that in the limit it always correctly verifies the mediator’s advice.

1. INTRODUCTION
In settings where agents repeatedly interact with each other (for example, through a repeated game), there are great opportunities for learning since agents are able to adapt their strategies given the history of play. This problem has garnished a lot of attention from several research communities, including the AI community and the game theory community. While many criteria have been proposed for measuring the success of learning approaches, one commonly used measure is whether the agents learn how to best-respond to the strategies being played by the others. That is, does the learning process converge to an equilibrium.

In this paper we study the problem of agents interacting with each other in a repeated game setting, but we introduce a third party mediator or advisor who makes strategy suggestions to the agents. Ideally, by following the suggestions of the mediator, agents will be able to learn how to play against each other, possibly even reaching mutually beneficial outcomes which would not have been possible without the mediation. That is, our goal is for the agents to learn and adapt so that they find a correlated equilibrium [1].

However, a mediator is only useful if it can make good suggestions. Even if a mediator tries to make good suggestions it may be prevented by coding errors, memory limitations, etc. For an agent to accept a mediator’s suggestions, there must be some way for the agent to verify that the suggestions are reasonable. A mediator might not be willing to share its code with the agents, or be aware of its own memory limitations. Therefore, for a truly robust system, the agents themselves must have a way of checking the mediator’s suggestions.

Thus, this paper introduces a statistical test based on hypothesis testing that, with high probability, can verify the mediator’s suggestions. While hypothesis testing has been proposed in the multi-agent learning literature as a tool that agents might use to learn how to play Nash equilibria [6], to the best of our knowledge it has never been applied for validating a mediator’s advice. Based on our test, we propose an algorithm that allows agents to converge to the mediator’s suggestion if it is a correlated equilibrium and otherwise, in the limit, be no worse off for having used our algorithm. We then generalize this algorithm to a more theoretical setting where we show that with probability one in the limit, our test will always be able to correctly verify the mediator’s suggestions. This provides a method for achieving convergence to a specific correlated equilibrium.

2. BACKGROUND
In this section we introduce the key concepts and assumptions used in this paper.

A n-agent stage game is a tuple $G = (N, A = A_1 \times \ldots \times A_n, u_1, \ldots, u_n)$, where $N = \{1, \ldots, n\}$ is the set of agents, $A_i$ is the set of possible actions for agent $i$ and $A$ is the set of all possible joint actions, and $u_i : A \rightarrow \mathbb{R}$ is the utility function for agent $i$. Without loss of generality, we assume that all utilities are greater than or equal to 0. A specific action for agent $i$ is $a_i \in A_i$, and a joint action is $a = (a_1, \ldots, a_n)$. We assume that $A$ is public knowledge but the agents’ utility functions are private.

Each agent chooses its actions according to some strategy. A strategy for agent $i$, $\sigma_i$, is a probability distribution over $A_i$, stating with what probability the agent will play each possible action. The set of all possible strategies for agent $i$ is $\Sigma_i$. The vector $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a strategy profile which specifies a strategy for each agent and $\Sigma$ is the set of all possible strategy profiles. We use $\sigma_{-i}$ to denote $(\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$.

Given a strategy profile $\sigma$, we define the expected utility for agent $i$ as

$$u_i(\sigma) = \sum_{a=(a_1, \ldots, a_n) \in A} u_i(a)\Pi_{j=1}^{n} \sigma_j(a_j).$$

(1)

Each agent’s utility is dependent not just on its own actions, but also on the actions taken by all other agents. We assume agents are rational, i.e., given $\sigma_{-i}$, agent $i$ will choose a strategy which
maximizes its expected utility.

In our model we introduce a third-party mediator, \( M \). The mediator knows the utility functions for all agents, but is not affected by the game’s outcome. Instead \( M \) makes suggestions to each agent as to what action it should take, where these suggestions are instantiations of a correlated strategy.

**Definition 1.** A correlated strategy, \( \sigma_A \), is a probability distribution over \( A \). We let \( s \in A \) denote an instantiation of \( \sigma_A \).
The conditional correlated strategy \( \sigma_{A,s} \) is the conditional probability of the joint signal \((s_1,s_2,...)\) given the signal \( s_n \), and \( \sigma_{A,s} \) is the set of all conditional probabilities given \( s_n \).

Note that \( \sigma \) is a probability distribution over \( A \) while \( \sigma_A \) is a probability distribution over \( A \).

We assume that \( M \)’s correlated strategy is public knowledge, but the actual instantiation \( s_n \) is not. In particular we assume that \( M \) sends each agent \( i \) a private signal \( s_n \) based on \( s_n \).

The agents are under no obligation to follow the mediator’s signals. It is up to the mediator to pick a correlated strategy that a rational agent would be willing to follow. Note that our type of a mediator is different than Monderer and Tannenholz’s, where agents must agree to follow the mediator’s suggested actions before knowing what they are [15].

**Definition 2.** A correlated strategy \( \sigma_A^* = \{ \sigma_A(a) | a \in A \} \) is a correlated equilibrium if for every agent \( i \) and every \( s_i \in A_i \),

\[
\sum_{s_{-i} \in A_{-i}} \sigma_{A,s_i}^* (s_{-i}|s_i) u_i(s_i,s_{-i}) \geq \sum_{s_{-i} \in A_{-i}} \sigma_{A,s_i}^* (s_{-i}|s_i) u_i(s_i',s_{-i}),
\]

for all \( a_i' \in A_i \) [11]. The set of all correlated equilibria in \( G \) is \( C(G) \).

If all of agent \( i \)’s opponents are following a correlated equilibrium \( \sigma_A^* \), it is rational for agent \( i \) to also follow \( \sigma_A^* \).

In this paper, we are interested in a setting where agents have the ability to learn and adapt to the actions taken by others. Thus, we study repeated games. A repeated game \( G' = (G^1, G^2, ...) \) is an infinite sequence of the stage game \( G \) played repeatedly. Agent \( i \)’s action at time \( t \) is \( a_i^t \) and the joint action at time \( t \) is \( a^t \). The history of joint actions, \( \text{hist}(t) = \{a_1^1, \ldots, a_i^{t-1}\} \), is a record of the joint action taken at each iteration until time \( t \). The empirical, or observed, percentage of play of joint actions, \( \sigma_{hist(t)}^A \), is the percentage of time each joint action has been played as of time \( t \). Agents may learn from previous iterations of the game to try and improve their strategy. Specifically, we assume that agent \( i \) has a learning algorithm \( L_i : \text{hist}(t) \rightarrow \Sigma_i \), that helps agent \( i \) select a strategy for time \( t \).

Let \( \sigma_A^i \) be the actual correlated strategy at time \( t \), i.e. the one agents are actually using and not necessarily the one based on \( M \)’s suggestions. We say that \( \sigma_A^i \) converges to a correlated equilibrium if for some \( \sigma_A^* \in C(G) \), \( \lim_{t \to \infty} \sigma_A^i = \sigma_A^* \). Thus, our algorithm is differentiated from algorithms that achieve convergence to the set of correlated equilibrium for example [5, 8, 12].

3. Setup

The setting for our paper is a repeated game \( G' \) with a mediator, \( M \). As illustrated for the two agent case in Figure 1, time \( t \) will begin with the mediator giving each agent a suggested action, \( s_i^t \). Agents will then simultaneously choose their action, \( a_i^t \), which may or may not be \( s_i^t \). If agent \( i \) chooses not to follow \( M \)’s signal, it can instead use a learning algorithm, \( L_i \), which we assume is independent of \( M \)’s signals, to select an action. Based on the actual joint action, each agent will then receive some utility and the process repeats. The mediator’s signal to each agent is private information, known only to that agent and the mediator, as is the agent’s utility function. However, the action set for each agent is public knowledge, as is the action taken by each agent during a turn.

The mediator’s signals are based on a selected correlated strategy, \( \sigma_A^M \), which is constant throughout the repeated game. Although ideally the mediator will suggest a correlated strategy that is also a correlated equilibrium, each agent still needs to verify that the mediator has actually done so.

Our aim is to design an algorithm that achieves the following goals.

**First Goal:** If \( \sigma_A^M \) is a correlated equilibrium then \( \sigma_A^i \), the actual correlated strategy which is not necessarily \( \sigma_A^M \), will converge to \( \sigma_A^M \).

**Second Goal:** If \( \sigma_A^M \) is not a correlated equilibrium, agents should be no worse off, in the limit, for having used our algorithm.

In Section 4, we present an algorithm, \( \Lambda \), that achieves these goals with high probability. In Section 5, we generalize \( \Lambda \) so that, with probability one, in the limit, it will achieve both goals. Since each agent will be using \( \Lambda \) independently, we refer to \( \Lambda_i \) as the instance of the algorithm being run by agent \( i \) and \( \Lambda \) as the joint algorithm.

The algorithm is based on the concept of giving \( M \) the benefit of the doubt; until there is reason to believe otherwise, agents assume that \( \sigma_A^M \) is a correlated equilibrium and follow \( M \)’s signals. Specifically, agents will assume that the following conditions hold.

**Condition 1:** The correlated strategy \( \sigma_A^M \) is a correlated equilibrium.

**Condition 2:** All other agents are following the signals based on \( \sigma_A^M \).

Figure 1: A graphical representation of our setting with 2 agents at time \( t \).
Agents test whether these conditions hold during an initial period of play called a sampling test which has a fixed length of \( l_T \). If, at the beginning of the sampling test, agent \( i \) decides that one of the conditions does not hold, it will not follow \( M \)'s signals and instead will use an individual “fall-back” strategy, \( \gamma_i \), chosen uniformly at random. At the end of the sampling test, all agents who still believe that both conditions hold will continue to follow \( M \)'s signals. All other agents will start using their original learning algorithm. The algorithm \( \Lambda \) is correct if and only if, at the end of the sampling test, it correctly determines whether both conditions hold. The joint algorithm, \( \Lambda \), is correct if and only if \( \Lambda_i \) is correct for all \( i \).

4. THE INITIAL ALGORITHM

In this section, we describe how our initial algorithm works. As a first step in \( \Lambda \), agent \( i \) will check to see if Equation 2 holds for all \( s_i \in A_i \). Note that the only information agent \( i \) needs to know in order to check if Equation 2 holds is its own utility function, and \( M \)'s announced correlated strategy, \( \sigma_M^A \), which we assume is publicly announced. If Equation 2 does not hold, agent \( i \) will know that Condition 1 cannot be true. In this case, agent \( i \) will use a “fall-back” strategy, \( \gamma_i \in \Sigma_i \), picked uniformly at random, for the rest of the sampling test. If Equation 2 does hold, agent \( i \) must check to see if Condition 2 is true and will continue to follow \( M \)'s signals throughout the sampling test.

Since the utilities for each agent, as well as the signals they receive each turn, are private, there may be no way to prove or disprove Condition 2 with absolute certainty at any finite point during the game. The best \( \Lambda \) can do is reach a probabilistic conclusion. Since joint actions are public knowledge, \( \Lambda \) can compare the empirical percentages of play for the duration of the sampling test against the percentages predicted by \( \sigma_M^A \). If the difference between these two values is statistically significant, there is a high probability that at least one agent has stopped following the mediator’s signals.

To test if there is a difference, agent \( i \) assumes there is some fixed but unknown correlated strategy \( \tilde{\sigma}_A \) that all agents were actually using for the sampling test, where \( \tilde{\sigma}_A \) may or may not be \( \sigma_M^A \). We are now able to use hypothesis testing, where our null hypothesis is that \( \sigma_M^A \) is equal to \( \tilde{\sigma}_A \), i.e.,

\[
H_0 : \sigma_M^A = \tilde{\sigma}_A,
\]

and our alternative hypothesis is that \( \sigma_M^A \) is not equal to \( \tilde{\sigma}_A \), i.e.,

\[
H_1 : \sigma_M^A \neq \tilde{\sigma}_A.
\]

The test statistic used is Pearson’s \( \chi^2 \) test,

\[
T = \sum_{a \in A_i} \frac{(X(a) - E(a))^2}{E(a)},
\]

where \( A_i \) is any subset of \( A \) such that \( |A'| = |A| - 1 \), \( X(a) = l_T \sigma_M^{A\setminus\{a\}}(a) \) is the actual frequency of play of \( a \in A' \) during the sampling test, \( E(a) = l_T \sigma_M^A(a) \) is the expected frequency of play according to \( \sigma_M^A \), and where \( l_T \) is the length of the sampling period [16]. Note that \( \sigma_M^{A\setminus\{a\}}(a) \) is based on a sampling from \( \tilde{\sigma}_A \) of size \( l_T \). For now we assume that \( \sigma_M^{A\setminus\{a\}}(a) > 0 \) for all \( a \in A \). We relax this assumption later. The Pearson’s \( \chi^2 \) test has (in the limit) a probability distribution function of

\[
\chi^2_2 + \chi^2_{NCP,1},
\]

where the first distribution has \( df = |A| - 2 \) degrees of freedom, and the second distribution has 1 degree of freedom and a non-centrality parameter of \( NCP \) [13].

If \( H_0 \) is true, \( NCP = 0 \). Assuming that \( H_0 \) is true, we choose a significance level for rejection of the null hypothesis of \( \alpha < 1 \) and a corresponding critical value of \( c(\alpha) \), i.e., we reject the null hypothesis when \( T \geq c(\alpha) \). In this case, the probability of incorrectly rejecting \( H_0 \) (known as a Type 1 error) is \( p_1 = \alpha \). If \( H_1 \) is actually true, we err when \( T < c(\alpha) \) and we do not reject \( H_0 \) (a Type 2 error). When \( H_1 \) is true, \( NCP > 0 \). Since the non-centrality parameter determines how much the probability distribution in Equation 6 gets adjusted, determining \( NCP \) helps determine the probability of a Type 2 error.

The equation for \( NCP \) is \( NCP = t \cdot \delta \), where \( \delta \), the sensitivity parameter, is a measure of the difference between \( \sigma_M^A \) and \( \tilde{\sigma}_A \) given by

\[
\delta(\sigma_M^A, \tilde{\sigma}_A) = \frac{\sum_{a \in A} (\tilde{\sigma}_A(a) - \sigma_M^A(a))^2}{\sigma_M^A(a)}.
\]

For a given value of \( \delta \), say \( \delta^* \), if

\[
\delta(\sigma_M^A, \tilde{\sigma}_A) \geq \delta^*,
\]

then the probability of a Type 2 error is bounded by some value \( \beta(\delta^*) < 1 \), whose value is normally found via numerical computation [13]. Since \( \beta \) is also a function of \( l_T \) and \( \alpha \), we refer to it as \( \beta(l_T, \alpha, \delta) \).

Since agents do not know whether their opponents are following the mediator’s suggestions, agents do not know the exact value for \( \tilde{\sigma}_A \), and therefore, it is impossible choose an appropriate value for \( \delta \) so that Equation 8 is guaranteed to hold. Instead, agents can take a different approach by asking what is the worst case situation under which Equation 8 does not hold. To answer this question, consider the set of all agents for whom Equation 2 does not hold, \( N_B \subseteq N \). Let \( (\sigma_{A\setminus N'}, \gamma_N) \) be the actual correlated strategy for the duration of the sampling test, i.e., a combination of those agents who will follow \( M \)'s signals and those who will rely on their fall-back strategy. Let \( \Sigma_N \) be the set of all possible joint strategies for agents in \( N_B \), and

\[
\Sigma_{N_B} (\sigma_M^A, \delta) = \{ \xi \in \Sigma_N : \delta(\sigma_M^A, (\sigma_{A\setminus N'} \gamma_N)) < \delta \}
\]

be the set of all possible joint strategies for agents in \( N_B \) which would result in Equation 8 not holding. Let \( \mu(\Sigma_N) \) and \( \mu(\Sigma_{N_B} (\sigma_M^A, \delta)) \) be the Lebesgue measures of \( \Sigma_N \) and \( \Sigma_{N_B} (\sigma_M^A, \delta) \), respectively. Then, since \( \gamma_N \) is chosen uniformly at random, the probability of \( \sigma_{N_B} \) being in \( \Sigma_N \) is

\[
\psi(\Sigma_N) = \frac{\mu(\Sigma_N)}{\mu(\Sigma_{N_B} \sigma_M^A, \delta))}.
\]

Since agents do not know \( N_B \), they consider the worst case scenario,

\[
\psi = \max_{N_B \subseteq N} \psi(\Sigma_N).
\]

If we assume that whenever Equation 8 does not hold and \( \tilde{\sigma}_A \neq \sigma_M^A \), a Type 2 error is always made, then the probability of a Type 2 error is at most

\[
p_2 \leq (1 - \psi) \cdot \beta(\delta) + \psi.
\]

That is, Equation 8 holds with at least a probability of \( \psi \) and when it does, the probability of a Type 2 error is at most \( \beta(\delta) \) and with a probability of at most \( \psi \), Equation 8 does not hold.
If we do not assume that \( \sigma_M^A(a) > 0 \) for all \( a \in A \), then Equations 5 and 7 may contain division by zero. To deal with this, we ignore all \( a \in A \) such that \( \sigma_M^A(a) = 0 \). If \( \zeta = \{ a \in A | \sigma_M^A(a) = 0 \} \), then the summations in Equations 5 and 7 need to exclude all \( a \in \zeta \), and \( df \) in Equation 6 now equals \( |A| - |\zeta| \). If the null hypothesis is correct then \( \sigma_M^A(a) = 0 \) implies that \( \sigma_M^{hist}(r)(a) = 0 \) for all \( a \in \zeta \). Alternatively, if there exists \( a' \in A \) such that \( \sigma_M^{hist}(r)(a') > 0 \) while \( \sigma_M^A(a') = 0 \), the alternative hypothesis must be correct. Hence, both of these cases do not present problems.

The only other case is if for all \( a \in A \) such that \( \sigma_M^A(a) = 0 \), \( \sigma_M^{hist}(r)(a) = 0 \) but, unknown to the agents, the alternative hypothesis is correct. In this case, a Type 2 error may occur. To find the probability of this case happening, we first determine the probability of \( a' \in \zeta \). Since any agent who rejects \( M \)'s suggested strategy chooses its new strategy uniformly at random, the probability, \( \mathcal{P} \), that \( a' \in \zeta \) for \( t \leq l_T \) is

\[
\mathcal{P} \geq \sum_{N \subseteq N} \min_{a \in \zeta} \sigma_{A_N}^{hist}(a) \frac{1}{|A_N|},
\]

where \( \min_{N \subseteq N} \) is considered since agents do not know \( N_H \). Therefore, the probability that \( a' \notin \zeta \) for all \( t \leq l_T \) is at most \( (1 - \mathcal{P})^r \). Therefore, the overall probability of a Type 2 error is at most

\[
p_2 \leq (1 - \mathcal{P})^r \cdot [1 - (1 - \psi) \cdot \beta + \psi].
\]

To accommodate the worst case, we assume equality holds in Equation 14. Note that \( p_1 \) has not changed. For simplicity, we assume that \( p_1 = p_2 = p \), and refer to \( p \) as the overall probability of error.

It is possible to rearrange \( \beta(l_T, \alpha, \delta) \) to express \( l_T \) as a function of \( \alpha, \beta \) and \( \delta \), i.e. \( l_T(\alpha, \beta, \delta) \). As a result, \( l_T \) is the sample size needed to perform the test with at most a probability of error (of either Type 1 or Type 2) of \( p \).

If all agents are to use the same value for \( l_T \), they must also have the same value for \( \beta \). This in turn requires them to have the same value for \( \psi \). To achieve this, in Equations 11 and 13, agent \( i \) will consider all possible \( N' \), including those containing agent \( i \). Although we require all agents to use the same value for \( l_T \), as well as \( p \) and \( \delta \), in the long run, no agent is harmed by doing so. This is because the sampling test is of a fixed length.

### 4.1 Examples

In this section we provide two examples to illustrate how our test would work.

**Example 1:** Consider the game in Figure 2.

**Figure 2:** A simple game

<table>
<thead>
<tr>
<th>Agent 1</th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a_2,1 )</td>
</tr>
<tr>
<td>( a_{1,1} )</td>
<td>0.1</td>
</tr>
<tr>
<td>( a_{1,2} )</td>
<td>5.2</td>
</tr>
</tbody>
</table>

Suppose that \( M \) announces a correlated strategy, \( \sigma_M^A \), as shown in Figure 3. Note that \( \sigma_M^A \) is a correlated equilibrium.

Suppose the agents choose \( p = 0.1 \) and \( \delta = 0.01 \). Agents must now determine the critical value for rejection, \( c(\alpha) \), and the length of the sampling test, \( l_T \). Since \( p_1 = \alpha, \alpha = 0.1 \). For 3 degrees of freedom, \( c(\alpha) = 6.25 \). Since \( \sigma_M^A(a) > 0 \) for all \( a \), we can calculate \( \beta \) by Equation 12. We calculate Equation 11 by numerical computation to find \( \psi \approx 0.00429 \). Therefore, \( \beta = 0.0063 \). In practice, \( l_T(\alpha, \beta, \delta) \) would now be solved by some method of numerical computation [13]. For simplicity, we used the tables in Cohen to obtain a value of \( l_T = 2100 \) [3].

Suppose after 2100 iterations, we have obtained an empirical frequency of play as shown in Figure 4. Using Equation 5, we obtain a test statistic value of 4.678. Since this is lower than the critical value, both agents do not reject the null hypothesis and continue to use \( M \)'s signals.

**Figure 3:** \( M \)'s correlated strategy

<table>
<thead>
<tr>
<th>Agent 1</th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a_2,1 )</td>
</tr>
<tr>
<td>( a_{1,1} )</td>
<td>1/18</td>
</tr>
<tr>
<td>( a_{1,2} )</td>
<td>2/18</td>
</tr>
</tbody>
</table>

**Figure 4:** A possible empirical frequency of play

**Example 2:** Consider a different example based on the same game where \( M \) announces a correlated strategy, \( \sigma_M^A \), as shown in Figure 5. In this case, \( \sigma_M^A \) is not a correlated equilibrium. Specifically, while Equation 2 is satisfied for Agent 1, it is not satisfied for Agent 2. Hence, Agent 2 will use a random fall-back strategy. Suppose \( \gamma_2 = (3/4, 1/4) \).

**Figure 5:** \( M \)'s correlated strategy for the second example

<table>
<thead>
<tr>
<th>Agent 1</th>
<th>Agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a_2,1 )</td>
</tr>
<tr>
<td>( a_{1,1} )</td>
<td>2/18</td>
</tr>
<tr>
<td>( a_{1,2} )</td>
<td>1/18</td>
</tr>
</tbody>
</table>

**Figure 6:** A possible empirical frequency of play for Example 2

Note that, as we have stated our algorithm, Agent 1 will only know that there is a probability of at most 0.1 of incorrectly rejecting the
null hypothesis. We have not accounted for the fact that the test statistic value is much higher than the critical value. An additional test that could be run after the null hypothesis is rejected is the calculation of the p-value. The p-value is the smallest α value that would still allow us to reject the hypothesis [16]. In the case of the above example, the p-value would be very small, and Agent 1 could be very certain that the above example, the game, agents will use multiple iterations of repeated testing.

For example, we can let $R_i$ be the set of repeated sampling tests is $R_i = \{b_{R_i}, l_{R_i}\}$, $b_{R_i}$ is the first time period in $R_i$, and $l_{R_i}$ is the length of $R_i$. The instance of $\Lambda_i$ during test $R_i$ is denoted by $\Lambda_{R_i}$. The repeated tests are not contiguous. A simple example is shown in Figure 7, where the timeline represents a repeated game up to 7 iterations. The grey areas represent sampling test iterations. For example, $R_2 = \{b_{R_2}, l_{R_2}\} = \{4, 2\}$, meaning that the second test iteration begins at time period 4 and lasts for 2 iterations of the repeated game.

The parameters, $\delta$ and $p$, can be set to depend on the test iteration, i.e. $\delta(R_j)$ and $p(R_j)$. Each test period must be identical for each agent, i.e. $R_j$ must be the same for all agents. This means that $\delta(R_j)$ and $p(R_j)$ must be the same for all agents. The parameters are chosen such that

$$\lim_{j \to \infty} \delta(R_j) = 0,$$

$$\sum_{j=1}^{\infty} p(R_j) < \infty.$$  

For example, we can let $\delta(R_j) = 1/j$ and $p(R_j) = 1/2^j$. Finally, we assume that each agent's fall-back strategy is fixed. That is $\gamma_{R_j}^i = \gamma_{R_j}^i$, for all $j, j'$. Our first result is that an agent will not draw the wrong conclusion about the mediator too often.

**Theorem 1.** In the limit, with probability one, there will only be a finite number of tests where $\Lambda_{R_j}$ is incorrect.

**Proof.** Let $\sigma^M_i$ be the correlated strategy suggested by $M$. Consider the following two cases:

**$\sigma^M_i$ is a correlated equilibrium:** For test $R_j$, the probability of $\Lambda_{R_j}^i$ making a Type 1 error, $p_1(R_j)$, is equal to $p(R_j)$. By the Borel-Cantelli lemma, with probability one, there will only be a finite number of times $\Lambda_{R_j}^i$ is incorrect, i.e. makes a Type 1 error.  This reasoning can be applied to all agents, and therefore with probability one there will only be a finite number of times $\Lambda_{R_j}$ is incorrect.

**$\sigma^M_i$ is not a correlated equilibrium:** If $\sigma^M_i$ is not a correlated equilibrium, then some subset of agents, $N \subseteq N$, will use their fall-back strategies instead of following the mediator’s signals. The resulting correlated strategy for every test iteration will be $(\sigma^M_{N \setminus R_j}, \gamma_{N \setminus R_j})$.

Since $\gamma_{N \setminus R_j}$ is fixed, by Equation 15, there exists a finite $j^*$ such that for all $j \geq j^*$,

$$\delta(\sigma^M_{N \setminus R_j}, (\sigma^M_{N \setminus R_j}, \gamma_{N \setminus R_j})) \geq \delta(R_j).$$  

Let $\psi(R_j)$ be the value of $\psi$, according to Equation 11, during the sampling test $R_j$. Starting at $R_j$, we know that, with probability one, Equation 8 holds and therefore, since $\psi(R_j)$ is the probability of Equation 8 not holding, $\psi(R_j) = 0$, for all $j \geq j^*$. Therefore, the probability of a Type 2 error starting at $R_j$ is

$$p_2 = \sum_{j^*}^\infty (1 - \psi)^j \beta.$$  

Note that $\psi, l_T$ and $\beta$ are all functions of $R_j$, however we omit the notation $(R_j)$ for clarity. Since $\beta$ is less than 1,

$$p_2 \leq \sum_{j^*}^\infty (1 - \psi)^j [(1 - \psi) \cdot \beta + \psi]$$

$$= \sum_{j^*}^\infty p(R_j),$$

where $\psi$, as calculated by Equation 11, is also a function of $R_j$. Therefore, by Equation 16 and the Borel-Cantelli lemma, with probability one, there will only be a finite number of times $\Lambda_{R_j}$ is incorrect, i.e. makes a Type 2 error. Again, this reasoning can be generalized to all agents and therefore, there will only be a finite number of times $\Lambda_{R_j}$ is incorrect.  

We now examine the behaviour of agents between sampling tests. The periods between test iterations are called free periods. The set of free periods is $F = \{F_1, \ldots\}$ where $F_j = \{b_{F_j}, l_{F_j}\}$. Thus $G^* = \{R_1, F_1, R_2, F_2, \ldots\}$. For example, in Figure 7, the first free period, $F_1$, would be $\{b_{R_1}, l_{R_1}\} = \{2, 2\}$. If $\Lambda_{R_j}$ did not reject the null hypothesis, agent $i$ continues to follow $M$’s signals for all of $F_j$. If $\Lambda_{R_j}$ did reject the null hypothesis, agent $i$ relies on its learning algorithm $L_i$ for $F_j$. We assume that $L_i$ is flexible at the beginning of each free period [2, 4].

**Definition 3.** The learning algorithm $L_i$ is flexible if at the beginning of every free period $F_j$,

$$L_i(\text{hist}(b_{F_j})) = L_i(\text{hist}(1)).$$  

**Borel-Cantelli Lemma:** Let $\{E_i\}_{i=0}^\infty$ be a sequence of independent events and $P(E')$ is the probability of the event $E'$ occurring. If $\sum_{i=0}^\infty P(E_i) < \infty$, then with probability one, only a finite number of the events will occur.
Therefore, during each free period, $L_i$ does not base its actions on what has happened before time $b_j$. For example, $L_i$ may be a trigger strategy, but that trigger may not be based on anything that has happened in a previous sampling test or free period.

We require that
\[
\lim_{j \to \infty} \sum_{m=1}^{j} \frac{1}{l_{F_m}} = 0, \quad (22)
\]
for example $l_{F_m} = \frac{1}{m}$. This means that, in the limit, the length of the sampling periods is negligible compared to the length of the free periods. We also require that
\[
\lim_{j \to \infty} \frac{l_{B_j}}{j} = \infty. \quad (23)
\]
This means that the length of the sampling tests grows at faster than a linear rate. The specific values for $l_{R_i}$ and $l_{F_j}$ would have to be agreed upon by all agents.

**DEFINITION 4.** Let $\theta^{a_{(t_1, t_2)}}_A$ be the expected frequency of play from time $t_1$ to $t_2$, i.e., the expected number of times each joint action $a \in A$ gets played between times $t_1$ and $t_2$ inclusive. If $t_1$ is not given, we assume $t_1 = 1$. Similarly, let $\theta^{F_j \ldots F_{j'}}_A$ be the expected frequency of play during the free periods $F_j$ through $F_{j'}$, inclusive.

Since the frequency of play depends on the algorithms the agents are using, let $\theta^{a_{(t)}}_A(L)$ be the expected frequency of play from time $t$ to $t$ assuming that agents use the joint learning algorithm $L$ for the whole period.

For simplicity in all of the following proofs, we assume that $t$ always corresponds to the beginning of a sampling period. Let $j(t)$ be the index of the last free period before $t$.

The first step is to show that if $\mathcal{M}$ suggests a correlated equilibrium, agents will converge to it.

**THEOREM 2.** If the correlated strategy suggested by $\mathcal{M}$, $\sigma^M$, is a correlated equilibrium, then with probability one,
\[
\lim_{t \to \infty} \theta^{a_{(t)}}_A = \sigma^M_A. \quad (24)
\]

**PROOF.** If $\sigma^M_A$ is a correlated equilibrium then by Theorem 1, with probability one, after some finite point $\Lambda$ will always correctly determine that $\sigma^M_A$ is a correlated equilibrium. As a result, with probability one, after some finite point, all agents will choose to follow the mediator’s signals during the free periods. □

Our next result is a technical lemma which shows that in the limit, agents are not harmed by taking time out to do the sampling tests.

**LEMMA 1.** In the limit, there is no difference between the average utility from agents using $L$ for the whole repeated game and just for the free periods, i.e.,
\[
\lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right) - u_i \left( \frac{\theta^{a_{(t)}}_A(F_j \ldots F_{(j+1)})}{t} \right) \right] = 0. \quad (25)
\]
Furthermore, this is true even when excluding the first $j^* = 1$ free periods, for some $j^* > 1$, i.e.,
\[
\lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right) - u_i \left( \frac{\theta^{a_{(t)}}_A(F_j \ldots F_{(j+1)})}{t} \right) \right] = 0. \quad (26)
\]

The proof is given in the Appendix.

Finally, we need to show that if $\sigma^M_A$ is not a correlated equilibrium, agents are no worse off, on average, for having used $\Lambda$.

**THEOREM 3.** If the correlated strategy suggested by $\mathcal{M}$, $\sigma^M_A$, is not a correlated equilibrium, then with probability one,
\[
\lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(\Lambda)}{t} \right) - u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right) \right] \geq 0. \quad (27)
\]
Therefore, in the limit, agent $i$ will be no worse off for using $\Lambda$ instead of $L_i$.

**PROOF.** If $\sigma^M_A$ is not a correlated equilibrium, by Theorem 1, with probability one, starting at some sampling test, say $R_t^*$, $\Lambda$ will always correctly determine that $\sigma^M_A$ is not a correlated equilibrium.

Consider $\theta_A$ with respect to some arbitrary $a \in A$, denoted by $\theta_a$. We start by breaking the game down into the sequence of sampling tests and free periods. That is, $\theta^{a_{(t)}}_A(\Lambda) = \theta^{a_{(t)}}_A(R_j \ldots F_{(t+1)})$. For $t \geq t(j^*)$, the utility can be split up into the utility for the sampling tests and free periods before $R_{j^*}$ and for those starting at $R_{j^*}$ i.e.,
\[
\lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(R_j \ldots F_{(t+1)})}{t} \right) \right] + \lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right) \right] = 0.
\]
Since $\theta^{a_{(t)}}_A(R_j \ldots F_{(t+1)})$ is constant, in the limit, the first term is 0, and so we are interested in
\[
\lim_{t \to \infty} u_i \left( \frac{\theta^{a_{(t)}}_A(R_j \ldots F_{(t+1)}) \Lambda}{t} \right)\]
The expected frequency can be split up into the expected frequency for the sampling periods and for the free periods. Since $\Lambda$ always determines that $\sigma^M_A$ is not a correlated equilibrium, during all the free periods agents will always use $L$, and so we are interested in
\[
\lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(R_j \ldots F_{(t+1)}) \Lambda}{t} \right) \right] + \lim_{t \to \infty} \left[ u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right) \right]\]
Since we assumed that all utilities are nonnegative, we may discard the first term, and thus have
\[
\lim_{t \to \infty} u_i \left( \frac{\theta^{a_{(t)}}_A(L)}{t} \right)\]
Therefore, by Lemma 1, the theorem follows. □

Together, Theorems 2 and 3 show that, with probability one, if $\sigma_a^{M_i}$ is a correlated equilibrium, agents will converge to it and if $\sigma_a^{M_i}$ is not a correlated equilibrium, agents will be no worse off in the long run for using $A$.

6. CONCLUSION

The setting for this paper was a repeated game with a mediator. The mediator makes suggestions to the agents as to what actions to take. We presented a test that agents could use so that, with high probability, they could determine if the mediator’s suggestion was a correlated equilibrium. We then generalized our algorithm to incorporate repeated testing so that in the limit, with probability one, the test will always correctly determine whether the mediator’s suggested strategy is a correlated equilibrium. As a result, if the mediator suggests a correlated equilibrium, then agents will converge to it, and otherwise, be no worse off in the long run for having used our algorithm.

We envision several directions for future research. First, it might be possible to extend our algorithm to work in radically uncoupled environments, where agents are not aware of the existence of others. This would significantly decrease the knowledge requirements of our test. One possible way to generalize our algorithm for a radically uncoupled setting would be to use experimental testing [7, 10]. Second, we would like to extend our approach so that the mediator receives feedback from the agents themselves, which can be used to help select appropriate correlated strategies. We believe that the incentive issues in such an approach will be challenging. It may also be interesting to apply our approach to other solution concepts such as mediated equilibria [15].

In a more applied direction, it might be possible to generalize our approach so it can be used in a stochastic game setting. Thus, our approach could be combined with methods such as Q-learning [11]. Correlated equilibria have also been used in graphical games, which can be used to model many different settings [14]. Hence, applying our technique to graphical games may yield some interesting results. For example, network games use graphical games to help represent a variety of problems, from public good provision and trade to information collection [9]. These models can be hindered by a “fundamental theoretical problem: even the simplest games played on networks have multiple equilibrium[sic] which display a bewildering range of possible outcomes” [9]. Our model may help integrate correlated equilibria as a possible solution to this problem.

7. ACKNOWLEDGEMENTS

Our thanks to Gord Hines for his statistical advice.

8. REFERENCES


APPENDIX

A. PROOF OF LEMMA 1

Proof: Consider $\theta$ with respect to $a \in A$, denoted by $\theta_a$. Since $j^*$ is fixed, $\theta_a^{x_1,\ldots,x_j^*+1}(L)$ is constant, and therefore,

$$\lim_{t \to \infty} \frac{\theta_a^{x_1,\ldots,x_j^*+1}(L)}{t} = 0,$$

and therefore, Equations 25 and 26 are equivalent.

Since the utility functions are linear transformations, proving the following is sufficient, although not necessary, to prove that Equation 25 holds,

$$\lim_{t \to \infty} \frac{\theta_a^{x_1,\ldots,x_j^*+1}(L)}{t} = 0.$$

Since $L$ is flexible, it will, in expectation, always behave the same way during each free period. Specifically,

$$\theta_a^{x_1,\ldots,x_j^*+1}(L) = \theta_a^{x_1,\ldots,x_j^*+1}(L),$$

for all $j'$ such that $t_{j'} \geq t_{j^*}$. This relationship can be represented graphically, as shown in Figure 8, where for simplicity, we let $w(j) = \exp(b_{j-1} + b_{j} + 1)$, where $t_{j0} = 0$. Therefore,

$$\theta_a^{x_1,\ldots,x_j^*+1}(L) = \sum_{j=1}^{j^*} (j(t) - j + 1)\theta_a^{w(j)}(L).$$
Figure 8: A graphical representation of how the expected frequency of play will be repeated each free period.

Note that \( \theta^{w(j)}_a \) will be “represented” more than \( \theta^{w(j')}_a \) for \( j < j' \) and any finite \( t \). In order for Equation 29 to hold, in the limit, all \( \theta^{w(j)}_a \) must be represented equally, i.e.

\[
\lim_{t \to \infty} \frac{j(t) - j + 1}{t} = \lim_{t \to \infty} \frac{j(t) - j' + 1}{t},
\]

for all \( j, j' \). Consider \( t(j) = j^{-1}(t) \), i.e. the first time index after the \( j^{th} \) free period has ended:

\[
t(j) = \sum_{j' = 1}^j (l_{R_j} + l_{F_j}) \geq \sum_{j' = 1}^j l_{R_j}.
\]

By Equation 23, \( \lim_{j \to \infty} \frac{t(j)}{j} = \infty \), and therefore,

\[
\lim_{t \to \infty} \frac{j(t) - j + 1}{t} \leq \lim_{t \to \infty} \frac{j(t)}{t} = 0.
\]

Therefore, in the limit, all \( \theta^{w(j)}_a \) will be represented equally. However, since \( \sum_{j=1}^j l_{F_j} < t \), each \( \theta^{w(j)}_a \) will be “underrepresented” compared to \( \theta^a(L) \) for any finite \( t \). However, in the limit, this is not the case since,

\[
\lim_{t \to \infty} \frac{\sum_{j=1}^j l_{F_j}}{t} = \lim_{t \to \infty} \frac{\sum_{j=1}^j l_{F_j}}{\sum_{j=1}^j (l_{R_j} + l_{F_j})}
\]

\[
= \lim_{t \to \infty} \frac{1}{\sum_{j=1}^j l_{R_j} + 1}
\]

\[
= 1 \text{ (by Equation 22)}. \quad (34)
\]

Therefore, in the limit \( \theta^{w(j)}_a \) will be represented equally compared to \( \theta^a(L) \). \( \square \)
Norm emergence with biased agents

Partha Mukherjee  
MCS Department  
Univ. of Tulsa  
partha-mukherjee@utulsa.edu

Sandip Sen  
MCS Department  
Univ. of Tulsa  
sandip-sen@utulsa.edu

Stéphane Airiau  
MCS Department  
Univ. of Tulsa  
Stéphane@utulsa.edu

ABSTRACT

Effective norms can significantly enhance performance of individual agents and agent societies. Previous research have studied social learning of self-enforcing behavioral norms from interaction experiences. In this model, individual agents repeatedly interact with other agents in the society over instances of a given scenario. Each interaction is framed as a stage game. An agent learns its policy to play the game over repeated interactions with multiple agents. We term this mode of learning social learning, which is distinct from an agent learning from repeated interactions against the same player. We are particularly interested in situations where multiple action combinations yield the same optimal payoff. The key research question is to find out if the entire population learns to converge to a consistent norm. While previous research assume agents have no prior biases to any of its actions, in real-life agents may have pre-formed biases or preferences which may hinder or even preclude norm emergence. We study the success and speed of norm emergence when different subsets of the population have different initial biases. In particular we characterize the relative speed of norm emergence under varying biases and the success of majority/minority groups in enforcing their biases on the rest of the population given different bias strengths.

1. INTRODUCTION

Norms or conventions routinely guide the choice of behaviors in human societies. Conformity to norms reduces social frictions, relieves cognitive load on humans, and facilitates coordination. “Everyone conforms, everyone expects others to conform, and everyone has good reason to conform because conforming is in each person’s best interest when everyone else plans to conform” [10]. Conventions in human societies range from fashions to tipping, driving etiquette to interaction protocols. Norms are ingrained in our social milieu and play a pivotal role in all kinds of business, political, social, and personal choices and interactions. They are self-enforcing: “A norm exists in a given social setting to the extent that individuals usually act in a certain way and are often punished when seen not to be acting in this way” [1].

While these aspects of norms or conventions have merited in-depth study of the evolution and economics of norms in social situations [6, 14, 21, 22], we are particularly interested in the following characterization: “... we may define a convention as an equilibrium that everyone expects in interactions that have more than one equilibrium.” [22]. This observation has particular significance for the study of norms in the context of computational agents. Computational agents often have to coordinate their actions and such interactions can be formulated as stage games with simultaneous moves made by the players [9]. Such stage games often have multiple equilibria [12], which makes coordination uncertain. While focal points [16] can be used to disambiguate such choices, they may not be available in all situations. Norms can also be thought of as focal points evolved through learning [22]. Hence, the emergence of norms via learning in agent societies promises to be a productive research area that can improve coordination in and hence functioning of agent societies.

While researchers have studied the emergence of norms in agent populations, they typically assume access to significant amount of global knowledge [6, 14, 21, 22]. For example, all of these models assume that individual agents can observe sizable fraction of interactions between other agents in the environment. While these results do provide key insights into the emergence of norms in societies where the assumption of observability holds, it is unclear if and how norms will emerge if all interactions were private, i.e., not observable to any other agent not involved in the interaction.

To study the important phenomenon of emergence of social norms via private interactions, we use the following interaction framework. We consider a population of agents, where, in each interaction, each agent is paired with another agent randomly selected from the population. Each agent then is learning concurrently over repeated interactions with randomly selected members from the population. We refer to this kind of learning social learning to distinguish from learning in iterated games against the same opponent [7]. Our experiments involve symmetrical games with multiple pure-strategy equilibria with the same payoff.

1Conventions can therefore be substituted as external correlating signals to promote coordination.

2Henceforth we use the term norm to refer to social norms and conventions.

in: Proceedings of ALAMAS&ALAg, Klügl, Tuyls, Sen (eds.)  
Workshop at AAMAS 2008, May, 12th, Estoril, Portugal
In previous work on learning in games, the opponent is fixed. In our social learning framework [11, 17], the opponent is unknown and different at each iteration. In past work [11, 17], we have considered a society of agents where opponents use the same learning algorithm, but we assume that agents did not have an initial bias to play a particular policy. We have also studied the effect of non-learning agents playing a fixed strategy, and we observed that few agents were able to bias the emergence of one norm. In both cases, we observe that a norm always emerges when the learning agents’ policy is initially set with a uniform probability distribution. In this paper, our goal is to provide experimental results that throw light on the dynamics of the emergence of norms by studying the effect of an initial bias in the policy of the learners. For example, do we observe the convergence of a norm in a 2-action game when the policy of 50% of the agent is initially set to play action 1 and the policy of the other agent is initially set to play action 2? We investigated a number of related issues: the effect of population size playing with an initial bias, agents playing with varying initial biases, multiple populations with opposite policies, multiple populations with bias differentials etc.

2. RELATED WORK

The need for effective norms to control agent behaviors is well-recognized in multiagent societies [3, 20]. In particular, norms are key to the efficient functioning of electronic institutions [8]. Most of the work in multiagent systems on norms, however, has centered on logic or rule-based specification and enforcement of norms [5, 20]. Similar to these research, the work on normative, game-theoretic approach to norm derivation and enforcement also assumes centralized authority and knowledge, as well as system level goals [2, 3]. While norms can be established by centralized dictat, a number of real-life norms evolve in a bottom-up manner, via “the gradual accretion of precedent” [22]. We find very little work in multiagent systems on the distributed emergence of social norms. We believe that this is an important niche research area and that effective techniques for distributed norm emergence based on local interactions and utilities can bolster the performance of open multiagent systems. We focus on the importance for electronic agents solving a social dilemma efficiently by quickly adopting a norm. Centralized social laws and norms are not sufficient, in general, to resolve all agent conflicts and ensure smooth coordination. The gradual emergence of norms from individual learning can facilitate coordination in such situations and make individuals and societies more efficient.

In our formulation, norms evolve as agents learn from their interactions with other agents in the society using multiagent reinforcement learning algorithms [13, 19]. Most multiagent reinforcement learning literature involves two agents iteratively playing a stage game and the goal is to learn policies to reach preferred equilibrium [15]. Another line of research considers a large population of agents learning to play a cooperative game where the reward of each individual agent depends on the joint action of all the agents in the population [18]. The goal of the learning agent is to maximize an objective function for the entire population, the world utility.

The social learning framework we use to study norm emergence in a population is somewhat different from both of these lines of research. We are considering a potentially large population of learning agents. At each time step, however, each agent interacts with a single agent, chosen at random, from the population. The payoff received by an agent for a time step depends only on this interaction as is the case when two agents are learning to play a game. In the two-agent case, a learner can adapt and respond to the opponent’s policy. In our framework, however, the opponent changes at each interaction. It is not clear a priori if the learners will converge to useful policies in this situation.

3. SOCIAL LEARNING FRAMEWORK

The specific social learning situation for norm evolution that we consider is that of learning “rules of the road”. In particular, we will consider the problem of which side of the road to drive in and who yields if two drivers arrive at an interaction at the same time from neighboring roads. We will represent each interaction between two drivers as a two-player, m-action stage game. These stage games typically have multiple pure strategy equilibria. In each time period each agent is paired with a randomly selected agent from the population to interact. An agent is randomly assigned to be the row or column player in any interaction. We assume that the stage game payoff matrix is known to both players, but agents cannot distinguish between other players in the population. Hence, each agent can only develop a single pair of policies, one as a row player and the other as a column player, to play against any other player from the agent population. The learning algorithm used by an agent is fixed, i.e. an intrinsic property of an agent.

When two cars arrive at an intersection, a driver will sometimes have another car on its left and sometimes on its right. These two experiences can be mapped to two different roles an agent can assume in this social dilemma scenario and corresponds to an agent playing as the row and column player respectively. Consequently, an agent has a private bi-matrix: a matrix when it is the row player, one matrix when it is the column player. Each agent has a learning algorithm to play as a row player and as a column player and learns independently to play as a row and a column player. An agent does not know the identity of its opponent, nor its opponent’s payoff, but it can observe the action taken by the opponent (perfect but incomplete information). The protocol of interaction is presented in Algorithm 1.

Algorithm 1: Interaction protocol.

```latex
for a fixed number of epoch do
  repeat
    remove randomly agents $p_{row}$ and $p_{col}$ from the population ask each agent to select an action;
    send the joint action to $p_{row}$ and $p_{col}$ for policy update;
  until all agents have been selected during the epoch ;
```

We have considered a homogeneous society of agents where it might seem to the modern reader that “rules of the road” are always fixed by authority, but historical records show that “Society often converges on a convention first by an informal process of accretion; later it is codified into law.” [22].
the WoLF-PHC [4] learning algorithm is chosen for learning norms. WoLF-PHC (Win or Learn Fast - policy hill climbing) can learn mixed strategies. Though WoLF is guaranteed to converge to a Nash equilibrium of the repeated game in a 2-person, 2-actions game against a given opponent, it is not clear whether it is guaranteed to converge in social learning.

4. EXPERIMENTAL STUDY

We are now presenting experimental results exploring the influences of the following issues on the norm emergence.

**Same initial bias:** we study the case where a subset of the population has the same bias to choose one norm over the other, the remaining agents in the population do not have an initial bias. We study the influence of two parameters: the size of the biased population and the strength of the initial bias.

**Agents starting with opposite bias:** we study the case where 50% of the agents in the society have a strong initial bias for one norm and the remaining 50% have an initial bias for the other norm. We vary the proportion of agents initially set with one norm to investigate how fast a majority norm will emerge in the social environment.

**Population sizes with varying bias differentials:** we have investigated the emergence of norms when a bias differential is introduced among the learners. If a certain size of population has an initial bias of $x\%$ to choose an action then the remaining learners have that of $y\%$ to choose the other action as we have considered 2-person 2-action game. So a bias differential exists in the society of learners. We like to study the outcome in terms of percentage of runs averaged for emergence of majority norm by varying the bias differentials and the number of learners playing with opposite biases.

We are now presenting the scenario used for our study before presenting the results.

4.1 Social dilemma Scenario

One typical example of the use of norms or convention is to resolve social dilemmas. A straightforward example of this is when two drivers arrive at an intersection simultaneously from neighboring streets. While each player has the incentive of not yielding, myopic decisions by both can lead to undesirable accidents. Both drivers yielding, however, also creates inefficiency. Ideally, we would like norms like “yield to the driver on right”, which serves all drivers in the long run. Hence, the dilemma is resolved if each member of the population learns to “yield” as a row (column) player and “go” as a column (row) player. The player that yields gets a lesser payoff since it is losing some time compared to the other player. The players know whether they are playing as a row or a column player: the row player sees a car on its right, and the column player sees a car on its left. The action choices for the row player are to go ($G$) or yield to the car on the right ($Y_R$), and they are go ($G$) or yield to the car on the left ($Y_L$) for the column player.

We model this coordination game using the payoffs presented in Table 1. Note that for a social norm to evolve, all agents in the population has to learn any one of the following policy pairs: (a) (row: $G$, col: $Y_L$), i.e., yield to the car on the left, or (b) (row: $Y_R$, col: $G$), i.e., yield to the car on the right. We say a norm has emerged in the population when all learners make the corresponding choice except for infrequent random exploration.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4, 4</td>
<td>-1, -1</td>
</tr>
<tr>
<td>1</td>
<td>-1, -1</td>
<td>4, 4</td>
</tr>
</tbody>
</table>

Table 1: Stage game corresponding to social interactions.

4.2 Influence of agents biased in one option with varying bias values

When the agents have no initial bias, our previous work shows that a norm always emerge. It takes some time for the agents to decide which norm is going to be used, and it is not clear what causes the decision? Is it the influence of few learners that converge faster? Or is it due to a large portion of the agents having a small bias for one norm? In this experiment, we test the latest hypothesis: if a subset of the population has an initial bias, the norm should emerge faster. We study the speed of emergence of a norm when we vary the proportion of the agent with the same initial bias, and the strength of the bias.

In this experiment we vary the number of agents playing with different biases for choosing the norm to yield to the car on the left (i.e., $G$ when the agent plays as a row player and $Y_R$ when the agent is playing as a column player), while the remaining agents are initially unbiased. The norm that emerges is always the norm that had an initial bias. Figure 1(a) and Figure 2(a) represent the dynamics of the agents with an initial bias, and Figure 1(b) and Figure 2(b) represent the dynamics of the agents with no initial bias. In Figures 1(a) and 1(b), 20% of the population has an initial bias. We observe that the norm emerges faster among the agents that started with an initial bias compared than among the agents that started with no bias. In addition, the stronger the bias, the faster the speed of emergence of the norm. For example, the emergence of the norm is faster when the initial bias of the agent is 0.8 ($\approx 500$ iterations) than when the value of the bias is 0.55 ($\approx 675$ iterations). In Figures 2(a) and 2(b), the proportion of agent that has an initial bias is 80%. Among the agent that were initially biased, the speed of emergence of the norm is comparable when the proportion of agent with initial bias is 20% or 80%. However, among the agent that started unbiased, the speed of emergence is faster when 80% of the population has an initial bias.

4.3 Influence of agents starting with opposite policies

So far we have considered a society where a certain percentage of learners are biased in choosing one particular norm and the remaining population is unbiased. We have observed that the norm that always emerges is the norm which had an initial bias. Now we consider 50% of the population will
Figure 1: Iterations required to emerge a norm in a society where the 20% of population is only biased in one action.

Figure 2: Iterations required to emerge a norm in a society where the 80% of population is only biased in one action.

Figure 3: Society with 225 agents using WoLF, averaged over 1,000 runs. The population converges 490 times to \((G, Y_L)\) and 510 times to \((Y_R, G)\).
Figure 4: Society with 225 agents using WoLF, averaged over 1,000 runs. The population converges 100% of the times to \((G,Y_L)\).

Figure 4 that the norm that emerges is always the norm that was initially chosen by the majority of agents. Comparing Figure 3 and 4, we observe that the speed of emergence of the norm is significantly faster when the split of the population is 60%/40% than when it is 50%-50% (≈ 750 iterations instead of ≈ 1250)

4.4 Influence of population sizes with varying bias differentials

In the previous set of experiments, we have studied some special cases. First we considered the case where all agents that have an initial bias have the same bias for the same norm, and we studied the influence of the strength of the bias. Then, we studied the case where agents have a maximum initial bias (initially, the agent chooses one norm with probability 1), though different agents may prefer different norms, and we studied the influence of the proportion of agents that initially prefer one norm over the other on the speed of emergence of a norm. In this section, we propose a more general study of the influence of the bias. We assume that

- all agents start with an initial bias;
- all agents that have a bias for the same norm have the same value for the initial bias.

If the proportion of agents that initially prefer norm \(a\) is more than 50%, we say that \(a\) is the majority norm, and the other norm is the minority one. Hence our study has three parameters:

\[ x: \text{is the proportion of the population corresponding to the majority, hence } 0.5 \leq x \leq 1; \]
\[ y: \text{the bias of the preferred norm for the set of agents that are in the majority}; \]
\[ z: \text{the bias of the preferred norm for the set of agents that are in the minority}; \]

We refer to \((y - z)\) as the bias differential and \(2x - 1\) as the population differential (note that as \(0.5 \leq x \leq 1, 0 \leq 2x - 1 \leq 1\)).

<table>
<thead>
<tr>
<th>majority bias</th>
<th>minority bias</th>
<th>bias differential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.65</td>
<td>0.35</td>
</tr>
<tr>
<td>0.90</td>
<td>0.70</td>
<td>0.20</td>
</tr>
<tr>
<td>0.85</td>
<td>0.75</td>
<td>0.15</td>
</tr>
<tr>
<td>0.80</td>
<td>0.775</td>
<td>0.05</td>
</tr>
<tr>
<td>0.775</td>
<td>0.75</td>
<td>0</td>
</tr>
<tr>
<td>0.75</td>
<td>0.8</td>
<td>-0.05</td>
</tr>
<tr>
<td>0.70</td>
<td>0.85</td>
<td>-0.15</td>
</tr>
<tr>
<td>0.65</td>
<td>0.9</td>
<td>-0.25</td>
</tr>
<tr>
<td>0.60</td>
<td>0.95</td>
<td>-0.35</td>
</tr>
<tr>
<td>0.55</td>
<td>1</td>
<td>-0.45</td>
</tr>
</tbody>
</table>

Table 2: Bias differential table.

We now investigate, in a systematic way, the influence of the bias and population differentials on the nature of the norm that emerges. We present the results in Figure 5(b) 5(a). In Figure 5(b),

- The x-axis is the population bias.
- The y-axis is the bias difference. For a bias difference of 0.45, the majority has an initial bias of 1 and the minority one of 0.55; thereafter, the minorities bias is incremented by 0.5 and the majority bias is decremented by 0.5 (produces a bias difference of 0.1 in one step) until the bias difference is -0.45. The 0 bias corresponds to the case where the majority and minority have an initial bias of 0.775 (see Table 2).
- The z-axis represent the percentage of times that the norm of the majority emerges

In Figure 5(a), we represent the percentage of time the norm of the majority emerges as a function of the population bias for some fixed values of the bias difference (i.e., we slice Figure 5(b) for a fixed value of the bias difference).

Let us consider the scenario where the population differential is 90% and the bias differential is -0.45. That case corresponds to the case where the majority is formed by 95% of the population and has a bias of .55, and the minority is formed by 5% of the population with a bias of 1 (see Table 2). For this scenario, we observe that the norm of the majority always emerge.

When the bias differential is positive, the majority has a higher bias compared to the minority (see Table 2). In that case, the norm that emerges is the norm of the majority, irrespective of the population difference. This is to be expected as the agents in the majority have also the stronger bias.

When the bias differential is negative, it means that the bias of the minority is stronger than the bias of the majority. In other words, the minority contains less agents, but their bias is much stronger and may influence undecided agents. In such cases the norm preferred by the minority may ultimately be adopted by everyone. When the bias of the minority is very strong and the one of the majority is
weak, the norm of the majority emerges only if the size of the majority is extremely large. For example, when the bias difference is -0.45, the minority has a bias of 1, the majority has a bias of .55, if the population differential is lower than 70 (i.e., if the size of the majority smaller than 85%), the norm of the minority will always emerge!

With this bias differential, the size of the majority has to be 95% before its preferred norm always emerges. In general, for negative bias differential, there are three regions:

1. a minimum population differential below which the minority norm always emerges;
2. a range of population differential in which increasing the population differential increasing the likelihood of the majority norm emerging,
3. and a maximum population differential above which the majority norm always emerges.

Also as the bias difference between the minority and the majority norm followers increase, it takes more population differential for the majority to bring about a certain likelihood of its preferred norm emerging (the curves shift to the right in Figure 5(a)).

5. CONCLUSION
We investigated a bottom-up process for the evolution of social norm that depends exclusively on individual experiences rather than observations or hearsay. Our social learning framework requires each agent to learn from repeated interaction with anonymous members of the society. In past work, we have assumed that the learners did not have an initial bias for one norm over another one, and under that hypothesis, we always observed the emergence of a norm.

In this work, we investigate the influence of the bias of the initial policy of a learner on the norm that emerges. We empirically showed that if all agents have a bias for the same norm, the norm emerges faster than when agents do not have initial bias. Then we showed that when all agents have a strong bias, a norm always emerge and is the one of the majority. The larger the margin between the majority and the minority, the faster the convergence. Finally, we provided a more general study of the influence of the initial bias where all agents have an initial bias, and agents that initially prefer the same norm have the same bias. For example, our study shows that when the majority has a weak bias and the minority a strong one, the size of the majority must be over a threshold for the norm preferred by the majority to emerge.

In future works, we would like to complete the general study of Section 4.4 as we have only presented results about a particular relationship between the initial bias of the majority and minority. When the agents initially start with no bias, after some iterations, they start to have a small bias. We would like to use the study to see whether we can determine when the norm is actually chosen by the agents: after a few iteration, the bias induced by the random interactions between the agents may be enough for one norm to emerge for sure.

Acknowledgment: US National Science Foundation award IIS-0209208 partially supported this work.

6. REFERENCES
Majority norms in biased society

% of runs to majority norms

(a) Nature of the percentage of runs that varies with different biased differentials.

(b) The surface, generated by the influence of bias and population differential on % of runs averaged for convergence of majority norms.

Figure 5: 2-D and 3-D plot for norm convergence when bias differentials exist in the society.


ABSTRACT
In this work, an hybrid, self-configurable, multilayered and evolutionary subsumption architecture for cognitive agents is developed. Each layer of the multilayered architecture is modeled by one different Reinforcement Machine Learning System (RMLS) based on bio-inspired techniques. In this research an evolutionary mechanism based on Gene Expression Programming to self-configure the behaviour arbitration between layers is suggested. In addition, a co-evolutionary mechanism to evolve behaviours in an independent and parallel fashion is used too. The proposed approach was tested in an animat environment (artificial life) using a multi-agent platform and it exhibited several learning capabilities and emergent properties for self-configuring internal agent’s architecture.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning – Connectionism and neural nets, Induction, Knowledge acquisition, Parameter learning.

General Terms
Algorithms

Keywords

1. INTRODUCTION
Recently, Cognitive Architectures have been an area of study that collects disciplines as artificial intelligence, cognitive science, psychology and more, to determine necessary, sufficient and optimal distribution of resources for the development of agents exhibiting emergent intelligence. One of the most referenced is the Subsumption Architecture proposed by Brooks [1].

According to Brooks [1], the Subsumption Architecture is built in layers. Each layer gives the system a set of pre-wired behaviours, where the higher levels build upon the lower levels to create more complex behaviours: The behaviour of the system as a whole is the result of many interacting simple behaviours. Another characteristic is its lack of a world model, which means that its responses are always and only reflexive as proposed by Brooks.

However, Subsumption Architecture results in a tight coupling of perception and action, producing high reactivity. Poor adaptability to new environments, no learning capabilities, no internal representation and the need of all patterns of behaviours must be pre-wired, are some weaknesses of the Subsumption theory.

Several extensions have attempted to add representation and behaviour arbitration to Subsumption like Behavior-Based Control Architecture [2] and Hormonal Activation Systems [3], but pre-wired behaviours and non-learning characteristics still remain becoming the architecture applicable and restricted only for a specific pre-configured environments.

The present research focuses on developing an Hybrid Multilayered Architecture for Cognitive Agents based on Subsumption theory. Additionally this work proposes an Evolutionary Model which allows the Agent to self-configure and evolve its arbitration of processing layers through the definition of processes (like inhibition, suppression and aggregation), kind of behaviours and number of layers. That means each agent instead of having a pre-configured structure of layers and processes it will have an Artificial Evolutionary Process which is responsible for defining the multilayered structure. On the other hand, instead of using an Augmented Finite Machine System as Subsumption theory states in [3] where no internal representation is done, in this paper we propose that each behaviour layer is driven by a different bio-inspired reinforcement machine learning system RMLS (chosen from a repertoire where behaviour co-evolution occurs) which learns from the environment and generates an internal world-model by means of an unsupervised and reinforced learning. The RMLSs used in the approach are: Extended Classifier System XCS [5], Learning Classifier System LCS [6], Artificial Immune System AIS [7], [8] and Neuro Connectionist Q-Learning System NQL [9], [10].

The remainder of the paper is organized as follows. The description of the approach proposed is detailed in Section 2. Section 3 outlines and discusses the experimental results and emergent properties obtained. Finally concluding remarks are shown in Section 4.
2. PROPOSED HYBRID, SELF-CONFIGURABLE AND EVOLUTIONARY MODEL

In order to design an hybrid, self-configurable, scalable, and evolutionary architecture for cognitive systems which exhibits emergent behaviours and learning capabilities, the proposed work is exposed as follows.

Consider a virtual environment where there are several cognitive agents interacting with each others using a typical Subsumption Architecture. Some mayor constraints arise:

- Environmental conditions changing continuously.
- The number of behaviours inside each agent is variable.
- Arbitration of behaviours is pre-wired and not depends on agent’s motivational states.
- The cognitive agent can inhibit or suppress behaviours “only” if an applicability predicate is preestablished and new environment changes are not considered.
- Agent’s behaviours do not generate a model of the world, do not couple with the environment via the agent’s sensors and actuators, do not learn about its own interaction with the environment and do not evolve the internal state of the behaviour.

These constraints address the following proposed approach of an hybrid, self-configurable and bio-inspired architecture for cognitive agents, depicted in Figure 1.

![Figure 1. Hybrid and Evolutionary Architecture for Cognitive Agents.](image-url)
The Figure 1 shows an hybrid architecture from which all the constraints mentioned before can be solved. An internal architecture based on subsumption principles but with few variations can be observed in every agent:

- Each processing layer is connected randomly with a different learning machine system (XCS, LCS, AIS, NQL, and scalable to others) which replaces the typical AFSMs proposed by Brook’s architecture in [1].
- After being trained, each agent’s behaviour is sent to a behaviour repertoire according to its type, where a co-evolutionary mechanism is applied so that every behaviour not only will learn in a local way inside of each agent but also will evolve in a global way, to be selected afterwards by another agent in the next generation.
- There is an evolutionary process driven by a Gene Expression Programming Algorithm GEP [12], which is in charge of self-configuring the agent (defining the number of layers, the behaviours that the agent will use, the connections and hierarchies between them -inhibit, suppress, aggregate-, the applicability predicates that determine which behaviour is activated at a certain situation and an activation time controlled by a timer).

2.1 Hybrid Learning Layer: Behaviours driven by different Machine Learning Systems

Every behaviour layer in the multilayered architecture will be associated to a Reinforcement Machine Learning System RMLS, that allows the architecture being hybrid and not only reactive since each behaviour will be able to exert deliberative processes using the acquired knowledge. Besides, this mechanism gives plasticity to the architecture because every behaviour “learns” in an unsupervised, independent and parallel way through its interaction with the environment, generating internal representations, rules and both specific and generalized knowledge. This mechanism is favored by the RMLSs characteristics: robustness, fault tolerance, use of bio-inspired techniques, adaptability and they do not require a previous definition of knowledge (unsupervised learning).

There are two principles formulated by Stone [13] that have motivated the proposed layered learning approach:

- Layered learning is designed for domains that are too complex for learning a mapping directly from an agent’s sensory inputs to its actuator outputs. Instead the layered learning approach consists of breaking a problem down into several behavioral layers and using RMLSs at each level. Layered learning uses a bottom up incremental approach to hierarchical task decomposition.
- RMLS is used as a central part of layered learning to exploit data in order to train and or adapt the overall system. RMLS is useful for training behaviors that are difficult to fine-tune manually.

The sensory inputs of each RMLS read the objects sensed around the agent while the actuator outputs indicate actions that the agent must to execute on the environment.

Accordingly, a common interface for all RMLSs (XCS, AIS, NQL, LCS, etc.) is proposed so although each RMLS has a different internal process, they all have a similar structure that it lets the system to be scalable introducing new RMLSs if is required and connecting them in an easy way with each behaviour layer in the agent’s multilayered architecture, as depicted in Figure 2.

Each RMLS has its advantages and disadvantages. However, no one RMLS is always better than others, so it is difficult to determine a good RMLS to drive each behaviour. On the other hand, the Cognitive Agent determines which behaviors must be inhibited or suppressed in a specific situation, but not always just one behaviour remains activated after inhibition, sometimes the Agent can require several behaviours to be activated concurrently. In order to merge the outputs of these behaviours activated in just one output, we propose an Aggregation Mechanism similar to proposed by Jiang in [11] but using different RMLS’s. This mechanism is based on Borda Counting Method. Aggregation of RMLS can improve the learning qualities as a whole because they can share some knowledge and utilize the strengths of the others to alleviate individual weaknesses.

2.2 Evolutionary Layer: Behaviour Arbitration

If each agent has an arbitrary behaviour set, how to determine: the interaction between them, the hierarchy levels, the Subsumption process (inhibition and suppression) and the necessary layers to do an adequate processing? These questions are solved next.

The internal multilayered structure of each agent is decomposed in atomic components which can be estimated and used to find the optimal organization of behaviors during the agent’s lifetime [4]. The main goal is that the agent in an automatic way self-configures its own behaviours structure. The model proposed by Ferra Reese in [12] called Gene Expression Programming GEP is used to evolve internal structures of each agent and generate a valid arbitration of behaviours.

GEP uses two sets: a function set and a terminal set. The proposed function set is: AND, OR, NOT, IFMATCH, INHIBIT, SUPPRESS and AGGREGATE. The AND, OR and NOT functions are logic operators used to group and exclude subsets of elements. The conditional function IFMATCH is an applicability predicate that matches with a specific problem situation. This function has five arguments; the first four
arguments belong to the rule’s antecedent: they all indicate motivational levels in the agent (internal states, moods, etc.), for instance: energy level, bravery/cowardice level, hunger/thirstiness level, etc. If the first four arguments are applicable then the fifth argument, the rule’s consequent, is executed. The fifth argument should be a INHIBIT/ SUPPRESS/AGGREGATE function, or maybe and AND/OR function if more elements are necessary. The INHIBIT, SUPPRESS and AGGREGATE functions have two arguments (behaviourA, behaviourB) and indicate that behaviourA inhibits/suppresses/aggregate behaviourB.

On the other hand, the terminal set is composed by the behaviour set and the motivational levels set. Additionally “do not care” elements are included so whichever behaviour or motivational levels can be referenced. Behaviour Arbitration is driven by Agent’s Motivational Levels which try to simulate moods or humor states in the Cognitive Agent. These moods are changing continuously whereas the Agent interacts with the environment.

Each agent has a chromosome with information about its self structure, e.g. Agent A can have a chromosome as: [{IFMATCH}, {ml1}, {ml2}, {ml3}, {ml4}, {INHIBIT}, {behaviour1}, {AND}, {behaviour2}, {behaviour3}], and this chromosome is a valid rule because both the antecedent and the consequent of IFMATCH function match to each required argument type, where {ml} is the abbreviation for motivational level. The above chromosome traduces in the following rule:

IFMATCH:

ml1, ml2, ml3, ml4

THEN:

behaviour1 INHIBIT behaviour2 AND behaviour3.

Analyzing this rule we can infer that the agent has three behaviour layers: behaviour1, behaviour2, and behaviour3, and the two last ones are inhibited by the first one when agent has the motivational levels ml1, ml2, ml3, ml4. However, these chromosomes (applicability predicates) do not have always a valid syntax, so the GEP mechanism is used to evolve the chromosome until it becomes in a valid syntactic rule.

Each individual (agent) has a multigenic chromosome, that means, each chromosome has a gene set where each gene is an applicability predicate like the example, so the agent has several rules (genes) as part of its genotype and each one is applied according to the situation that matching the rule antecedent. Each gene is become to a tree representation and then a genetic operator set is applied between genes of the same agent and that this emergence is to a large extent accidental.

Finally, new random rules are generated until complete the maximum size of rules that behaviours can have in their own knowledge base, so a new pair of behaviors is created and left in the corresponding behaviour pool to be selected by an agent in the next generation.

2.4 Emergent Properties of the Architecture

Brooks postulates in his paper [3] the possibility that intelligence can emerge out of a set of simple, loosely coupled behaviours, and emergent properties arise (if at all) due to the complex dynamics of interactions among the simple behaviours and that this emergence is to a large extent accidental.

The proposed architecture articulates a behaviour set that learns about environmental conditions in an independent and parallel fashion, and on the other hand evolve inside a categorized pool. Each simple behavior can be applied to a subset of specific situations but not to the whole problem space, however the individual level interaction between behaviours (inside each agent) allows covering multiple subsets of problem states and some characteristics are generated: robustness, redundancy in acquired knowledge: fault tolerance and a big plasticity level, so emergent properties in the individual and inside of the society (Multiagent systems) appear. So, the emergent properties arise from three points of view in a bottom-up approach:

o Atomic Level: in each behaviour of the multilayered architecture, when the associated RMLS learns from the environment how to associate sensory inputs and actuator outputs, in an automate way.

o Individual Level: when the agent self-configures its internal structure (chromosome), hierarchy and arbitration of behaviours through an evolutionary process driven by GEP.
Social Level: when an hybrid behaviour co-evolution mechanism is applied to all agent’s behaviours, so behaviours learn not only themselves via the RMLS associated but also cooperating with other agents and communicating the acquired knowledge between them.

It is important to notice that emergence in different levels, from atomic to social point of view, provokes an overall emergence of the system, where some kind of intelligence we hope to arise. The experimentation focused on discovering some emergent characteristics in the agents. Nevertheless, expected emergent properties can vary according to the environment and the behaviour set.

3. EXPERIMENTATION

In order to evaluate the proposed architecture, following aspects were considered in each level:

- Learning convergence rate of each proposed systems: XCS, AIS, LCS and NQL.
- Learning and evolution convergence rate of each behaviour pool.
- Variation of success rate vs. number of genes in GEP
- Syntactically well-formed gene convergence rate

About overall System:

- Subsumption architectures obtained on individuals after n iterations and emergent properties identified.

An artificial life environment called Animat (animal + robot) described in [6] is proposed to test the experiments. The environment simulates virtual agents (prey-depredator model) competing for getting food and water, avoiding obstacles, hunting, escaping from depredators, etc. Each animat driven by an agent in the environment disposes a set of 10 proximity sensors (see Figure 3) simulating a limited sight sense. 8 sensors read a safe zone and 2 sensors read a danger zone (to avoid collisions) as proposed by Romero [8].

Thus, some experiments designed to evaluate the performance aspects mentioned above are described next.

3.1 Learning convergence of each RMLS

In this experiment we chose an environment where the animat has to interact with using one different RMLS on a time. Table I shows the learning parameters used.

Table 1. Learning parameters of each RMLS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>XCS</th>
<th>AIS</th>
<th>NQL</th>
<th>LCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life Tax</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
<td>0.005</td>
</tr>
<tr>
<td>Bid Tax</td>
<td>-</td>
<td>0.003</td>
<td>-</td>
<td>0.003</td>
</tr>
<tr>
<td>Cloning Rate x rule</td>
<td>1</td>
<td>4</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Mutation Rate x rule</td>
<td>2</td>
<td>2</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Similarity Threshold</td>
<td>-</td>
<td>0.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Alpha α</td>
<td>0.1</td>
<td>-</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>Beta β</td>
<td>0.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Delta δ</td>
<td>0.1</td>
<td>-</td>
<td>0.02</td>
<td>-</td>
</tr>
<tr>
<td>Gamma γ</td>
<td>0.95</td>
<td>-</td>
<td>0.8</td>
<td>-</td>
</tr>
<tr>
<td>Lamda λ</td>
<td>-</td>
<td>-</td>
<td>0.8</td>
<td>-</td>
</tr>
<tr>
<td>Layers in NN</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Number of Epochs</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nº runs x epoch</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4 shows a chart of the learning curve of the RMLSs: XCS, AIS, LCS, simple NQL and multilayered NQL.

Figure 4 illustrates that AIS and NQL are more adaptive and robust than the others converging more quickly when changes in the learned environmental pattern are introduced. The peaks were registered because of changing patterns, but each RMLS adapted to new conditions quickly.

3.2 Learning and evolution convergence of each behaviour pool.

The goal of this experiment is to examine if the fitness of every separate behaviour pool increments gradually until reaches a convergence point while evolution takes place. The experiment was carried out with the parameters on Table 2

Three behaviour pools were selected for the experiment: Avoiding-obstacles, Looking-for-food and Escaping-From-Depredators, the results are depicted in Figure 5.

Figure 5 depicted some differences in each learning curve, due to environmental conditions, however the pools always tried to converge and reach certain stability in the same number of...
epochs (approximately after 30 epochs), that means the evolution has been effective and each behaviour pool has established a coherent knowledge base getting a consensus between its own behaviour instances, about what the “behaviour category” should do.

<table>
<thead>
<tr>
<th>Table 2. Co-evolution Learning Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>Epochs</td>
</tr>
<tr>
<td>Nº runs x epoch</td>
</tr>
<tr>
<td>Crossover Prob.</td>
</tr>
<tr>
<td>Mutation Prob.</td>
</tr>
<tr>
<td>Mutation Rate $\eta$</td>
</tr>
<tr>
<td>Mutation Rate $\theta$</td>
</tr>
<tr>
<td>Mutation Rate $\kappa$</td>
</tr>
<tr>
<td>Mutation Rate $\gamma$</td>
</tr>
</tbody>
</table>

3.3 Syntactically well-formed gene convergence
In this experiment, the progression of the number of syntactically well-formed structure (multigenic chromosomes) of each individual was analyzed. Figure 6 shows how the number of valid chromosomes increments whereas generations evolve through the time. The experiment was executed with a population of 300 individuals.

Figure 6 shows that a point of convergence (that means all chromosomes in population are valid) is given in the generation 27 approximately. Then, the system will need between 25 and 30 generations to evolve all individuals in the population.

3.4 Analysis of evolved architectures
Finally, after the whole system has evolved during a specific number of generations, we have analyzed the final structures of the best adapted agents where emergent properties arose.
Figure 7 shows the genotype (Expression Trees ETs) and phenotype respectively of an initial architecture of a random agent without any evolutionary phase; in contrast, Figure 8 shows the genotype and phenotype respectively of the evolved architecture of the same agent.

In Figure 7 the chromosome represents four behaviours: looking-for-water LFW, looking-for-food LFF, avoiding-obstacles AO and sleeping SL, where LFW inhibits LFF and SL and LFW suppresses AO, but there is a contradictory process when LFF tries to suppress LFW and LFF has been inhibited by LFW already. This is solved with the evolved architecture in Figure 8, which proposes a new structure adding escaping-from-depredators EFD behaviour and excluding sleeping behaviour.

As depicted in Figure 8, the initial contradictory inhibitory/suppressor processes in the agent’s architecture are solved, and only hierarchical inhibitory processes are proposed by the evolved architecture. Furthermore, we can deduce too that evolved architecture has collected a set of specific behaviours becoming the agent to an animat with a prey identity.

It is important to notice in evolved architecture that EFD behaviour inhibits both LFF and LFW behaviours, but if the animat is escaping and its sensors read a “wall” or a “tree”, then EFD behaviour is inhibited by AO behaviour until the obstacle is not in front of the animat anymore, and after that the animat continues its getaway, so we can say that emergent behaviour arises.

Finally, the experimentation demonstrate that specific parameter configurations in RMLs, GEP and Co-evolutionary mechanism are required to reach certain robustness, adaptability and learning capacities in the overall system. Nevertheless, emergent properties did not arise every time or in a quick way, in several experiments animats died quickly and they could not learn to survive.

4. CONCLUSIONS

The integration of multiple Reinforcement Machine Learning Systems in controlling the behaviours layers of an hybrid Subsumption Architecture approach, instead of using the typical Augmented Finite State Machines, have demonstrated important advantages in learning about the world of the agent, making obstacles AO and sleeping SL, where LFW inhibits LFF and SL and LFW suppresses AO, but there is a contradictory process when LFF tries to suppress LFW and LFF has been inhibited by LFW already. This is solved with the evolved architecture in Figure 8, which proposes a new structure adding escaping-from-depredators EFD behaviour and excluding sleeping behaviour.

As depicted in Figure 8, the initial contradictory inhibitory/suppressor processes in the agent’s architecture are solved, and only hierarchical inhibitory processes are proposed by the evolved architecture. Furthermore, we can deduce too that evolved architecture has collected a set of specific behaviours becoming the agent to an animat with a prey identity.

It is important to notice in evolved architecture that EFD behaviour inhibits both LFF and LFW behaviours, but if the animat is escaping and its sensors read a “wall” or a “tree”, then EFD behaviour is inhibited by AO behaviour until the obstacle is not in front of the animat anymore, and after that the animat continues its getaway, so we can say that emergent behaviour arises.

Finally, the experimentation demonstrate that specific parameter configurations in RMLs, GEP and Co-evolutionary mechanism are required to reach certain robustness, adaptability and learning capacities in the overall system. Nevertheless, emergent properties did not arise every time or in a quick way, in several experiments animats died quickly and they could not learn to survive.

The evolutionary mechanisms used in this work, provided a plasticity feature allowing the agent to self-configure its own multilayered behaviour-based architecture; thus it can avoid creating exhaustive and extensive knowledge bases, pre-wired behaviour-based multilayered structures and pre-constrained environments. Instead of this, a cognitive agent using our architecture only needs to interact with an arbitrary environment to adapt to it and take decisions in a reactive and deliberative fashion.

In the experimentation, the emergent properties were difficult to discover because it takes a lot of time to evolve the overall system despite of using a multiagent platform in a distributed configuration. Maybe, it can be similar to the natural evolution where adaptation occurs slowly and sometimes produces poor adapted creatures.

In our future work we expect to continue working on designing more adaptive and self-configurable architectures, using fuzzy techniques in the RMLs to improve the sensors readings and to manipulate motivational levels (moods). One concrete application of this research will be the development of a Cognitive Module for Emotive Pedagogical Agents where the agent will be able to self-learn about its own perspectives, believes, desires, intentions, emotions and perceptions.
5. ACKNOWLEDGMENTS
Supported by the Programme Alban, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E05D056455CO”.
Diego Romero of Mechanical Engineering Department at National University of Colombia, who made numerous contributions to this research.

6. REFERENCES
Multi-Agent Reinforcement Learning for Intrusion Detection: A case study and evaluation

Arturo Servin
University of York
Heslington York
YO10 5DD, United Kingdom
aservin@cs.york.ac.uk

Daniel Kudenko
University of York
Heslington York
YO10 5DD, United Kingdom
kudenko@cs.york.ac.uk

ABSTRACT
In this paper we propose a novel approach to train Multi-Agent Reinforcement Learning (MARL) agents to cooperate to detect intrusions in the form of normal and abnormal states in the network. We present an architecture of distributed sensor and decision agents that learn how to identify normal and abnormal states of the network using Reinforcement Learning (RL). Sensor agents extract network-state information using tile-coding as a function approximation technique and send communication signals in the form of actions to decision agents. These in turn generate actions in the form of alarms to the network operator. By means of an on line process, sensor and decision agents learn the semantics of the communication actions. In this paper we detail the learning process and the operation of the agent architecture. We also present tests and results of our research work in an intrusion detection case study, using a realistic network simulation where sensor and decision agents learn to identify normal and abnormal states of the network.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multiagent systems; I.2.6 [Artificial Intelligence]: Learning; K.6.5 [Management of Computing and Information Systems]: Security and Protection

General Terms
Machine Learning

Keywords
Intrusion Detection, Reinforcement Learning, Multi-Agent Systems

1. INTRODUCTION
Intrusion Detection Systems (IDS) play an important role in the protection of computer networks and information systems from intruders and attacks. Despite previous research efforts there are still areas where IDS have not satisfied all requirements of modern computer systems. Specifically, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks have received significant attention due to the increased security vulnerabilities in end-user software and bot-nets [10]. A special case of DoS are the Flooding-Base DoS and Flooding-Base DDoS attacks. These are generally based on a flood of packets with the intention of overfilling the network resources of the victim. It is especially difficult to create a flexible hand-coded IDS for such attacks, and machine learning is a promising avenue to tackle the problem. Due to the distributed nature of this type of attacks and the complexities that involve its detection, we propose a distributed reinforcement learning (RL) approach.

Multi-Agent RL has been successfully used to solve some challenging problems in various areas. Despite its apparent appeal, MARL needs to deal with problems such as the size of the action-state space which makes scalability an issue; the partial information that agents have of other agents’ observations and actions; a non-stationary environment as result of the actions of other agents, and the credit assignment problem. To overcome these problems we present an architecture of distributed sensor agents that get information from the environment and share it in the form of communication signals with other agents higher up the hierarchy. Without any previous semantic knowledge about the signals, higher-level hierarchical agents interpret them and consequently interact with the environment. This results in a learning process where agents with partial observability make decisions and coordinate their own actions to reach a common goal.

In order to evaluate our proposal we explore its use of Distributed Intrusion Detection Systems (DIDS). Distributed Intrusion Detection Systems (DIDS) is a group of IDS or sensors coordinated to detect anomalies or intrusions. The system can be homogeneous with every sensor of the same kind and type or heterogeneous with a mixture of types. We build our DIDS approach by training of a group of heterogeneous sensor agents that must identify normal and abnormal states of the network resulting from Flood-Base DoS and DDoS. We have used the detection of these attacks to test our learning approach for the following reasons:

- Many researchers [13, 2, 19] remark that a variety of sensor information is required to detect attacks with high levels of confidence.
• These type of attacks disrupt the operation of the network by modifying state information. Spotting these abnormal states can lead to the detection of a flooding attack.

• These abnormal states are characterised by several factors that are normally present in different part of the network and they are only visible to specific networks devices. To identify these events, it is not possible to use a single device or entity.

• To identify events around the network that are visible to only some type of agents it is necessary to use distributed specialised agents. These agents only have partial observability of the whole environment (network).

• In the future the IDS could not only detect intrusions but it also could try to stop them. This would give the opportunity to automatically generate feedback from the environment without the need to hand code the reward function.

In addition to proposing a distributed RL approach for intrusion detection, we adapt and evaluate it in a realistic network simulation using the ns-2 [20] simulator. In this way, we are able to demonstrate the practical applicability of our approach.

The paper is organized as follows: In section 2 we present an overview of the Intrusion Detection System (IDS), MARL and tile coding. We follow with a detailed description of the agent architecture and operation in section 3. In section 4 we explain the approach used to train and test the proposed agent architecture. We also show and explain the results obtained. Section 5 briefly presents some related research work in MARL and the application of RL on the IDS field. Finally we conclude this paper and discuss future research directions.

2. TECHNOLOGY OVERVIEW
Flood-Base DoS and DDoS attacks change the normal behaviour of the network in different ways and spotting these differences could help us to detect the presence of attacks [12]. The distributed operation of these attacks brings on the opportunity to use a distributed and adaptable platform to detect them. We propose to use an architecture based on MARL agents.

2.1 Intrusion Detection Systems
IDS are commonly divided in two functional categories; Anomaly Intrusion Detection and Misuse/Signature Intrusion Detection. Anomaly IDS states that intrusions are deviations of normal traffic. These systems create profiles of different variables over time to get a usage pattern. The difference between the pattern and the current activity triggers an alarm. The advantage of these systems is that they are capable of detecting unknown attacks, however non-malicious activity that does not match normal behaviour can also trigger the intrusion mechanism. This results in a high rate of false alarms. On the other hand, misuse or signature intrusion detection systems use rule matching to detect intrusions. These systems compare system activity with specific intrusion rules that are generally hard coded. When the observed activity matches the intrusion pattern an intrusion is detected and an action is executed. The flaw of these systems is that regardless of their accuracy and reliability they lack the ability to detect new types of attacks.

Anomaly Intrusion Detection Systems use a variety of schemes to detect normal user patterns from simple statistical to complex machine learning methods. Although most of the research on IDS using machine learning has been done under an Anomaly Intrusion Detection approach, recent research work incorporates Machine Learning to automatic rule generation on misuse/signature intrusion detection. IDS using machine learning try to learn a function that maps input information into different categories. The learning process can be supervised, unsupervised or reinforced.

2.2 Multi-Agent Reinforcement Learning
In Reinforcement Learning an agent learns to act optimally via observations and feedback from the environment in the form of positive or negative rewards [22]. A widely used RL technique is Q-learning [23]. In Q-learning the agent iteratively tries to estimate a Q value function that tells the agent how good it is to perform a specific action in a given state. In Q-learning the agent chooses an action a in any given state s, observes the reward r and the next state s′. Then it updates the estimated Q value denoted by ̂Q by Eq. 1.

\[ ̂Q(s, a) = (1 - \alpha) ̂Q(s, a) + \alpha (r + \gamma \max_a ̂Q(s′, a′)) \] (1)

In order to discover which actions lead to the best rewards over time, the agent needs to explore and to exploit actions.

In our experiments we have used Boltzmann or softmax action selection rules as the exploration/exploitation strategy. When RL is used in real world applications, it is not feasible to store values for all states individually. To tackle this problem we use function approximation techniques. These techniques commonly used in supervised learning are adapted to RL in order to generalise the state representation. One especially successful technique is Tile Coding. Tile coding is a type of sparse coarse coding where the features or characteristics that we want to analyse are grouped in a set of partitions. These partitions form the input space and they are called tiling. Each tiling is divided into small pieces called tiles. To produce the state-action feature representation several tilings are overlapping. The value of the state-action \((V\pi(s))\) is the sum of all the weights \((\vec{w})\) of the tiles activated by the input.

MARL has shown promise in solving distributed problems, but there are many challenges to overcome when applying it in a realistic network domain, e.g., feature selection, communication, and synchronisation. In a DIDS architecture we require a large number of distributed agents collecting complex network information and coordinating their action under restricted communication.

3. AGENT ARCHITECTURE
In previous research [17], we used a highly abstract IDS scenario to test how a group of agents learn to interpret
and coordinate their action signals to detect normal and abnormal activity. We proposed a hierarchical architecture of agents composed by groups of agents or cells. These cells were composed by sensor agents (SA) and decision agents (DA). SA collect and analyse state information about the environment. Each SA receives only partial information about the global state of the environment and they map this local state to communications action-signals. These signals are received by the DA and without any previous knowledge it learns their semantics and how to interpret their meaning. In this way, the DA tries to model the local state of cell environment. Then it decides which signal-action to execute to a higher level agent outside the cell or in single cell environments the final action to trigger (in our case study it triggers an alarm to the network operator). To expand the number of agents we used multi-cell environments composed by cells of DAs. In these multi-cell environments each DA inside the cells sends action-signal to a central DA, which in turn can trigger a final action or it can send an action-signal to a higher level DA. When the top agent in the hierarchy triggers the action and this is appropriate accordingly with the goal pursued, all the agents in the cell receive a positive reward. If the action is not correct, all the agents receive a negative reward. The goal is to coordinate the signals sent by the SA to the DA in order to represent the global state of the environment. After a certain number of iterations every agent must know the action that they need to execute in a specific state to obtain positive rewards.

This agent architecture may be used in a diverse set of environments to solve different kind of problems. In our past research work we designed a highly abstract simulation of a distributed sensor network. This environment gave us the opportunity to test the basic feasibility of the agent learning architecture using an abstract environment containing simple network agents. However, the question remained open how the approach would work with more complex and realistic network topologies, traffic patterns and connections. In order to evaluate our learning architecture of agents and to add elements and the complexity of real applications, in this paper we used the network simulator ns-2 [20], a specifically designed library for ns-2 and the Tile Coding Software [21].

To detect the abnormal states that DoS and DDoS generate in a computer network we have slightly modified the original agent architecture as shown in Fig. 3. This architecture is composed by single cell with a Congestion Sensor Agent (CSA), a Delay Sensor Agent (DSA), a Flow Sensor Agent (FSA) and the Decision Agent (DA). We need this diversity of sensor information to develop more reliable IDS. The idea is that each sensor agent perceives different information depending on their capabilities, their operative task and where they are deployed in the network. Furthermore not all the features are available in a single point in the network. Flow and congestion information may be measured in a border router between the Internet and the Intranet whilst delay information may be only available from an internal router. What is more, Flood-Based DDoS attacks are launched from several remote controlled sources trying to exhaust a target’s key resource. A stand-alone IDS does not have all the information to accurately identify sources and destinations of DDoS attacks.

Figure 1: Agent Architecture

3.1 Sensor and Decision Agents for Intrusion Detection

In our test domain, the CSA analyses link information on a particular node in the network. It is advisable to use a representative node inside the network topology such as a backbone router or a border router in front of the node or service to protect from intrusions. Specifically this agent samples link utilisation in bytes per second, the size of the queue in packets and the number of packets drop by the queue. These three metrics (link utilisation, queue size and packets drop) are what we call our feature domain. To obtain a state representation of the network according with these features we use tile coding. The DSA monitors TCP connections between nodes. As previously stated DoS and DDoS attacks modify the normal behaviour of the network in many ways. Some of these changes can be spotted analysing TCP information from connections in the path of the attack. This agent has the same internal structure than the CSA but its feature domain is different. The features analysed for the TCP connections are the average number of ACK packets received, the average window size and the average Round Trip Time (RTT).

The FSA has a different internal structure than the other sensor agents as can be seen in Fig.1. This agent is composed by two logical sub-agents, the Flow Monitor (FM) and the Flow Aggregator (FA). The FM analyses the traffic flows that pass through the FSA and its feature domain is composed by protocol number, port number and the average packet size of the flow. Using this information the FM learns which flows are normal traffic and which ones may lead to an attack. The FA aggregates flow information by keeping a flow table with the signals reported by the FM. The feature domain of the FA is very simple. It is the number of attack flows reported by the FM. Finally the original DA described previously does not suffer any modification in its structure, functionality or operation.

3.2 Architecture Nuts and Bolts

To process the input features to generate the action signals we use tile coding. At the beginning of the algorithm each
agent initialises the weight array \( \vec{w} \) and the value function \( V(\pi(s)) \) of each action \( a \) to 0. At any given time \( t \) and state \( s_t \), the agent takes the values of the \( n \) monitored features. The features, the number of tiles and the divisions per tile (tilings) are processed by a subroutine \( \text{GetTiles} \) [21]. The output of the subroutine is the index of weights \( \vec{w} \) activated by the input features. Using this information the agent sums the activated weights in each action. Following a modified Boltzmann strategy the agent selects the action-signal. In this modified Boltzmann strategy we apply a decay factor \( T \) during the first 50% of the simulation time. In the rest of the simulation the agents follow a greedy policy (i.e. no exploration, only exploitation). If the action is executed by the DA, this is sent to the network operator and the agents receive a reward \( r \). The reward is positive if the action was correct, it is negative otherwise. On time \( t + 1 \) and state \( s_t+1 \) the agents updates the activated weights according to Eq. 2.

\[
\vec{w}_t = \vec{w}_{t-1} + \alpha \left( \text{Target} - \sum \vec{w} \right) \tag{2}
\]

where:

\[
\text{Target} = \sum \vec{w} + \alpha \left( r - \sum \vec{w} \right) \tag{3}
\]

The process is repeated every time step (in our experiments 200 ms) until \( t_{final} \). On \( t_{final} \) we stop the learning process and we store the values of the weight array \( \vec{w} \) for future use.

4. TESTS AND RESULTS

To find out whether the agent architecture along with the proposed learning process were capable of detecting abnormal states of the network we performed a series of tests. To add some realistic elements and the complexity of real applications, in this paper we used the network simulator ns-2 [20]. We generated the network topology of Fig. 4 composed by 7 agents or nodes. Node 0 generates normal FTP-like traffic while node 1 produces normal UDP traffic. Node 4 is an attacker producing a flood of UDP traffic. Node 5 is logically divided in two RL sensor agents, one CSA and one FSA. Their tasks are to forward traffic and collect data about the network. Node 6 is the DA and it solely works as a RL agent. Finally Node 3 is the DSA. It receives valid data from nodes 0 and 1 and it is the node under attack as well.

![Figure 2: Tested Network](image)

### 4.1 Intrusion Detection Metrics

To measure the success of the performed tests we used a variety of metrics (See Table 1). The most common metrics used to measure the detection performance of IDS are the false alarm rate and the attack detection rate. The false alarm rate is the fraction of the total alarms that do not represent an intrusion. We will refer to them as False Positives (FP) as well. False Negatives is the fraction of the total number of intrusions that were not categorized as intrusion. The intrusion detection rate or precision is the fraction of the total number of alarms that were identified as intrusions. To assist us in the design and evaluation of our results we also introduced other prediction metrics commonly used in bioinformatics and machine learning. Recall is introduced to show the number of malicious events that the IDS fail to categorise as negative instances. To verify that the IDS is learning how to detect attacks this measure is important to observe. In a similar fashion, accuracy relates all the variables together to an intuitive idea of the performance of the IDS system in relation with the number of correct events categorised. It is important to mention that all the described measures will not properly reflect performance well where the probability of intrusion is very low.

**Table 1: Performance Metrics 1**

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Positive Rate</td>
<td>FP / (TP + FP)</td>
<td>The fraction of non negative instances that was predicted as negative</td>
</tr>
<tr>
<td>Intrusion Detection Rate</td>
<td>TP / (TP + FP)</td>
<td>The percentage of negative labeled instances that was predicted as negative</td>
</tr>
<tr>
<td>Events</td>
<td>TP + TN + FP</td>
<td>The total number of events</td>
</tr>
<tr>
<td>Accuracy</td>
<td>(TP + TN) / (TP + TN + FP + FN)</td>
<td>The percentage of positive predictions that is correct</td>
</tr>
<tr>
<td>Recall</td>
<td>TP / (TP + FN)</td>
<td>The percentage of negative labeled instances that was predicted as negative</td>
</tr>
<tr>
<td>Specificity</td>
<td>TN / (TN + FP)</td>
<td>The percentage of predictions that is correct</td>
</tr>
</tbody>
</table>

### 4.2 Tests

We set up several tests to verify the learning capabilities of the agents as shown in Table 2. We used a control test (Baseline) to train the agents to categorise basic normal and abnormal activity in the network. To simulate the normal traffic we randomly started and stopped connections from node 0 (TCP/FTP) and node 1 (UDP stream). Using another random pattern of connections we used node 4 to simulate the attacks to the network characterised by a flood of UDP traffic. At time \( t = 0 \) each one of the agents started gathering information from the network and learning as previously explained. At time \( t_{final} \) we stopped the learning process and we stored the values of the weight array \( \vec{w} \) in order to use them in each one of the tests of Table 2.
To evaluate the adaptability of the agents we ran test 2 to 8. During these tests the agents are not learning anymore and they are exploiting the knowledge acquired during the training with the baseline test (test 1). Test 2 considers an identical network topology as in test 1 but with different traffic patterns. In this test we modify the start-stop times of the data traffic from the no-attack and attack nodes. Tests 3 to 5 were designed to create a more complex scenario where the attacker changes its attack to mimic authorised or normal traffic. Test 3 simulates when the attacker changes the attack port to any other given port while in test 4 we change the attack port to be the same as the authorised application. In test 5 we simulated when the attacker goes further and changes the attack port and the packet size to mimic the no-attack application. Tests 6 to 8 modify the network topology adding more sources of traffic. In test 6 we added multiple UDP sources and in test 7 we added multiple FTP sources, both of them are valid applications. Finally in test 8 we added multiple attackers as UDP sources to simulate a DDoS attack.

### 4.3 Results

Unless stated otherwise, we have performed our entire set of tests using the feature domains for sensor and decision agents described in section 2. In Fig. 3 we show the performed tests (1 to 8) evaluated using false positive rate, recall and accuracy. A low false positive rate indicates that our agents will not overwhelm the network operator. A high recall indicates that the agents are able to identify attacks while they maintain a low number of false negatives (unidentified attacks). Finally a high level of accuracy indicates that the system is capable of identifying attacks while generating few false positives. The intrusion detection rate (IDR) is another important metric but it can be misleading given a certain type of traffic (e.g. the IDR can be high when the system recognises few attacks but the number of FP is low). It is important to observe that even in test 1, which is our training environment, the agents do not perform without any error. This is due to a synchronisation problem between the information received by sensor agents and it being processed by the DA. Sensor agents receive input information at time $t$, they process it and send their action signals to the decision agent. At time $t + 1$ the decision agent processes the sensor agent’s signals resulting from the information retrieved at time $t$. The FP and FN in test 1 are the result of a DA executing actions in $t + 1$ with network information of $t$. As long as the time step difference remains short these errors are insignificant.

Excluding test 5, the remaining tests show acceptable levels of all the intrusion detection metrics including accuracy and recall. Test 2 shows remarkably good levels of accuracy and recall as result of the modified traffic pattern with longer and fewer no-attack/attack cycles. A smaller number of no-attack/attack cycles means a small number of FN and FP due to the synchronisation issue between the DA and the collected network information. Contrary to test 2, tests 5 shows a low level of accuracy and recall. Remember that test 5 simulates when the attacker changes the information of the IP packet (protocol, port and packet size) of its attack to mimic a valid connection. In this case when there is an attack the FA interprets the flow information as a no-attack. However the CSA and DSA interpret the network information correctly. When the action signals are transmitted to the DA any of these scenarios may happen:

1. Even though the FA is reporting a no-attack, the signals for the CSA and DSA activate the DA weights that trigger an alarm-action.
2. The signals for the CSA and DSA are not strong enough to activate the alarm-action and the DA triggers a no-alarm-action.
3. The signals from the sensor agents activate weights with similar values for both actions and the DA trigger a do-not-know action. In other words, a do-not-know action denotes that the DA does not have enough evidence to trigger a committed action such as an alarm or no-alarm.

In test 5 when the attacks start the congestion and delay value measured by sensor agents are similar to the no-attack states causing the DA to trigger an incorrect action generating a FN. As the attack progresses the congestion and delay information make the current state appear to the DA as a no-attack but not strong enough to trigger a no-alarm action. Instead, the DA triggers the do-not-know action. Finally when the attack is at its peak, the signals from the DSA and CSA make the value of the alarm action better than the no-alarm and the DA triggers the alarm. A similar behaviour takes places when the attacker slows down its attack.

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baseline test</td>
</tr>
<tr>
<td>2</td>
<td>Traffic pattern change</td>
</tr>
<tr>
<td>3</td>
<td>Attack port change</td>
</tr>
<tr>
<td>4</td>
<td>Attack port same as no attack application</td>
</tr>
<tr>
<td>5</td>
<td>Attack port and packet size same as no attack application</td>
</tr>
<tr>
<td>6</td>
<td>Multiple valid UDP sources</td>
</tr>
<tr>
<td>7</td>
<td>Multiple valid FTP sources</td>
</tr>
<tr>
<td>8</td>
<td>Multiple attack sources</td>
</tr>
</tbody>
</table>
Trying to improve the intrusion detection metrics we ran more tests changing the feature domain of the sensor agents. This task showed the complexity of choosing the correct set of features. While some sets improved the metric for some tests, they also showed worse results for other tests. We end up with the combination of features described in the past section. This set was yield the best results overall. As in any other fields where machine learning is used the selection of features is a complex and difficult process. Further research in this area is necessary to guaranty reliable intrusion detection in more general environments.

We compared our learning algorithm against two hard-coded approaches. The first hard-coded approach (Hard-Coded 1) emulated a misuse IDS. In this case the IDS is looking for the patterns that match an attack. The Hard-Coded 2 approach integrates the same variety of input information as our learning algorithm. We evaluated the learning and hard-coded approaches using test 2 and test 5. We used test 2 because it only changes the traffic pattern of the attack and it must be very simple to detect. Attacks in test 5 are the hardest to detect because they emulate some of the signatures of normal traffic. The learning curves of the test are shown in Fig. 4. Hard-Coded 1 had no problem to identify attacks and have low false negatives for test 2 but it completely failed to detect attacks test 5. This is the same problem that misuse IDS have when the signature of the attack changes or when they face unknown attacks. The results for Hard-Coded 2 and our learning approach confirm our argument that for more reliable intrusion detection we need a variety of information sources. Both solutions were capable of detecting the attacks even though one of the sensors was reporting incorrect information. This scenario also could be seen as the emulation of a broken sensor sending bogus information or a sensor compromised by the attacker and forced to send misleading signals. Either way it demonstrates that a system using more than one source to detect intrusions could be more reliable than single-source IDS.

Both the Hard-coded 2 and learning approaches present very good results regarding the identification of normal and abnormal states in the network. While the learning algorithm requires some time to learn to recognize normal and abnormal activity, it does not require any previous knowledge about the behaviour of the measured variables. Hand-coded 2 reaches maximum performance since the beginning of the simulation but it requires in-depth knowledge from the policy programmer about the the network traffic and the variables measured to detect intrusions.

5. RELATED WORK

Problems such as the curse of dimensionality; partial observability and scalability in MARL have been analysed using a variety of methods and techniques [15, 18] and they represent the foundation of our research. More recent work related with our research include the use of Hierarchical Reinforcement Learning [8], learning automata [11] and game theory [16].

An application of MARL to networking environments is presented in [5] where cooperative agents learn how to route packets using optimal paths. Using the same approach of flow control and feedback from the environment, other researchers have expanded the use of RL in routing algorithms [14], explore the use of MARL to control congestion in networks [6, 9], routing using QoS [7] and more recently to control DDoS attacks [24].

The use of RL in the intrusion detection field has not been widely studied and even less in distributed intrusion detection. Some research works are [4, 5] where the authors trained a neural network using RL and applying CMAC as the function approximation technique and [1] where game theory is used to train agents to recognise DoS attacks against routing infrastructure. Other recent research work include the use of RL to detect host intrusion using sequence system calls [25] and the previously mentioned [24].

6. CONCLUSION AND FUTURE WORK

In this paper we have shown how a group of agents can coordinate their actions to reach the common goal of network intrusion detection. During this process decision agents learn how to interpret the action-signals sent by sensor agents without any previously assigned semantics. These action-signals aggregate the partial information received by sensor agents and they are used by the decision agents to reconstruct the global state of the environment. In our case study, we evaluate our learning approach by identifying normal and abnormal states of a realistic network subjected to various DoS attacks. Overall the following conclusions can be drawn:

- We have successfully applied RL in a group of network agents under conditions of partial observability, restricted communication and global rewards in a realistic network simulation.
- The use of a variety of network data has generated good results to identify the state of the network. The system presents high reliability even in cases when some sensor information is missing or compromised.
- Finding the correct feature domain for each agent is not a trivial task. To find the state representation that allows identification of the status of the network requires a long and complex trial and error process.
• The learning approach yields better results than the simple hand coded alternative. It also yields similar results to a more complex hand coded alternative using a variety of sensor information. The main advantage of the learning approach is that it does not need a trainer with prior knowledge of the network environment.

Future work include scaling up our learning approach to a large number of agents using the hierarchical approach. This architecture will be similar to our previous research work in intrusion detection with the addition of more real world challenges such as the use of function approximation techniques or its implementation in a highly realistic simulation environment. The new characteristics will allow us to create more complex network topologies emulating geographical cells of agents, security domains composed by cells or groups of cells, complex DDoS attacks and eventually the emulation of real packet streams inside the network environment.

Finally, IDS have been criticised of being a passive alternative that only detects intrusion but it does not take any action to stop it. Recent trends in IDS take these systems to a more active role to detect, stop and prevent intrusions. Following this path a next step in our research is to adapt our agent architecture to detect and respond to abnormal states of the network. This may allow us to generate some network information as the reward signal for the agents' feedback. This would result in an active architecture of agents, eliminating the manually generated feedback currently used as reward.

7. REFERENCES
ABSTRACT
Lightweight agents distributed in space have the potential to solve many complex problems. In this paper, we examine a model where agents represent individuals in a genetic algorithm (GA) solving a shared problem. We examine two questions: (1) How does the network density of connections between agents affect the performance of the systems? (2) How does the interaction topology affect the performance of the system? In our model, agents exist in either a random network topology with long-distance communication, or a location-based topology, where agents only communicate with near neighbors. We examine both fixed and dynamic networks. Within the context of our investigation, our initial results indicate that relatively low network density achieves the same results as a panmictic, or fully connected, population. Additionally, we find that dynamic networks outperform fixed networks, and that random network topologies outperform proximity-based network topologies. We conclude by showing how this model can be useful not only for multi-agent learning, but also for genetic algorithms, agent-based simulation and models of diffusion of innovation.

Categories and Subject Descriptors: I.2.m [Artificial Intelligence] Misc.

General Terms: Algorithms

Keywords: Multi-Agent Learning, Genetic Algorithms, Networks, Innovation, Diffusion

1. MOTIVATION
Given lightweight agents, that is agents without a large amount of computational power, and an interaction topology that allows those agents to communicate, how well can these agents perform at solving a shared problem? The general idea is that it can be difficult for a simple agent to solve a complex problem on its own, but working together a group of simple agents may be able to solve the problem. This question is interesting for a number of fields including multi-agent learning, robotics, and agent-based modeling (ABM). In this paper, we will assume that we have a group of agents who can propose solutions to a problem and evaluate how good that solution is when compared to other solutions. In addition, we will assume that these agents have the ability to compare their solutions with a subset of the other agents in the group, and they can copy or combine other agent’s solutions in an effort to improve their own solution.

If we assume that the problem is tractable enough that it can be solved by such a group of agents in a reasonable amount of time, then a natural question to ask is: How do the characteristics of the interaction topology affect the performance of the group, i.e., the time until the discovery of an optimal solution? Using this question as our motivation, we further refined the question to two sub-questions: (1) how does the network density of the topology affect the group performance, and (2) how does the structure and dynamics of the network affect the group performance? In the first question, we asked how does the number of agents each agent communicates with affects the overall performance of the group. In the second question, we asked if it matters which agents each agent interacts with, and how relationships between agents can affect group performance.

In this paper, we explore these questions by examining how a group of distributed individual agents can operate as a genetic algorithm (GA) to solve a problem. We will allow these individuals to be influenced and to influence a set of other individuals in the group, specified by a network topology. Our parameters of interest will be how many other individuals each agent communicates with, and how these other individuals are chosen.

We will begin by discussing related work. Then we will present the model that we used to examine these two questions and the initial results of two experiments. We will conclude by discussing how this model applies to a wide range of phenomena, and discuss future research.

2. RELATED WORK
As we mentioned, the way we have operationalized our driving questions is by examining a group of agents that operate together as a GA. As a result, there is relevant research in the field of evolutionary computation (EC). In particular, since we are assuming that the agents can eventually
solve the problem that we are giving them, the measurement of “takeover time” from EC is relevant. Takeover time is the amount of time it takes for a good solution to spread throughout a population. There has been work in the past that has looked at how takeover time is affected by different network topologies [8] [9] [3]. However, this work has focused on how a perfect solution percolates through a fixed network and not how network topologies affect the search for a perfect solution. Our model is also related to the study of spatially structured populations in GAs (e.g., evolution taking place on lattices) [12]. To date, however, the majority of research has focussed on fixed network structures. We are interested in comparing fixed and dynamic networks.

Besides EC, there is work in network theory that is relevant, specifically with respect to the diffusion of innovation that has influenced the development of the model that we are presenting. In particular, Watts’s work on how innovations diffuse on networks [14]. However, this work also assumes that there is only one innovation, and the focus is on the spread of that innovation. Our research seeks to understand how the network affects the development of innovative solutions, and not just their diffusion. Moreover, Watts’s focused on fixed random networks, we are interested in examining a wider range of networks. There has also been interest in understanding how processes of diffusion operate on dynamic network structures. Moody [7], for instance, has criticized the practice of studying features of fixed networks while ignoring the issue of timing and the dynamic nature of real-world data. Since our work is more theoretical than Moody’s, we aim to provide more general guidelines to practitioners of social network analysis about the ways in which dynamic networks differ from their fixed counterparts.

3. THE MODEL AND SETUP

As we mentioned, to operationalize our examination of how network interactions and topology affect the performance of agents solving a shared problem, we construct a GA constituted of agents. In a standard GA [4], agents represent a solution to a problem; they are evaluated on how well they solve the problem, and better solutions continue to propagate through cloning or by recombining with other agents in the population. We will use this simple description as our starting point, but instead of agents being potential solutions to problems, our agents each possess one potential solution. The agents in our model do not contain genetic algorithms, but they participate in a genetic algorithm. The population of 256 agents works together to solve a problem. Each agent has a local neighborhood (defined by the network topologies described below) of other agents that can share information with it. During each generation agents examine the solutions of a uniformly random subset (3 agents in the experiments below) of their local neighbors (which include themselves) and choose the best solution in that subset. This selection process resembles tournament selection in a standard GA, but the possible subsets are restricted by the local network neighborhood. With a certain probability (30% in the experiments below) the agents take the selected solution for their own. Otherwise (i.e., 70% of the time), they carry out the selection process again, and combine the two solutions that they have selected using one-point crossover [4]. After all agents have chosen a new solution, each agent probabilistically mutates its solution. Mutation is per-bit at a rate of \( \frac{1}{l} \), where \( l \) is the length of the solution string.

In these experiments, we examine a population of agents trying to solve a bit-matching problem called the hyperplane-defined function. Hyperplane defined functions (hdf’s) were constructed to facilitate the study of GAs. The difference between this new test suite and most other test functions is that the underlying representation of this suite is schemata [4]. By utilizing functions that reflect the way the GA searches, the performance of the GA can be easily observed. Created by Holland, the hdf’s [5] are an extension of the Royal Road functions developed by Mitchell et al. [6].

An hdf is composed of positive schemata and negative “pothole” schemata. For each schema that is matched by an agent’s solution, the agent is rewarded (the fitness value is incremented). And for each pothole that is matched, the individual is punished (the fitness value is decreased). There are elementary level schemata, which are the foundational elements, and intermediate-level schemata, which are composed of pieces of the elementary schemata.

We examined two hdf’s with two levels of difficulty. These two hdf’s are restricted instances of the general class of hdf’s and corresponded to static instances of the shaky ladder hyperplane-defined functions (sl-hdf’s) [10]. The first problem was a 100-bit hdf that contained 10 elementary schemata, 7 intermediate schemata, and 10 potholes. Elementary schemata were of length 10 and order 8. We also used a 200-bit hdf that contained 20 elementary schemata, 17 intermediate schemata, and 20 pot-holes. Elementary schemata were of length 20 and order 8. The 100-bit hdf is substantially easier than the 200-bit hdf. Because of the way the sl-hdf’s are constructed we know a priori their optimal value and optimal string set [10].

In the simple GA the population is panmictic, meaning that any solution may breed with any other solution to create offspring. In this model, we restrict the breeding neighborhoods by imposing several types of network structure on the population. In particular, we consider two types of network structures: (1) random networks and (2) “geographically” defined proximity networks among agents distributed on a torus world. We will refer to this first type of network as a random or long-distance network, since the locations of two agents in space is irrelevant to whether or not they can communicate. The neighborhood for each agent in the random networks is a random selection of other agents in the population; in particular, the structure is created using the Erdős-Rényi model [2] with an edge probability equal to the expected network density rate being examined. The expected network density is the key parameter of control throughout all of our experiments. An instance of a random network is shown in Figure 1. We will refer to the second type of network as a proximity-based network since agents only communicate with other agents that are proximate to them on the basis of some distance-measure. In these networks, agents are scattered randomly on a wrapping bounded (toroidal) world. The neighborhood for each

---

1. Agents only know the performance of their solutions relative to others; they never know their absolute performance.

2. We are agnostic as to whether this distance represents a physical or trait-based distance. Sometime these proximity networks are referred to in the organizational science literature as “silo” networks, since the individuals are biased toward interacting within their contained silos.
agent is defined by a communication radius; a link is created between two agents if the distance between them is less than this radius. An instance of this type of network is visible in Figure 2. We compute the communication radius such that the circular neighborhood will contain, on average, the correct number of agents to achieve the desired expected network density. This allows us to directly compare the results from the same expected network densities in both the random and proximity-based networks.

We also considered one additional parameter, which was whether the networks are fixed or dynamic. In fixed networks agents always communicate with the same other agents each generation, i.e., the breeding neighborhoods for the genetic algorithm remain constant. In dynamic networks, agents may communicate with different agents every generation, i.e., the breeding neighborhoods change. These two factors together result in four network topologies that we will examine: (1) fixed proximity-based, (2) fixed random, (3) dynamic proximity-based, and (4) dynamic random.

Proximity-based and random networks used different forms of dynamics. In dynamic proximity-based networks, agents move slowly (about 1% of the world's diameter) forward across the world, turning randomly (up to 15 degrees right or left) each generation. Agents' local neighborhoods update based on the agents currently within their communication radius. In dynamic random networks, a new network structure is generated each generation, using the same Erdős-Rényi model as described above. These particular forms of dynamics are a subset of a larger class of dynamics. For example, instead of reassigning every link in the dynamic random networks we could change a smaller fraction of links each generation, and in the proximity-based networks, the rate and method of agent movement could be altered.

We chose these two network topologies to be representative of two ends of a spectrum from unstructured (random) to structured (proximity-based) topologies. In future work, these networks topologies will act as base cases for comparison to other topologies. We were also interested in the effects of dynamics on networks, and examining these base cases assists in isolating the effects of dynamics from other complicating factors of network structure.

We conducted two experiments with this model to investigate how the density affects the overall performance of the GA. All of the components of the model are implemented in NetLogo [16]. In all of the experiments described below, we will run the model with a different level of density until an optimum solution is found, and we measure how long it takes to find that optimum solution, or we give up if it takes more than 3000 generations.

4. EXPERIMENT 1: NETWORK DENSITY

In the first experiment, we vary the network density from 0% to 100% at 5% increments, and examine all four network topologies. We also examine both the easy problem (100 bit hdf) and the harder problem (200 bit hdf). For each parameter set we ran the model sixty (60) times, and record the time until a perfect solution is found. We average the results, and present them for the easy problem in Figure 3 and for the harder problem in Figure 4.

4.1 Experiment 1 Results

Most runs find the optimum well before 3000 generations.

Figure 1: Random Network with density of 0.01 after discovery of optimum. Agents are colored according to the fitness of the solution they possess. Higher luminance values indicate better solutions.

Figure 2: Proximity Network with density of 0.01 after discovery of optimum. Agents are colored according to the fitness of the solution they possess. Higher luminance values indicate better solutions.
In the easier problem (Figure 3), it can be seen that all four topologies result in similar performance. In all four cases, as long as the network density is at or above 5%, the population is able to find solution in roughly the same amount of average time, around 250 generations.

In the harder problem (Figure 4), a similar qualitative phenomenon can be observed. With a network density 5% or greater, the population is always able to find a solution somewhere between 1000 and 1500 generations. The results on this harder problem are less consistent than they are on the easier problem, and there is no discernible pattern as to when the topologies perform differently.

4.2 Experiment 1 Discussion

Despite the lack of differences in the network topologies there are several interesting phenomenon to observe. First of all, the system is robust, even for sparse networks (≤ 5% density) it performs equivalent to 100% network density.

This panmictic (all mixing) population is equivalent to global communication. These results indicate that there is no reason that every individual needs to be in communication with every other individual in the population. Instead, each individual needs to only communicate with a small local neighborhood, regardless of the interaction topology.

It should also be noted that 0% network density is the same as every agent working alone. In this model, agents are only able to measure the performance of their solution indirectly, by comparing it to the performance of solutions of other agents they communicate with. Thus, agents operating alone wind up mutating bits at random.

Values of network density between 0% and 100% reflect varying degrees of local communication. The results of these experiments illustrate that little local communication is necessary to achieve the same effects as global communication. Moreover, these results seem to hold independent of the difficulty of the problem. In applications, this might guide decision-making about the amount of effort that should be invested in inter-agent communication.

5. EXPERIMENT 2: A CLOSER LOOK

We designed a follow-up experiment to determine two things. First, at what network density (between 0% and 5%) does agent communication break down significantly enough that the group problem solving efficiency is diminished. Second, how do the different network topologies affect the performance of the system, given very low network densities. In this experiment, we varied the network density from 0% to 5% at 0.1% increments, and again examined results for all four network topologies on both the easier (100 bit hdf) and the harder (200 bit hdf) problems. For each parameter set we ran the model sixty (60) times, and recorded the time until a perfect solution is found. We averaged the results, and present them for the easy problem in Figure 5 and for the harder problem in Figure 6. Only network densities between 0% to 3% are shown, to highlight the region of interest.

Additionally, we measured the diversity of the population’s solutions at the end of the run (3000 generations). Our diversity measure was the average pairwise Hamming distance between agents’ solutions (strings of bits), normalized to be between 0.0 (completely homogenous) and 1.0 (maximally diverse).2 We average the results of the 60 repetitions of the model, and present them for the easy problem in Figure 7 and for the harder problem in Figure 8.

5.1 Experiment 2 Results

In the easier problem (Figure 5), we now find a difference in the performances of the four topologies. The dynamic random structure requires the least amount of network density in order to achieve the same results as the panmictic population. After this, the dynamic proximity and fixed random structures have equivalent behavior. Finally, the fixed proximity structure has the worst performance of the four interaction topologies. However, in all four cases, as long as the network density is over 1.8%, the population is able to find a solution in roughly the same amount of average time, around 250 generations.

This is one of many possible diversity measures.
In the harder problem (Figure 6), a similar qualitative phenomenon can be observed. The dynamic random structure performs the best, with the fixed random and dynamic proximity structures coming in second, and the fixed proximity structures performing the worst. However, with a network density greater than 1.9%, the population is consistently able (on average) to find a solution somewhere between 1000 and 1500 generations regardless of the network structure. Again, the results on this harder problem are less consistent than they are on the easier problem, and there is no statistically significant pattern as to when the topologies perform differently after this 1.9%.

The diversity plots (Figures 7 and 8), show that decreases in diversity occur in the same order as the decreases in time to optimal. As the expected network density increases, first the dynamic random network diversity decreases, then the fixed random and the dynamic proximity, with the fixed proximity having the slowest decrease in diversity. Moreover, these decreases in diversity occur at roughly the same network densities as the decreases in time to optimal.

We ran additional experiments identical to this one, except with variations on the selection process (i.e., choosing the best of two (2) or four (4) solutions from the neighborhood, instead of three (3), or using "roulette" selection which chooses probabilistically from the whole local neighborhood proportional to each solution’s fitness). These experiments yielded the same qualitative results as those presented here.

5.2 Experiment 2 Discussion

There are several clear results from this data. First, dynamic topologies require less network density to achieve the same level of performance as do fixed topologies. Second, random topologies require less network density to achieve the same level of performance as do proximity-based topologies. Moreover, these results appear to be independent of the difficulty of the problem.

Several hypotheses explain these results. First, proximity-based fixed topologies are still segmented at higher network densities than the other topologies. Since the network is defined by other agents within a certain local radius, at low levels of density it is common for not all of the agents to be connected to each other in one giant component. These isolated components cannot share information with each other, which prevents global coordination to collectively solve the problem. Dynamic topologies are not subject to this problem because they are constantly changing their network connections so isolated groups will eventually be connected to the rest of the group. The fixed random network topology is also less susceptible to this problem because the construction of the random networks means that neighbors of agents are not likely to be connected to other neighbors of the same agent, i.e., they have a low clustering coefficient. As a result, for fairly low network densities there is still a giant component connecting many of the agents.

A second hypothesis is that the key ingredient to successful group problem solving is the ability for good solutions to propagate quickly through the whole population. One important measure for this on fixed networks is the average length of the path between any two nodes in the network (average path length). Random networks have a shorter average path length than proximity-based networks. In dynamic networks, average path length is not well-defined, but since agents are constantly changing their partners, the number of generations it takes for an innovative idea to spread across the network should be smaller.

A third hypothesis is similar to the second, but instead the emphasis is on the initial rapidity with which a solution can reach a reasonably broad audience. In particular, if the clustering coefficient of the network is too high, then agents will pass information to their neighbors, who will mostly pass it on to other agents that haven’t been exposed to it yet. Random networks have lower clustering coefficients than proximity-based networks. In the dynamic topologies, the clustering coefficient is not well-defined, but since agents are not exposed to the same individuals every generation, good information is more likely to initially spread more quickly.

The diversity results support these hypotheses that the key to solving these problems is to have good communication across the network. High diversity indicates that there are pockets of the network that are not communicating well with other pockets of the network and thus have evolved their own different solutions to the problem. When a topology promotes quick communication of good ideas across the network then you would expect less diversity than when it takes more time for these partial good solutions to permeate.

These results clearly indicate that random-network topologies outperform proximity-based network topologies, and that dynamic topologies outperform fixed topologies.

6. CONCLUSIONS AND FUTURE WORK

We have presented a model of multi-agent learning that is embedded in a network context, and have discussed three main results. First, the system is robust (maintains optimal performance) for a large range of network densities ($\geq 2\%$), but below a certain density threshold the performance decreases sharply (i.e., there is a phase transition). Second, this threshold is lower for the dynamic networks than for the fixed networks. Third, this threshold is lower for the random topology than the proximity topology.

Beyond these specific research questions, our research has potential applications in three different disciplines: evolutionary computation (EC), social science, and agent-based modeling (ABM). From an EC perspective, the presented model is an investigation into the effect of various breeding networks in a GA. Such an investigation allows researchers interested in GAs to develop a new understanding of how breeding topologies affect the performance of the GA, and is directly relevant to researchers interested in the question of takeover time in GAs [8]. Within this field our findings supports the idea that panmictic populations are not necessary in order to achieve maximal performance in a GA. There may also be ways to use breeding topologies to increase the performance of a GA. Thus, these results may have applications in the design of robust distributed GAs.

From a social science perspective, this model examines the diffusion of innovation in social networks. The agents can be viewed as individuals in an organization, where each individual is involved in the process that is called “reinvention” in the diffusion of innovation literature [11]. Reinvention is where innovations are modified by individuals in order to solve a new problem, or to solve an existing problem better. Our results indicate that little communication between individuals is necessary for reinvention to work. Long-range (i.e., cross-silo) communication is superior to proximity-based communication. Dynamic communication
Figure 5: Average time to optimum versus network density from 0% to 3% for all four network topologies on the 100-bit hdf problem. Standard error bars are shown.

Figure 6: Average time to optimum versus network density from 0% to 3% for all four network topologies on the 200-bit hdf problem. Standard error bars are shown.
Figure 7: Average final solution diversity versus network density from 0% to 3% for all four network topologies on the 100-bit hdf problem. Standard error bars are shown (but may be too small to be discernable).

Figure 8: Average final solution diversity versus network density from 0% to 3% for all four network topologies on the 200-bit hdf problem. Standard error bars are shown (but may be too small to be discernable).
networks are also beneficial; talking to different people everyday facilitates reinvention.

From an ABM perspective, we consider how several types of agent interaction dynamics affect the exchange of information. This model can be viewed generally as a model of agent communication, and our results describe how levels of communication influence performance. This model provides researchers, who are interested in communication processes within ABMs, guidelines for what to expect based upon how their agents are distributed and how the agents communicate. For instance, for engineers interested in using a cloud of “smart-dust” to collectively solve difficult problems, our results recommend choices for communication topologies and connectivity requirements to achieve success.

There are many additional questions that could be considered when examining this model. For instance, we speculated that dynamic topologies can more quickly distribute information across the network, and that they are exposed to a wider range of information quickly. However, as mentioned above, the average path lengths and clustering coefficients are not well-defined for dynamic networks and so the creation of dynamic or generalized versions of these measures might facilitate the understanding of how dynamic network structures affect performance of these systems.

Another phenomena that warrants investigation is the role that particular nodes play in the system performance. Researchers interested in the diffusion of innovation would be particularly interested in this question. It has been speculated that “influential” nodes play a key role in the diffusion of innovation, though this role has recently been brought under speculation [15]. Influentials could be examined as highly connected nodes, or as nodes with special properties (e.g., nodes that only innovate and do not copy from other nodes), and this model provides a computationally tractable way of asking this question.

We have also only investigated two classes of network structures (proximity and random) in this model. Additional topologies like small-world topologies [13] and scale-free networks [1] should be investigated. Since scale-free and small-world both maintain higher clustering coefficients than expected given their lower average path lengths, they may provide a way for agents in networks to gain the benefits of local communication, and long-distance communication at the same time. Evaluating performance on these networks would also help us diagnose the primary structural factors that contribute to stable multi-agent learning in our model. At very low network densities, how much is performance controlled by average path length as opposed to clustering coefficient? If we could answer this question, then we may be able to design better multi-agent learning systems.

Besides changing the network parameters, we could also vary the GA parameters. For instance, we could examine the interaction between the mutation rate and the network structure. A high mutation rate allows for a greater probability of innovation, but it can also cause good solutions to be destroyed before spreading through the population. However, high network densities may lessen the negative effects of mutation by raising the rate of spreading, suggesting a trade-off between mutation rate and network density.

There are many interesting questions that can be addressed by this model. In general, the goal of our research project is to take a step back from particular applications and build higher-level models that can be employed in a wide variety of circumstances. By studying these “meta-models” we can provide advice, guidelines, and frameworks to researchers interested in a wide variety of fields.

Acknowledgments: We thank the Northwestern Institute on Complex Systems for providing support for WR, and Luis Amaral for access to his Computing Cluster.

7. REFERENCES

Transferring Instances for Model-Based Reinforcement Learning

Matthew E. Taylor, Nicholas K. Jong, and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{mtaylor, nkj, pstone}@cs.utexas.edu

ABSTRACT
Reinforcement learning agents typically require a significant amount of data before performing well on complex tasks. Transfer learning methods have made progress reducing sample complexity, but they have only been applied to model-free learning methods, not more data-efficient model-based learning methods. This paper introduces TIMBREL, a novel method capable of transferring information effectively into a model-based reinforcement learning algorithm. We demonstrate that TIMBREL can significantly improve the sample complexity and asymptotic performance of a model-based algorithm when learning in a continuous state space.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning

General Terms
Algorithms, Experimentation

Keywords
Transfer Learning, Model Transfer, Reinforcement Learning

1. INTRODUCTION
In many situations, an agent must learn to execute a series of sequential actions, which is typically framed as a reinforcement learning [18] (RL) problem. Although RL approaches have enjoyed past successes (e.g., TDGammon [22], inverted Helicopter control [9], and robot locomotion [7]), they frequently take substantial amounts of data to learn a reasonable control policy. In many domains, collecting such data may be slow, expensive, or infeasible, motivating the need for sample-efficient learning methods.

One recent approach to speeding up RL so that it can be applied to difficult problems with large, continuous state spaces is transfer learning (TL). TL is a machine learning paradigm that reuses knowledge gathered in a previous source task to better learn a novel, but related, target task. Recent empirical successes in a variety of RL domains [12, 20, 23] have shown that transfer can significantly increase an agent’s ability to learn quickly, even if agents in the two tasks have different available sensors or actions. However, to the best of our knowledge, TL methods have thus far been applied only to model-free RL algorithms.

Model-free algorithms such as Q-Learning [24] and Sarsa [13, 15] learn to predict the utility of each action in different situations, but they do not learn the effects of actions. In contrast, model-based (or model-learning) methods, such as PEGASUS [10], R-MAX [3], and Fitted R-MAX [5], use their experience to learn an internal model of how the actions affect the agent and its environment, an approach empirically shown to often be more sample efficient. Such a model can be used in conjunction with dynamic programming [2] to perform off-line planning, often enabling superior action selection without requiring additional environmental samples. Building these models may be computationally intensive, but using CPU cycles to reduce data collection time is a highly favorable tradeoff in many domains (such as physically embodied agents). In order to further reduce sample complexity and ultimately allow RL to be applicable in more complex domains, this paper introduces Transferring Instances for Model-Based Reinforcement Learning (TIMBREL), a novel approach to combining TL with model-based RL.

The key insight behind TIMBREL is that data may be reused between different tasks. Data is first recorded in a source task, transformed so that it applies to a target task, and then used by the target task learner as it builds its model. In this paper we utilize Fitted R-MAX, an instance based model-learning algorithm, and show how TIMBREL can help construct a target task model by using source task data. TIMBREL combines the benefits of transfer with those of model-based learning to reduce sample complexity. It works in continuous state spaces and is applicable when the source and target tasks have different state variables and action spaces. We fully implement and test our method in a set of Mountain Car tasks, demonstrating that transfer can significantly reduce the sample complexity of learning.

The rest of this paper is organized as follows. Section 2 provides a brief background of RL and Fitted R-MAX, as well as discussing a selection of related TL methods. The experimental domain is detailed in Section 3. Section 4 introduces TIMBREL and discusses its implementation when using Fitted R-MAX. Experimental results are presented in Section 5. Section 6 discusses possible future directions and concludes.

2. BACKGROUND AND RELATED WORK
In this paper we use the notation of Markov decision processes (MDPs). At every time step the agent observes its state \( s \in S \) as a vector of \( k \) state variables such that \( s = (x_1, x_2, \ldots, x_k) \). In episodic tasks there is a starting state \( s_{\text{initial}} \) and often a goal state \( s_{\text{goal}} \), which terminates the episode if reached by the agent. The agent selects an action from the set of available actions \( A \) at every time step. The start and goal states may be generalized to sets of states. A task also defines the reward function \( R : S \times A \rightarrow \mathbb{R} \), and
the transition function \( T : S \times A \rightarrow S \) fully describes the dynamics of the system. The agent will attempt to maximize the long-term reward determined by the (initially unknown) reward function \( R \) and the (initially unknown) transition function \( T \).

A learner chooses which action to take in a state via a policy, \( \pi : S \rightarrow A \). \( \pi \) is modified by the learner over time to improve performance, which is defined as the expected total reward. Instead of learning \( \pi \) directly, many RL algorithms instead approximate the action-value function, \( Q : S \times A \rightarrow \mathbb{R} \), which maps state-action pairs to the expected real-valued return. If the agent has learned the optimal action-value function, it can select the optimal action from any state by executing the action with the highest action-value.

In this paper, we introduce and utilize TIMBREL to improve the performance of Fitted R-MAX [5], an algorithm that approximates the action-value function \( Q \) for large or infinite state spaces by constructing an MDP over a small (finite) sample of states \( X \subset S \). For each sample state \( x \in X \) and action \( a \in A \), Fitted R-MAX estimates the dynamics \( T(x, a) \) using all the available data for action \( a \) and for states \( s \) near \( x \). Some generalization from nearby states is necessary because we cannot expect the agent to be able to visit \( x \) enough times to try every action. As a result of this generalization process, Fitted R-MAX first approximates \( T(x, a) \) as a probability distribution over predicted successor states in \( S \). A value approximation step then approximates this distribution of states in \( S \) with a distribution of states in \( X \). The result is a stochastic MDP over a finite state space \( X \), with transition and reward functions derived from data in \( S \). Applying dynamic programming to this MDP yields an action-value function over \( X \times A \) that can be used to approximate the desired action-value function \( Q \). For the original 2D Mountain Car task, Fitted R-MAX learns policies using less data than many existing model-free algorithms [5].

Approaches that transfer between model-free RL algorithms are most closely related to TIMBREL. Torrey et al. [23] show how to automatically extract advice from a source task by identifying actions which have higher Q-values than other available actions; this advice is then mapped by a human to the target task as initial preferences given to the target task learner. In our previous work [20], we learn an action-value function for a source task, translate the function into a target task via a hand-coded inter-task mapping, and then use the transferred function to initialize the target task agent. Other work [12] shows that in relational reinforcement learning, object-specific action-value functions can be used for initialization when the number of objects change between the source and target tasks. In all three cases the transferred knowledge is effectively used to improve learning in the target task, but only using model-free learning methods that inherently require more data than model-based learning.

3. Generalized Mountain Car

This section introduces our experimental domain, a generalized version of the standard RL benchmark Mountain Car task [15]. Mountain Car is an appropriate testbed for TIMBREL with Fitted R-MAX because it is among the simplest continuous domains that can benefit from model-based learning, and it is easily generalizable to enable TL experiments.

In Mountain Car, the agent must generalize across continuous state variables in order to drive an underpowered car up a Mountain to a goal state. We also introduce 3D Mountain Car as extension

\(^1\)Fitted R-MAX is an instance-based learning method; our implementation currently retains all observed data to compute the model. In the future we plan to enhance the algorithm so that instances can be discarded without significantly decreasing model accuracy.

\(^2\)Both Mountain Car tasks are deterministic, as is Fitted R-MAX. To introduce randomness and allow multiple learning trials, when each domain is initialized, \( x \) (and \( y \) in 3D) in the start state is perturbed by a random number in \([-0.005, 0.005]\).

\(^3\)Available at http://rlai.cs.ualberta.ca/RLR/MountainCarBestseller.html

\(^4\)Although we call the agent’s vehicle a “car,” it does not turn but simply accelerates in the four cardinal directions.
should be able to help an agent learn the 3D task, we do expect that some amount of learning will be required after transfer.

3.3 Learning Mountain Car

Our experiments used Fitted R-MAX to learn policies in the Mountain Car tasks. We began by replicating the methods and result of applying Fitted R-MAX to 2D Mountain Car task as reported in the literature [5]. To apply Fitted R-MAX to 3D Mountain Car, we first scaled the state space so that each dimension ranges over the unit interval, effectively scaling the state space to a unit hypercube. We sampled a finite state space from this hypercube by applying a grid where each position state variable can be one of 8 values, and each velocity state variable can be one of 9 values. The 3D version of Mountain Car has 2 of each type of state variable; we obtained a sample X of 8² × 9² = 5184 states that approximated the original state space S. For any state x ∈ X and action a ∈ A, Fitted R-MAX estimates \( T(x, a) \) using a probability distribution over observed \( (s_i, a, r_i, s'_i) \) instances in the data available for action a. Each instance i is given a weight \( w_i \) depending on the Euclidean distance from \( x \) to \( s_i \), and on the model breadth parameter \( b \), according to the following formula:

\[
    w_i \propto e^{-\frac{(x-s_i)^2}{2b}}.
\]

Intuitively, \( b \) controls the degree of generalization used to estimate \( T(x, a) \) from nearby data. In 3D Mountain Car experiments, we used a parameter of \( b = 0.1 \). In theory, all instances that share the action a could be used to help approximate \( x \), where each instance i’s contribution is modified by \( w_i \) (i.e., a Gaussian weighting that exponentially penalizes distance from \( x \)). To reduce the computational cost of the algorithm, for a given state \( x \) we computed the weights for the nearest instances first. Once an instance’s weight failed to increase the cumulative weight by at least 40%, we ignored the remaining instances’ contribution as negligible. Finally, when the accumulated weight failed to reach a threshold of 1.0, we used Fitted R-MAX’s exploration strategy of assuming an optimistic transition to a maximum-reward absorbing state.

Changing the learning parameters for Fitted R-MAX outlined above affect three primary aspects of learning:

- How accurately the optimal policy can be approximated.
- How many samples are needed to accurately approximate the best policy, given the representation.
- How much computation is required when performing dynamic programming.

For this work, it was most important to find settings which allowed the agent to learn a reasonably good policy in relatively few episodes so that we could demonstrate the effectiveness of TIMBREL on sample complexity. We do not argue that the above parameters are optimal. They could be tuned to emphasize any of the above goals, such as achieving higher performance in the limit. In preliminary results (not shown), we compared using Fitted R-MAX and to using model-free \( \epsilon \)-greedy Sarsa(\( \lambda \)). Fitted R-MAX learned to consistently find the goal state with roughly two orders of magnitude less data than Sarsa, although learning with Fitted R-MAX takes substantially more computational resources than Sarsa.

4. MODEL TRANSFER

Model-based algorithms learn to estimate the transition model of an MDP, predicting the effects of actions. The goal of transfer for model-based RL algorithms is to allow the agent to build such a model from data gathered both in a previous task, as well as in the current task. To help frame the exposition, we note that transfer methods must typically perform the following three steps:

I. Use the source task agent to record some information during, after, or about learning. Successful TL approaches include recording learned action-value functions or higher-level advice about high-value policies.

II. Transform the saved source task information so that it applies to the target task. This step is most often necessary if the states and actions in the two tasks are different, as considered in this paper.

III. Utilize the transformed information in the target task. Successful approaches include using source task information to initialize the learner’s action-value function, giving advice about actions, and suggesting potentially useful sequences of actions (i.e., options).

Section 4.1 introduces TIMBREL, a novel transfer method, which accomplishes these steps. Section 4.2 gives an overview of the method details how TIMBREL is used in the Mountain Car domain with Fitted R-MAX, our chosen model-based RL algorithm.

4.1 Instance-Based Model Transfer

This section provides an overview of TIMBREL. In order to transfer a model, our method takes the novel approach of transferring observed instances from the source task. The tuples, in the form \((s, a, r, s')\), describe experience the source task agent gathered while interacting with its environment (Step I). One advantage of this approach as compared to transferring an action-value function or a full environmental model (e.g., the transition function) is that the source task agent is not tied to a particular learning algorithm or representation: whatever RL algorithm that learns will necessarily have to interact with the task and collect experience. This flexibility allows a source task algorithm to be selected based on characteristics of the task, rather than on demands of the transfer algorithm.

To translate a source task tuple into an appropriate target task tuple (Step II) we utilize inter-task mappings [20], which have been

Specifically, we used a CMAC [1] function approximator with 14 4-dimensional linear tilings, which is analogous to how Singh and Sutton [15] used 14 2-d dimensional linear tile codings for their 2D task.
successfully used in past transfer learning research to specify how pairs of tasks are related via an action mapping and a state variable mapping. This pair of mappings identifies source task actions which have similar effects as target task actions, and allows a mapping of target task state variables into source task state variables.

When learning in the target task, TIMBREL specifies when to use source task instances to help construct a model of the target task (Step III). Briefly, when insufficient target task data exists to estimate the effect of a particular \((x, a)\) pair, instances from the source task are transformed via an action-dependent inter-task mapping, and are then treated as a previously observed transition in the target task model. The TIMBREL method is summarized in Algorithm 1.

Table 1: This table describes the mapping used by TIMBREL to construct target task instances from source task data.

\[
\begin{array}{|c|c|}
\hline
\text{Action Mapping} & \text{State Variable Mapping} \\
\hline
\chi_A(\text{Neutral}) = \text{Neutral} & \chi_S(x) = x \\
\chi_A(\text{North}) = \text{Right} & \chi_S(x) = \dot{x} \\
\chi_A(\text{East}) = \text{Right} & \text{or} \\
\chi_A(\text{South}) = \text{Left} & \chi_S(y) = x \\
\chi_A(\text{West}) = \text{Left} & \chi_S(y) = \dot{x} \\
\hline
\end{array}
\]

As an illustrative example, consider the case when the agent wants to approximate \(T(x, a_T)\), where \(x = (\dot{x}_T, \dot{y}_T) = (-0.6, -0.2, 0.0, 0.1)\) and \(a_T = \text{East}\). TIMBREL considers source task transitions that contain the action \(\text{Right}\). \(\chi_S\) is defined so that either the \(x\) or \(y\) state variables can be mapped from the target task to the source task, which means that we should consider two transitions selected from the source task instances. The first tuple is selected to minimize the Euclidean distances \(D(x_T, x_S)\) and \(D(y_T, x_S)\), where each distance is scaled by the range of the state variable. The second tuple is chosen to minimize \(D(y_T, x_S)\) and \(D(y_T, x_S)\).

Continuing the example, suppose that the first source task tuple selected was \((-0.61, 0.01), \text{Right}, -1, (-0.59, 0.02)\).

If the inter-task mapping were defined so that both the \(x\) and \(y\) state variables simultaneously, the inverse inter-task mapping \(\text{could be used to convert the tuple into}\)

\((-0.61, -0.61, 0.01, 0.01), \text{East}, -1, (-0.59, -0.59, 0.02, 0.02)\).

However, this point is not near the current \(x_T\) we wish to approximate. Instead, we recognize that this sample was selected from the source task to be near to \(x_T\) and \(y_T\), and transform the tuple, assuming that \(y_T\) and \(\dot{y}_T\) are kept constant. With this assumption, we form the target task tuple

\((-0.61, 0.01, 0.01), \text{East}, -1, (-0.59, 0.02, 0.02, 0.02)\).
The analogous step is then performed for the second selected source task tuple; we transform the source task tuple with \( \chi \) while assuming that \( x_T \) and \( x'_T \) are held constant. Finally, both transferred instances are added to the approximation of \( T(x, a) \).

TIMBREL thus transfers pairs of source task instances to help approximate the transition function. Other model-learning methods may need constructed trajectories instead of individual instances, but TIMBREL is able to generate trajectories as well. Over time, the learner will approximate \( T(x_T, a_T) \) for different values of \((x, a)\) in order to construct a model for the target task environment. Any model produced via this transfer may be incorrect, depending on how representative the saved source task instances are of the target task (as modified by \( \chi \)). However, our experiments demonstrate that using transferred data may allow a model learner to produce a model that is more accurate than if the source data were ignored.

As discussed in Section 3.3, Fitted R-MAX uses the distance between instances and \( \chi \) to calculate instance weights. When an instance is used to approximate \( \chi \), that instance’s weight is added to the total weight of the approximation. If the total weight for an approximation does not reach a threshold value of 1.0, an optimistic value \( (R_{max}) \) is used because not enough data exists for an accurate approximation. When using TIMBREL, the same calculation is performed, but now instances from both the source task and target task can be used.

As the agent interacts with the target task, more transitions are recorded and the approximations of the transition function at different \((x, a)\) pairs need to be recalculated based on the new information. Each time an approximation needs to be recomputed, Fitted R-MAX first attempts to use only target task data. If the number of instances available (where instances are weighted by their distance from \( x \)) does not exceed the total weight threshold, source task data is transferred to allow an approximation of \( T(x_T, a_T) \). This process is equivalent to removing transferred source task data from the model as more target task data is observed and therefore allows the model’s accuracy to improve over time. Again, if the total weight from source task and target tasks instances for an approximated \( \chi \) does not reach 1.0, \( R_{max} \) is assigned to the model for \( \chi \).

As a final implementation note, consider what happens when some \( x \) maps to an \( s_T \) that is not near any experienced source task data. If there are no source task transitions near \( s_T \), it is possible that using all available source task data will not produce an accurate approximation (recall that instance weights are proportional to the square of the distance from the instance to \( x \)). To avoid a significant reduction in performance with limited improvement in approximating \( T \), we imposed a limit of 20 source task tuples when approximating a particular point (line 5). This threshold serves a similar purpose as the 10% cumulative weight threshold discussed in Section 3.3.

5. TRANSFER EXPERIMENTS

In order to test the efficacy of transfer, we conducted an experiment to measure the learning speed of Fitted R-MAX in the Mountain Car domain both with and without TIMBREL. To transfer from 2D Mountain Car into the more complex 3D Mountain Car, we first allow Fitted R-MAX to train for 100 episodes in the 2D task while recording all observed \((s,a,r,s')\) transitions. The agent’s learning parameters were set so that the agent thoroughly explored the source task state space and only discovered the goal near the end of learning.6

6We experimented with 5 different parameter settings for Fitted R-MAX in the 2D Task. Recall that every episode lasts 500 time steps if the goal is not found. When learning 2D Mountain Car, the agent experienced 48,669 source task transitions during 100 episodes.

Roughly 100 preliminary experiments were run on the 3D task, each lasting a few hundred episodes, in order to select Fitted R-MAX settings for the non-transfer learner, which were discussed in Section 3.3. We ran 10 trials of Fitted R-MAX without transfer and 10 trials with transfer, each of for 1,500 episodes. After learning, we averaged each set of 10 independent learning curves, but due to the low number of trials, the learning curves were quite noisy. To improve the clarity of our results, we also smoothed the two summary learning curves by averaging over groups of 10 episodes.

Figure 3 shows the summary of our two sets of experiments, along with the standard error at each point. We ran paired t-tests on the 151 graphed points and found that every difference was statistically significant \( (p < 1.7 \times 10^{-4}) \), which confirms that utilizing transfer between our pair of Mountain Car tasks yield a significant advantage for Fitted R-MAX.

Our algorithm and implementation have been designed to minimize the sample complexity. However, it is worth noting that there is a significant difference in the computational complexity of the transfer and non-transfer methods. Every time the transfer agent needs to use source task data to estimate \( T \), it must locate the most relevant data and then insert it into the model. Additionally, the transfer agent has much more data available initially, and thus its dynamic programming step is significantly slower than the non-transfer agent. These factors cause the transfer learning trials to take roughly twice as much wall clock time as the non-transfer trials. While our code could be better optimized, using the additional transferred data will always slow down the agent, relative to an agent that is not using transfer, but is running for the same number of episodes. However, in many domains a tradeoff between computational and sample complexity is highly advantageous, and is one of the benefits inherent to model-based reinforcement learning.

Also note that the transfer and non-transfer learning curves do not end at the same performance. We do not claim that transfer has produced a superior asymptotic performance, however, because neither learning curve has fully converged. We expect that the non-transfer Fitted R-MAX agents would reach the same, or perhaps superior, performance. However, these results do demonstrate that transfer can provide a significant speed advantage.

6. CONCLUSION AND FUTURE WORK
In this paper we have introduced TIMBREL, which we believe to be the first transfer method compatible with model-based reinforcement learning. We demonstrate that when learning 3D Mountain Car with Fitted R-MAX, TIMBREL can significantly reduce the sample complexity and demonstrated how transfer is achieved by changes to the source task’s reward and transfer functions.

There are a number of research directions suggested by this work. When learning the 2D source task in this paper, we explicitly set the parameters to maximize exploration. It would be informative to study how transfer efficacy changes when the amount of exploration is decreased in the source task. This is an issue related to, but distinct from, discovering how the target task performance is affected when the number of source task episodes changes. A final question left for future work is whether one could determine if collecting additional samples in the source task would help learn the target, which could help reduce the total amount of data required to learn both tasks.

We predict that TIMBREL will work, possibly with minor modifications, in other model-based RL algorithms. In the future we would like to experiment with other model-based RL algorithms, such as R-MAX, to see if transfer is as effective as in Fitted R-MAX, and see if our methods need to be modified to accommodate the different model representation. Additionally, we intend to apply TIMBREL to more complex domains that have continuous state variables; we expect that transfer will provide even more benefit as task difficulty increases.

Acknowledgments

We would like to thank the anonymous reviewers for helpful comments and suggestions. This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, and NSF award IIA-0303609.

7. REFERENCES

Author Index:

S. Airiau  49, 81
R. Arrabales  57
R. P. Barbosa  1
O. Belo  1
H. Coelho  41
M. Grzes  9, 17
Y. Guo  65
D. Hennes  25
G. Hines  73
D. Kudenko  9, 17, 97
K. Larson  73
A. Ledezma  57
R. Li  65
M. Mnif  33
P. Mukherjee  81
A. de Antonio Jiménez  89
N. K. Jong  113
L. Padham  49
W. Rand  113
U. Richter  33
O. J. Romero López  89
A. Sanchis  57
S. Sardina  49
S. Sen  49, 81
A. Servin  97
P. Stone  105
F. Stonedahl  113
M. E. Taylor  105
P. Trigo  41
K. Tumer  25
K. Tuyls  25
U. Wilensky  105
A. Zeman  65