Preposterior Decision Analysis in the game Cluedo

Robert Iepsma
6139108

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr T.M.V. Janssen

Institute for Language and Logic
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July 10th, 2012

Abstract

Cluedo is a detective board game where players have to obtain information about an imaginary murder by making suggestions about this murder. The aim of this research is to make a computer player for the game Cluedo and investigate what is needed for such computer player to compete with human players. In this paper three programs are described, each with a way different way to store and maintain knowledge about the cards in the game in a knowledge base, introducing knowledge about knowledge of opponents into the game Cluedo. Each program makes use of different reasoning for deciding which observation grants the most utility for winning the game, using the gain of information as measure for utility. The game results show that the increase of information gain from observations does not increase the performance of the program. The main result of the research is that the kind of knowledge base used for maintaining knowledge about the cards held by opponents does effect the chances of winning the game, and that the programs do outperform human players.

1 Introduction

A: *The game of Cluedo*

The board game Cluedo is a detective game with the purpose of determining the guilty suspect, the weapon and room of an imaginary murder inside a mansion. Players are able to get information about this murder by entering rooms, making suggestions about the murder and obtaining responses from other players refuting or supporting these suggestions. A game of Cluedo includes 21 different playing cards representing possible suspects, weapons and rooms.

At the start of a game of Cluedo three cards, one suspect card, one weapon card and one room card are randomly picked, and placed in the middle of the board. These three cards are not shown to any player in the game and determine the actual guilty suspect, the murder weapon and the room in which the murder took place. The players of the game have to find out which these three cards are in order to win the game. The rest of the cards are shuffled and equally distributed among the players.

When a turn goes to a player, he gets to role a dice which indicates the amount of steps that player can take to move around on the board. There are nine different rooms on the board, and if a player enters a room, he gets the option to make a suggestion about the murder. A suggestion about the murder involves naming three cards; one suspect, one weapon and one room. The room named in the suggestion has to be the room where player is making the suggestion from. Once a player made a suggestion, other players have to respond to this suggestion, in a clockwise direction, until the suggestion is refuted or reaches the player who made the suggestion. There are two ways in which players can respond to suggestions; supporting it or refuting it. A player has to refute a suggestion if that player holds one of the cards mentioned in the suggestion by showing the player that made the suggestion that card. The other players do not get to see this card. When a player is not able to refute a suggestion, he supports the suggestion by saying so, in a way that every player can hear it.

When a player thinks he knows which cards are in the middle of the board, he can make an accusation, naming the three cards. That player then checks whether his accusation is right. If so, he wins the game, if not, he is not allowed to make another accusation and is no longer able to win the game, but still has to keep responding to suggestions.

B: *Research*

In this article the development of an intelligent computer program that can play Cluedo is researched, where the requirements of such program, to be able to successfully defeat experienced human players, are investigated. The moves of a player in Cluedo allow the player to get access to observations, which will grant him an increase in utility for winning the game. Therefore the game of Cluedo can be viewed at as a preposterior decision problem, in which the expected increase in utility obtained from gaining access to additional observations is calculated before making a decision. The same principles arise in modern surveillance systems, where autonomous vehicles move around in an environment in order to collect measurements.

In Cluedo the observations are the responses of opponent players to suggestions made by all players. When a player has the option to make a suggestion, the responses from each opponents have to be predicted to calculate the increase in utility from the observations. The problem that arises in deciding which observation grants the most utility in a game of Cluedo is that the responses from opponent players are based on which cards they hold. Since the cards are randomly distributed among the players, a player does not know which cards his opponents hold. Therefore the response from that opponent, and the increase in utility, is a matter of chance.

So far the reasoning about which suggestion grants the most utility has only been researched in such way, that the calculation of the increase in utility from an observation only involves the knowledge a player gets from this observation. In the game Cluedo, the observations made by one player also give knowledge to opponent players. Thus trying to maximize the amount of knowledge obtained from an observation might not give an advantage and therefore might not be the optimal way to win the game. Consequently an addition to such preposterior decision solution can be made where the utility of an observation is calculated using not only the increase of knowledge that the player gets, but also the knowledge about the increase of knowledge all opponent players get. This way, maximizing the utility, it is possible to maximize the advantage a player gets over his opponents. This could also be used in the modern surveillance systems, where for example in case of hostile environments, we want to minimize the knowledge others get about our obtained knowledge.

The performance of such program that also takes knowledge about knowledge of opponent players into account will be compared to the performance of a program that only uses the knowledge of the program itself. The performance against human players as well as the performance against each other is measured, to come to an answer to the question: What is needed for a computer program to compete with human players in the game Cluedo?.

2 Related work

Previously to this work, some articles have been published about the game Cluedo. A way for players to store and handle the knowledge ob-

tained during a game of Cluedo is developed by H.P. van Ditmarsch [6], where he shows by example what knowledge can be derived from different observations. H.P. van Ditmarsch has also made an article fully dedicated to the description of game actions in Cluedo [7], describing what actions are possible and what kind of knowledge can be deduced from these actions. The research of van Ditmarsch is used in the program design for the research in this paper.

Clare Dixon has made a similar paper showing how to specify Cluedo using temporal logics of knowledge and proving statements about the knowledge of the players using a clausal resolution calculus for that logic [2]. The same clausal resolution is used in the program design in this paper. Chenghui Cai and Silvia Ferrari made a Bayesian network approach to develop an automated computer player for Cluedo, viewed at as a preposterior decision problem [3]. Later Silvia Ferrari, Senior Member, IEEE, and Chenghui Cai researched an information-driven sensor management problem, where hidden variables are inferred by selecting a sequence of measurements associated with multiple fixed targets distributed in a sensor workspace [4]. The methodology of the research is illustrated through the board game Cluedo, where they created a connectivity graph to plan the best paths for Cluedo pawn movement.

3 Program design

Four different type of programs are implemented:

1. A simple program, which does not try to find an observation which grants the highest utility.
2. A dumb program, which does make suggestions like the simple program, but does not make any accusations during the whole game.
3. A program, which we refer to as: "Program K", that only uses obtained knowledge for calculating the observation which grants the highest utility.
4. A program that in addition to the obtained knowledge, also uses knowledge about the knowledge of the opponents for calculating the observation that gives the highest utility, which we refer to as: "Program K&KaK".

A: *Knowledge base*

The intelligent computer program needs a knowledge base to store obtained knowledge, and derive new knowledge from this stored knowledge. The knowledge stored in this knowledge base is used to reason about the moves the program will make during the game. The knowledge base is constructed with propositional logic [1] clauses containing knowledge about the cards in the hands of opponents. For each opponent a main clause which includes multiple knowledge clauses about cards is maintained, where the number in front of the main clause represents the player of which the knowledge is about. The variables in the knowledge clauses represent the cards. The main clause for each opponent has the logical value $true$, and so do all knowledge clauses inside this main clause. When a knowledge clause inside the main clause of certain opponent contains one variable of a card without negation, it means that opponent holds this card. When a knowledge clause contains one variable with negation of a card, it means that opponent does not hold this card. When a knowledge clause contains multiple variables, it means at least one of the

variables within that clause is true. For example, a main clause with knowledge clauses about the cards of one opponent:
1:$[\{A\}, \{\neg B\}, \{C, D, E\}, \{\neg F, \neg G, \neg H\}]$

The knowledge base of the simple program and Program K contain one of these main clauses for each opponent. For example in a game with 3 players where Program K is player 1, Program K holds the cards A and B and has the knowledge that player 2 holds the card C. This results into a knowledge base containing the following database:
2:$[\{\neg A\}, \{\neg B\}, \{C\}]$
3:$[\{\neg A\}, \{\neg B\}, \{\neg C\}]$

The knowledge base of Program K&KaK contains this database, plus another database like this for each opponent, containing the knowledge of the program about their information about their opponent cards. For example in a game with 3 players where Program K&KaK is player 1, Program K holds cards A and B and he has the information that his opponents know that Program K holds card A, this results in:
1$\{$2:$[\{\neg A\}, \{\neg B\}]$ 3:$[\{\neg A\}, \{\neg B\}]\}$
2$\{$1:$[\{A\}]$ 3:$[\{\neg A\}]\}$
3$\{$1:$[\{A\}]$ 2:$[\{\neg A\}]\}$

The knowledge base gets updated every time new knowledge is obtained by the program. New knowledge can be obtained in three different ways, which is also described in the paper of van Ditmarsch [7] :

1. At the beginning of the game. When the cards are distributed, both programs add one knowledge clause per card they hold, containing the negation of that card, to each main clause of the opponents, because their opponents can no longer hold these cards.

2. When the program makes a suggestion, and other players respond to this suggestion, there are multiple cases situations where the knowledge base gets updated:

   (a) When an opponent player supports the suggestion, referred to as "supporter", this means he does not hold any of the three cards mentioned in the suggestion. Therefore both programs add three knowledge clauses, one for each card, containing the negation of this card, to the main clause of the supporter in their database. Since the response to this suggestion is known by every player of the game, Program K&KaK now knows that all his opponents know that the supporter does not hold any of the three cards. Therefore Program K&KaK also adds the same three knowledge clauses to the main clause of the supporter in the database of all other opponents.

   (b) When an opponent player refutes the suggestion, referred to as "refuter", he shows one of the cards mentioned in the suggestion to the program. This means the program now knows the refuter has that card and all other opponents do not have this card. Thus both programs add multiple clauses to their knowledge base:

      i. One knowledge clause, containing the shown card, to the main clause of the refuter.

      ii. One knowledge clause, containing the negation of that card to the main clause of all other opponents.

   Since all opponents see the refuter showing a card to the program, they

know the refuter has one of the three cards mentioned in the suggestion. Therefore Program K&KaK:

i. Adds one knowledge clause, containing the three cards mentioned in the suggestion, to the main clause of the refuter in the database of all other opponents.

ii. Adds one knowledge clause, containing the three cards mentioned in the suggestion with negation, to the main clause of all opponents excluding the refuter in the database of all other opponents.

3. When an opponent makes a suggestion, referred to as "suggester", and other players respond to this suggestion.

   (a) When the program supports the suggestion, Program K does not obtain any knowledge, but Program K&KaK knows all his opponents know he does not hold any of the three cards mentioned in the suggestion. Therefore he adds three knowledge clauses, one for each card, containing the negation of this card, to the main clause of himself in the database of all opponents.

   (b) When the program refutes the suggestion, the program shows one card to the supporter. Program K does not obtain any knowledge, but now Program K&KaK knows that the supporter knows the shown card, and all other opponents know he has one of the three cards. So Program K&KaK:

      i. Adds one knowledge clause, containing the shown card, to the main clause of himself in the database of the suggester.

      ii. Adds one knowledge clause, containing this card with a negation, to the main clause of all other opponents of the suggester in the database of the suggester.

      iii. Adds one knowledge clause, containing all three cards mentioned in the suggestion, to the main clause of himself in the database of all opponents, excluding the suggester.

   (c) When another opponent player supports the suggestion, the knowledge base update is the same as described above, where the program itself makes the suggestion, and gets supported by an opponent.

   (d) When another opponent player refutes the suggestion, he shows the suggester one of the cards mentioned in the suggestion. The program now knows that the refuter has one of those three cards. Because the refuter holds one of these cards, the program knows that the other opponents can not hold one of these cards. Now both programs:

      i. Add one knowledge clause, containing the three cards mentioned in the suggestion, to the main clause of the refuter in the programs database.

      ii. Add one knowledge clause, containing these three cards with a negation, to the main clause of the other opponents in the programs database.

   Since all players see that the re-

futer shows a card to the suggester, they know the refuter has one of the three cards mentioned in the suggestion, and all other opponents of those players do not hold one of the three cards. Program K&KaK does not know which of the three cards was shown to the suggester, he still only knows that the suggester knows that the refuter has one of the three cards. Therefore Program K&KaK:

i. Adds one knowledge clause, containing the three cards mentioned in the suggestion, to the main clause of the refuter in the database of all opponents excluding the refuter.

ii. Adds one knowledge clause, containing these three cards with a negation, to the main clauses of all players, excluding the refuter, in the database of all opponents, excluding the refuter.

After a knowledge base update, both programs check the clauses of each opponent to see whether resolution can be applied. Resolution can be applied to clauses that contain multiple variables, referred to as "multi-clauses", with the use of clauses that contain only one variable, referred to as ' 'single-clauses", and is applied in the same way for both programs. Resolution is only applied within one main clause of a certain player and is done in two steps, which is also explained in the paper of Dixon [2]:

1. First, when there exists a multi-clause, which contains a certain variable of a card, and there also exists a single-clause with a variable of the same card with a negation, a new clause can be created. This new clause includes the same variables of the original multi-clause, excluding the variable of which the negation was found in a single-clause.

   When a clause of a certain opponent is reduced to containing one card without negation, none of the other players can hold this card. Thus when this happens, both programs add a clause with the negation of that card to the clauses of all other players. When Program K&KaK adds such clause to a clause of a player of an opponent, he also adds the the negation of that clause to the clause of all other players of that opponent.

2. If there exists a certain clause which subsumes a multi-clause within a main clause, that multi-clause is deleted from the knowledge base.

Whenever a new clause is added to a main clause in the knowledge base, resolution is again applied to that clause.

With the use of this knowledge base, the first two different kind of reasoners can be made to decide which suggestions they make during the game:

1. Simple reasoning:
   The program uses the knowledge base to answer suggestions from other players. To make a suggestion, the program checks the database to see which cards are not hold by any of the players. The first suspect, weapon and room found which are not known to be hold by any player will be the suggestion he tries to make.

2. Dumb reasoning:
   The program makes the same suggestions as the Simple program, but does not make any accusations at all.

B: *Utility Reasoning*

In order to reason about which suggestion to make, the program has to calculate the average amount of clauses it will obtain into his knowledge base from each possible suggestion. The number of clauses obtained from a suggestion depends on the response from the opponents. When a suggestion is refuted by a certain player, the other players sitting clockwise next to that player do not get to respond to that suggestion. This means the suggestion can be refuted by only one player, or by none of the players. When there are s players in a game, there are $s-1$ opponents that can refute the suggestion, thus there are s ways a suggestion can be handled.

The amount of new clauses the program obtains from a response from an opponent depends on whether the opponent refutes or supports the suggestion. A player p, responding to a suggestion is noted as Response_p, thus p refuting a suggestion is noted as Refute_p, and p supporting a suggestion is noted as Support_p. Where p indicates the seat of the player with respect to the program, with 1 being the first player clockwise next to the program. To calculate the number of new clauses obtained from a response from a player p, the average amount of clauses obtained from the possible responses is calculated, noted as $\textit{Clauses}(\text{Response}_p)$. Some of the clauses obtained by a response might already be contained in the knowledge base of the program, and are thus no longer new clauses obtained from that response, the amount of such clauses is noted as $\textit{Containedclauses}(\text{Response}_p)$. To calculate the amount of new clauses obtained from a specific response, the program takes the maximum number of clauses obtained from that response and subtracts the number of contained clauses from that response.

The maximum number of clauses that are obtained from an opponent supporting a suggestion is three, one clause for each card which the player does not hold when he supports a suggestion. Since there is only one possible response when a player is supporting a suggestion, he can only say he has neither of the cards, this gives us the following formula:

$$\textit{Clauses}(\text{Support}_p) = 3 - \textit{Containedclauses}(\text{Support}_p)$$

The maximum number of clauses that are obtained from a single refute, in a game with s players, equals $s-1$, because one clause is added to the main clause of the opponent refuting the suggestion, and one clause is added to main clause of all other opponents. Since a suggestion involves three cards, and a player refutes a suggestion by showing one of the cards mentioned in that suggestion, there are three ways for an opponent to refute the suggestion. For each card mentioned in the suggestion, the maximum number of possible clauses from refuting equals $s-1$. Since the program only gets to see one of the cards, the average amount of clauses obtained from a player refuting a suggestion is calculated by dividing the total amount of clauses by 3. Therefore the amount of clauses obtained from an opponent p refuting a suggestion for the knowledge base of program K is calculated by the following formula:

$$\textit{Clauses}(\text{Refute}_p) = \frac{\sum_{Card} [(n-1) - \textit{Containedclauses}(\text{Refute}_p(Card)]}{3}$$

Since Program K&KaK also takes the knowledge of its opponents into account, it calculates the the amount of clauses obtained from a suggestion, compared to the amount of clauses its opponents gain from this suggestion. Program K&KaK calculates the amount of clauses

an opponent gains from a player supporting a suggestion in the same way as calculating the amount of clauses the program itself gains, but then applied to the databases of those opponents noted as $\textit{Containedclauses}_{opp}$, where opp is the seat number of an opponent with respect to the program. The program calculates, and adds, the amount of clauses gained for each opponent, excluding the opponent who supports the suggestion because he does not get any new information:

$$\textit{OpponentClauses}(\text{Support}_p) = \sum_{\substack{opp \\ opp \neq p}} 3 - \textit{Containedclauses}_{opp}(\text{Support}_p)$$

When a player refutes a suggestion made by the program, the program gets to see the card shown by that player, but the other opponents do not get to see this card. Therefore a different formula is used for calculating the amount of clauses obtained by the opponents when a player refutes a suggestion. The opponents only know that the player showed one of the three cards mentioned in the suggestion, therefore a multi-clause, with three cards, will be added to the database of those opponents. Since this clause has less information than a single-clause, a multi-clause will have a value of one divided by the amount of cards within this clause. When such clause is added to a database, there is a possibility of resolution onto this clause, which will make the clause more valuable. In order to apply resolution onto this clause, that database has to contain single-clauses with the negation of the cards mentioned by the suggestion noted by *Resolutionclauses*(Suggestion):

$$\textit{Value}(\text{Clause}_{\text{Suggestion}}) = \frac{1}{3 - \textit{Resolutionclauses}(\text{Clause}_{\text{Suggestion}})}$$

Therefore the maximum amount of clauses obtained by an opponent from a player refuting a suggestion, in a game with s players, still equals $s-1$. For opponents there is only one way for a player to refute (showing a card), because the opponent does not get to see which card is showed by the refuting player. This leads to the following formula for calculating the amount of clauses contained by the opponents:

$$\textit{OpponentClauses}(\text{Refute}_p) = \sum_{\substack{n=1 \\ n \neq p}}^{s-1} \sum_{m=1}^{s-1} \begin{cases} \textit{Value}_{n\|m}(\text{Clause}_{\text{Suggestion}}) & \text{if } m = p \\ \textit{Value}_{n\|m}(\neg\, \text{Clause}_{\text{Suggestion}}) & \text{if } m \neq p \end{cases}$$

With n being the opponents of the program, and m the opponents of those opponents, and $\textit{Value}_{n\|m}$ meaning the value of the clause obtained in the database of opponent n, in the main clause of player m.

To calculate the amount of clauses obtained from a player supporting or refuting a suggestion for Program K&KaK can now be calculated with the following formula:

$$\textit{Clauses}_{K\&KaK}(\text{Response}_p) = \frac{\textit{Clauses}(\text{Response}_p)}{\textit{OpponentClauses}(\text{Response}_p)}$$

To calculate the average amount of clauses obtained from a suggestion, the chances for each player to respond to a suggestion in a certain way have to be calculated. To calculate these chances, the chance for each opponent to support, as well as the chance for each opponent to refute the suggestion have to be calculated. An opponent will support a suggestion when he does not hold any of the cards mentioned in the suggestion. Therefore the chance for an opponent holding a card has to be calculated. If the knowledge base of the program contains a clause which holds the information that a certain opponent holds a card, the chance that this opponent holds that card equals 100%. If the knowledge base contains a clause which holds

the information that a certain opponent does not hold a card, the chance that this opponent holds that card equals 0%.

When such clauses do not exist, first the chance that the card is not in the middle of the board is calculated. Since both the three cards in the middle and the suggestion include a card of each category $(\textit{Weapon}, \textit{Suspect}, \textit{Room})$, the chance for a card, *Card*, of a certain category mentioned in the suggestion to be in the middle, equals one divided by the possible amount of cards of that category that can still be in the middle, noted as $\textit{Poss}(Cat(\textit{Card}))$:

$$\textit{Chance}(\text{Card}_{mid}) = \frac{1}{\textit{Poss}(Cat(\text{Card}))}$$

A card is no longer able to be in the middle if the knowledge base of the program contains a clause which contains information that a certain player holds that card. The chance for a card not to be in the middle, equals one minus the chance this card is in the middle:

$$\textit{Chance}(\text{Card}_{players}) = 1 - \textit{Chance}(\text{Card}_{mid})$$

When it is known that the card is not in the middle, it can still be in either players hands. The chance a certain player holds a card given the card is not in the middle, equals the amount of cards that player p holds of which the program does not know which card that is, noted as $Count_{unknown}(p)$, divided by the total amount of cards of which the program does not know which card that is:

$$\textit{Chance}(\text{Card}_p|\text{Card}_{players}) = \frac{Count_{unknown}(p)}{\sum\limits_{Player} Count_{unknown}(Player)}$$

The chance for a card to be a specific card in one of all players hands, equals the chance this card is not in the middle times the chance that card is in that players hand given it is not in the middle. Therefore the following formula is used to calculate the chance player p holds card "Card":

$$\textit{Chance}\left(\text{Card}_p\right) = \textit{Chance}(\text{Card}_p|\text{Card}_{players}) * \textit{Chance}(\text{Card}_{players})$$

The chance a player does not hold a card, equals one minus the chance a player does hold that card:

$$\textit{Chance}\left(\text{Card}_{\neg p}\right) = 1 - \textit{Chance}\left(\text{Card}_p\right)$$

Since a player only supports a suggestion when he does not hold any of the three cards mentioned in the suggestion the chance player p supports a suggestion is calculated with the following formula:

$$\textit{Chance}\left(\text{Support}_p\right) = \prod_{\text{SuggestionCard}} \textit{Chance}\left(\text{SuggestionCard}_{\neg p}\right)$$

The chance player p refutes a suggestion, equals one minus the chance that player supports the suggestion:

$$\textit{Chance}\left(\text{Refute}_p\right) = 1 - \textit{Chance}\left(\text{Support}_p\right)$$

The average amount of clauses obtained from a suggestion can now be calculated by making a summation over the multiplication of the chances for each possible way the suggestion can be handled, with the amount of clauses are averagely obtained in that way. In a game with s players there are $s-1$ ways in which the suggestion can be handled where an opponent refutes the suggestion. When a certain player refutes a suggestion, the players in front of that player must have supported the suggestion. Therefore the total amount of clauses obtained for each player to refute a suggestion is calculated with the following formula:

$$TotCl(\text{Refute}_p) = Clauses(\text{Refute}_p) + \sum_{n=1}^{p-1} Clauses(\text{Support}_n)$$

Since all players in front of the player refuting the suggestion have supported the suggestion, the total chance for a certain player to refute a suggestion is calculated with the following formula:

$$TotCh(\text{Refute}_p) = Chance\,(\text{Refute}_p) * \prod_{n=1}^{p-1} Chance(\text{Support}_n)$$

Now there is still one way left for the suggestion to be handled, where all players support the suggestion. The amount of clauses for this way is calculated with the following formula:

$$Clauses(\text{Support}_{ALL}) = \sum_{p=1}^{s-1} Clauses(\text{Support}_p)$$

The chance for all players to support the suggestion is calculated with the following formula:

$$Chance(\text{Support}_{ALL}) = \prod_{p=1}^{s-1} Chance(\text{Support}_p)$$

Now the average amount of clauses can be calculated by calculating the amount of clauses obtained for each way the suggestion can be handled and multiplying that with the chance for this to happen:

$$Averageclauses(\text{Suggestion}) = \sum_{p=1}^{s-1}(TotCl(\text{Refute}_p) * TotCh(\text{Refute}_p)) + Clauses(\text{Support}_{ALL}) * Chance(\text{Support}_{ALL})$$

Where s is the amount of players in the game.

Since a player has to move towards a certain room, which is included in the suggestion, in order to make that suggestion, the utility from a suggestion depends on the amount of turns it takes for the program to reach that room. Therefore for each suggestion, the distance to the room mentioned in the suggestion has to be calculated. The playing board of Cluedo includes multiple secret passages that lead from room to room, and players are also able to move trough rooms to create shortcuts. In order to calculate the distance from a player on the board to a certain room a connectivity graph is used, where the available number of adjacent nodes or rooms for each possible node or room on the board are connected (see Figure 1).

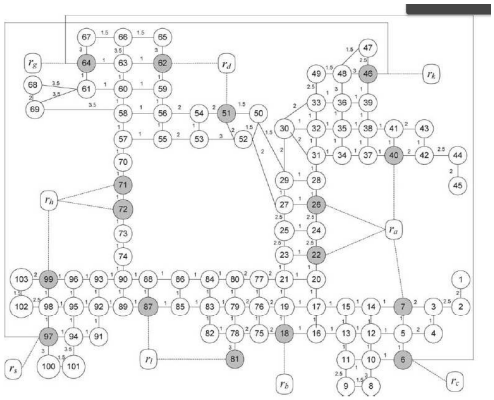

Figure 1: Connectivity graph with connections from nodes and rooms to adjacent nodes and rooms from the paper of Chenghui Cai and Silvia Ferrari [3] where a r_n node is a room, the other nodes are places where players can walk.

Before the program reasons about which suggestion to make, he throws the dice. After throwing the dice, there are two possibilities to calculate the amount of turns it takes for the program to make the suggestion:

1. If the distance to the room mentioned in the suggestion is shorter than the number rolled with the dice, the amount of turns to get to that room equals one.

   $$Turns(\text{Suggestion}) = 1$$

2. If the distance to the room mentioned in the suggestion is longer than the number rolled with the dice, the program has to calculate the amount of turns by taking the total distance, subtracting the current number rolled with the dice, and expecting an average 3.5 rolled from the dice for each following turn and adding one for the current turn.

$$\begin{aligned} &\mathit{Turns}(\text{Suggestion}) \\ &= 1 + \frac{\mathit{distance}(\text{Suggestion}) - \text{currentRoll}}{3.5} \end{aligned}$$

With the use of these formulas a way for Program K to reason about which suggestion to make can be developed. In order to make a move, the Program K and Program K&KaK have to decide which suggestion it wants to make. First the program will throw the dice to know the distance it can move. Then the program reasons about a suggestion, and move towards the room he needs to be in to make the suggestion. To calculate the suggestion which grants the most utility for winning the game Program K and K&KaK search through all possible suggestions they can make and find the suggestion which causes program K to obtain the most average number of clauses or Program K&KaK to obtain the most average number of clauses compared to his opponents, divided by the amount of turns it takes to make that suggestion. When a program has enough information about the cards held by his opponents to be certain about the cards that are in the middle, the program makes an accusation about these cards, causing him to win the game.

4 Game setup

The performance of the programs are measured with an interactive simulation of the game Cluedo in Eclipse [5]. Each simulation, two different type of players; human, Simple, Program K or Program K&KaK, are set up to compete against each other. When a game of Cluedo is played with only two players, there is not much reasoning involved, since the cards which are not held by the player himself can either be in the middle, or in the hand of his only opponent. To make the game more interesting and complex, a player must have multiple opponents. At the start of a game the players may not get an advantage over other players, and need to have equal chances of winning if the same strategy is used. To ensure this, at the end of every game the players switch seats, so that a different player starts the game. This way all players will be in the same position equally. A game of Cluedo includes 18 cards to be distributed among the players, which need to be distributed equally. Therefore the simulation of each game of Cluedo involves three players, to ensure multiple opponents as well as equally distributed cards. Since we only want to measure the performance of two programs against each other, the third player must have the least effect possible on those performances.

It is possible to make the third player not make any suggestions at all, and just sit on the board doing nothing but answering to other players suggestions. This way the two players are only playing against each other, with the increased complexity of multiple opponents. This seems like a perfect way to measure the performance of those two programs, but in practice shows great advantage to players sitting clockwise next to that player. The reason that player gets an advantage is because when a player refutes a suggestion, the other players do not get to react to this suggestion. For example in a game with three players, 1, 2 and 3, where player 2 is the player who makes no suggestions:

1. When player 1 makes a suggestion, it can either be refuted by player 2 or 3, or supported by both. When the suggestion is refuted by player 2, this will also give knowledge about the cards to player 3. When it is refuted by player 3 it means player 2 supported the suggestion, and player 3 will also obtain knowledge about the cards when player 2 supported the suggestion. Also when both players support the suggestion, player 3 will obtain knowledge about the cards for player 2 supporting the suggestion.

2. When player 3 makes a suggestion, it can either be refuted by player 1 or 2, or supported by both. When the suggestion is refuted by player 1, this will not give player 1 any knowledge about the cards, since he already knows what cards he holds himself. Only when player 1 supports the suggestion, so player 2 gets to respond to it, player 1 will obtain knowledge about the cards.

Therefore player 1 will get significantly less knowledge about the cards, and is less likely to win the game. This is shown in an experiment where player 1 is Program k, player 2 is the program that makes no suggestions and player 3 also is Program k. After each game the players 1 and 3 switch seats, to change the start position, which give the results shown in table 1. The columns of the tables declare the type of player, the seat of that player and the total amount of wins. Where p#1 means the player at seat 1, starting the game, with p#2 and p#3 respectively sitting clockwise next to him. The percentages below the seats are the games won at that seat, and the percentage below Total is the total win percentage of all games.

Table 1: Player 2 making no suggestions
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Program K	14%	0%	35%	49%
Program K	15%	0%	36%	51%
No Sugg	0%	0%	0%	0%

The player sitting clockwise next to player 2, who makes no suggestions, is more than double as likely to win the game. Since the chances of winning at each place is about equal with each player being the same type (see Table 2), the player of which the performance is not measured has to make suggestions to make sure the other players have equal chances to win. To make sure this player does not win the game, he is not allowed to make any accusations, so the measurement of performance is still only between two programs. This is done by making that player the Dumb program (Table 3). The results in table 3 show that each program now has equal chances of winning.

Table 2: Three players of the same type
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Program K	12%	11%	13%	36%
Program K	10%	12%	11%	33%
Program K	11%	10%	10%	31%

Table 3: Introducing the Dumb program
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Program K	24%	0%	25%	49%
Program K	25%	0%	26%	51%
Dumb	0%	0%	0%	0%

5 Results

As shown in Table 4, when the Simple program is put up against Program K, both programs win about equal amount of the games. The Simple program is able to beat the Program K because he guesses possible accusation suggestions about three cards that can all still be in the middle, then when all opponents support that suggestion, he knows that those cards have to be in the middle. This gives the program a chance to instantly win the game, by making the perfect suggestion. The reason the Simple program performs just as good as Program K could be that the suggestions made by Program K also give too much information to the Simple program. Then again Table 5 shows that Program K&KaK also performs just as good as the Simple program, while making suggestions that give the least amount of information to his opponents. Table 6 shows the performance of Program K put up against Program K&KaK, where they again win equal amount of the games.

Table 4: Simple vs Program K
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Simple	17%	16%	14%	47%
Program K	15%	20%	18%	53%
Dumb	0%	0%	0%	0%

Table 5: Simple vs Program K&KaK
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Simple	18%	15%	17%	50%
Program K&KaK	15%	17%	18%	50%
Dumb	0%	0%	0%	0%

Table 6: Program K vs Program K&KaK
10.000 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Program K	16%	16%	17%	49%
Program K&KaK	13%	20%	18%	51%
Dumb	0%	0%	0%	0%

The human players that are put up against the programs are given a Detective Notebook (see Figure 2), that is originally from the game Cluedo. In this notebook they can cross off suspects, weapons and rooms to eliminate them from the possible cards in the middle. When asked about how the human players reason about which suggestions they make, they said: ”Any card that can still be in the middle, in the most nearby room”. Table 7 shows the performance of the Simple program against some human players, Table 8 shows the performance of Program K against some human players, and Table 9 shows the performance of Program K&KaK against some human players. The human players knew the rules of the game and had experience in playing the board game.

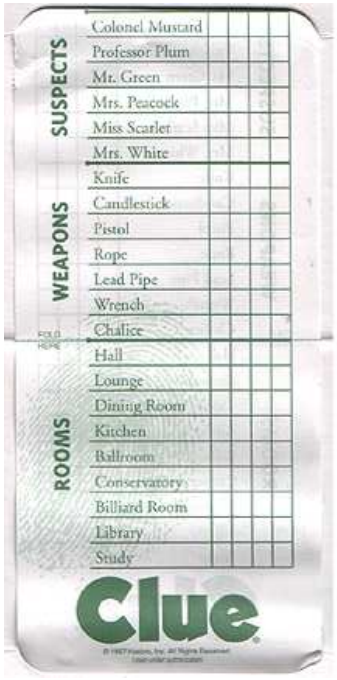

Figure 2: Original Clue Detective Notebook page used by the human players.

Table 7: Simple vs Human
25 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Simple	12%	16%	12%	40%
Human	16%	20%	24%	60%
Dumb	0%	0%	0%	0%

Table 8: Program K vs Human
25 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Human	12%	12%	8%	32%
Program K	20%	20%	28%	68%
Dumb	0%	0%	0%	0%

Table 9: Program K&KaK vs Human
25 games played, showing the win percentages

Player	p#1	p#2	p#3	Total
Human	8%	12%	8%	28%
Program K&KaK	20%	28%	24%	72%
Dumb	0%	0%	0%	0%

When the programs are put up against human players, different results occur compared to when the programs are put up against each other. The programs all seem to do very well against human players, having a significant majority of wins.

6 Conclusion

In the results we see that when the programs are put up against each other, the simple program performs just as good as the other two programs, which have improved reasoning about making a suggestion which grants them the most utility. This means that in the game Cluedo, trying to make suggestions that grant you the most information, or the most information compared to your opponents, is not the best strategy for winning the game. The reason for the Simple program to perform so good against these two reasoning programs, is that the Simple program makes a lot of suggestions about cards, which are not held by any of the players, making him instantly win the game. The other programs also make suggestions about cards of which they already have knowledge about, if the suggestion involving that card grants more clauses per turn. The simple program gets the same chance for making perfect suggestions as the other programs, because he still gets a lot of information from the suggestions made by the other programs. This means that trying to get the most information from suggestions, is not working out as intended. The reason for this is that the calculation for these suggestions is based on chance. And even tough the program might get lucky and find the most information he wants from a suggestion, this does not make him win the game, but another program making that lucky perfect suggestion about the cards in the middle, will win that program the game. The same goes for the program trying to obtain the most information compared to his opponents. From this we can conclude that getting the most information from a suggestion does not improve your chances of winning in the game Cluedo, and guessing the possible cards that can be in the middle is a sufficient strategy for winning the game.

Then again, when we put up the programs against human players, the programs are outperforming the humans. The reasoning about a suggestion from the human players is about the same as the reasoning of the Simple program. Since we already concluded that the reasoning about the suggestions from the other programs do not improve the chances of wining the game, the reason the programs beat the human play-

ers is not found in the reasoning about making the suggestions. This means the difference in performance is found in the way the players maintain their knowledge base. The human players used the standard Clue Detective Notebook, which only gives them the ability to cross off the cards, which they know are held by certain players, of which they are sure they are not in the middle. The programs, on the other hand, also maintain information about which cards are not held by certain players, and the cards that are possibly in certain players hands(*multi-clauses*). Therefore the programs gain their knowledge about which cards are possibly in the middle increases significantly faster than the knowledge about these cards of the human players. This leads to the programs having a higher chance of making the perfect suggestion about the cards in the middle, and thus having a higher win-percentage. Since Program K&KaK does not have a better performance than Program K, the improvement of the knowledge base where the knowledge of opponents is also stored does not increase the performance.

From this we can conclude that for the game Cluedo, increasing the utility for winning the game by increasing the amount of knowledge gain from suggestions is not an effective win strategy. We can also conclude that improving your knowledge base, in such way that more information about the cards held by the players can be stored and maintained, does improve your chances for winning the game. Therefore all that is needed for a program to compete with human players, is a simple reasoner for making suggestions, where the cards mentioned in the suggestions are cards which can still be in the middle, and a knowledge base to store and maintain knowledge about the cards held by the opponents.

We can also conclude that the standard Cluedo Detective Notebook has insufficient possibilities to store the information which is obtained from answers to suggestions. To make the notebook good enough to be able to handle this information more columns have to be added, where each column represents the cards held by the players. In these columns players can then set crosses to note that a player does not hold that card, or check marks when they know that players does hold that card. To handle the multi-clauses as described in the knowledge base section above, players can set characters in the columns. For example when a player sees that an opponent shows a card to another opponent, he can set an a at each of the cards mentioned in the suggestion in the column of the opponent who shows the card. Then the player can cross off the a whenever the knowledge that the player does not hold that card is obtained, until only one a remains. The player then knows the opponent holds that card, and must have shown that card. An example of such notebook is shown in figure 3. A similar notebook is also described by H.P. van Ditmarsch [6].

Figure 3: Improved Clue Detective Notebook page to allow equal chances for human players against the programs.

References

[1] Rachel Ben-Eliyahu and Rina Dechter. Default logic, propositional logic and constraints. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 1*, AAAI'91, pages 379–385. AAAI Press, 1991.

[2] Clare Dixon. Specifying and verifying the game cluedo using temporal logics of knowledge. 2004. Source: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.7191.

[3] Silvia Ferrari and Chenghui Cai. On the development of an intelligent computer player for clue: A case study on preposterior decision analysis. In *Proc. Amer. Control Conf.*, pages 4350–4355. IEEE, 2006.

[4] Silvia Ferrari and Chenghui Cai. Information-driven search strategies in the board game of clue®. *Trans. Sys. Man Cyber. Part B*, 39(3):607–625, June 2009.

[5] Eclipse SDK Version: 3.7.2 Build id: M20120208-0800. [online]. available: http://www.eclipse.org.

[6] H.P. van Ditmarsch. Killing cluedo. *Natuur en Techniek*, 69(11):32–40, 2001.

[7] H.P. van Ditmarsch. The description of game actions in cluedo. *Game theory and Applications*, 8:1–28, 2002. L.A. Petrosian, V.V. Mazalov (eds.).