Project Plan

James King

September 2018

Abstract

For 6 marks copy from the project list.
For 15 marks describe and motivate the final project.
For 30 marks describe and motivate, and also give well thought out individual project goals.

Aims

- To create an agent mind as a service (AMAAS) system in Prolog that is able to interact with any other system that is capable of creating an environment for that agent to work in, and contacting the service.
- To explore a mechanism to aid in the evolution of cooperation known as indirect reciprocity.
- To create a web application that allows users to set up and run tournaments of both direct and indirect reciprocity.
- To use the web application to demonstrate an environment for the AMAAS system.

Background

The Iterated Prisoner's Dilemma uses a mechanism known as direct reciprocity to aid in the evolution of cooperation, this was popularized in Axelrod's tournaments and research on it [1]. There is a well tested library that is able to run matches, tournaments and the Moran Process for the Iterated Prisoner's Dilemma [2]. This is not the only mechanism used to aid in the evolution of cooperation. Nowak presents 4 alternative methods [3] including: kin selection, group selection, network reciprocity and indirect reciprocity.

Of these indirect reciprocity stands out as an interesting model for interaction between humans or intelligent agents. Pairs of agents are chosen at random to interact (they may not necessarily interact again), in each pair one agent is the donor and the other is the recipient. A donor chooses whether to defect or cooperate, defecting gives them a higher payoff so is always tempting, however the model utilizies the idea of reputation (also known as image score [5]) to create a condition where an agent may want to cooperate. If the donor cooperates their reputation will be enhanced, if they defect it will fall. Some strategies known as discriminators, will choose to defect if the recipient has a low reputation or cooperate if they have a good reputation. In some cases of tournaments all players have viewed all other players' interaction and have full knowledge of their reputation, or in other tournaments, only a subset of the population of 'onlookers' have viewed a certain interaction.

This idea of onlookers becomes particularly interesting when you look at the spread of information. The onlookers may gossip to other agents, gossip being the passing on of information. The information that onlookers pass on may not necessarily be correct, allowing the spread of misinformation. How agents react to this information, reveals a personality to them. This spread of information and misinformation effectively becomes a metagame to the interactions in indirect reciprocity.

In thinking about how to implement this model, I and my supervisor Kostas felt using Prolog to implement an agents mind was most appropriate, whilst implementing the environment itself in a procedural language would be more appropriate (Python has been chosen). The issue with this is interfacing between the two languages, there are libraries such as PySwip [6] that are able to do this but are limited. For instance PySwip struggles working in multi-threaded environments.

Project Description

I will explore indirect reciprocity, by both researching and reporting on Nowak and other authors research on the mechanism, drawing on their research and possibly adding on to the ideas of onlookers and gossip to create a model for implementation. I shall be using the Axelrod-Python library to create a proof of concept that I am able to create a web application that allows users to set up and run games and tournaments. This web application will be hosted on heroku and created using Flask and Python.

The solution that I have devised with Kostas to the interfacing issues between Python and Prolog is to create a web service for the agent's mind (The AMAAS system). This will have a Prolog server containing "agent's minds". The minds themselves will represent information on the state of the environment using an efficient version of the events calculus that Kostas has worked on. Percepts can be added to the agent's mind, and based on these percepts and agent will be able to decide on an action, and act on it by returning an action to the environment.

The capabilities of AMAAS are to be demonstrated on the web application, which will run indirect recirprocity tournaments creating the environment that agents reside in and use AMAAS to input percepts and get actions from.

These have been the core parts of the project, however I could extend this to the other mechanisms presented by Nowak [3] if I have time. Axelrod-Python has an implementation for network reciprocity - which would be interesting to implement so I could attempt to create an especially nice frontend for the patterns created in the population [4]. Due to the nature of group selection working on top of direct reciprocity [7] it would be possible to implement it on top of the Axelrod-Python library. Leaving kin selection to implement using AMAAS.

Motivation

The development of the AMAAS system will provide a solution to problems interfacing between Prolog and other languages. It will also be useful to develop my logic programming skills, this is key to my interest in intelligent agents and AI

Timeline

For 8 marks copy milestones from the project list without dates.
For 20 marks copy milestones from the project list but with dates or extra milestones.
For 30 marks milestones are adequate, with dates but without sufficient motivation.
For 40 marks a good list of well explained milestones.

Early Deliverables

Familiarisation with the idea of strategies and the identification of set of them.
Implementation of strategies and the definition of an interface that allows agents to be selected and play against each other
Definition of a tournament
Report on the design of a tournament.
Implementation of a tournament, where a user selects the number of agents, associates them with existing strategies, specifies the number of iterations and deploys a tournament
Report on the Iterative Prisoner's dilemma problem
Report on previous work with the Iterative Prisoner's dilemma

Final Deliverables
A program that allows us to simulate tournaments of the iterative prisoner's dilemma.
A GUI that allows a user to create and manage tournaments.
GUI should provide summary statistics that are tournament specific, i.e. a summary for each player/strategy and how it is compared with the other players/strategies in the tournament.
The report should describe some of the theory behind strategies in cooperative settings.
The report should provide an analysis and an evaluation of the different strategies, including an indication of which one is the best strategy, if any.
The report should contain a design of the program and a discussion of any software engineering issues encountered.
The report should describe any useful and interesting programming techniques that have been employed to develop the final prototype.

Suggested Extensions
Make use of multi-agent system platform to support the tournament in a completely distributed setting.
Use logic-based rules to specify the strategies and wrap them in the Java implementation.
Use strategy recognition techniques so that agents can detect what strategies opponents are playing, and adapt their behaviour accordingly.
Create a methodology for developing tournaments of this type.

Milestone: Report on the design of the AMAAS system and an Agent's Mind

Date: Method: Explanation and motivation: Link to other milestones:

Milestone: Definition of a GUI to run direct reciprocity matches

Date: Method: Explanation and motivation: Link to other milestones:

Milestone: Definition of a GUI to run direct reciprocity tournaments

Date: Method: Explanation and motivation: Link to other milestones:

Milestone: Definition of a GUI to run indirect reciprocity matches

Date: Method: Explanation and motivation: Link to other milestones:

Milestone: Implementation of a database to record results of indirect and direct reciprocity matches and tournaments

Date: Method: Explanation and motivation: Link to other milestones:

Risk Assessment

For 3 marks general risks are given such as "may get behind".
For 6 marks risks are associated to deliverables and show clear thought.
For 10 marks risks provided with reasonable mitigations.
For 15 marks a good case is made for understanding how the project may fail to proceed and what should be done about it. Some risks have likelihoods and importance.

Risk: Using SWI-Prolog to run a web service

This is new to both me and my supervisor. It has been done before and there are resources online, but it is still new territory to me. This could delay the delivery of the agent mind as a service system.
Mitigation: If I fail to be able to deliver this on time I shall switch my plan to one of the following two options: write an API in Python to run the AMAAS system or forget the AMAAS system and code the program to run indirect reciprocity directly in the web application.
Likelihood: Possible
Importance: Moderate

Risk: Failure to understand and use mvfcec

Mvfcec or the multi-value fluent cached event calculus, is a version of the event calculus where querying is faster. I have knowledge on the event calculus itself, but this is a more complicated form of it. If I fail to understand and use it, I cannot write a report on it and either the AMAAS system will be far slower or I will fail to deliver it
Mitigation: I shall attempt to avoid this by working with Kostas to understand it. However if I still fail to grasp mvfcec, I shall either switch to using the simplified events calculus, or use one of the two mitigations from the risk: Using SWI-Prolog to run a web service. Likelihood: Possible
Importance: Moderate

Risk: My inexperience of writing large Prolog base applications

I have used Prolog in the AI course in second year, however these applications were nowhere near as large or complex as the AMAAS system will have to be. I am doing it this way to push myself, but recognise this could cause a failure to deliver the AMAAS system.
Mitigation: switch from Prolog to Python for writing the AMAAS system or simply include the program for indirect reciprocity in the Python web application.
Likelihoood: Possible
Importance: Moderate

References

[1] Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[2] Vince Knight; Owen Campbell; Marc; eric-s-s; VSN Reddy Janga; James Campbell; Karol M. Langner; T.J. Gaffney; Sourav Singh; Nikoleta; Julie Rymer; Thomas Campbell; Jason Young; MHakem; Geraint Palmer; Kristian Glass; edouardArgenson; Daniel Mancia; Martin Jones; Cameron Davidson-Pilon; alajara; Ranjini Das; Marios Zoulias; Aaron Kratz; Timothy Standen; Paul Slavin; Adam Pohl; Jochen Müller; Georgios Koutsovoulos; Areeb Ahmed. Axelrod: 4.3.0. `http://dx.doi.org/10.5281/zenodo.1405868`, September 2018.

[3] Martin A. Nowak. Five rules for the evolution of cooperation. *Science*, 314, 2006.

[4] Martin A. Nowak and Robert M. May. Evolutionary games and spatial chaos. *Nature*, 359:826–829, 1992.

[5] Martin A. Nowak and Karl Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393:573–577, 1998.

[6] Yuce Tekol. Pyswip. `https://pypi.org/project/pyswip/`, 2018.

[7] Arne Traulsen and Martin A. Nowak. Evolution of cooperation by multilevel selection. *Proceedings of the National Academy of Sciences of the United States of America*, 103:10952–10955, 2006.

For 3 marks just include web pages.
For 6 marks clear use of more than one source, not just web sites.
For 10 marks nice evidence of good research and understanding how this motivates the timeline.
For 15 marks items are very good and each has a short relevant discussion of it's value to the project.