

COOPERATIVE STRATEGIES IN MULTI-AGENT SYSTEMS

Full Unit Project: Plan
James King
Supervisor: Kostas Stathis

September 2018

Abstract

Background

The Iterated Prisoner's Dilemma uses a mechanism known as direct reciprocity to aid in the evolution of cooperation, which was popularized in Axelrod's tournaments and research on it [1]. There is a well-tested library that is able to run matches and tournaments for the Iterated Prisoner's Dilemma [2]. This is not the only game-theory mechanism used to model the evolution of cooperation. Nowak presents 4 alternative methods [4] 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 utilizes the idea of reputation (also known as image score [6]) 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 are known as discriminators, these will choose to defect if the recipient has a low reputation or cooperate if they have a good reputation.

Nowak and Sigmund present the idea of onlookers [6] a notion limits the number of direct observers of interactions. As is noticeable in real life, direct observation of other's interactions is not always possible, so this seems to more closely match real life. In societies with advanced communication however direct observation is not the only method of finding out about interactions, gossip has been put forward as an alternative to direct observation [7]. Previously the outcome depended upon the probability of witnessing another's decision, now with gossip, it depends upon how the information has been spread and how agents base their behaviour on gossip. Ralf D. Sommerfeld *et al* state that gossip needs to truthfully describe other's behaviour, and when hearing correct gossip an agent should act accordingly or gossip will fail.

We will develop this model as follows. To implement the agent's decision-making component we will consider using the logic programming language Prolog because it can support the implementation of transparent decisions. To implement the environment and how a user will control its evolution we will consider the procedural language Python because it has the potential to deal with the user interface and other useful control features. The issue with this is interfacing between the two languages, there are libraries such as PySwip [9] that are able to do this but are limited. For instance, PySwip struggles to work in multi-threaded environments (such as a web application).

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 we are exploring for the interfacing issues between Python and Prolog is to create a web service for the agent's decision-making component. This will have a Prolog server containing what [8] refers to as "agent's minds". A mind will represent information on the state of the environment using an efficient version of the Event Calculus reported in [3]. Percepts can be added to the agent's mind and based on these percepts an agent will be able to decide on an action and act on it by returning an action to the environment. Separating out the head and body as in Stathis *et al.* 2002 [8], though this solution would take this separation further, having the head based in the Prolog service and the body in the environment (Python web application).

The capabilities of the Prolog service are to be demonstrated on the web application, which will run indirect reciprocity tournaments creating the environment that agents reside in and containing the body of the agent. The web application will use the service to input percepts and get actions.

These have been the core parts of the project however, I could extend this to the other mechanisms presented by Nowak [4] if I have time. Axelrod-Python has an implementation for network reciprocity - which would be interesting to use so I could create an especially nice frontend for the patterns created in the population [5]. Due to the nature of group selection working on top of direct reciprocity [10] it would be possible to implement it on top of the Axelrod-Python library. Leaving kin selection to implement using the Prolog service.

Goals

- To create a Prolog service that is able to support the decisions of a game-theoretic agent which interacts with other systems that wish to choose an agent, provide percepts from an environment and get actions from the agent.
- To explore in a game-theoretic context 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 game-theory tournaments of both direct and indirect reciprocity.
- To use the web application to demonstrate an environment that combines the Prolog service and a set of game-theory strategies and evaluate the results.

Motivation

For many years researchers from many fields have speculated and attempted to explain as to why cooperation is so widespread in our world. Cooperation is extremely varied, examples include interactions between humans, between meerkats and even interspecies such as between the drongo bird and meerkat. Cooperation has been a major factor in our development as a species, allowing us to create fantastic civilisations and develop massive trade networks. It is also becoming important in technology, cooperation is key in the development and spread of multi-agent systems. Agents working together can help us solve more difficult problems.

The mechanisms that aid in the evolution of cooperation are important to understand as a matter of understanding the development of life itself and guide us in future actions. These future actions may include the development of multi-agent systems or interactions with other biological/intelligent agents. Implementing game-theoretic tournaments and agents will help further my understanding of cooperation and game-theory, allowing me to correctly aid in situations where cooperation is key. Due to the nature of the product I will be implementing, I will be furthering my programming skills especially in the areas of web development and AI. Two areas in which I have a keen interest, the development of these skills will hopefully aid me in securing a job in one of the two sectors.

Timeline

I'm approaching this project in an agile way, using 2-week sprints to complete tasks, as such the milestones here shall be assigned to a sprint week or multiple sprint weeks if they are more likely to take longer. This does not mean that I will not be working on a task beforehand if possible, but that I plan to finish that task by the end of the later sprint. In the winter period, I will take my time to consolidate and finish current work, whilst planning the next terms work. Here are the timings for each sprint:

Sprint 1: 1-12 October	Sprint 7: 14-15 January
Sprint 2: 15-26 October	Sprint 8: 28 January - 8 February
Sprint 3: 29 October - 9 November	Sprint 9: 11-22 February
Sprint 4: 12-23 November	Sprint 10: 25 February - 8 March
Sprint 5: 26 November - 7 December	Sprint 11: 11-22 March

First Term

Milestone: A proof of concept program for running a Prolog web service

I will be using tutorials online to learn how to create a Prolog web service, and from this learning, I will attempt to create my own service. This service should resemble a very early version of the Prolog service I wish to provide. This will prove that I can create a Prolog web service and give me confidence, but also reduce the risk of the project failing if I cannot create one.

Link to goal: This will act as an early version of the Prolog web service.

Date: Sprint 1

Milestone: A proof of concept web application to run direct reciprocity games and tournaments and provide statistics on them

The environment for the agents will be on a web application, this proof of concept will show that I am capable of creating a web application that can run similar games without the need to program the environment inside it yet. I will be using the Axelrod-Python library to run the games for a convincing mock-up.

Link to goal: This will aid in the creation of my web application.

Date: Sprint 1

Milestone: Report on the evolution of cooperation in relation to game-theory and different game-theoretic mechanisms to aid it

Report on past work in this area such as Axelrod's tournaments [1] and Nowak's five rules for the evolution of cooperation [4].

Link to goal: This will act as a starting point for my exploration of the mechanisms for the evolution of cooperation and help motivate the implementation of the web application.

Date: Sprint 1

Milestone: Report on indirect reciprocity, strategies for agents and the development of a concrete model to implement

Report on past work on indirect reciprocity including the strategies agents have employed playing this game. From this collated work I shall define a model which I am to implement and a number of key strategies to implement as well.

Link to goal: This is a part of my exploration of the mechanisms that aid in the evolution of cooperation and will help provide a model to implement in system for running indirect reciprocity tournaments across the Prolog service and environment.

Date: Sprint 2

Milestone: Report on the design of the Prolog service and Agents, and the web application and agent's environment

Will include the following: an agent definition and the inner workings of an agent's decision-making component, a definition of the structure of the Prolog service and management of agents (as well as the design of the API), a definition of the agent's environment and how this will work with the Prolog service, and a

design of the web application. This is key to a well-planned architecture for the Prolog service and agent environment. It will also help me research better methods of implementation for the Prolog service.

Link to goal: This will lay out a concrete way of implementing the Prolog service, web application and link between them.

Date: Sprint 2

Milestone: Design and implementation of a GUI for indirect reciprocity tournaments

This will follow on from my proof of concept web application, to which I will add a GUI to set up, run and provide analysis on indirect reciprocity tournaments.

Link to goal: This is a part of the creation of the web application.

Date: Sprints 2 and 3

Milestone: Design and implementation of the structure of the Prolog service

Taking inspiration from my proof on concept Prolog service and previous architectures for intelligent agents and multi-agent systems [8] I will define the structure of the Prolog service programmatically.

Link to goal: This is a part of the creation of the Prolog service.

Date: Sprints 3 and 4

Milestone: Design, implementation and testing of the management of agents and agent templates in the Prolog service

Once the structure of the Prolog service has been implemented I can then develop the components that will manage agents in the system. The system will have to deal with the management of multiple agents at one time, so this must be carefully implemented and thoroughly tested (I will need to learn how to test a Prolog system for this). For agents to work correctly in the system they will have to follow a regular pattern (naming and functionality). This pattern I will define in a template in the Prolog service.

Link to goal: This is a part of the creation of the Prolog service.

Date: Sprints 3 and 4

Milestone: Design, implementation and testing of the environment for a simple indirect reciprocity tournament

The environment is a key aspect of running indirect reciprocity tournaments. It will run the cycles of the game and record information about the game. This will be written in the web application and needs to be thoroughly tested.

Link to goal: This is a part of the creation of the web application, and will help me explore my model for indirect reciprocity.

Date: Sprints 3 and 4

Milestone: Design, implementation and testing of defector, cooperator and discriminator strategies for a simple indirect reciprocity tournament

We need strategies in the Prolog service to play indirect reciprocity tournaments. The simplest of which are defector and cooperator strategies, whilst discriminator strategies make indirect reciprocity tournaments varied. For these reasons I will be implementing them first, using the agent template in the Prolog service.

Link to goal: This is a part of the creation of the Prolog service, and will help explore how agents work in the indirect reciprocity mechanism.

Date: Sprint 4

Milestone: Linking together the Prolog service and web application environment

After the implementation of a subset of indirect reciprocity functionality - without onlookers and gossip - I should now be ready to link together the tested web application environment and Prolog service.

Link to goal: This will be the starting point in which the web application becomes a demonstrator of the Prolog service.

Date: Sprints 4 and 5

Milestone: Preparing the interim report and presentation

With the programs working together and finished first term reports I shall prepare the reports and presentation for the interim review.

Date: Sprint 5

Second Term

Milestone: Design, implementation and testing of the onlooker and gossip features in the web application environment

The model that I'm exploring becomes more interesting with onlookers and gossip features, with the simple version of the web application and Prolog service linked together I can now move onto developing the onlooker and gossip features. These features need support from the web application environment.

Link to goal: This is a part of the implementation of the web application and will help me explore my model for indirect reciprocity.

Date: Sprints 7 and 8

Milestone: Design, implementation and testing of the onlooker and gossip features in the Prolog service

While developing the onlooker and gossip features in the web application environment, in parallel I can develop the Prolog service support for these features. All the time ensuring that both the environment and the Prolog service can work together.

Link to goal: This is a part of the implementation of the Prolog service and will help me explore my model for indirect reciprocity.

Date: Sprints 7 and 8

Milestone: Design, implementation and testing of a development upon current strategies to play the gossip and onlookers meta-game

The onlookers and gossip features need support from agents, so they can spread gossip, listen to gossip and have appropriate reactions for this gossip.

Link to goal: This is a part of the implementation of the Prolog service and will help me explore agents in indirect reciprocity.

Date: Sprints 7 and 8

Milestone: Relinking the Prolog service and web application environment with the onlooker and gossip meta-game implemented

With the onlookers and gossip features fully support by agents, the Prolog service and the environment, I can move onto linking them together. Ensuring there aren't issues communicating and that the applications work together correctly.

Link to goal: This is a part of using the web application as a demonstrator for the Prolog service.

Date: Sprint 8

Milestone: Implementation and testing of further strategies for an indirect reciprocity tournament

So far we have 3 strategies, but there are other strategies available for these tournaments that don't just base their actions on the recipient's reputation but also their own reputation or even both. These will again use the agent template and be implemented in the Prolog service.

Link to goal: This is a part of the implementation of the Prolog service and will help me explore more advanced strategies used for indirect reciprocity.

Date: Sprints 8 and 9

Milestone: Enhanced analysis post-tournament for indirect reciprocity

Part of the reason for using Prolog for the agent's decision-making component is that we are hopefully able to query why an agent has made the decision that it has. In the analysis post-tournament using an Ajax request, users should be able to query the decisions. Furthermore, analysis of games should be enhanced

using graphs and possibly animation to display a timeline of actions. This will enhance the capabilities of the system in explaining what interactions occurred and why they occurred as they did.

Link to goal: This is part of the development of the web application and will help me understand how this mechanism for the evolution of cooperation acts.

Date: Sprints 8 and 9

Milestone: Enhanced GUI features

I will have focused a lot on developing the features of the application. With these features now implemented I can work on giving a facelift to the web application using bootstrap, React.js and CSS. This will enhance the web applications abilities as an educational system and also be a good preparation for the demonstration.

Link to goal: This is a part of the development of the web application.

Date: Sprints 8 and 9

Milestone: The report should describe the theory behind indirect reciprocity and it's strategies in relation to game-theory

Indirect reciprocity is a game-theoretic modelling tool for interactions, it is key to understanding the project to understand this model and the strategies used in it.

Link to goal: This will help me explore mechanisms for the evolution of cooperation.

Date: Sprints 9 and 10

Milestone: The report should provide an analysis and evaluation of strategies in indirect reciprocity tournaments

While implementing the project I will have gained in-depth knowledge of the strategies in indirect reciprocity and should be able to analyse them and evaluate their effectiveness in indirect reciprocity tournaments. This should use examples of tournaments run by my system.

Link to goal: This will help me explore strategies used in the indirect reciprocity mechanism for the evolution of cooperation.

Date: Sprints 9 and 10

Milestone: The report should describe the onlookers and gossip aspects of indirect reciprocity

These extensions on indirect reciprocity are an area of interest for understanding how the spread of information affects decisions made by agents. Understanding these extensions is key to understanding a lot of the motivation behind my project.

Link to goal: This will help me explore aspects of the indirect reciprocity mechanism for the evolution of cooperation.

Date: Sprints 9 and 10

Milestone: The report should describe the link between indirect reciprocity theory and real life biological and intelligent agent interactions

Indirect reciprocity is a modelling tool, but how is this modelling tool useful to understand real interactions between agents? In this section, I wish to describe the link between the theory and the real world, its uses and also it's limitations.

Link to goal: This will help me explore mechanisms for the evolution of cooperation and their application.

Date: Sprints 9 and 10

Milestone: The report should contain a design of the web application and environment, Prolog service and agents, and the connection between them

Program design is key to developing understandable, working applications. The design of the system as a whole and also individual programs have been motivated by both past research on multi-agent systems and my experiences when creating them.

Link to goal: This will describe the final product including the web application, Prolog service and their link.

Date: Sprints 9 and 10

Milestone: The report should contain a discussion of the software engineering techniques, tools and processes used and issues encountered

The use of software engineering techniques, tools and processes is key to writing good code that works. In my project, I will be using a variety of them in the hope of achieving this. Here I will review this use, both successes and failures.

Link to goal: This will allow me to review what I have learnt from the creation of the web application and Prolog service.

Date: Sprints 9 and 10

Milestone: The report should contain any interesting programming techniques employed to develop the final prototype

To create the system I will need to employ techniques used in web development and logic programming among other fields. With such a wide variety of techniques available this will be interesting to review and report on.

Link to goal: This will allow me to review some of the successes of the web application and Prolog service.

Date: Sprints 9 and 10

Milestone: Draft report

A draft report will be a useful point to review what I have written so far. Allowing me to check my progress and develop on what I have written.

Date: Sprint 9

Milestone: Development of content for the web application to make it into an educational system for the evolution of cooperation

The website should have content that enhances the user's experience, explaining the motivation behind concepts and features of the application.

Link to goal: This is a part of the development of the web application.

Date: Sprint 10

Sprints 10 and 11

These two sprints will be dedicated to finishing unfinished parts of the end products, the report as well as preparing for the demonstration. This period should act as a buffer if I fall behind, or a period to perfect or extend what I have produced if not.

Risk Assessment

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, as is writing web services in general. This could delay the delivery of the Prolog service.

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 a service in Python to replace the Prolog service or forget the service and code the program to run agents decision making directly in the web application.

Likelihood: Possible

Importance: Moderate

Risk: Failure to understand and use mvfcec

Mvfcec or the multi-valued fluent cached event calculus is a version of the event calculus where querying is faster. I have knowledge of 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 Prolog service 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, write a service in Python to replace the Prolog service or forget the service and code the program to run agents decision making directly in the web application.

Likelihood: Possible

Importance: Moderate

Risk: My inexperience of writing large Prolog base applications

I have used Prolog in the AI course in my second year, however, these applications were nowhere near as large or complex as the Prolog service will have to be. I am doing it this way to push myself, but recognise this could cause a failure to deliver the service.

Mitigation: switch from Prolog to Python for writing the service or simply include the agent decision-making component in the web application.

Likelihood: Possible

Importance: Moderate

Risk: My inexperience in writing large web applications

In my first year, we used HTML, CSS, Javascript etc. to learn about writing web applications. However, writing a large back end and front end of a web application is new to me. Meaning there will be obstacles that I have not yet anticipated.

Mitigation: I have taken the Miguel Grinberg Flask Mega-Tutorial, to further practice my skills in writing web applications. I feel that my practice in this has allowed me to give a low likelihood that this will affect my project, and due to the resources available for Flask the importance of any obstacles to my project will be minor.

Likelihood: Unlikely

Importance: Minor

Risk: The scale of the project

With so many new technologies and techniques to learn, programs to implement and so much theory to understand, I believe this project to be quite large. If the project is too large it could delay the delivery of multiple areas of the project.

Mitigation: I have already started on much of the learning for the project, which should help reduce the time for learning. If there is too much to implement I will have to carefully scale down the project parts at a time, this may include: reducing the features of the indirect reciprocity tournaments, removing the Prolog service and replacing it with functionality inside the web application or a reduction to the 3 key strategies (defector, cooperator and discriminator). To manage this possible scaling down or up of the program I have gone for an agile approach using scrum techniques.

Likelihood: Likely

Importance: Minor (due to the many mitigations I can employ, the impact should be minimal)

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] Özgür Kafali, Alfonso E. Romero, and Kostas Stathis. Agent-oriented activity recognition in the event calculus: an application for diabetic patients. *Computational Intelligence*, 33:899–925, August 2017.

- [4] Martin A. Nowak. Five rules for the evolution of cooperation. *Science*, 314, 2006.
- [5] Martin A. Nowak and Robert M. May. Evolutionary games and spatial chaos. *Nature*, 359:826–829, 1992.
- [6] Martin A. Nowak and Karl Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393:573–577, 1998.
- [7] Ralf D. Sommerfeld, Hans-Jürgen Krambeck, Dirk Semmann, and Manfred Milinski. Gossip as an alternative for direct observation in games of indirect reciprocity. *Proceedings of the National Academy of Sciences of the United States of America*, 104:17435–17440, 2007.
- [8] Kostas Stathis, Antonis C. Kakas, Wenjin Lu, Neophytos Demetriou, Ulle Endriss, and Andrea Bracciali. *PROSOCS: a platform for programming software agents in computational logic*, pages 523–528. 4 2004.
- [9] Yuce Tekol. Pyswip. <https://pypi.org/project/pyswip/>, 2018.
- [10] 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.

Of what value are these references to the project?

- [1] - This paper and Axelrod’s tournaments helped popularize the iterated prisoner’s dilemma, being a good introduction to game theory that has a programmatic approach for computer science.
- [2] - This project is a relatively new implementation of the iterated prisoner’s dilemma, pushing me towards focusing on other mechanisms, whilst helping me write a proof of concept web application.
- [3] - The efficient version of the Event Calculus as described in this paper is what I plan to understand and use for the agent’s memory and decision making component.
- [4] - This paper introduced me to the other mechanisms that aid in modelling the evolution of cooperation.
- [5] - This paper extended my research on network reciprocity and pushed me towards an interest in indirect reciprocity.
- [6] - This paper is key to understanding and reasoning about indirect reciprocity models and is central to past research on this mechanism.
- [7] - This paper gives a good explanation of the concept of gossip in the spread of information as well as how gossip should be used.
- [8] - This paper is on an architecture for the design of a multi-agent system and has helped motivate my ideas on how to design and implement the Prolog service.
- [9] - This library attempts to implement an interface between Python and Prolog. It has issues and this has motivated me to find a solution using a Prolog web service.
- [10] - This paper is a good description of a model for group selection and has given me an idea for the group level selection extension to my project.