Coordinating Learning Agents via Utility Assignment

Steven Lynden and Omer F. Rana
Department of Computer Science, University of Wales, Cardiff, UK.
{s.j.lynden, o.f.rana}@cs.cf.ac.uk

Abstract. In this paper, a coordination technique is described for fully cooperative learning based multiagent systems, based on the Collective Intelligence work by Wolpert et al. Our work focuses on a practical implementation of these approaches within a FIPA compliant agent system, using the FIPA-OS agent development toolkit. The functionality of this system is illustrated with a simple buyer/seller agent application, where it is shown that the buyer agents are capable of self-organising behaviour in order to maximise their contribution to the global utility of the system.

1 Introduction

Various approaches to multiagent system (MAS) coordination exist, many of which are based on off-line techniques, in which predefined cooperation protocols [7] and social laws [9] are engineered, and agent behaviour is specified using rule based decision making processes. We believe that such approaches limit the potential for useful emergent, self-organising behaviour, and are less adaptable than MAS consisting of agents that use machine learning techniques such as reinforcement learning (RL) [3]. Learning agents require some form of payoff function in order to guide their learning process. In many approaches, the agent’s payoff function is determined by its local utility function, which rates the performance of the agent according to various desirable characteristics. Utility functions are designed to give an indication of the degree of success with which an agent is achieving the goals it was designed to perform. In this work, the focus is on engineering fully cooperative MAS, where agents work towards common goals; an additional aim of this work is that agents are not restricted by explicit centralised control. A global utility function rates the performance of the MAS as a whole, based on various aspects of the collective performance of the agents. Agents learn to adapt their behaviour in order to maximise their local utility, so the challenging issue here is to assign local utility functions that reflect the contribution of each agent towards the global utility, also known as the credit-assignment problem [6]. The obvious approach is to use team-game utility functions, which simply assign the global utility function to each agent as the local utility function. This does not scale well in large MAS, due to the fact that it becomes increasingly difficult for an agent to determine the effect of its actions on its utility value as the number of agents increases. More promising results are achieved using the Collective Intelligence (COIN) [1] framework,
in which various utility function assignment techniques are investigated that overcome this signal-to-noise problem.

The focus of this paper is to illustrate the application of LEAF (the Learning Agent based FIPA compliant agent toolkit), a toolkit for developing coordinated learning based multiagent systems. The motivation for the development of LEAF is to provide a generic toolkit that can be used to develop agent based systems applicable to a diverse range of domains. Coordination in LEAF is based on the utility function assignment techniques of COIN, and support for this is incorporated into the LEAF infrastructure. It is intended that the LEAF toolkit be used to design, implement and monitor the performance of learning based MAS, and support is provided for each of these three key areas. The importance of interoperability with other agent systems and the role of standards is recognised, and for this reason LEAF builds on the FIPA-OS [4] agent development toolkit. FIPA-OS is an open source agent platform supporting the development of agents implementing the standards that must be met in order to develop FIPA compliant [5] agent systems. The use of FIPA-OS therefore enables interaction between LEAF agents and other FIPA compliant agents via FIPA ACL (FIPA agent communication language).

2 Utility Assignment

The essential concept on which the LEAF toolkit is based is that machine learning based agents can learn to maximise their personal local utility functions, which are assigned to agents with the aim of engineering a system in which improvements made to local utility are beneficial to the system globally. In the LEAF system, we endeavour to provide a toolkit for the rapid development of coordinated, FIPA compliant, learning based MAS utilising the utility function assignment ideas of COIN.

Collective Intelligence (COIN) [1] is a framework for designing large-scale learning based MAS, where centralised control of individual agents is extremely difficult or impossible due to scale. The COIN framework assumes a defined world utility function that rates the functional performance of the MAS as a whole. In order to engineer coordinated MAS where agents should not exhibit behaviour that is detrimental to world utility, COIN introduces a technique where local utility functions are assigned to agents, which are then maximised by the agent’s internal learning processes. The framework investigated here, based on COIN, lies between centralised control and complete autonomy, where utility functions can be assigned to agents, and the functions parameterised by group/world dynamics, which necessitates some form of centralised communication to transfer parameters to the agent’s local utility functions. Agents are autonomous in the sense that they are able to make their own decisions concerning how to maximise their utility function. Unlike centralised control, these utility assignment techniques do not involve a centralised entity with detailed models of agents, and no explicit instructions are given to agents.
Utility function assignment takes place via FIPA ACL. Utility function parameters are updated by the ESN via socket connections.

(a) The development process

(b) Deployment of agents and the ESN

Among the various utility functions assigned to agents in COIN, some require centralised communication, whereas others can be computed by information available locally to an agent [2]. It is therefore clear that any practical implementation of the COIN techniques requires some means of dynamically providing information (that is not available locally) to agents concerning MAS state. In the LEAF system, this task is performed by an environment service node (ESN), which is responsible for maintaining the system information necessary for updating local and global utility functions. Multiple ESN’s can co-exist in the system, each of which can specialise in a particular aspect of utility. We therefore foresee the existence of multiple environments (or markets – as described in section 3), with ESN’s being federated across these environments.

The LEAF toolkit has been developed in Java, utilising the FIPA-OS agent development toolkit. A basic knowledge of Java programming is required to develop agents using our framework as agent behaviour, comprised of various learning algorithms, must be programmed in Java. The LEAF system is illustrated in figure 1.

3 Marketplace Application

In order to demonstrate an application of the LEAF system, we outline an example application based on a buyer/seller marketplace. It is not the intention here to present an environment simulating realistic market behaviour; the aim is to illustrate a simple market based application to which the LEAF system can be applied. In this application, the learning based MAS consists of a set of buyer agents, which aim to maximise their local utility by making certain purchases from seller agents within a number of markets. Agents exist within a variety of
markets, each of which contains a number of buyer agents and a number of seller agents. Seller agents offer a number of items, which can be purchased by buyer agents within the same market only. Buyer agents are given one credit each day, with which one purchase can be made from a seller agent, from a number of different categories of items which can be purchased. The buyer agents are placed in markets with the purpose of making purchases that will achieve the global objective of making as many purchases as possible, with the constraint that at least some quantity of each type of item is purchased each day. Each buyer agent’s learning process is defined by simple RL algorithms.

3.1 Agent Behaviour

Seller agents existing within a market sell items to buyer agents within the same market. Each seller agent maintains a price list which determines the quantity of items it will sell for one credit, however this information is not made public and buyer agents must ”gamble” by giving the seller agent one credit (specifying the desired item) before knowing the quantity of that item the seller agent will return. The price list of a seller agent is generated randomly when the seller agent is created, where the quantity of an item returned for one credit is a random integer between 0 and 100. Seller agents are assumed to have a large number of items to sell, the quantity of items they offer do not vary over time, and they never refuse to sell items. All seller agents sell items categorised as food, textiles, luxuries, computers, machinery and alloys. There is no difference in the quality of the items sold by the agents. Seller agents do not receive local utility functions as they are considered to be part of the environment in which the buyer agents exist, and not part of the MAS that is being coordinated.

Buyer agents purchase items from seller agents with the aim of maximising their local utility functions, which are distributed by the ESN. Each buyer agent attempts to maximise its local utility function using a simple RL algorithm, which weighs purchases according to the effect they have on local utility. Buyer agents make the assumption that it is desirable to obtain the greatest quantity of an item from each purchase (they aim for the maximum value-for-money in each purchase); making this assumption allows a buyer agent to maintain a set of gating weights, related to the number of items received from seller agents in previous interactions. The local utility function of each buyer agent therefore influences the type of item that the buyer is likely to purchase.

The Agent Learning Process The actions that a buyer agent can perform are determined by the items it can buy - food, textiles, machinery, computers, luxuries or alloys. Once the decision is made to buy an item, the gating network is used to decide the agent from which to purchase. A simple RL technique is used to select which item to buy at the start of each day. A set of action-values are used, which record the average change in local utility that occurs after an action is selected - these values are then used to select an action each day using a simple Softmax action selection technique [3].
A set of weights (referred to here as gating weights) are used by buyer agents to determine which seller agents to purchase specific items from (in order to get the greatest number of items from the purchase), while the RL algorithm allows the buyer agent to determine which items to purchase (in order to benefit the system globally). The gating weights simply record the number of items returned by the seller agents when a purchase is made. \( Gat(s, i) \) gives the quantity of item \( i \) received from agent \( s \) in a purchase. The following steps are repeatedly performed, with one iteration executed each day, where a day is an arbitrary system wide standard unit of time: (1) An item \( i \) is selected to be purchased based on the action-values held. (2) A seller agent \( s \) is selected to purchase from based on the gating weights held, which give an indication of the value of previous interactions with sellers. (3) The purchase of the item \( i \) is made from seller \( s \). (4) The agent computes its local utility function, and rewards the action (purchasing item \( i \)) by the change in utility, which is determined by comparison with the result of the previous utility computation.

### 3.2 Utility Functions

The overall objective of the MAS is to achieve a situation in which each agent makes the best purchases it possibly can given the characteristics of its market, and the daily global utility of the system is computed each day as \( U_g(d) = \sum_{i \in I} \text{Max}(N^i_d) \) where \( I \) is the set of item types, and \( \text{Max}(N^i_d) \) is the maximum quantity of item \( i \) purchased in any individual purchase made on day \( d \) by any of the buyer agents.

The global utility of the system is calculated as the sum of daily global utility values, hence \( U_g = \sum_{d=1}^{n} U_g(d) \), where the system has existed for \( n \) days. The following local utility functions are assigned to the buyer agents: \( U_i(a) = \sum_{d=1}^{n} U_g(Purchased_{a,d} = 0) - U_g \), where \( Purchased_{a,d} \) is the number of items purchased on day \( d \) by agent \( a \). \( U_g(Purchased_{a,d} = 0) \) is the global utility function for day \( d \) computed without considering any of the purchased made by agent \( a \), where the system has existed for \( n \) days. In order to compute local utility functions, agents are dynamically supplied with the following parameters: \( U_g(Purchased_{a,d} = 0) \) and \( U_g \). The ESN performs the tasks involved in maintaining and updating these values.

### 3.3 Results & Observations

Experiments were performed with 6 buyer agents, 6 markets (one buyer agent per market), 2 seller agents in each market, and 6 categories of items. The experiments were allowed to run for 250 days in each case. In all cases, the buyer agents experimented with different purchases initially, and then learned to concentrate on one specific purchase. This behaviour was a common feature of all buyer agents in all simulations. The results are illustrated in figure 2, which shows the value (sum of the maximum quantities of each item purchased) of the purchases settled on by the LEAF agents, and compares this to the best possible combinations of purchases that could have been achieved. It can be
The results from 10 experiments are displayed.

The column labelled “LEAF” contains the value of the combination of purchases settled on by the buyer agents in each experiment.

The column labelled “Optimal” shows the best possible combination that could have been achieved with a centralised controller with complete knowledge of the states of all markets.

<table>
<thead>
<tr>
<th>LEAF</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>469</td>
</tr>
<tr>
<td>2</td>
<td>453</td>
</tr>
<tr>
<td>3</td>
<td>530</td>
</tr>
<tr>
<td>4</td>
<td>509</td>
</tr>
<tr>
<td>5</td>
<td>480</td>
</tr>
<tr>
<td>6</td>
<td>532</td>
</tr>
<tr>
<td>7</td>
<td>538</td>
</tr>
<tr>
<td>8</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>470</td>
</tr>
<tr>
<td>10</td>
<td>483</td>
</tr>
</tbody>
</table>

Fig. 2. Results

seen that only in experiment number 3 do the buyer agents learn to select the optimal combination of purchases, but all other experiments achieve at least 84% of the value of their optimal combinations. The benefit of the suboptimal combinations is that they are achieved without complete centralised control, via the individual learning processes of agents, combined with local utility function assignment. More complex, realistic applications will be the focus of future work.

References

5. FIPA (Foundation of Intelligent Physical Agents) homepage: http://www.fipa.org