UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DANIEL DOS SANTOS BOSSLE

Finding Optimal Strategies for Group Deception Games

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Marcus Ritt

Porto Alegre
January 2018

ABSTRACT

We present a solution method for deception-based games, which are zero-sum games without perfect information, and apply it to the popular party game The Resistance. The methods presented include adaptations of the techniques from (KOLLER; MEGIDDO; von Stengel, 1994), together with novel improvements such as symmetry reductions. These methods allow us to solve the game with up to 8 players, which would require the analysis of $8.5 * 10^{11}$ game tree nodes, if without symmetry reductions, and even more without knowledge trees.

Keywords: Linear programming. CPLEX. Nash equilibrium. zero-sum games. imperfect information games.

Encontrando Estratégias Ótimas para Jogos de Estratagemas em Grupo

RESUMO

Apresentamos um método de solução para jogos de estratagemas — jogos de soma zero sem informação perfeita — e o aplicamos ao popular jogo The Resistance. Os métodos apresentados incluem adaptações das técnicas de (KOLLER; MEGIDDO; von Stengel, 1994), além de novas melhorias como, por exemplo, reduções de simetrias. Esses métodos permitem soluções para até 8 jogadores, o que necessitaria a análise de $8.5 * 10^{11}$ nodos da árvore do jogo caso não utilizássemos reduções de simetria, e um número maior ainda caso não utilizássemos árvores de conhecimento.

Palavras-chave: programação linear, CPLEX, equilíbrio de Nash, jogos de soma zero, jogos de informação imperfeita.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CRM	Counterfactual Regret Minimization
LCP	Linear Complementarity Problem

CONTENTS

1 INTRODUCTION

The life of any person is riddled with decision-making opportunities, which can have great impact to all involved. Decision-making is also of great importance to any business, requiring plenty of employees, most often called managers, who are tasked solely with performing it. Many have profited from the rapid technology advances in the latest years by using these new technologies to help improving their decisions. These advances have also brought a rise in Artificial Intelligience, which brings with it many unanswered questions about decision-making, such as on how to evaluate it mathematically.

In order to provide those decision improvements, and to solve those questions, we can turn to Game Theory. Games can be viewed as mathematical models for decision-making interactions. They range from very simple, such as the children's game "Rock-Paper-Scissors", to real-life examples, such as auctions, and to complex games, such as Go, Chess and Poker.

Artificial Intelligence has advanced to the point of being much better than any human player on games as Chess and Go, and has recently beaten some top Poker players as well. However, it still has many questions left unanswered, such as on how to adapt solutions like AlphaGo, which masters a wide variety of games, such as Chess and Go, to games where deception is involved, like Poker, where the best AIs are based on specifically restricted models of the game.

This work is an analysis and implementation of the state-of-the-art of methods for solving games played by two agents, where there is no cooperation possible, and where deception is involved. These restrictions, other than the possibility of deception, are particularly useful for the definition of an “optimal strategy”, as common shortcomings of the classic notion of Nash Equilibrium don't manifest in these games.

We apply known combinatorial optimization techniques together with novel improvements in order to solve the game The Resistance, a popular party game based on deception and asymmetry of information.

2 DEFINITIONS FROM GAME THEORY

In this chapter we will present our definitions of games and strategies, which are the same as in Game Theory literature, such as in (KOLLER; MEGIDDO, 1992).

Games and their Players

Game theory studies mathematical models of games. In these models we have agents, called players, who may perform actions, from a modeled set, that affect the state of the game. After a series of rounds of actions, the game may end, and results in a set of payoffs for the players, where the payoff for each player is modeled as a numerical value. Note that this makes the payoff a one-dimensional value, which means that situations that would incur in multiple kinds of value, such as a cost in labor and a benefit in money, require considerations on how to measure the final payoff in a single measurement unit.

As an example of games that can be studied in this manner, we have the popular game of chess. For chess, we have 2 players who alternate turns, and on their turn they choose an action to perform from the set of valid piece movements. The payoff for each player can then be modeled as 1 in case of a win, 0 in case of a tie, and -1 in case of a loss.

One important aspect of chess is that the sum of all its payoffs is always 0: if one player wins, we have payoffs of 1 and -1, and, in case of a tie, both payoffs are 0. This property, together with the fact that there are only 2 players, shows that the game is competitive, that is, does not involve cooperation, since, in order to obtain a higher payoff, one player must only aim to lower the other's payoff by the same amount. Games with this property are called zero-sum games.

Another important aspect of chess is that, at any point, both players have knowledge of the game's entire state. Games with this property are called perfect information games. That is not the case in other games, such as poker, where one player does not know which cards the other is holding. When players may have private information about the state of the game, the game is called an imperfect information game.

Strategies

When modeling a game, it is assumed that players act according to a strategy. Being more mathematically precise, a strategy defines, given a player's knowledge about the game state, which actions the player may take, and with what probability the player will take each of those actions. For example, in the game of Rock-Paper-Scissors, a strategy may be to "always play Rock", or "play each choice with probability 33%". Strategies with only one possible action per state are called pure strategies, while ones where the players use random chance to choose their actions are called mixed strategies.

Nash Equilibria

In order to provide optimal strategies for a game, we can study a very important concept from game theory, the Nash Equilibrium (NASH, 1951).

Definition 1. *Let $P_i(S_1, S_2, \ldots)$ be the payoff for player i if players use strategies from the set $S_1, S_2, \ldots$, that is, player j uses strategy S_j. This set of strategies is a Nash Equilibrium if $\forall i, \forall S'_i, P_i(S_1, S_2, \ldots, S_{i-1}, S'_i, S_{i+1}, \ldots) \leq P_i(S_1, S_2, \ldots)$*

This means that a Nash Equilibrium is a set of strategies, one for each player, such that no player can improve his own payoff by changing just his own strategy. This is an indicator of optimality, in a way, since it means that each player is playing their best, assuming the other players' strategies are known. This can also be seen as a saddle point in the payoff function for the first player, since the payoff cannot increase when moving along the possible strategies for the first player, and it cannot decrease along the second player's strategies. Nash demonstrated that, for any game with a finite amount of pure strategies, there is at least one Nash Equilibrium, though it may be composed of mixed strategies.

One basic example is the game of Rock-Paper-Scissors. In this game, both players choose simultaneously between three actions, called Rock, Paper, and Scissors. If both players choose the same action, the game is a tie. Otherwise, if one player chose Rock and the second chose Paper, the player that chose Paper wins; similarly, a player choosing Scissors wins against a player choosing Paper; and Rock wins against Scissors. We then model payoffs, again, as being 0 for ties, 1 for a winning player, and -1 for a losing player.

There is no optimal pure strategy for this game, since, for any of the three that exist, there is another that beats it, and, since the game is symmetric, the optimal payoff should be 0, not -1. This means that any Nash equilibrium must consist in a pair of mixed strategies. In fact, we have only one: where both players play each action with $\frac{1}{3}$ probability each. You can note that this strategy means that, no matter what the opponent does, there is $\frac{1}{3}$ chance of a tie, $\frac{1}{3}$ chance of a win and $\frac{1}{3}$ chance of a loss, for an average payoff of 0.

Note that optimal strategies are not always this obvious. For instance, if the game was adapted so that a win of paper against rock had payoffs of $\frac{1}{2}$ and $-\frac{1}{2}$ rather than 1 and -1, the Nash Equilibrium would shift to playing paper with 40% chance, rock with 40% chance, and scissors with 20%. Note that this strategy is a Nash Equilibrium since paper wins with 40% chance, ties with 40%, and loses with 20%, for an average payoff of $40\% * \frac{1}{2} + 40\% * 0 + 20\% * -1 = 0$, while rock has an average payoff of $40\% * -\frac{1}{2} + 40\% * 0 + 20\% * 1 = 0$, and scissors $40\% * 1 + 40\% * -1 + 20\% * 0 = 0$.

Mixed strategies also have a payoff of 0, due to the linearity of expectation: if and opponent plays rock, paper and scissors with probabilities r, p and s, his expected payoff is $r * P(\mathit{Rock}, S) + p * P(\mathit{Paper}, S) + s * P(\mathit{Scissors}, S) = r * 0 + p * 0 + s * 0 = 0$. In fact, this analysis applies to any game, meaning that we can find the best response to a strategy by searching only among pure strategies. This will be formalized in Theorem 2.

Representations of Games

The usual representations for single-round games, such as Rock-Paper-Scissors, are with a payoff matrix, where rows represent the pure strategies for one player, and columns represent the ones for the other player, as exemplified in Table 2.1 and Table 2.2.

Table 2.1: Normal form for Rock-Paper-Scissors

Player 2 \Player 1	*Rock*	*Paper*	*Scissors*
Rock	0	1	-1
Paper	-1	0	1
Scissors	1	-1	0

Table 2.2: Normal form for modified Rock-Paper-Scissors

Player 2 \Player 1	*Rock*	*Paper*	*Scissors*
Rock	0	0.5	-1
Paper	-0.5	0	1
Scissors	1	-1	0

Figure 2.1: Extensive form for Rock-Paper-Scissors

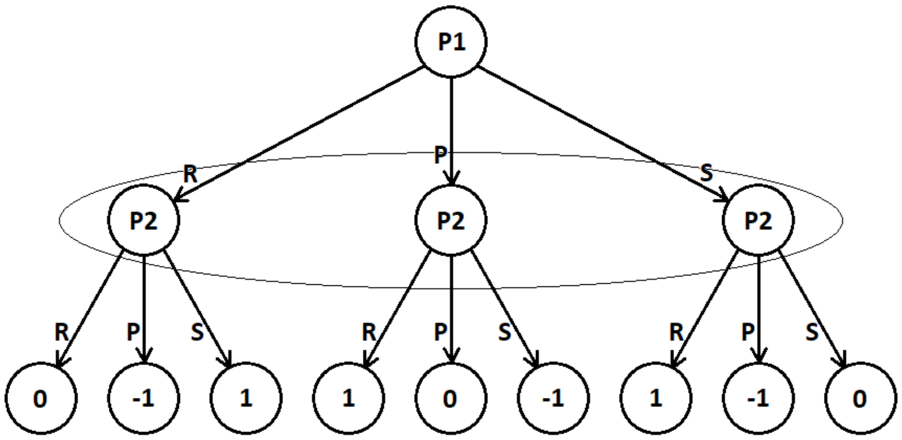
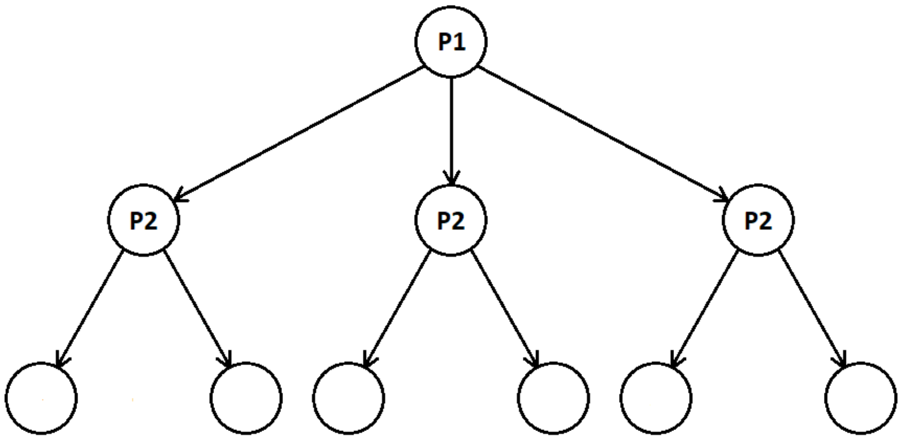


Figure 2.2: Example of extensive form more compact than its normal form



This representation is commonly known as the normal form (NEUMANN; MORGENSTERN, 2007). For multi-round games, such as chess or poker and variations, a much more compact representation is the extensive form, as exemplified in Figure 2.1.

The extensive form can be modeled as a tree with labeled edges, with edge labels corresponding to actions, and where each node is also labeled with the player that should choose at that moment. Furthermore, nodes are grouped into partitions that represent the player's knowledge: two nodes are in the same group if the player cannot distinguish between the two situations they represent. In particular, all nodes in the same group have the same player label and the same action labels on their children edges. There can also be an additional kind of label in nodes, representing random events. For example, in a card game, the dealing of a card can be modeled as a node where each child represents one of the possible cards dealt. The edges that are children of a random event node can be labeled directly with the probability of that edge actually being traversed.

Extensive form games may be converted to normal form, with each pure strategy being a set of choices that say, given each possible game state, which action the player should pick. Note that this may be exponential on the size of the tree, as exemplified on Figure 2.2: for that game, the second player has 8 pure strategies, even though the tree only has 6 leaves. A pure strategy there consists in choosing which of 2 actions to take for each of 3 possible states. This simple example can be trivially extended to contain 2^n pure strategies with only $2n$ leaves.

3 TECHNIQUES FOR FINDING GAME SOLUTIONS

In this chapter we present what are optimal strategies for two-player zero-sum games, through the use of theorems commonly known in Game Theory, and we present state-of-the-art techniques to find these strategies.

Solutions to Games

In this work we limit our study to two-player zero-sum games, since in these games there is an unambiguous definition of “optimal strategy”:

Definition 2. *$P(S1, S2)$ is the payoff for the first player if he plays with strategy $S1$ and the second player plays with strategy $S2$. In this case, the payoff for the second player will be $-P(S1, S2)$, since we are considering only zero-sum games.*

Theorem 1. *All Nash Equilibria result in the same payoff.*

Proof. If we have two equilibria $(S1, S2)$ with payoff p and $(S1', S2')$ with payoff p', from the definition of Nash Equilibrium we know that

$$P(S1, S2') \geq P(S1, S2) = p$$

and that

$$-P(S1, S2') \geq -P(S1', S2') = -p'$$

thus $p \leq p'$. Similarly,

$$P(S1', S2) \leq P(S1, S2) = p$$

$$-P(S1', S2) \leq -P(S1', S2') = -p'$$

$$p \geq p'$$

Thus $p = p'$ □

With this we may call any strategy that belongs to a Nash Equilibrium as “optimal”, since it gives the player an optimal payoff if the opponent has an optimal strategy too, and this payoff may only increase if the opponent uses another strategy. It is possible for the player to obtain a better payoff, but only if it is known that the opponent is not using an optimal strategy. In this work, we assume that is not the case, since, while there are biases in human play, even for simple games such as Rock-Paper-Scissors, as explored by (WANG; XU; ZHOU, 2014), they can be somewhat corrected by aware players, and fully corrected by the use of external tools, such as randomness generators like dice or mobile apps, and can be assumed to not exist in non-human players, like artificial intelligences.

Useful properties of optimal strategies include that they work against unpredictable opponents, and that they work against opponents that can predict your strategy.

Theorem 2. *A strategy is optimal if it achieves at least the optimal payoff against each pure strategy.*

Proof. Since the payoff function against a fixed strategy is a linear function on the probabilities of each pure strategy, the payoff for a mixed strategy is the weighted average of the payoffs of the pure strategies it uses. Thus, it cannot surpass the payoff of the best pure strategy it uses. □

Thus we may, when validating the optimality of a solution, only check its outcome against pure strategies, without having to consider mixed ones.

Definition 3. *Given a strategy S_1, we call Best Response any strategy $S_2 = \arg\max P_2(S_1, S_2) = \arg\min P_1(S_1, S_2)$.*

In particular, from the previous theorem we know that there is always a pure best response to any strategy. Also, from the definition, all strategies in a Nash Equilibrium are best responses to each other.

Theorem 3. *The Nash Equilibria define a continuous, convex surface in the strategy space.*

Proof. If we have two optimal strategies, $S1$ and $S1'$, by playing the first with some probability p and the second with probability $1-p$ we arrive at another Nash Equilibrium, since the payoff against each pure strategy is at least the optimal payoff. □

We do not consider multi-player and non-zero-sum games, since any definition of optimality for them is very ambiguous and controversial. For instance, one such game is the famous Prisoner's Dilemma, a class of non-zero-sum games, with the payoff matrix of one instance represented in Table 3.1. In it, each player may choose to benefit all players at a small cost to himself, or may choose instead to hurt the other players with a small gain to himself. The Nash Equilibrium for this game is for all players to defect, however everyone would have a better payoff if instead everyone cooperated, which shows a flaw in the concept of Nash Equilibrium. In this case, the choice is between selfishness and altruism, which is not a mathematical choice, but rather a philosophical one, so we keep these games out of the scope of this work.

Table 3.1: Payoff matrix for an instance of the Prisoner's Dilemma

	Cooperate	*Defect*
Cooperate	1,1	2,-1
Defect	-1,2	0,0

Solving Perfect Information Games

There are many methods to find optimal strategies, each better suited to different kinds of games.

For games with perfect information, that is, games in extensive form, and where every knowledge partition contains a single node, there is always at least one optimal and pure strategy, which can be found through backward induction (FUDENBERG; TIROLE, 1991).

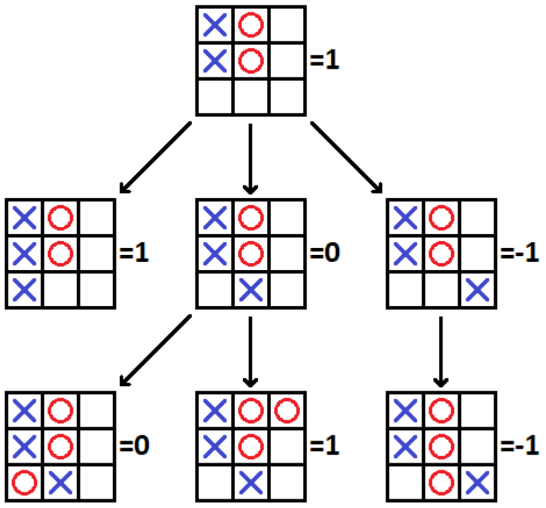
Definition 4. *The Subgame Payoff for a game state is the payoff function for the game that starts at that state.*

For a perfect information game, the optimal subgame payoff for a state is the expected payoff if both players play optimally starting from that state. This value can be found recursively, with backward induction:

- On a leaf, this value is the payoff of the game, which is fixed;
- For any node that corresponds to a first player choice, the value is the maximum value among its children;
- For a second player choice, the value is the minimum among its children states;
- And for a random choice, the value is the expected value of its children, meaning the weighted average, with weights equal to the probability that the corresponding edge will be taken.

With this, we can see that the optimal strategy for perfect information games is to always follow the action that maximizes the subgame payoff, or that minimizes it if you are the second player, since by this recursive definition above this set of strategies is a Nash Equilibrium.

Figure 3.1: Part of the game tree for Tic-Tac-Toe, with subgame payoffs noted



In Figure 3.1 we have a small portion of the result of applying this algorithm to the popular game of Tic-Tac-Toe. The game consists in two players, one placing X's on the board, and the other placing O's, alternately, and without the ability to overwrite previous placements. The winner is who manages to place three of their symbol in a straight line first, including a diagonal.

On games where the outcome is either a win, a tie or a loss, and with no randomness involved, such as chess, go, or tic-tac-toe, this subgame payoff is always 1, 0 or -1, representing respectively that, from that state, the first player can force a win, that optimal play will lead to a tie, and that the second player can force a win.

Approximation Techniques for Larger Games

That method can find the optimal strategy on any perfect-information game, as long as the game tree is not too large. For instance, checkers (SCHAEFFER et al., 2007). However, many games such as Chess or Go have trees that are too large to be computed. In those cases, it is not possible to explicitly compute a complete, perfect strategy. However, there have been increasing advances in obtaining approximate strategies for these games. More specifically, usually a heuristic is used to approximate the subgame payoff, and then this approximation is improved through techniques such as Alpha-Beta Pruning (KNUTH; MOORE, 1975) and Monte Carlo Tree Search (TESAURO; GALPERIN, 1997). These techniques perform the propagation of these approximate subgame payoffs, but only in part of the game tree, such as only for the immediate children of the current node, since the whole tree would be too big to be explored normally.

While chess has very optimized engines such as StockFish (ROMSTAD et al., 2017), for the game of Go, Google has developed AlphaGo (SILVER et al., 2016), a neural-network based AI trained via reinforcement learning, with the goal of being more general purpose, that is, being able to solve perfect information games in general. AlphaGo has recently been adjusted into AlphaZero (SILVER et al., 2017), which can play other games, such as Chess, and has quickly surpassed by far previous champion Chess-specific engines.

Issues to Adapt to Imperfect Information Games

On most games without perfect information, there is no pure optimal strategy, and the notion of subgame payoff changes drastically, not being able to be approximated properly. For instance, in Rock-Paper-Scissors, the optimal subgame payoff after the first player plays is always -1, since the second player can counter the move. However, the optimal payoff for the game is 0. This apparent contradiction is due to the fact that separate subtrees affect each other's optimal strategies.

Approximation Techniques for Imperfect Information Games

Even though that may be the case, there is an approach based on machine learning that computes approximate solutions to games without perfect information: the Counterfactual Regret Minimization (CRM) method (ZINKEVICH et al., 2008). It computes a function that, given the current game state and player knowledge, returns the probability of choosing each action. The core of the algorithm is how it improves a strategy with reinforcement learning (SUTTON; BARTO, 1998). After having it play against an opponent, who is also being trained with this strategy, a regret value is calculated for each action that was not taken. To calculate the value, the game is replayed with the only difference being that the action is taken instead of not taken. The regret is, then, the difference of the payoffs of these two games, or 0 if taking the action lowered the payoff. The strategy then is shifted towards picking actions with higher regret value, with a shift proportional to the regret value.

Fixpoints for this strategy shifting algorithm are Nash Equilibria, since they correspond to strategies where all regrets are 0, meaning no action could be taken that would improve the player's payoff. However, since regrets are computed when playing one pure strategy at a time, not all possible paths of a mixed strategy, in most games these fix-points don't exist, since they would mean we have an optimal pure strategy. Rather, the algorithm usually converges to a strategy where the shifts cancel out, on average. This can give an intuition that CRM is not very efficient, since the computed function never properly converges. Indeed, only recently has this method been optimized enough to be competitive against human players in a limited variant of poker (BOWLING et al., 2015).

The main reason why these approximative methods don't work as well for imperfect information games is the fact that the exact solution methods for imperfect information games are very different from the methods for solving perfect information games. While the latter use essentially a dynamic programming method, the former have to take into account that most subgames may interfere with each other's solutions, and use linear programming models instead.

Solving Imperfect Information Games with Linear Programming

The basic method for finding an exact solution is taking the game model in normal form, and then translating Theorem 2 into linear programming. Essentially, the formulation tries to maximize the final payoff, and restricts this payoff to be the minimum payoff against any pure strategy.

Let Ps_i be the set of pure strategies for player i. Then the formulation becomes

$$\max z$$

subject to:

$$\forall p_2 \in Ps_2, \sum_{p_1 \in Ps_1} S[p_1] P(p_1, p_2) \geq z$$

$$\sum_{p1} S[p_1] = 1$$

$$S[p_1] \in \mathbb{R}^+, \forall p_1 \in Ps_1$$

This formulations finds z, the optimal payoff, and S, the strategy for the first player.

While this provides a very good solution for normal form games, it is not as efficient for extensive form games, since, as already observed, the normal form formulation usually has size exponential on the size of the game tree. However, (KOLLER; MEGIDDO, 1996) demonstrates the existence of some solution with linear support, and in fact there are methods for finding more compact solutions.

The Lemke-Howson Algorithm

The basic algorithm for studying two player games that may not be zero-sum is the Lemke-Howson algorithm, which can be adapted to extensive form zero-sum games, as demonstrated in (KOLLER; MEGIDDO; von Stengel, 1996). It works similarly to the Simplex algorithm, pivoting pure strategies that are best reponses to the opponent's current strategy into the current solution, while pivoting out strategies that are not best reponses anymore.

The model used in that work involves modeling a mixed strategy not by assigning probabilities not to pure strategies, but rather by assigning probabilities p to the leaves of the game tree. Then, the probability p of an internal node is defined by the probabilities of its children, as will be described later in this section. The strategy represented by these probabilities means that, when reaching some node n, each possible action corresponds to a child node c, and the probability that the strategy picks this action is $p(c)/p(n)$.

In order to apply this formulation, the game tree is converted into two trees, one for each player, corresponding to their knowledge. Then the strategy of the first player affects only the probabilities in his tree, the strategy for the second player affects only the second tree probabilities, and the probabilities of random events only affect the payoffs on the leaves.

Definition 5. *The knowledge tree for a player is a tree where each node belongs to either the set of action nodes, A, the set of information nodes, I, or the set of leaf nodes, L. This tree is defined by having a bijection between its nodes and the original game's knowledge states for the player. This means that A represents the states where the player may take an action, with a bijection between children and possible actions. I represents states where it is not the player's turn, with a bijection between children and possible information that the player may receive at that state. And L is the set of states when the game is over. We represent the children of a node n as the set $C(n)$.*

Each node in this tree corresponds, thus, to a knowledge partition in the original game tree. This formulation uses the fact that players have knowledge of their actions and of the history of their perception. The study of games where players don't have perfect recall, meaning that they may end up at the same knowledge state from more than one knowledge path, that is, meaning that the knowledge does not follow a tree, is out of the scope of this work, specially since it is proven NP-complete by (KOLLER; MEGIDDO, 1992).

The policy for having random event probabilities only affect leaves, as mentioned earlier, simplifies the modeling, since then there is no need to consider how these events interact between the two knowledge trees. It makes it so that the probability of a node is the sum of the probabilities of its children, if the node is in A, or the probability is equal to each of its children's probabilities, if the node is in I. Without the policy for random events, these values could be calculated in much more complex ways, due to concerns with conditional probabilities. By only affecting the leaves, we can model the final payoff for each pair of leaves, one leaf from each tree, and make it not just the original payoff for the corresponding end state, but that payoff multiplied by the product of the probabilities that random events led to that state. Note that this payoff is thus not being affected by either player's strategies, and is 0 if those leaves cannot be reached simultaneously, as that would mean some random event probability would be 0. We can summarize these rules in the following formulation:

$$\max_{S_1} \min_{S_2} \sum_{l_1 \in L_1, l_2 \in L_2} S_1[l_1] P(l_1, l_2) S_2[l_2]$$

subject to:

$$\forall a_1 \in A_1, S_1[a_1] = \sum_{c_1 \in C(a_1)} S_1[c_1]$$

$$\forall i_1 \in I_1, \forall c_1 \in C(i_1), S_1[i_1] = S_1[c_1]$$

$$S_1[Root_1] = 1$$

$$\forall a_2 \in A_2, S_2[a_2] = \sum_{c_2 \in C(a_2)} S_2[c_2]$$

$$\forall i_2 \in I_2, \forall c_2 \in C(i_2), S_2[i_2] = S_2[c_2]$$

$$S_2[Root_2] = 1$$

$$S_1[p_1] \in \mathbb{R}^+, \forall p_1 \in A_1 \cup I_1 \cup L_1$$

$$S_2[p_2] \in \mathbb{R}^+, \forall p_2 \in A_2 \cup I_2 \cup L_2$$

As an example, consider the following poker simplification: both players privately receive a random card, saying 0 or 1 each, then the first player decides to bet 1 extra point or not, and if he does, the second player decides to call the bet or forfeit. If there is no forfeit, the bets go to the player with the higher card, and are split in case of a tie. The game trees from the point of view of the players are represented in Figure 3.2, and so the LCP formulation becomes:

$$\begin{aligned}\max_{S_1}\min_{S_2} & -0.5S_1[0P]+0.5S_1[1P]+0.25(S_1[0BF]+S_1[1BF])(S_2[0F]+S_2[1F] \\ & -0.5S_1[0BC]S_2[1C]+0.5S_1[1BC]S_2[0C]\end{aligned}$$

subject to:

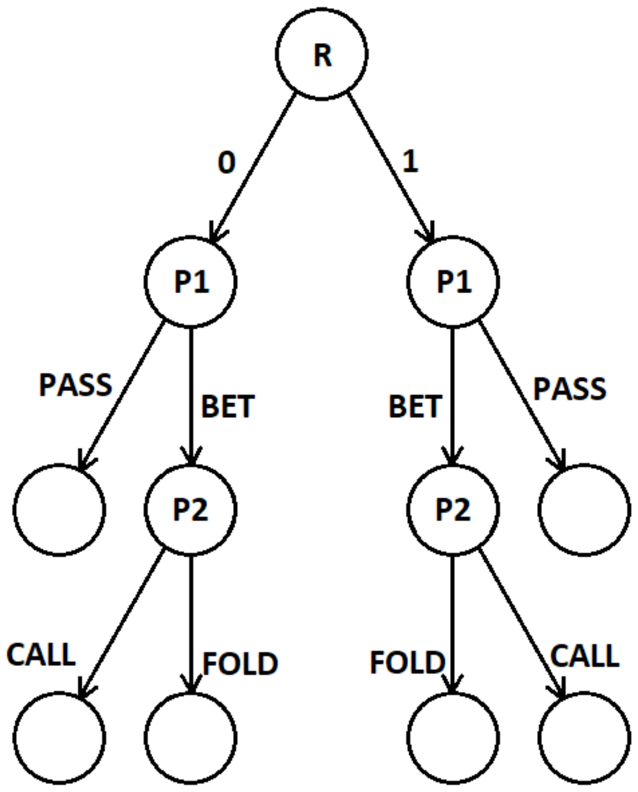
$$\begin{aligned}S_1[root1] =& S_1[0]\\ S_1[root1] =& S_1[1]\\ S_1[0] =& S_1[0P]+S_1[0B]\\ S_1[0B] =& S_1[0BC]\\ S_1[0B] =& S_1[0BF]\\ S_1[1] =& S_1[1P]+S_1[1B]\\ S_1[1B] =& S_1[1BC]\\ S_1[1B] =& S_1[1BF]\\ S_1[root1] =& 1\end{aligned}$$

$$\begin{aligned}S_2[root2] =& S_2[0]\\ S_2[root2] =& S_2[1]\\ S_2[0] =& S_2[0C]+S_2[0F]\\ S_2[1] =& S_2[1C]+S_2[1F]\\ S_2[root2] =& 1\end{aligned}$$

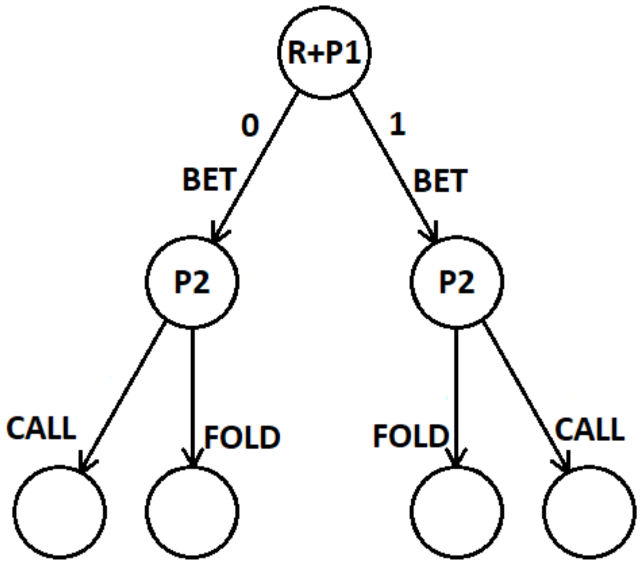
$$S_1[p_1]\in\mathbb{R}^+, \forall p_1\in A_1\cup I_1\cup L_1$$
$$S_2[p_2]\in\mathbb{R}^+, \forall p_2\in A_2\cup I_2\cup L_2$$

In this formulation, we labeled every node with the initials of the edge labels in the path from the root to the node. While the optimal strategy is quite trivial, being to always bet or call if you have a good card and to never bet nor call if you have a bad card,

Figure 3.2: The tree for the poker-like game described, split into the points of view for both players



(a) The tree for the first player



(b) The tree for the second player

the analysis required to demonstrate that fact is not so straightforward, but is automatic when using the LCP formulation above.

Adapting to a Linear Programming Model

Lemke-Howson, as the simplex method, is not suitable for very degenerate systems, and is also quite complex to be optimized when dealing with larger games. (KOLLER; MEGIDDO; von Stengel, 1994) has solved these issues by adapting the linear complementarity model into a linear programming model, thus enabling optimizations achieved in linear programming that are not yet available for linear complementarity, since the latter is not nearly as studied as the former.

The adaptation found involves the use of the dual program for the restrictions of the second player. By fixing the first player's strategy, we have that the second player's strategy should satisfy a normal minimization linear program. By taking the dual of that program, it becomes a maximization program, which stays linear even after making the first player's strategy into variables again. That is, this modified system contains variables for the first player's strategy and for the dual of the second player's, and is a purely maximization linear program, meaning it can be solved with simplex or other methods for normal linear programs. We provide the full derivation and a more straightforward formulation of this method in Section 5.2.

4 A GROUP DECEPTION GAME

In this work, we solved the party game The Resistance, since it is a somewhat popular game, is based on knowledge economics, and is not too complex.

Rules

The original game is played with 5 to 10 players, and consists of 5 consecutive rounds. At the start of the game, one third of the players become spies, and become aware of who the spies are, while the other players, the resistance, lack that knowledge. Then, in each round, some amount of players are chosen by the group, through a democratic process. Then these chosen players go in a "mission", and can privately decide to help or sabotage the mission. If enough sabotages happens, the spies gain one point, otherwise the resistance gains one point. The first team to reach 3 points wins.

Table 4.1: Mission sizes for The Resistance

Players	*5*	*6*	*7*	*8*
Round 1	2	2	2	3
Round 2	3	3	3	4
Round 3	2	4	3	4
Round 4	3	3	4*	5*
Round 5	3	4	4	5

* means the missions requires two sabotages to give a point to spies

For example, a game among 5 players may play out as follows:

- Players 2 and 4 become spies
- Round 1: Players 1 and 2 are voted to go on a mission
- No sabotages happen, the resistance gains one point
- Round 2: Players 1, 2 and 3 are voted to go on a mission
- This time, player 2 decides to sabotage the mission, giving one point to the spies. The resistance is aware of the point, but not of who did the sabotage
- Round 3: Players 1 and 2 are voted into the mission
- Player 2 sabotages the mission again, now the spies have 2 points
- Round 4: Players 1, 3 and 4 are voted into the mission
- Player 4 sabotages the mission, winning the game for the spies

We can see that the strategy for the spies is to sabotage missions as long as that doesn't give the resistance too much information, and the resistance's strategy is to agree on missions that avoid players who are probably spies. Note that mission assignments are dominated by the resistance, since they form the majority of players.

Modeling a Multi-Player Game as a Two Player Game

Even though this game has more than two players, we can model it as a two-player zero-sum game, thus allowing us to apply the methods discussed in Chapter 3 to find optimal strategies to it.

We consider that we have one player who represents the resistance and the other represent the spies. Then, the mission voting can be controlled by the resistance player, while the spies player decides on whether to sabotage each mission or not, but cannot decide to sabotage a mission where not enough spies were assigned.

This is a simplification of the game, as it assumes that all players are coordinated in their strategy, and that the spies can also make synchronized decisions without alerting the resistance. To be more precise, we can simplify the game in this manner by having all players agree on a strategy before the game starts. In a real scenario, this is often achieved after a series of games, where strategies can converge after enough discussions among the group of players.

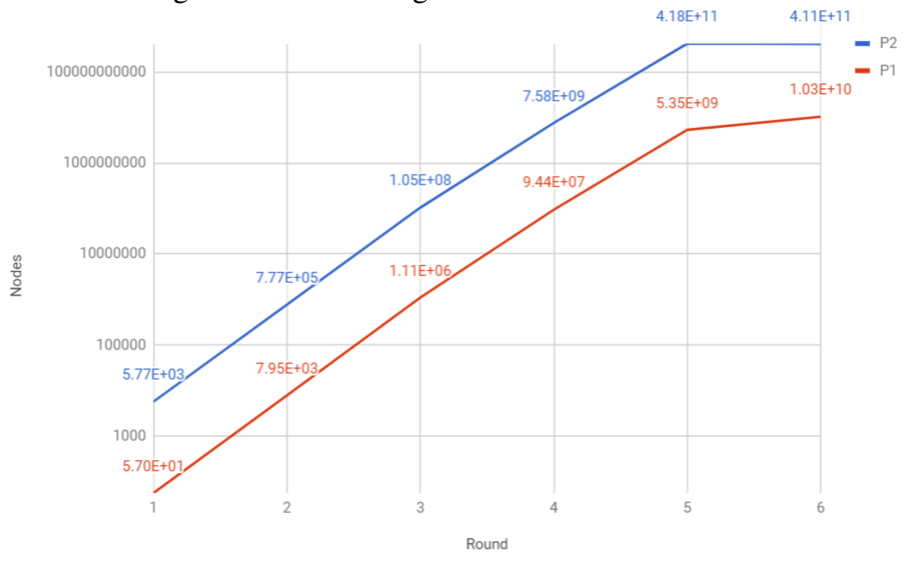
Then the resistance can make their decisions based on agreed protocols, such as die-rolling, coin-tossing, or cryptographic exchanges. This makes the voting aspect of the game irrelevant, since all resistance players would vote on the decision they agreed upon, and any spies that vote on another option would have no effect on the final result, but would improve the knowledge of the resistance. As for the spies' strategy, they can just sabotage missions the two-player optimal strategy says them to.

There is an important problem, however, when more than one spy is chosen to a mission. In that case, optimal 2-player strategy would dictate that, if they wanted to sabotage the mission, exactly the amount necessary would do it, and if not, no spy would sabotage the mission. So they face two issues: when not all chosen spies would need to sabotage the mission, they have to know who would do it. This can easily be prearranged, before the game starts. The second issue is when more than one spy would need to sabotage the mission, but their strategy dictates that they should do it but not with 100% probability, meaning they would need to take a synchronized decision. This can still be solved, albeit with more complex pre-game preparations, such as deciding on communication protocols that would not be detected by the other players. We can also solve the model ignoring this problem, then adapt the strategy found accordingly, though this adaptation falls out of the scope of this work.

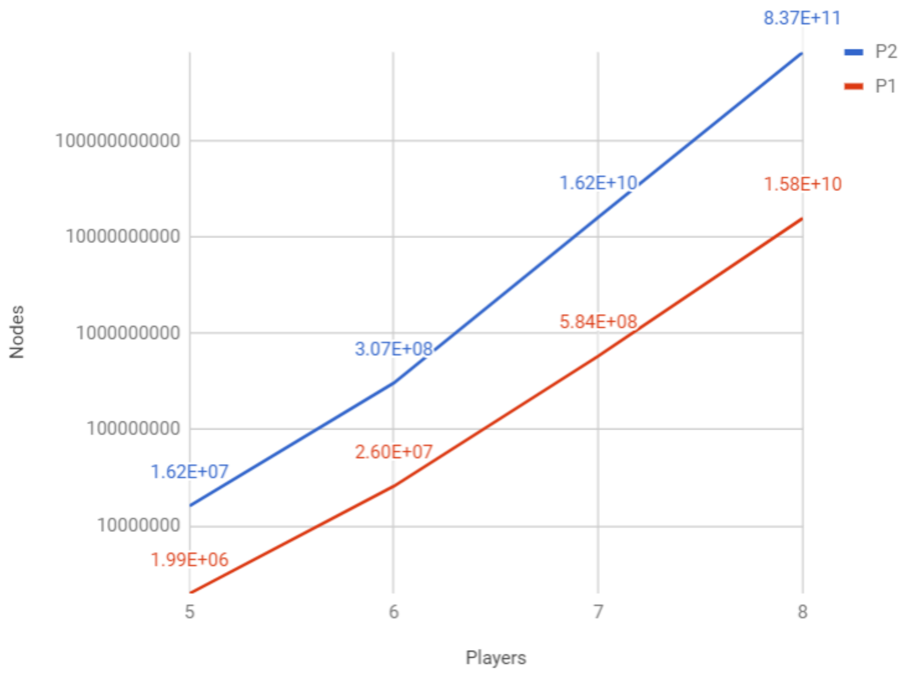
Size of the Game's Trees

In Figure 4.1 we can see how the game scales with the amount of players. We can note, particularly, that the size of the game grows exponentially as rounds pass, with the exception that many game stories finish after the 4th round. We can also see that the game size increases super-exponentially with the amount of players. Indeed, we could not generate any results for instances with more than 8 players, as even the reductions presented in Section 5.3 were not enough to make those instances fit in our server's memory.

Figure 4.1: Knowledge tree sizes for Resistance



(a) Sizes for 8 players



(b) Total sizes

5 STRATEGIES AND THEIR APPLICATION TO SOLVING THE RESISTANCE

In this chapter we present a reformulation of the techniques presented in (KOLLER; MEGIDDO; von Stengel, 1994), showing that they are the same as the linear programming model presented in (BOSANSKY et al., 2014). We then proceed to show the novel technique of symmetry reduction, which was used to great effect on solving the game The Resistance.

Adapting Games to Two Players

Our strategy for solving games is to first model them as a two-player zero-sum game. With a multi-player game, such as Resistance, we can, as discussed, group together players who aim to cooperate, and who are competing against the other team. We have observed that, in general, this is possible, albeit with varying degrees of faithfulness between the model and the actual game, depending on the game. For instance, we could model a political scenario, if we wanted to have an idea of a strategy, by limiting it to two competing parties, at the cost of not taking into consideration bipartisan agreements, betrayals, ethics, among many other factors.

Adapting Koller's Dual Method

We then adapt the method from (KOLLER; MEGIDDO; von Stengel, 1994), in order to have a more straightforward formulation, thus enabling a more effective implementation and future improvements. We do so by figuring out what kind of constraints may arise from performing the dual.

The first step from the cited work is to isolate the second player's variables. Thus we have the partial formulation:

$$\min \sum_{l_2 \in L_2} (\sum_{l_1 \in L_1} S_1[l_1]P(l_1, l_2))S_2[l_2]$$

subject to:

$$\forall a_2 \in A_2, S_2[a_2] = \sum_{c_2 \in C(a_2)} S_2[c_2]$$

$$\forall i_2 \in I_2, \forall c_2 \in C(i_2), S_2[i_2] = S_2[c_2]$$

$$S_2[Root_2] = 1$$

$$S_2[p_2] \in \mathbb{R}^+, \forall p_2 \in A_2 \cup I_2 \cup L_2$$

First we note that the variables in the primal model correspond to nodes, and constraints may be of two kinds: representing an edge where the parent is an information node, and representing a choice node and its children. While every choice node only appears in two constraints, its parent's and its own, information nodes appear in one constraint for every neighbor edge.

When taking the dual, these roles reverse. Each primal constraint from a choice node, since it contained many primal variables, becomes a dual variable that appears in many dual constraints. Meanwhile, an information node primal variable, since it appeared in many primal constraints, becomes a dual constraint that contains many dual variables. We can observe that in the dual:

$$\max M[Root_2]$$

subject to:

$$\forall l_2 \in L_2, M[l_2] \leq \sum_{l_1 \in L_1} S_1[l_1]P(l_1, l_2)$$

$$\forall a_2 \in A_2, \forall c_2 \in C(a_2), M[a_2] \leq M[c_2]$$

$$\forall i_2 \in I_2, M[i_2] \leq \sum_{c_2 \in C(i_2)} M[c_2]$$

$$M[p_2] \in \mathbb{R}, \forall p_2 \in A_2 \cup I_2 \cup L_2$$

When analyzing these new constraints and variables, we can see that an information node dual constraint is a sum, meaning the dual value for the corresponding node is the sum of its children. Also, the dual constraints where a choice node dual variable appears form a minimum constraint, meaning a choice node dual value is the minimum value of its children.

As for the root and leaves, in the primal the root appeared in the single constraint involving a constant (root = 1), and the leaves are the ones that appeared in the final value definition. In the dual, these two roles are also reversed: the root value becomes also the final value of the system, and the leaves have values assigned to them.

The complete interpretation of this dual is that, while in the primal we were interested in the optimal probability a strategy should choose for each node, $S[node]$, in the dual we only calculate the best payoff possible when we reach each node, $M[node]$. If a node represents a choice, the player may choose the child node that has the best payoff, thus the value of a choice node is the minimum of its children. And if a node represents waiting for information, the child node that is followed is taken at random, thus the value of the parent node is a weighted average of the values of its children. However, we already weigh the nodes accordingly, since random probabilities and the other player's strategy already affect the leaves' values, thus this weighted average is translated into simply a sum.

We can return the first player's restrictions to the system now, since we know that the dual we just modeled finds the same final values as its primal (WOLFE, 1961):

$$\max M[Root_2]$$

subject to:

$$\forall a_1 \in A_1, S_1[a_1] = \sum_{c_1 \in C(a_1)} S_1[c_1]$$

$$\forall i_1 \in I_1, \forall c_1 \in C(i_1), S_1[i_1] = S_1[c_1]$$

$$S_1[Root_1] = 1$$

$$\forall l_2 \in L_2, M[l_2] \leq \sum_{l_1 \in L_1} S_1[l_1] P(l_1, l_2)$$

$$\forall a_2 \in A_2, \forall c_2 \in C(a_2), M[a_2] \leq M[c_2]$$

$$\forall i_2 \in I_2, M[i_2] \leq \sum_{c_2 \in C(i_2)} M[c_2]$$

$$S_1[p_1] \in \mathbb{R}^+, \forall p_1 \in A_1 \cup I_1 \cup L_1$$

$$M[p_2] \in \mathbb{R}, \forall p_2 \in A_2 \cup I_2 \cup L_2$$

So the final system optimizes "probability" variables for one player, S, and "payoff" variables for the second, M. The constraints for the first player are:

- His root must have probability 1;
- Each choice node is constrained to have its probability be the sum of its children's;
- And each information node is constrained to have probability equal to each child

Then we have the second player's constraints on his leaves: they must have payoff equal to the expected payoff if the second player reached that leaf, but again the random event probability policy of affecting only leaves, as described in Section 3.5, also affects this payoff. After that, these leaf payoffs are propagated towards the root of the second player's tree, through the other second player constraints:

- An information node has payoff equal to the sum of its children;
- A choice node has payoff equal to the minimum of its children;
- The value being maximized by the system is the payoff value for the root of the second player's tree.

In the linear system, these constraints are translated as $M[n_2] \leq f(C(n_2)$ for f representing sum or minimum, respectively, which is not the equality $M[n_2] = f(C(n_2)$ one could expect. These two formulations, however, are equivalent, since, when maximizing the final payoff, these variables end up with the maximum value they can get, meaning they will be equal to the limit imposed on them.

While this formulation and similar can be found in other works, such as (BOSANSKY et al., 2014), it is important to link it to the dual method from (KOLLER; MEGIDDO; von Stengel, 1994), since this brings new points of view to this formulation, and may help with improving it and its usages. For example, in this work we are interested in obtaining the optimal strategies for both players, but this simplified formulation only obtains the strategy S of the first player, giving only the much less useful payoff values M for the second. This would mean that both this system and its dual would need to be solved separately in order to obtain the strategies for both players, S_1 and S_2. However, by understanding the duality in the formulation, we may take the dual constraints for the second player, which use variables M, and calculate their associated primal variables, thus obtaining the strategy S_2 for the second player, without the need for a complete reformulation and re-solving of the system. This improves the efficiency of the solution by more than double, since solving one formulation of the system may be much faster than solving its dual, if the game is quite asymmetric, as is the case for Resistance.

Symmetry Reduction

One important decision when solving a game is on how to treat symmetry. AlphaGo, for example, takes all symmetries of its training sets and uses them as part of the training set too, growing it by a factor of 8. In this work, however, our bottleneck is not that there is too little data being produced, but that there is too big of a game tree being generated. With that in mind, we take the opposite stance to AlphaGo, pruning symmetry, in order to reduce the size of our game tree. We do so with two techniques: by generating the knowledge trees directly, without generating the complete game tree; and by merging equivalent nodes and subtrees.

The strategy of generating knowledge trees directly is quite straightforward: instead of generating the complete game tree, and only then using that tree to generate the two knowledge trees, it is possible to skip that first step, at the cost of a more complex implementation. The main benefit, and also the main challenge, of generating knowledge trees directly, is that events not controlled by the player only manifest into the creation of new nodes when they give new information to the player.

This means that, for example, in a card game where both players start with the private knowledge of their N cards, the full game tree starts with one node for each combination of 2N cards, while knowledge trees only consider the combinations of N cards dealt to the player that owns the tree. Since the amount of combinations grows exponentially, cutting the amount of cards by half represents a great improvement in the amount of nodes generated. This technique is not immediately useful for the game of Resistance, since it only reduces the size of knowledge trees of players who don't have too much information. This means that it reduces the size of the resistance player's tree, but the spies still end up with a knowledge tree of size very similar to the full game's tree, since they have complete knowledge of the game's state.

We can, however, reduce the spies player's tree by noticing that he has many equivalent subtrees in his knowledge tree, and so does the resistance player. We define two subtrees as equivalent if they have an ismorphism, with possible relabeling of actions. Note that this isomorphism also requires the payoffs for leaf pairs to also match up. For Resistance, these isomorphisms are relabelings of the players. For example, if the resistance player sends players 1 and 2 into the first mission, that brings the same results as if he had sent players 1 and 3, if we swapped the names of players 2 and 3. We can then assume that there is an optimal strategy where every equivalent action is taken with the same probability, similarly to Rock-Paper-Scissors, as doing so only increases unpredictability of the player.

In order to implement this reduction, we have to be careful with conditional probabilities. For example, the resistance's first play is to send two random players on a mission. With 5 players, this means that the probability that no spies were sent into this mission is 30%, that both spies were sent is 10%, and that exactly one spy was sent is 60%. These calculations are far from trivial, so we solve this problem instead by, in any symmetry-reduced game state, keeping track of all possible spy combinations, along with the resistance player's knowledge. This knowledge consists in knowing which players went in exactly the same missions together and which went in different ones, meaning that players are partitioned into groups of non-differentiable players.

With these pieces of data, we can more easily calculate the conditional probabilities. In the example, the resistance's choice translates into breaking the group of 5 players into two groups, the two players that went in the first mission and the three players that did not go in that mission. Then we can filter the possible spy combinations according to those partitions: of the 10 pairs, 1 consists in both spies belonging to the first player partition, 6 have one spy in the first partition and the other on the second partition, and 3 have both spies in the second partition.

We do not have, however, a framework for discovery of equivalence properties in general games, requiring this optimization to be applied manually for other games. This is because finding them automatically would require deeper understanding of sub-game analysis, which, as described in (MORAVČÍK et al., 2017), is far from straightforward. It does rest, however, as a topic of interest for future work on the field.

Figure 5.1: Comparison of knowledge tree sizes with symmetry reduction on Resistance with 8 players

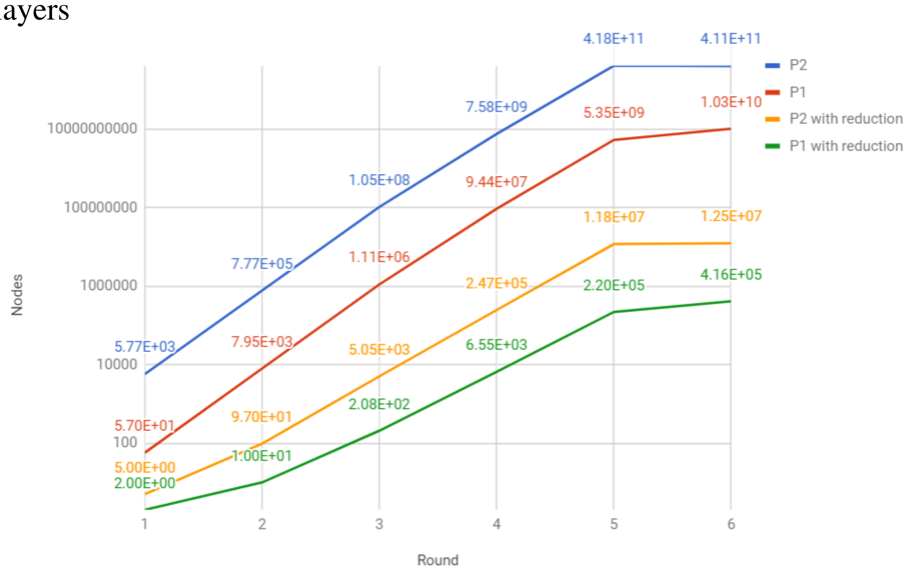
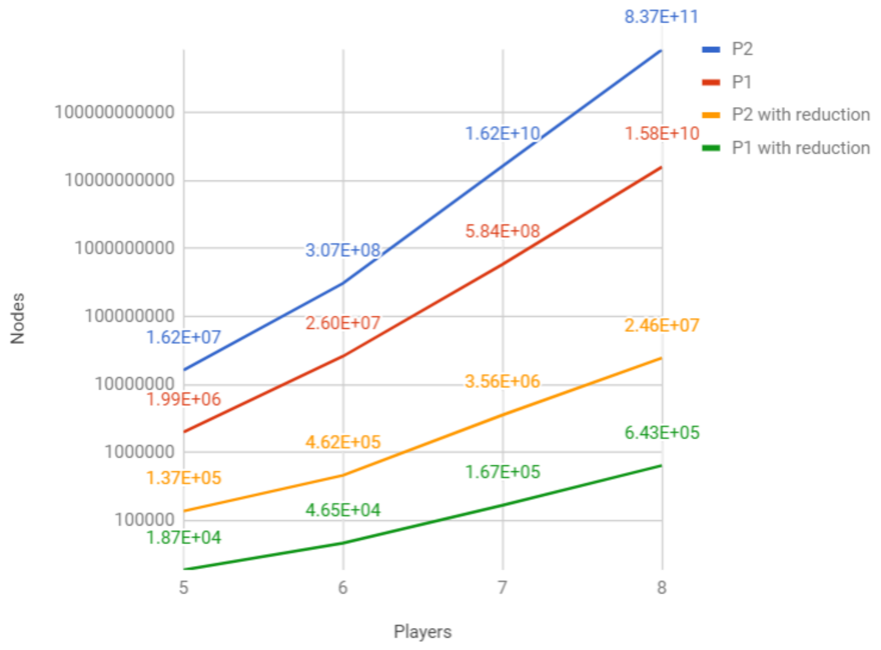


Figure 5.2: Comparison of total knowledge tree sizes with symmetry reduction on Resistance



6 RESULTS

In order to solve the linear programming models derived form the game's description, we used the GLPK solver, and the CPLEX solver. For GLPK, we generated the model in its MathProg language, while for CPLEX we used its C++ API. GLPK was not able to handle very large instances of the game, being very slow to solve the models, so its numbers were not interesting, thus we only present the results with CPLEX.

The tests were run in a server equipped with the 8-core FX-8150 AMD processor, which runs at 3.6GHz. It has 32GB of memory, Ubuntu 16.04 (xenial), on Linux 4.4.0.

CPLEX was much more efficient than GLPK, by using multiple threads, and using a barrier method. The barrier method is specially useful because the models generated by our techniques are quite degenerate. Indeed, only one independent variable is not 0, in the constraint $S[Root] = 1$.

With these results we can see the advantage of obtaining the strategy of the second player via dual variables. Indeed, the time taken to solve the model with roles reversed was up to 9 times faster, due to the asymmetry between the two players, in information and strategy, that is inherent to this game.

Table 6.1: Nash Equlibrium payoff for Resistance

Players	*5*	*6*	*7*	*8*
Payoff	-0.4	-0.333	-0.486	-0.679

Table 6.2: Resources used to solve Resistance, with first player controlling the resistance

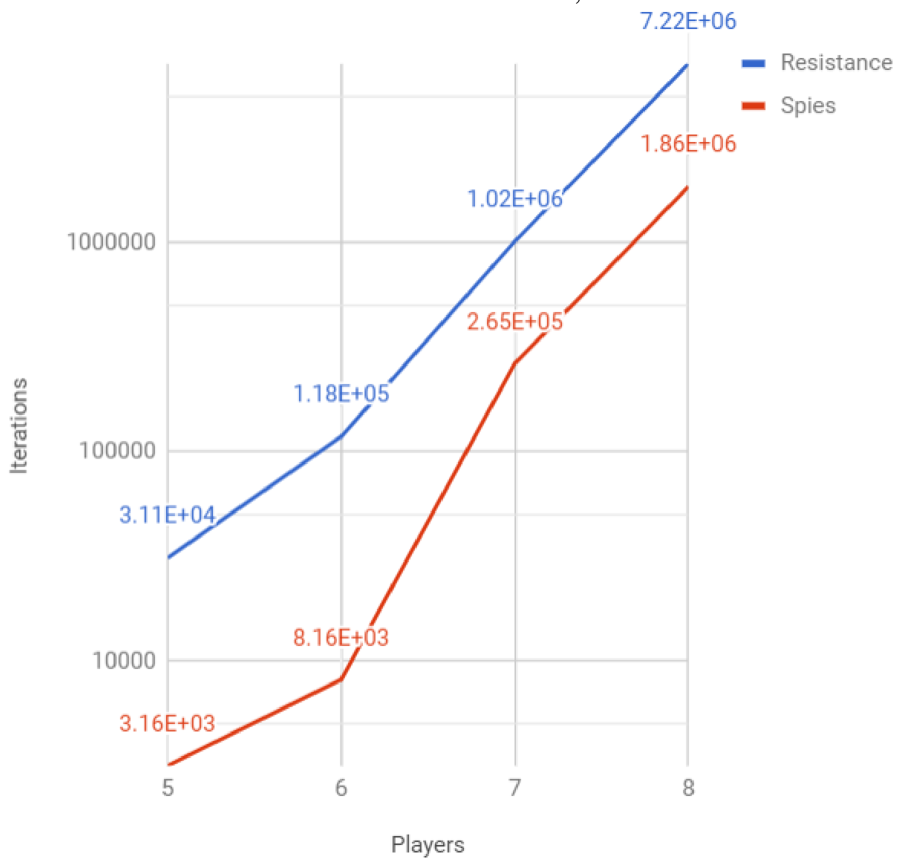
Players	*5*	*6*	*7*	*8*
CPU	220%	217%	249%	200%
Time	2.820s	13.620s	7m 21.200s	9h 47m 32s
Memory	311MB	1,057MB	7,491MB	31,720MB
Iterations	31,114	118,370	1,019,223	7,216,077

Table 6.3: Resources used to solve the dual of Resistance, that is, with first player controlling the spies

Players	*5*	*6*	*7*	*8*
CPU	229%	226%	365%	462%
Time	1.910s	10.080s	2m 58.650s	1h 7m 54s
Memory	214MB	691MB	5,352MB	31,748MB
Iterations	3,156	8,156	265,379	1,864,616

We can note that, in these results, the spies have an advantage. In games with human players, however, this difference is usually mitigated by the fact that it is quite hard

Figure 6.1: CPLEX iterations to solve Resistance, in both normal and dual form



for a human spy player to never act in a suspicious manner, as noted by (DEMYANOV et al., 2015), and also probably by the lack of knowledge of optimal strategies and by the lack of synchronization among spies, as discussed in Chapter 4, however these considerations fall out of the scope of this work.

7 CONCLUSION

We have solved the game The Resistance for up to 8 players, which is an imperfect information game with $8.5 * 10^{11}$ knowledge nodes. This was achieved through a combination of known techniques, specially the ones described in (KOLLER; MEGIDDO; von Stengel, 1994), with novel methods, particularly the symmetry reduction method we introduce.

Optimal solutions for games have quite a few uses, including the modification of protocols in order to improve their expected results, as studied in (LIU; MARSCHNER, 2017), and the development of artificial intelligences to play or aid the play of the situations from which the solved models were extracted from.

In our work, we found a partial solution for the problem of cross-interfering game subtrees in imperfect games, albeit requiring very restrictive assumptions, which is the symmetry reduction. Indeed, we found that if multiple choices lead to equal sub-game trees, other than permutations and relabeling, there is always some optimal solution where they receive the same probability. This result, if adapted into a generic solution, may prove important for the adaptation of algorithms usch as AlphaGo to imperfect information games, as this is a major issue faced in the transition from perfect to imperfect information.

We have also attempted to implement a method similar to the one described in (BOSANSKY et al., 2014): by restricting the game model temporarily, it is possible to obtain an optimal strategy for a simplified version of the game. Then this simplification can be expanded iteratively until the strategy found corresponds to a good approximation of an optimal strategy. Our implementation was inconsistent, meaning we did not obtain results from this approach, but it still seems very promising, as it heavily reduces the game tree sizes being worked with. We have modified the cited approach by improving the algorithm for the tree expansion. In particular, we used the strategy from (KOLLER; MEGIDDO, 1992), which efficiently discovers best responses, that could then be used as new expansion branches.

REFERENCES

BOSANSKY, B. et al. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. **Journal of Artificial Intelligence Research**, v. 51, p. 829–866, 2014.

BOWLING, M. et al. Heads-up limit hold'em poker is solved. **Science**, American Association for the Advancement of Science, v. 347, n. 6218, p. 145–149, 2015.

DEMYANOV, S. et al. Detection of deception in the mafia party game. In: ACM. **Proceedings of the 2015 ACM on International Conference on Multimodal Interaction**. [S.l.], 2015. p. 335–342.

FUDENBERG, D.; TIROLE, J. **Game theory**. [S.l.], 1991.

KNUTH, D. E.; MOORE, R. W. An analysis of alpha-beta pruning. **Artificial intelligence**, Elsevier, v. 6, n. 4, p. 293–326, 1975.

KOLLER, D.; MEGIDDO, N. The complexity of two-person zero-sum games in extensive form. **Games and Economic Bahavior**, v. 4, n. 4, p. 528–552, 1992.

KOLLER, D.; MEGIDDO, N. Finding mixed strategies with small supports in extensive form games. **International Journal of Game Theory**, v. 25, n. 1, p. 73–92, March 1996.

KOLLER, D.; MEGIDDO, N.; von Stengel, B. Fast algorithms for finding randomized strategies in game trees. In: **Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)**. [S.l.: s.n.], 1994. p. 750–759.

KOLLER, D.; MEGIDDO, N.; von Stengel, B. Efficient computation of equilibria for extensive two-person games. **Games and Economic Behavior**, v. 14, n. 2, 1996.

LIU, A. J.; MARSCHNER, S. Balancing zero-sum games with one variable per strategy. 2017.

MORAVČÍK, M. et al. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. **Science**, American Association for the Advancement of Science, v. 356, n. 6337, p. 508–513, 2017.

NASH, J. Non-cooperative games. **Annals of mathematics**, JSTOR, p. 286–295, 1951.

NEUMANN, J. V.; MORGENSTERN, O. **Theory of games and economic behavior**. [S.l.]: Princeton university press, 2007.

ROMSTAD, T. et al. **Stockfish: A strong open source chess engine**. 2017. Https://stockfishchess.org/.

SCHAEFFER, J. et al. Checkers is solved. **science**, American Association for the Advancement of Science, v. 317, n. 5844, p. 1518–1522, 2007.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. **Nature**, Nature Research, v. 529, n. 7587, p. 484–489, 2016.

SILVER, D. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. **arXiv preprint arXiv:1712.01815**, 2017.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press Cambridge, 1998. v. 1.

TESAURO, G.; GALPERIN, G. R. On-line policy improvement using monte-carlo search. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 1997. p. 1068–1074.

WANG, Z.; XU, B.; ZHOU, H.-J. Social cycling and conditional responses in the rock-paper-scissors game. **Scientific reports**, Nature Publishing Group, v. 4, p. 5830, 2014.

WOLFE, P. A duality theorem for non-linear programming. **Quarterly of applied mathematics**, v. 19, n. 3, p. 239–244, 1961.

ZINKEVICH, M. et al. Regret minimization in games with incomplete information. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2008. p. 1729–1736.