Inter-agent cooperation and betrayal:
Two approaches to decision making in a reality TV show

Jens Oknelid

May 19, 2005
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Michael Minnock
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

In this thesis a game based on the Swedish TV-show "Expedition: Robinson" (known as "Survivor" in the english-speaking world) is studied. It is a game in which players can either cooperate with each other or stab each other in the back. Both cooperation and betrayal are sometimes the best option. The success of a player depends on how well s/he can balance cooperating with the others versus deceiving them.

In this modified Robinson, the only two actions a player can take are communication and voting. The game play consists of two phases: First, the agents communicate privately with each other. This is when they try to gather as much information as possible in order to survive to the next round. Second, each agent cast their vote. The votes, but not who casted them, are displayed and the player with the most votes is removed from the game. Then, a new round begins.

In this thesis an agent that plays the game is implemented using propositional modal logic. The behaviour of this agent is shown to be lacking but at some points interesting. Weak results are presented suggesting some benefits for an agent with a deeper level of reasoning over one with a more shallow deduction.

Finally the voting decision is studied from a decision theoretic perspective. A program is implemented that calculates new probability distributions based on the expected utility of a vote with a previous distribution. A series of tests of different scenarios are carried out using this program. No clear conclusion can be drawn as to how to vote under different circumstances, but it is at least clear that it is not as simple as always voting for the agent the others are most likely to vote for.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis a game called the Robinson game is examined. It is based on the Swedish TV-show "Expedition: Robinson"[1]. This particular game is interesting because it involves inter-agent trust relationships and social interactions, but it is at the same time simple enough to make it feasible to implement agents that are relatively good at playing it. So the Robinson game is examined by trying to implement agents that play the game.

In general, the field of social AI is not very well understood so there are few concrete, practical applications of this kind of work today. But understanding trust and the mechanisms behind it are major research topics in fields from economics to psychology and studying simple games like these could provide new insights which in turn have practical applications. Furthermore, reality shows are not showing any signs of falling out of grace with the television networks and studies of this kind may be of interest to people planning or participating in such shows. For example to evaluate different playing strategies or when reasoning about new shows or changing the rules of established ones.

The outline of this report is quite straightforward. First, the problem and the game studied is briefly described. Then two methods for implementing agents playing the game are examined and results from experimentation with these implementations are presented. Finally these results are discussed and conclusions are made.

[1]The show is in no way only a Swedish phenomenon though, it is known as "Survivour" in the english-speaking world for example

Chapter 2

Problem Description

2.1 Problem Statement

The task in this thesis is to implement a set of agents that play a game where they can either cooperate with each other or stab each other in the back. The behaivour of the agents is to be studied and described in the report. For this to be interesting, a game where both cooperation and betrayal are sometimes the best option must be found. This decision must also be non-obvious.

The game used in this project is derived from the TV show ”Expedition: Robinson”, or ”Survivour” in english. In this show participants are placed on an island where they have to compete with each other in different events. Each episode also ends with a voting session where each participant will cast one vote for any of the other participants. The one who gets the most votes loses and has to leave the show. Who left a certain vote is not publicly disclosed.

In the version of the game used in this thesis the episodes have been translated into a number of rounds. Each round of the game starts with the players communicating in pairs. The communications are private and every player get to talk to every other player. Then they vote, the votes are disclosed and one player has to leave the game. Then a new round begins. In case of a split, the new round begins without removing any player and if the vote has been split two rounds in a row the game ends in a tie.

Every player has to cast one vote. It is not allowed to vote for yourself. The game ends when there are only two players left, and the objective of the game is to be one of those two players.

2.2 Objective

The ultimate goal of this thesis was to find a game with the potential of interesting things happening, and then implement agents indeed displaying some form of interesting behaivour. What is interesting is of course not something that can be determined objectively, so for the purpose of this thesis this word should perhaps be replaced with non-trivial or intelligent in some way. One far-reaching goal that proved to be to difficult to achieve was to create an agent capable of beating a human playing the game.

The secondary objective was to gain insight in what strategies would typically be best for playing the game and what this implies on a more psychological level.

2.3 Methods

Two methods were used to examine the Robinson game. First, an agent was implemented that used propositional modal logic to draw conclusions about whether the other agents were truthful or not. This agent was written in C++. A gui using the Gtk+ library [4] was also written to make looking at the results of a run of the game more convenient. A framework was also written to progress the game forward, actually eliminate agents that were voted out and such things.

When the logic-based agent was found a bit lacking the Robinson game was looked at from a decision theoretic angle and a Matlab program was constructed to do calculations of probabilities of the agents making different choices. These probabilities were calculated based on normalizing expected utility values. The Matlab program was later replaced by a C++ version for speed reasons. A python script was used together with gnuplot[6] to draw the graphs used in this report.

2.4 Related Work

An agent that plays the board game Diplomacy was constructed in [5]. Diplomacy is a game set in the years prior to the first world war were you play one of six nations and the goal is to take over the other nations territory using military force. During each round of the game, the players are allowed to pass notes to each other containing *any* message possible. Thus, a large part of the game is communicating and coordinating your attacks with the other players, making agreements and perhaps breaking them. It is this social aspect of Diplomacy that makes it similar in some ways to the Robinson game. The focus of the work done by Kraus et. al is on getting an agent that plays the game well, not on examining how to play it perfectly. For every front, that is every contact zone between two or more territories, their system evaluates the strategic value of each possible scenario for each player. If it finds one scenario that is beneficial for both itself and someone else, for example together attacking a common enemy, it will propose this strategy to the other player. It is then up to a higher level module to decide which agreements to keep and which to break. This module uses heuristics developed by speaking to expert human players and tuning it until it makes similar choices. Finally, something the Kraus et. al. agent never does is trying to bad-mouth it's opponents or spread false information or any other more advanced conversational tactic one would at least suspect takes place in a game between expert human players. They report good results when their agent plays against beginner and intermediate level humans however.

Chapter 3

Methods

3.1 Preliminaries

The first route taken to model the Robinson game was using modal logic [1]. The knowledge base of an agent was represented as modal logic propositions and the game rules were encoded as axioms in this logic. This approach was thought to have a number of nice characteristics:

– It would be able to model the entire game, with communication, voting and multiple rounds.

– There would be a general way to express communication, namely by encoding it using the same propositions as the rest of the logic.

– Solvers for different kinds of logic are well researched.

However, in practice this approach proved problematic. Encoding the entire game rules in the logic was difficult, specially determining who is excluded given knowledge about certain votes. This is because this requires counting of the number of votes received for a certain agent, something requiring a massive amount of propositions. Since the inference algorithm needs to test resolving all propositions against each other at least once, this would make the inference step far too complex to calculate.

Therefore, the decisions of the individual agent are not made by the logic system but instead by a common algorithm that bases it's decision on information in the logic knowledge base, both information observed and learned from reasoning. Looking at the operations of an agent in figure 3.1 the decide vote and communication stages are

[1]Modal logic can be described in many ways. For the purpose of this thesis the modal logic it is not used in it's traditional sense. Originally, the intention was to use the knowledge and belief axioms from [1], but that was later removed. Modalities as used in this thesis are really just a way of putting labels on formulas, so that you can look at a formula and say that this is a formula of modality X or Y. For a more in-depth discussion of modal logic see [2].

now based on an algorithm analyzing the knowledge base, but they are not made by processing the logics in some more formal way.

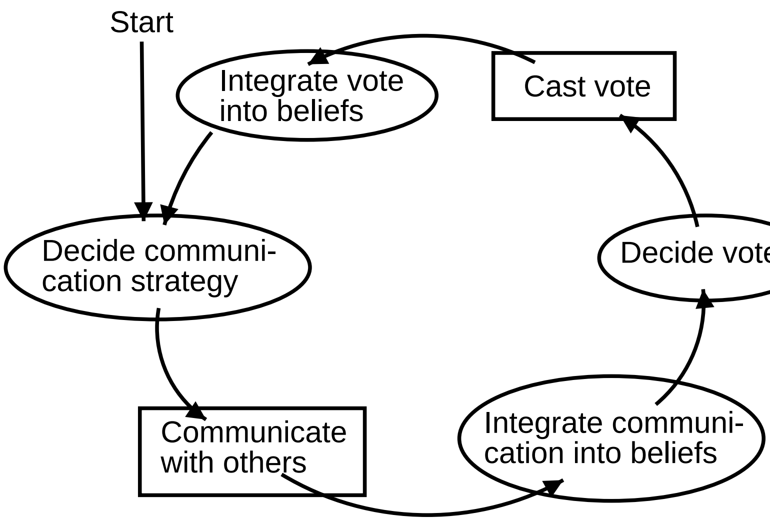

Start
Integrate vote into beliefs
Cast vote
Decide communication strategy
Decide vote
Communicate with others
Integrate communication into beliefs

Figure 3.1: An agent's main control loop. Squares indicate actions the agent takes and circles are internal processing by the agent.

The problem with this logic+programatic approach is that finding a good algorithm to decide who to vote for or what to say is not easy. The ones used in this thesis (described later) were not very impressive, as shall be seen. It was also a big problem that the agents first communicate, and then make a new decision about who to vote for based on what the others said. Weighting the benefit of changing the vote from the one that was spoken against the negative impact of lying is difficult and could be more natural if it was done gradually, ie. the communication decision was re-evaluated in between speaking to each agent. This makes communication order important but this could be solved by randomizing the order in which the agents talk to each other. In the end trying to improve the decision algorithms boiled down to lots of special-casing and tampering that didn't give very good results. Therefore, the Robinson game was studied from a decision theoretic approach in order to gain a better understanding of how to decide who to vote for. This study grew into more or less a new thesis all by itself, and it was beyond the scope of this thesis to unify these different topics again. The decision theoretic approach and the modal logic approach are therefore presented individually.

In order to apply decision theory to the Robinson game it needs to be a bit simplified compared to how it's described previously. First of all, it is only studied one round at a time. This can be seen as both a big drawback and just a minor techicallity: Since the game is studied for many different numbers of contestants, one could argue that this is sufficient because the second round in a four person game is equivalent to the first round of a three person game. It would however have been very interesting to incorporate probability values for all remaining rounds into the simulation and have them depend on each other in some way. This together with making utility depend on the success of an agent in the game as a whole would allow for many interesting effects. This way of looking at the problem was not considered.

The second simplification was to disregard the whole aspect of communication with the other agents. In the original game agents gain knowledge about the actions of the others by communicating with other agents. Thus the information an agent has access to is not always of very good quality, in fact in many cases it's intended to fool the agent into making a poor decision. In the simulations run for this report however, all agents simply know the probabilities of all other agents to make a certain decision. This is done in order to focus the study on one single problem, namely to study what action is optimal for different agents under different scenarios. One way of looking at this is to see the initial probability values as output from some sort of social module that interacts with the other agents and assigns probabilities based on factors like trustworthiness and how reasonable a statement is. This makes disregarding the communication aspect more of a specialization than a simplification of the game.

3.2 Modal Logic Approach

The original decision to look at the game using modal propositional logic was based on convenience. The modalities were needed in order to express that a certain formula is a belief of the agent or a statement of someone. If a statement of another agents as a result of communication is included in the knowledge base and there were no modalities to keep track of who said it, it would be impossible to determine who was the liar when a contradiction was found, because you would have no way of knowing who said it. Since an implementation of an inference module could not be found for any kind of modal logic one would have to be implemented, and the simplest form of logic was then chosen so that the entire body of work would not go into implementing an inference module.

When it came to implementing the strategic capabilities of the agents the first concept considered was game tree search. If we take the communication to be a set of formulas in the logic the game tree will consist of different states for each permutation of formulas an agent can possibly say. Similarly, every possible vote will render a different state. Assuming this search is always exhaustive finding a utility function is easy. This can namely be done by using the inverse of the agents finishing position , perhaps with a penalty if a win is shared between more than two agents since there is a possibility they will all settle for a split early in the game otherwise. However, since the game tree approach requires simulating all the other agents internal processes at every game state (to decide what they know and don't know about the truthfulness of other players) even using game tree search for the voting decision becomes impossible. The number of possible votes is quite low compared to the number of possible things to say even if we restrict what's possible to say to just telling the others who you will vote for, since a communication strategy needs to be decided for each other agent but a voting strategy just once. Therefore the logic needs to be either scaled down significantly, removed, or game tree search needs to be abandoned. Since without the logic there would be no way of telling if somebody is lying or not, and this is very central to the game, game tree search had to go.

After ruling out game tree search, the task was to decide how to implement the agents decision-making capabilities. Originally it was hoped that it would be possible to encode as much of the game rules in the logic that decisions about who to vote for or what to say could be obtained from the inference module. The only thing the agent framework would need to do in this case is to supply information about what's been said and how the votes fell out into the knowledge base, run the inference step and then look for certain propositions that tell what action the agent should take. For example do_vote_{112} could be used if agent 1 should vote for agent 2 in round 1. This was thought to make the agents general and intelligent, because a lot of information about the game and the actions of the other agents would be available to draw conclusions from. It wasn't feasible to do this however, mainly because counting is hard to do in logic and a lot of counting of votes etc. was needed to express when and why people are excluded. What was left was a system where a logic module was used to store knowledge about the game (put there by the agent code when communicating or voting) and to draw conclusions about this knowledge. An agent's decision would be made by an algorithm that evaluate the knowledge generated by the logic module and come up with a strategy in a more ad-hoc way.

3.2.1 Agent modeling

The control flow of the simulated agents can at the highest level be described by figure 3.1. In the figure circles indicate that deduction is being performed by the agent, something is happening in the internals of the agent. Squares indicate that an action is being taken by the agent.

When the program starts agents start out at the "Decide communication" position. They then make one lap around the cycle for each round of the game, ending either in elimination from the game (if they were the one to receive the most votes) or starting another lap at "Decide communication". In the integrate votes/communication steps the inference algorithm is run over the whole knowledge base (the list of formulas) to generate new propositions that constitute the conclusions made from that data. In the cast vote and communicate steps the actual voting and communication is carried out and the main program removes the agent that is excluded in case someone receives majority. Statements in the communication step can in principle be any set of propositions, but the agents currently only state an agent they will vote for.

When deciding what to say, the algorithm used first makes a preliminary decision who to vote for, by using the "Decide vote" stage of the cycle. This means using exactly the same methods for making the decision as is used when actually making the voting decision[2]. This results in one agent to vote for, and one who is the next best option. Then the decision is made to lie to the one who will get the vote and tell that agent that the vote will be for the next best one. All others are being told the truth.

To find someone to vote for the agent calculates a score for each remaining agent. The agent with the lowest score is chosen to be the one to get the vote. The score of each agent is calculated like this:

1. If I trust the agent
   (a) Add one to it's score
2. If the agent trusts me
   (a) Add two to it's score
3. If I don't trust the agent
   (a) Subtract it's score by one
4. If the agent doesn't trust me
   (a) Subtract one from it's score
5. Calculate how many votes I think the agent will get from others (using information from the communication stage)
6. Subtract this number from the agent's score

3.2.2 Axioms

These axioms are added to the list at the end of each round, in versions to suit the round that is coming. The axioms for the first round are added by the framework code, before the actual game cycle starts. The reason for not adding all axioms in the beginning of the games is to avoid adding axioms for agents who have been eliminated from the game. Some of these draw inspiration from [1].

Symbols

The base propositions of the system are described in table 3.1.

[2]There is one modification, and that is that calculating how many votes an agent is believed to receive is not done. Nothing is known about who others will vote for since this knowledge is based on what the agents state during communication, and no communication has of course taken place at the "Decide communication" step.

Symbol	Description
$cast\_vote_{round,vote,agent}$	Agent *agent* cast vote *vote* in round *round*.
$public\_vote_{round,vote,agent}$	Vote *vote* in round *round* was for agent *agent*.
$declared\_vote_{round,agent1,agent2}$	Agent *agent*1 has declared they will vote for agent *agent*2 in round *round*.
$trust_{round,agent1,agent2}$	Agent *agent*1 trusts *agent*2 at round *round*.
$in\_game_{round,agent}$	Agent *agent* is in still in the game at round *round*.

Table 3.1: Symbols used in the axioms

Modalities

Two modalities are used in the system, B and C. $B\ \alpha$ means that the agent doing the resolution believes α, where α is a formula of the propositional modal logic. $C_{round,agent1,agent2}\ \alpha$ means that agent *agent*1 has said the propositional modal logic formula α to agent *agent*2 at round *round*. It also means that the agent doing inference believes this.

Main axioms

$$B\ in\_game_{round,agent} \tag{3.1}$$

Axiom 3.1 is added to the list for each agent still in the game.

$$B(cast\_vote_{round,vote,agent} \Rightarrow \neg public\_vote_{round,vote,agent}) \tag{3.2}$$

Axiom 3.2 is added to the list of prepositions for each agent except the agent the list belongs to and and each vote. (*vote* and *agent*) The meaning of the axiom is that you can't vote for yourself.

$$B(cast\_vote_{round,vote,agent1} \Rightarrow \neg cast\_vote_{round,vote,agent2\neq agent1}) \tag{3.3}$$

Axiom 3.3 is added to the prepositions for each agent *agent*1, each vote *vote* and each agent *agent*2 that is not *agent*1. The axiom says that each vote can only be cast by one person.

$$B(cast\_vote_{round,vote1,agent} \Rightarrow \neg cast\_vote_{round,vote2\neq vote1,agent}) \tag{3.4}$$

Axiom 3.4 is added to the prepositions for each agent *agent*, each vote *vote*1 and each vote *vote*2 that is not *vote*1. This states that a certain vote can only be cast by one agent.

$$B(in\_game_{round,agent} \Rightarrow cast\_vote_{round,0,agent} \vee cast\_vote_{round,1,agent} \vee \cdots \vee cast\_vote_{round,n,agent}) \tag{3.5}$$

Axiom 3.5 is added to the prepositions once for each agent *agent*. n is the number of the last vote. It states that if you're still in the game, you have to vote at least once.

$$B(pulic\_vote_{round,vote,agent1} \Rightarrow \neg pulic\_vote_{round,vote,agent2 \neq agent1}) \tag{3.6}$$

The axiom 3.6 is added to the prepositions list for each vote *vote*, each agent $agent1$ and each agent $agent2$ different from $agent1$. The purpose is to make it illegal for one vote to be for more that one person.

$$B(declared\_vote_{round,agent1,agent2} \Rightarrow (cast\_vote_{round,vote,agent1} \Rightarrow pulic\_vote_{round,vote,agent2})) \tag{3.7}$$

The meaning of axiom 3.7 is that if an agent has declared it will vote for somebody, this means that a vote cast by that agent should indeed be for that somebody. It is added for each agent $agent1$, each agent $agent2$ different from $agent1$ and each vote *vote*.

$$C_{round,me,agent}\ false\ \Rightarrow B\ \neg trust_{round+1,me,agent} \tag{3.8}$$

Axiom 3.8 deals with the breaking of trust. It says that if what an agent has said to me - the agent doing the deduction - resolves to false I don't trust that agent anymore. It is added to the preposition list once for each agent.

Extra axioms

To be able to make a lot more conclusions, knowledge about the voting of the agent doing the deduction is added to the list of prepositions. This is done in the "integrate votes" step of the control cycle, before the inference is conducted. The symbols and modalities are the same as in the previous section.

$$B(cast\_vote_{round,vote,me} \Rightarrow pulic\_vote_{round,vote,myVote}) \tag{3.9}$$

Axiom 3.9 is added for each vote *vote*. It says that if the agent doing the induction, *me*, cast a vote *vote*, that vote must be for the agent ($myVote$) the inducing agent really voted for.

$$B(pulic\_vote_{round,vote,agent \neq myVote} \Rightarrow \neg pulic\_vote_{round,vote,me}) \tag{3.10}$$

Axiom 3.10 is added for each vote *vote* and each agent *agent* that is different from the one the agent doing the induction really voted for. ($myVote$) The purpose of the axiom is to say that if a certain vote wasn't for the right agent, then the vote can not have been cast by the agent doing the induction.

Finally, a gui was constructed to help understand what happens when a simulation of the game is run. Some screen shots will be shown of that gui to illustrate different aspects of the modal logic simulations but it will not be described in detail.

3.2.3 Implementation

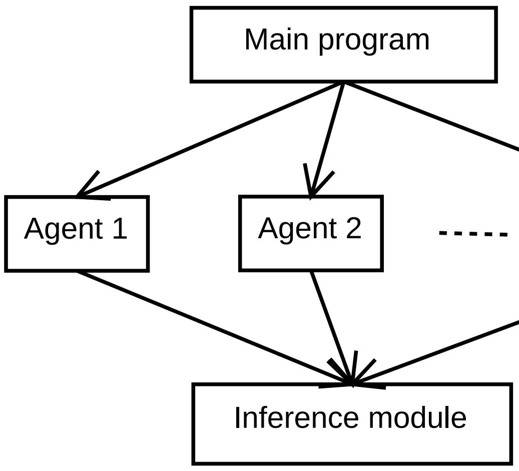


Figure 3.2: A graph of the program structure

The program is written in C++ and consists of three parts: An inference module, an agent part and a main program. A graph of their relationships to each other can be seen in figure 3.2.

The main program is responsible for running the game loop. It creates the agents and implements the game rules, such as making the agents communicate the regulated way (it goes through all agents and makes them communicate pairwise, as stated in the description of the game), counting votes and possibly removing agents from the game.

The Agent class implements all the internal processes of an agent. It maintains a list of formulas that can be thought of as a player of the Robinson games' internal state. The inference module is then used to make conclusions based on this list. When knowledge enters the agent from the main program, for example after the voting when the votes are disclosed, the agent class converts it into formulas and adds it to this list. Knowledge learned from other agents during the communication phase is also entered into this list. The agent part uses this list when it makes decisions about what to say to other agents or how to vote. The internal mechanisms of this module are described in more detail in the section "Agent modeling".

The task of the inference module is to generate the logical consequences of a list of formulas that is given to it. This is done by combining all formulas in the list in a breadth-first order. A maximum depth the combining is allowed to reach can be specified.

The inference rule used is the CNF clause collision used by standard resolution. Since this is normally part of a logics course or an introductory computer science course in a typical masters program it will not be described in detail. People interested in knowing more about it can look it up in [7]. For the purpose of this report it is sufficient to know that CNF is a method for representing logic formulas in a data structure, and that this allows for a technique called clause collision to be used to do inference on the formulas.

The CNF clause collision is however modified to suit the B and C modalities. If α and β are clauses that can be collided with the result γ, $B\alpha$ and $B\beta$ will also be possible to collide with the result $B\gamma$. $C\alpha$ and $B\beta$ or $B\alpha$ and $C\beta$ will also both be possible collisions with the result $C\gamma$. $C\alpha$ and $C\beta$ will however only be collidable if both C:s have identical subscripts.

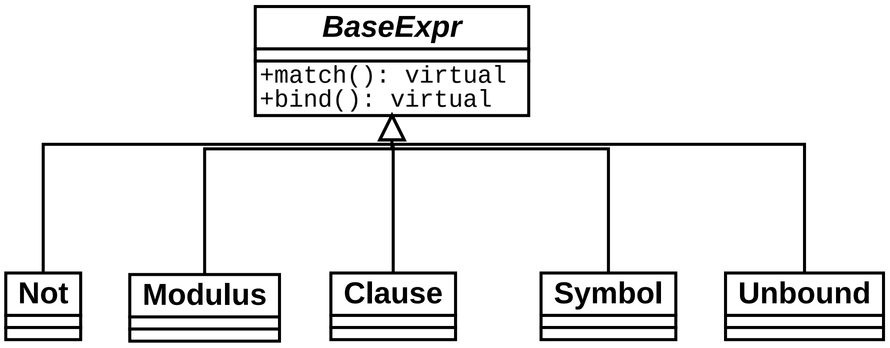


Figure 3.3: Class hierarchy of inference module

The propositions are implemented as a set of classes that define the different operators. Not is a class for example, and it has one pointer to another expression that is the expression that is negated. Note that neither and nor or are present as classes. This is because when using CNF, these operations are implicitly represented. (A CNF clause is a list of expressions that by definition represents that these expressions are or:ed together. Thus each CNF clause represents a formula. A list of clauses in CNF is defined to represent that these clauses are connected by and:s)

As can be seen in figure 3.3 all expression classes inherit from a common base class. This is done to enable pointers to point to any type of expression, when storing expressions in the list or as subexpressions of each other. There are two classes that have no subexpression, Symbol and Unbound. Symbol represents, as you might have guessed, a symbol in the expression. The Unbound class represents an unbound expression that when trying to match two expressions to see if they are equal will match any other not unbound expression. Unbounds are strictly not part of propositional logic, but they were introduced for convenience.

The expression classes have two main purposes: The first one is simply data representation. The second is to provide matching and substitution functions to the collision algorithm. Matching is used when trying to combine two clauses to form a new one, or to put it differently concluding the new clause from the previous two. The matching algorithm works as follows:

1. Look at top level expressions
2. If they are of the same type
   (a) If they are of type Symbol we have a match
   (b) If they are of type Unbound we have no match
   (c) Otherwise apply algorithm on subexpressions
3. If one is of type Unbound
   (a) Set bound_to to be the other expression (the one not being Unbound)
   (b) Set bound_num to be the number of the Unbound
   (c) Return match with bound_to and bound_num
4. Otherwise no match

The main inference function (the function in the inference module called by the agent class at the "Integrate votes" and "Integrate communication" steps) uses the following algorithm to determine if it can collide two clauses, and to generate the new clause if this is the case.

1. Looking at clauses c_1 and c_2.
2. Let n be the number of colliding pairs. Set this to zero.
3. For each pair of expressions $e_1 \in c_1$ and $e_2 \in c_2$:
   (a) If both e_1 and e_2 or neither of them are negations:
      i. Look at next pair.
   (b) If e_1 matches e_2, removing the negation from the one who had it:
      i. Increase n. If an unbound was bound during the match, record its number and what expression it was bound to. Remember e_1 and e_2.
4. If $n = 1$:
   (a) Create new clause $c_{new} = c_1 \cup c_2 \setminus \{e_1, e_2\}$
   (b) If a bind was created during the match, go through c_{new} and replace all unbounds with matching numbers with the bound expression.

3.3 Decision Theoretic Approach

To study the Robinson game from a probability theoretic point of view, a number of probability distributions are calculated and the chance that each agent makes a certain vote is visualized in a graph and examined. The probability that an agent chooses to vote for somebody is calculated by taking the expected utility of that vote given that the others vote according to the probability values already known. Thus calculating the probability values is an iterative process where the values at step $d+1$ depend on the values at step d. Therefore an initial distribution of probabilities is always needed, and the real question asked by this project becomes to analyze what happens with the probability values given a certain initial distribution. A program was therefore constructed to calculate a number of steps in the iterative algorithm and output probability values. This program was then run for a number of steps until some trend in the values could be seen and then these values were analyzed.

To implement a program that calculates probability values according to what was previously outlined, the ability to calculate the maximum expected utility of a certain vote for a certain agent is needed. How this is done is outlined in equation 3.11. In this equation $EU_i(a)$ denotes the expected utility of action a for agent i. $\bar{V}$ is a vote distribution $v_1 = x_1, v_2 = x_2, ..., v_n = x_n$. Note that since the votes are not dependent on each other other than indirectly, the probability of a voting distribution $P(v_1 = x_1, v_2 = x_2, ..., v_n = x_n)$ is the product of the probabilities of the individual votes, $P(v_1 = x_1) \cdot P(v_2 = x_2) \cdot ... \cdot P(v_n = x_n)$. For a more in-depth discussion of expected utility see [3].

$$EU_i(v_i = x_j) = \sum_{\bar{V}}^{\bar{V} \in \{\text{vote configurations where } v_i = x_j\}} U_i(\bar{V}) \cdot P(\bar{V}) \tag{3.11}$$

In this formula a utility function is used, so one must be defined. The one chosen for this project assigns zero utility to being excluded and one if somebody else is voted out of the game. If the vote is split, ie. if more than one agent receive the same number of votes and this is the most votes anybody gets the utility is 1/Number of agents. The function always assigns voting for yourself zero utility. This utility function is described more formally in equation 3.12. In the formal definition n is the number of agents, U_i is utility for agent i and v_i is the vote of agent i. $\bar{V}$ is defined in the same way as for expected utility. (equation 3.11)

$$U_i(\bar{V}) = \begin{cases} 1 & \text{If agent } j \text{ is excluded according to } \bar{V} \text{ and } j \neq i \\ 1/n & \text{If noone is excluded given } \bar{V} \\ 0 & \text{If agent } i \text{ is excluded according to } \bar{V} \end{cases} \tag{3.12}$$

The formula used to calculate the probability of agent i casting a certain vote in step $d + 1$ is described formally by equation 3.13. In this formula $EU_i(action)$ is the expected utility for agent i to take *action* and α is a normalization factor to make the probabilities sum to one. Note that EU_i is based on probability values from step d, and that this is how calculating an optimal probability distribution becomes an iterative process.

$$P_{d+1}(v_i = x_j) = \begin{cases} 0 & \text{If } i = j \\ \alpha \cdot EU_i(v_i = x_j) & \text{If } i \neq j \end{cases} \tag{3.13}$$

Finally these is one problem: if the expected utility EU_i is zero for all actions of an agent, they will be impossible to normalize with a multiplication as in equation 3.13. Therefore a special case is made for this situation (the expected utility of all votes of an agent are zero) and this is described by the formula in equation 3.14.

$$P_{d+1}(v_i = x_j) = \begin{cases} 0 & \text{If } x_j = i \\ 1/(n-1) & \text{Otherwise} \end{cases} \tag{3.14}$$

To implement this in practice the following algorithm is used:

1. Create a table EU with one entry for each agent and each vote.
2. Set all entries in EU to zero.
3. For each possible combination of votes $x_1...x_n$:
   (a) Calculate the probability of this vote distribution and store in p ($p = P(v_1 = x_1) \cdot ... \cdot P(v_n = x_n)$)
   (b) If this vote combination results in one agent being excluded:
      i. Add p to the entry in EU for all agents and their respective vote x_j except the one being excluded.
   (c) Otherwise:
      i. Add p/n to the entry in EU for all agents and their respective vote x_j.
4. For each agent a and vote x:
   (a) Sum the values in EU for all votes for the current agent.
   (b) If sum is zero:
      i. Set $P_{d+1}(v_a = x)$ to $1/(n-1)$ for all votes not for the agent itself. ($a \neq x$)
   (c) Otherwise:
      i. Set $P_{d+1}(v_a = x)$ to $EU(a, x)/sum$.

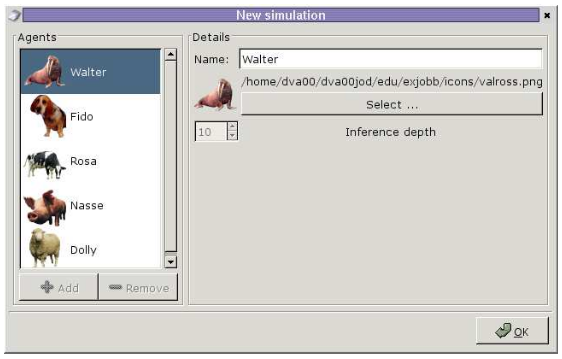
An interesting thing to note about the problem of calculating a new probability distribution is that it is very difficult. Since not even determining if a given probability distribution is a descendant of another one can be done in polynomial time, it is not possible to solve the problem in NP, or at least there is no obvious way. With a complexity of n^n, it is not possible to study very large numbers of agents without taking to extremes (like using a supercomputer or a cluster). The maximum number of agents studied in this report is eight and this is purely because of the time required to solve larger instances of the problem.

Chapter 4

Results

4.1 Modal Logic Approach

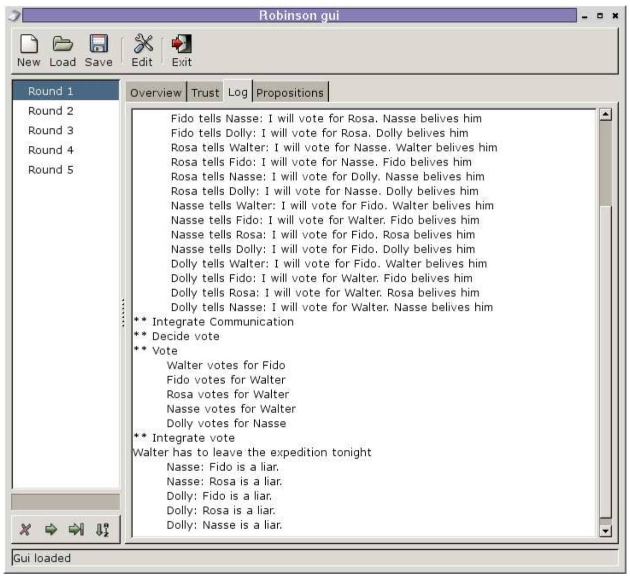
First of all, let's start out with some examples of what kind of things the agents in the logic implementation are capable of figuring out. The most basic thing, and the one that happens most often, is that an agent will say they are going to vote for someone and then no vote for that agent shows up. Then the other agents will understand that the first one was lying. Apart from that there are two facts an agent can use to realize someone is lying: it knows you can't vote for yourself, and it knows who itself voted for. These facts can be combined to form some more complicated conclusions. For example look at a game with five agents, Walter, Fido, Nasse, Rosa and Dolly. (see figure 4.1) A log of the first round of this game is also shown in figure 4.2. Here we see that Dolly figure out that both Fido, Rosa and Nasse have been lying. Dolly knows that she cast the vote for Nasse, and therefore Walter must have voted for Fido because he can not have voted for himself. Therefore both Fido, Rosa and Nasse must have voted for Walter.



Walter
Fido
Rosa
Nasse
Dolly

Name: Walter
/home/dva00/dva00jod/edu/exjobb/icons/valross.png
10 Inference depth

Figure 4.1: Agents in the game as shown by the gui.



```
        Fido tells Nasse: I will vote for Rosa. Nasse belives him
        Fido tells Dolly: I will vote for Rosa. Dolly belives him
        Rosa tells Walter: I will vote for Nasse. Walter belives him
        Rosa tells Fido: I will vote for Nasse. Fido belives him
        Rosa tells Nasse: I will vote for Dolly. Nasse belives him
        Rosa tells Dolly: I will vote for Nasse. Dolly belives him
        Nasse tells Walter: I will vote for Fido. Walter belives him
        Nasse tells Fido: I will vote for Walter. Fido belives him
        Nasse tells Rosa: I will vote for Fido. Rosa belives him
        Nasse tells Dolly: I will vote for Fido. Dolly belives him
        Dolly tells Walter: I will vote for Fido. Walter belives him
        Dolly tells Fido: I will vote for Walter. Fido belives him
        Dolly tells Rosa: I will vote for Walter. Rosa belives him
        Dolly tells Nasse: I will vote for Walter. Nasse belives him
** Integrate Communication
** Decide vote
** Vote
       Walter votes for Fido
       Fido votes for Walter
       Rosa votes for Walter
       Nasse votes for Walter
       Dolly votes for Nasse
** Integrate vote
Walter has to leave the expedition tonight
       Nasse: Fido is a liar.
       Nasse: Rosa is a liar.
       Dolly: Fido is a liar.
       Dolly: Rosa is a liar.
       Dolly: Nasse is a liar.
```

```
Gui loaded
```

Figure 4.2: A log of the first round of a game.

All is not well with the program however. In fact, some of the decisions made by the agents are quite silly. Looking at the previously presented example it's quite clear that some agents make bad decisions. When the voting algorithm considers voting for an agent points are added for each statement from other agents that they will vote for this agent. The problem is that it's not always beneficial to change your vote to suit the majority. If all other agents will vote for someone it is not necessary to change your vote for example, because that agent will be excluded anyway. It was realizing problems like this that led to the study of the raw decision theoretic aspects of the Robinson game.

To examine if the added sophistication in the form of the added logics module gave the agents any benefits in playing the game a small test was carried out. It was suspected that the outcome of the game was more or less random and that an agent playing according to some basic strategy would have equal chance of winning as the agents that try to analyze the game. The test consisted of running a game with five agents, where two of them searched twelve steps ahead in the integration steps in the cycles and three only searched to a depth of eight. The game was run ten times and the number of agents of each kind that remained in the last round can be seen in table 4.1. The agents from the more advanced group reach the last round ten times, and the others reach that round eleven times. This means that the "smarter" agents make up 47.6% of the winning groups but statistically, if everything was random, they should only be 40% of them.

Simulation number	Agents in winning round of depth 8	Agents in winning round of depth 12
1	1	1
2	2	0
3	2	0
4	0	2
5	1	1
6	1	1
7	2	0
8	1	2
9	0	2
10	1	1
Total number:	11	10

Table 4.1: Results from simulations with agents of different search depth of the inference module.

4.2 Decision Theoretic Approach

In this section, a number of graphs are displayed to show results from different scenarios. In these graphs, the axes are not labeled to avoid clutter and to allow more space for the actual graphs. A little explanation might therefore be in place: Each figure contain one subgraph for each agent in the simulation. (which one is indicated by a header) In this subgraph one curve is plotted for each agent, and this curve shows the probability of the agent "owning" the graph voting for one of the other agents. Which one is determined by the line style as indicated in each figure. Here it is important to pay close attention, since the curves often overlap, but looking carefully it is possible to see which agents are being referred to. Anyway, the X axis (horizontal) scales with the number of iterations. Thus, the values of the probabilities at X=0 are the starting conditions, the input to the program. These are also displayed in a separate figure with arrows between the agents indicating their initial probability of voting for each other. On the Y axis the probability value is shown. Following the curve labeled "$P(V_i) = 2$" in the graph labeled "Agent 3" will thus show how the probability that agent 3 will vote for agent 2 changes as the iterations progress. Looking at the Y value at X=24 will tell the probability that Agent 3 would vote for agent 2 after 24 iterations.

4.2.1 Cycles

Cycles are scenarios where the initial probability is very high that $v_1 = 2$, $v_2 = 3$, ... , $v_n = 1$ is very high. These scenarios were studied because it was thought that they might cause oscillating probabilities, ie. the probabilities would never converge. That proved not to be the case, instead they converged to a configuration of maximum entropy in all cases were there was a small probability distributed equally for each agent to vote for anyone but themselves. Setting the probability to one throughout the cycle leads to

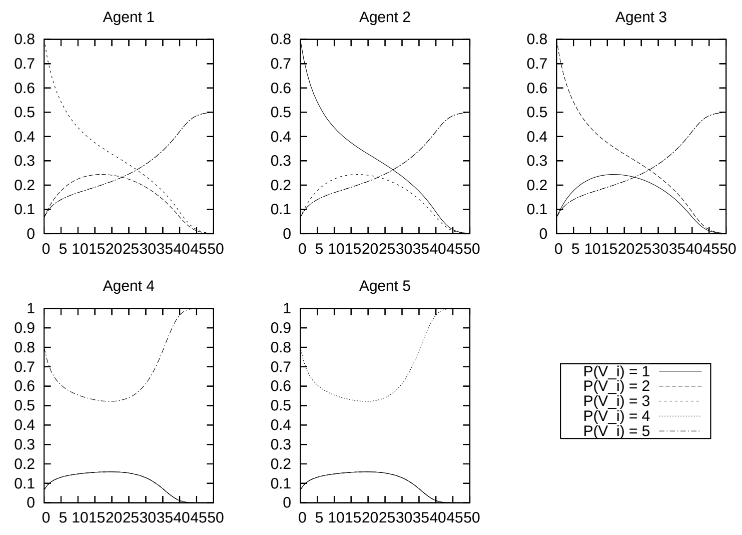


Figure 4.3: Five agents divided into two cycles.

a situation where the probabilities remain one all around the cycle forever.

4.2.2 Two cycles

When two cycles are present in the initial configuration, the results are more interesting. The hope when constructing these scenarios was that when the two cycles were not equal in size, the agents in the larger cycle would gain more and more probability to vote for the agents in the smaller cycle. This is also what happens. Interesting to note is that in the five agent example (figure 4.3, starting conditions described in figure 4.4) the agents in the smaller cycle first gain probability in voting for the agents in the larger one, but as these get more and more likely to vote for the agents in the smaller one these start going after each other instead. Something like this can not be seen in the example with seven agents however. In these scenarios a small probability that an agent is going to vote for anyone other than itself needs to be added for something to happen, if the edges in the cycles have a probability of one nothing ever changes. Also, if the two cycles have the same number of members they stabilize, the probability of the agents to vote for the agent next in their cycle reaches one and all other probabilities reach zero.

A final remark about cycles is to look at what happens if one agent is added with equal probability to vote for anyone of the others. This was studied for six, seven and eight agents divided in cycles of two and three, four and two, four and three and five and two. The results from these runs show that in the six agent case the single agent acts as a stabilizer that causes the cycles to strengthen until they reach probability one. With seven agents the single agent and the larger group become increasingly likely to vote for the agents in the smaller cycle. The two in the smaller cycle continue to vote for each other. When it comes to the eight agent scenarios, these show two different

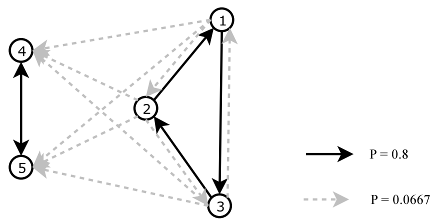


Figure 4.4: Starting conditions for scenario in figure 4.3.

Number of agents	Division	Result
4	2+2	Maximum entropy.
5	3+2	Group of tree votes for group of two, group of two votes for each other.
6	3+3	Maximum entropy.
6	4+2	Both groups continue to vote for agents in their group.
7	4+3	Small group votes for larger, larger unchanged.
7	5+2	Small group votes for larger, larger unchanged.

Table 4.2: Two groups of agents voting for agents in their own group.

trends: in the one were the largest cycle is four agents long all agents start distributing their votes evenly among the agents in the larger cycle. When the largest cycle consists of five agents, all agents end up spreading their votes among the two in the smaller cycle, except the single agent that didn't belong to any cycle. This agent starts out with increasing probability of voting for the two in the smaller cycle but later in the simulation switches until all probability is distributed among the five agents in the larger cycle.

4.2.3 Groups that vote for each other

To generally describe these scenarios, as the heading implies they consist of at least two groups of agents. These agents have a larger probability of voting for other members of the group than of voting for somebody else outside the group. Interesting aspects to study about groups like these is if agents remain likely to vote for members of the group or if the groups will break up. It will also be interesting to see what happens when a single agent with no preference for the agents in either group is added.

When it comes to plain groups without any extra agents no general trend can be found in the results. Test were run with four, five, six and seven agents. A total of six were run. The results are summarized in table 4.2. As can be seen, the two divisions where both groups are of equal size (2+2 and 3+3) behave in the same way, they collapse into maximum entropy. Also, both seven agent examples show the smaller group going

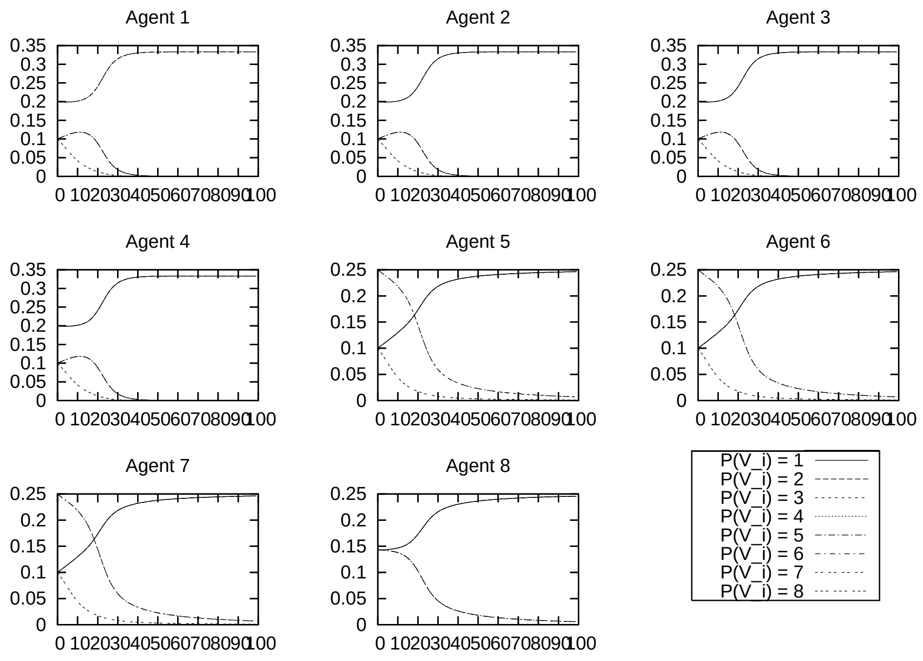


Figure 4.5: Eight agents divided into two groups that vote within themselves and one agent added.

after the larger one. This is consistent with what would be expected. However, contrary to this the five agent example show agents in the larger group become more and more likely to vote for the ones in the smaller group. The second example with six agents also show unexpected behavior, namely both groups remain faithful to the initial conditions and the agents show increasing probability of voting for people in their original group.

When it comes to adding another agent the results display a lot more consistency. In the four tests run (six agents, 3+2+1, seven agents divided in groups 4+2+1 and eight agents in groups of 4+3+1 and 5+2+1) all of them show the smaller group and the single agent gain probability in voting for the larger group. The agents in the larger group continue to vote among themselves. An example of this, with eight agents divided in two groups of five and two and one undecided can be seen in figure 4.5. The initial probabilities for the agents in this example can be found in figure 4.6

4.2.4 Groups that vote for other groups

A very common situation from TV-show is having two groups of agents who are very likely to vote for agents belonging to the other group. At first glance it seems like this will be a very stable situation if the groups are of the same size. It is a more open question what will happen if one group is smaller than the other. A final thing to look at is what happens when one agent who is undecided between the two groups is added.

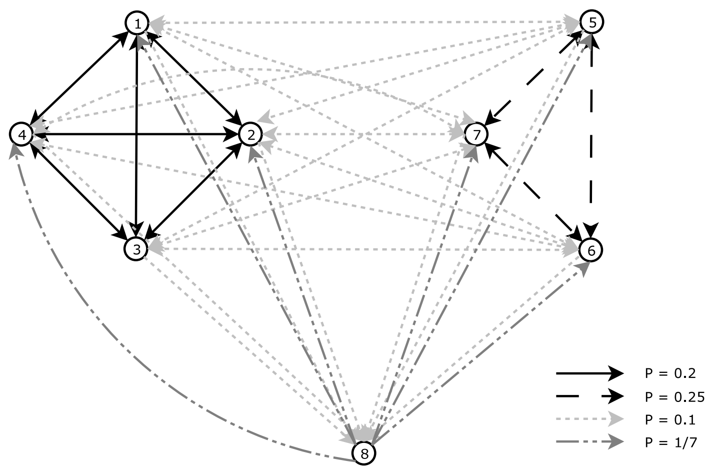


Figure 4.6: Starting conditions for figure 4.5.

To examine the first case when two groups who want to vote for each other three tests were run: four agents in groups of two, six agents in groups of three and eight agents in groups of four. What happens in all these cases is that the scenario converges into maximum entropy.

The scenarios where there are two groups that differ in size offer no surprises. In all cases the agents in the larger group gain more and more probability of voting for the agents in the smaller group. The agents in the smaller group become more and more likely to vote for the others in their group and they become unlikely to vote for agents in the larger group. The examples run were five agent in groups of three and two, six agents in groups of four and two and two seven agent examples divided in groups of four and three and five and two. In all except one case the probability of the agents in the larger group to vote for others in the same group reach zero. This doesn't happen in the case with six agents (figure 4.7 and 4.8) however. In all examples, the probability of agents in the smaller group voting for others in the smaller group reaches zero.

When an additional agent is added to the groups two things generally happens. When the number of agents in the groups are equal (2+2 or 3+3 were the cases tested) the lone agent doesn't favor any particular group, and the other agents ignore the lonely agent and their probabilities get distributed equally among all other agents.

In the scenarios with different sized groups something a bit different happens. This kind of scenario was tested with six, (3+2+1) seven (4+2+1) and eight (4+3+1 and 5+2+1) agents. In all cases, the agents in the larger group ended up increasing their probability of voting for agents in the smaller group until all other probabilities were zero. Also in all cases the smaller group broke up and the agents in that one became more and more likely to vote for each other as the iteration continued. The lonely agents behavior was a bit less consistent though. In all cases except one, namely the seven agent one, this agent ended up distributing it's probability of voting equally among

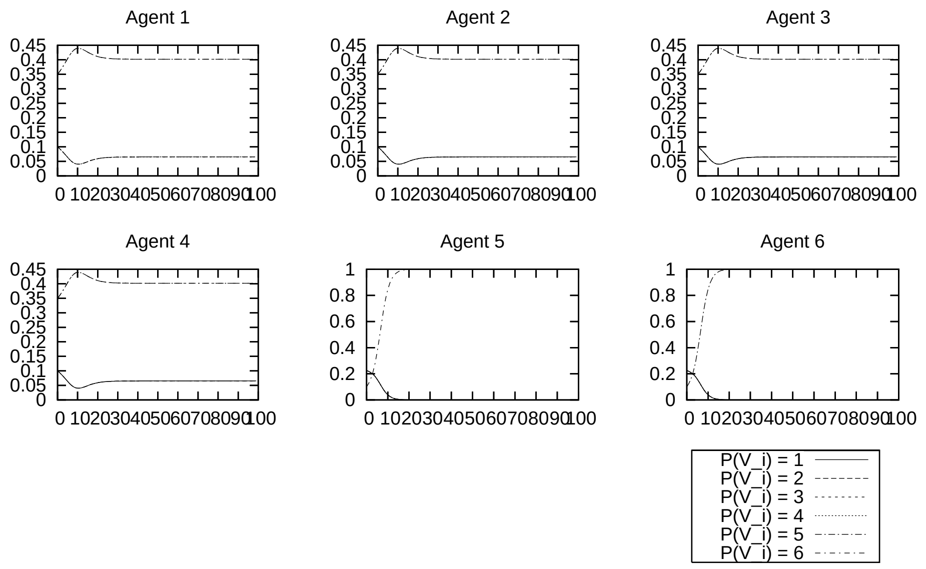


Figure 4.7: Six agents in two groups voting for each other. Note that the probability of agents 1-4 voting for agents 1-4 doesn't reach zero.

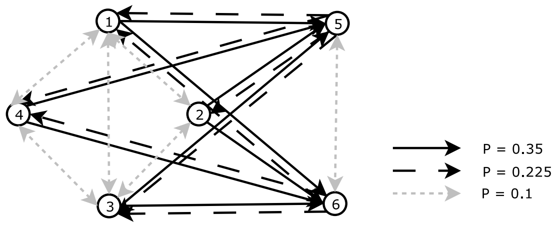


Figure 4.8: Starting conditions for the scenario in figure 4.7.

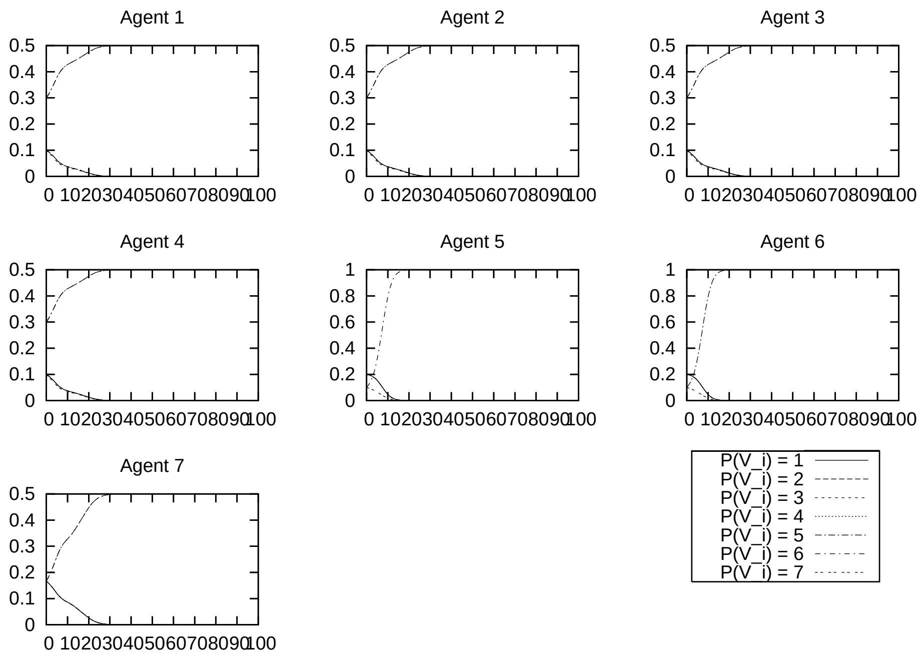


Figure 4.9: Seven agents in two groups voting for each other, with one undecided agent added. Note the behavior of agent 7 compared to agent 6 in figure 4.10.

the agents in the larger group. In the seven agent example however, the lonely agent instead became more and more likely to vote for the agents in the smaller group. Graphs of this example can be found in figure 4.9. Graphs from the six agent case are provided in figure 4.10 so that the difference can be seen.

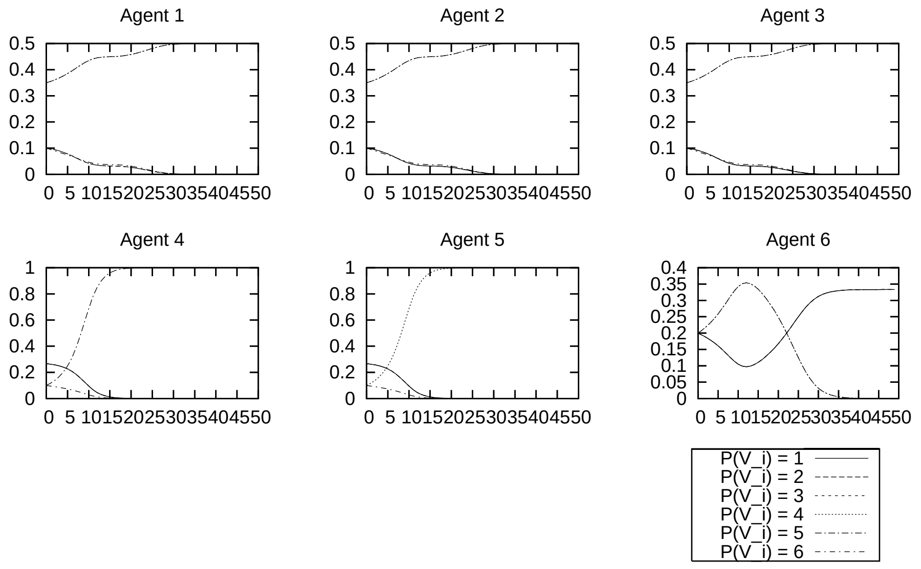
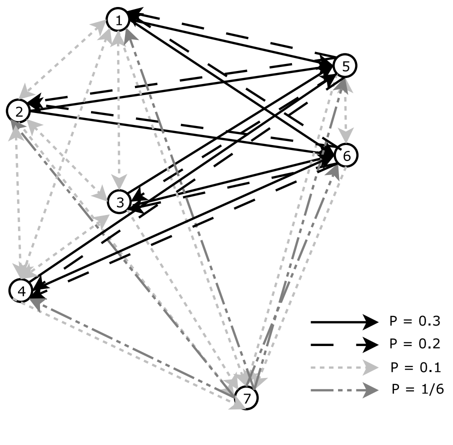
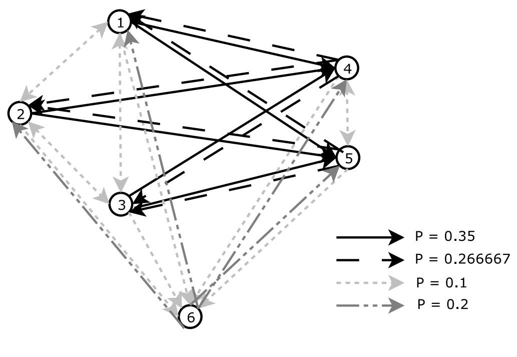


Figure 4.10: Six agents in two groups voting for each other, with one undecided agent added.



(a) 72g+12-cyc



(b) 62g+1-cyc

Figure 4.11: Starting conditions for (a) figure 4.9 and (b) figure 4.10

Chapter 5

Conclusions

Generally speaking a big conclusion of the author of this thesis has been a lesson learned about planning a project like this and making necessary limitations to the scope of the problem studied. Initially it was believed that a pure logic based approach that includes decision making would be possible, but this proved not to be the case. When this was found out, it was thought that with some tweaking a simple algorithm like the one used for making the voting decision would lead to relatively intelligent behaivour. When this also proved to be false the problem was approached at it's most fundamental level with the decision theoretical study. In retrospect, it would have been nice if this route had been taken earlier.

5.1 Modal logic approach

That the modal logic part of this thesis is not very impressive, that is something most would agree on. The more interesting question to ask is why this is the case. Two main reasons have been identified:

– The implementation of the communication phase is very restrictive (the only thing an agent can say is who it will vote for)

– The mechanisms for making a good voting or communication decision are not very well understood.

To clarify the first part of this statement, the model of the game becomes very shallow when the only thing an agent can say to another is "I will vote for X". Since the framework for communication allows for more or less anything to be said, a statement is just a list of propositions, it would have been easy to extend the communication. Adding the ability of referring to other agents ("X told me Y") and negation ("I will not vote for X") to the set of possible statements would have been quite straightforward and has the potential of making the game much more interesting. The problem is just that the additional propositions needed to reason about this new information would strain the inference module beyond what it's capable of dealing with in terms of computational complexity. As it is not well understood how to make good decisions about voting, having to decide what to say with this many options is difficult.

Also, when I say that it's not understood how to make voting decisions, this refers to how to write a simple algorithm that makes the decisions. Game tree search is well researched and would probably work if one didn't care about the time it would take to get an answer. Encoding the full game in the logic, with counting of votes and everything and then having a strategy come out of the inference step is also a possible way of making the decision. Since the agent would be able to draw every possible conclusion about the internal state of the other agents it would be able to avoid actions with bad outcomes.

Implementing a system like this in C++ results in a lot more code than if using some other languages. A logical programming language like prolog could probably make writing a program like this a lot easier.

5.2 Decision theoretic approach

The suspicion before actually running the tests and carrying out the study was that the optimal decision would be to vote for the agent the others were most likely to vote for. This would mean summing up the probabilities for each agent that the others would vote for this agent and distributing your own probability equally among those with the highest sum. It was believed all scenarios could very well come down to this. When the tests were run however, it became clear that this was not always what happened. So the major conclusion from the decision theoretical tests is that the voting decision is not trivial. One can not simply vote for the agent(s) who are the most likely to get votes and expect this to be an optimal strategy.

The cases where an additional agent is added are interesting from a psychological standpoint. It is sometimes the case in the TV show that additional participants are introduced into the game after a while when the participants have already formed alliances with each other. What typically happens in this scenario is that the newcomer is eliminated right away and everything continues as before. In the simulation runs however, no scenario with two groups willing to vote for each other and one undecided agent show the others going after the undecided agent. If this is because humans tend to act non-optimally in situations like these (they are after all known for being a bit wary of the new and/or unknown) or because the model used in the simulations doesn't capture this situation very well is hard to tell from the results however. It could certainly be a combination of both.

There were also some technical problems when performing these tests that may be worth knowing for people doing similar work in the future. First of all, probabilities of zero should be avoided in the input. This also means probabilities of one should be avoided, because this implies setting all but one probability to zero. The reason for this is because a zero field in the probability table means that the expected utility of a vote configuration containing that vote will be zero no matter what the utility of that configuration is. To put it differently, an agent having an initial probability of zero for a certain vote will never consider that vote because the expected utility will always be zero, even if that vote has a very high utility.

5.3 Future work

Generally, this thesis is full of loose ends. Results from the decision theoretical study were never integrated into the logics based agents. This was a major reason for studying the decision theoretic aspects in the first place, but a lack of time prevented any work in this direction. A lot more could also be done to improve the inference module in order to have the search reach a deeper level. The biggest limitation of the logics-based agents are not the resolution engine but the decision algorithm however. A more efficient logics module would have made it possible to take a more direct logic-based approach and make actual decisions by resolution. This would have been very interesting to see but would require a logics module so much faster that it's a very hard problem to construct one. Perhaps a predicate logic based approach would make this feasible, since this will at least reduce the number of formulas that are needed to describe the game. It remains to be seen if this will increase the actual throughput of the inference module, or if the benefit from having a smaller number of formulas will be swallowed by the larger overhead needed to unify variables etc.

Chapter 6

Acknowledgements

A big thanks to Michael for supervising this thesis, helping me stay motivated and keeping the project somewhat on track.

References

[1] Ronald Fagin et. al. *Reasoning about knowledge.* MIT Press, Cambridge, Mass., 1995.

[2] Patrick J. Hayes John McCarthy. Modal logic. Webpage, March 2005. `http://www-formal.stanford.edu/jmc/mcchay69/node22.html`.

[3] S. Russell P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, Saddle River, New Jersey, 1995.

[4] The Gtk Project. The gtk+ toolkit. Webpage, March 2005. `http://www.gtk.org`.

[5] Daniel Lehmann Sarit Kraus. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171, 1995.

[6] Colin Kelley Thomas Williams and contributors. Gnuplot hompepage. Webpage, March 2005. `http://www.gnuplot.info`.

[7] Ramin Yasdi. *Logic and Programming in Logic.* Immediate Publishing, Hove, 1997.