Universiteit Leiden
The Netherlands

Opleidingen W&I

An Analysis of the Cooperative Card Game The Crew

Aron de Jong

Supervisors:
Dr. W.A. Kosters & M.J.H. van den Bergh, MSc.

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden Mathematical Institute (MI)
www.liacs.leidenuniv.nl

14/07/2021

Abstract

The Crew is a cooperative trick taking game played with 3–5 players, in which players first choose objectives and must then proceed to win all tricks containing the cards specified by their objectives. Furthermore, players only see their own hand. Only by the playing behavior of other players (do they follow suit) and a hinting system can they know more about the hands of other players. We first consider the problem of winnability. For some configurations of cards, it is not possible to complete all objectives. We model the problem of winnability as a satisfiability problem and can then use a SAT solver to determine if a game of The Crew is winnable. Next, we define three different Monte Carlo players and compare their performance. In order to develop one of these, the smart Monte Carlo player, we first create a method for determinization. That is, a method for creating a card distribution conforming with what we know about other players' hands. In comparing the different players, we discover that such a more complicated, "smarter" player does not yield better performance than a simpler, "ignorant" player which uses card distributions that do not necessarily conform to what they could know.

Contents

1 Introduction

The focus of the field of game Artificial Intelligence has, for the longest time, been on *deterministic* games (no chance events) of *perfect information* (all players know the entire game state). The focus has also started to shift towards *non-deterministic* games (with chance events) and games of *imperfect information* (players can only observe part of the game state).

A popular approach to solving games that are non-deterministic or have imperfect information is *determinization*. Here, the game is reduced to (one or more) instances of a deterministic game of perfect information (called *determinizations*) by fixing all future chance events and hidden information. These determinizations can then be analyzed using techniques for deterministic games of perfect information. The results of these analyses can be combined to give information for the original game.

Figure 1: THE CREW card game box.

This paper will analyze THE CREW, a game which is a cooperative trick taking game of imperfect information. We will construct a method of determining if a game of THE CREW is winnable using SAT-solvers and we will create and compare three types of Monte Carlo players. We will also solve the problem of determinization for games of THE CREW (which comes down to determining possible hands of other players) and use this for one of our Monte Carlo players.

1.1 Thesis Overview

In Section 4, we first look into games where the objectives have been distributed already. We model the problem of winnability (is there a way to play the card such that the game is won) as a satisfiability problem. This allows us to use SAT-solvers to solve an instance of the winnability problem by first translating it to an instance of satisfiability and then checking if this instance can be solved. Next, we extend our translation method such that it can also check the winnability of games where objectives are yet to be distributed. This leads to the results discussed in Section 4.5.

In Section 5, we define three types of Monte Carlo players. One of these players first requires us to solve the problem of determinization, which is tackled in Section 5.2. The performance of these three Monte Carlo players is then compared in Section 5.3.

This thesis was written as part of the bachelor program of Mathematics and Computer Science at Leiden University under supervision of Mark van den Bergh (MI) and Walter Kosters (LIACS).

2 Related Work

An early example of a trick tacking game of imperfect information being tackled by an AI is *Bridge*. The program GID makes use of a Monte Carlo approach with determinization to play Bridge at approximately expert level [Gin01].

More recently, Monte Carlo simulations have also been used in the AlphaGo program, the first computer program to defeat a human professional player in the full-sized game Go, a deterministic game of perfect information [SHM+16]. The program AlphaGo Zero, a successor of AlphaGo, also makes use of Monte Carlo simulations and achieves superhuman performance in Go, beating AlphaGo 100–0 [SSS+17].

Determinization has also been used to implement Monte Carlo Tree Search for the popular Chinese card game *Dou Di Zhou* [WPC11].

An example of winnability for a cooperative game with imperfect information is *Hanabi*, where the question of the maximum score being reachable is considered [Ber15].

3 Definitions and Examples

The Crew is a co-operative trick taking game in which the players need to work together to complete a set of objectives. We will now give a concise description of the rules, followed by an example game.

3.1 Rules

The original game of The Crew, which we will call Classic Crew, is played by 3–5 players. The game lists 50 independent missions, increasing in difficulty. The players, also called *crew*, must try to complete all these missions. This can be done in one big session or over multiple session. Since all missions are independent, we shall consider a game as consisting of one mission.

Classic Crew is played with a deck of 40 cards, each having a value and a suit. There are four normal suits and one trump suit, where the normal suits have values 1–9 and the trump suit has values 1–4. These cards are divided evenly among the players, becoming their so-called *hand cards*, or *hand* for short. Notice that for the 3 player game, one player will have an extra card (thus they will end the game with 1 card left in their hand). Players may only see their own hand cards, not those of others. The starting player, or *commander*, is the player with the highest trump card, i.e., the trump 4 (hence, from the start of the game, all players know where the highest trump is). For our games of The Crew, we can of course vary the number of cards to suit our preferences. We may even eliminate trump cards altogether and appoint a starting player ourselves.

The game is a trick taking game, which means it consists of many small rounds (*tricks*) in which each player plays one card. In a trick, the trick starter can play any card from their hand. The suit of this starting card is called the *trick suit* or *suit of the trick*. Going clockwise, the other players must then play a card of the trick suit if they can, or play any card from their hand otherwise. If trumps were played, the player who played the highest trump wins the trick. If no trumps were played, the player who played the highest card of the trick suit wins the trick. The first trick is started by the commander and subsequent tricks are started by the winner of the previous trick.

In a game of The Crew, the goal is to complete all objectives. In Classic Crew, many types of objective exist. Examples are having specific players win tricks containing specific cards, having all 1s (the lowest card in the game) win a trick and winning no tricks with a 9 (the highest card of non-trumps). We will solely focus on the first of these three examples. Here, the mission specifies the number of objectives. Every objective is linked to a unique non-trump card in the game. We will call these linked cards the *objective cards*. After the specified number of objectives are randomly selected, players take turns picking objectives. If a player picks an objective, it becomes the goal of all players for this player to win the trick containing the associated objective card. The game is won if all objectives are completed. Hence, we can abort the game once one objective has failed (i.e., a trick containing an objective card was won by the wrong player) and we can declare victory once all objectives have been completed.

The only form of communication in The Crew is via a hinting system. Every player may, once per game, show one of their hand cards and indicate if it is their highest, lowest or only card of its suit. The card must meet one of these requirements.

3.2 Game Example

Consider a 4-player, 8-card game with normal cards ♠1–3,♣1–3 and trump cards ★1–2 where we are player South. After dealing the cards the situation might be as seen in Figure 2. In this case West has announced they are the commander; thus we know they have the highest trump, i.e., the trump 2. The cards ♠2 and ♣1 have been selected as objective cards (and put in the center of the table). Now West may first select an objective card, followed by North. Say West selects the ♠2 and North must then take the ♣1. This leads to the second situation.

A trick is then played in which West chooses to start with the ★2. Since West has an objective to win the trick with the ♠2, North plays the ♠2 (from which we can also conclude that North has no trumps since they didn't follow suit). East is obliged to follow suit and play their ★1. We (South) can play either card and choose to play our ♣3. Since trumps were played, the highest trump won the trick. Hence West won the trick and completed their objective to get the ♠2.

The remaining cards are played in the last trick. This trick has requested suit ♣ (since west played the ♣1) and North wins the trick with their ♣2. Since this trick contains the ♣1, all objectives are completed and the game is won.

4 Winnability

It is trivial to see that not all games of The Crew are winnable, that is to say, it is not always possible to complete all objectives. Consider any game in which one player will always win all tricks (since they have the highest cards). If any other player has an objective, the mission will inevitably fail. This leads to the question: "When is a game of The Crew winnable?" That is to say, when is there a way for the cards to be played such that all objectives are completed? To answer this question, we will formulate the problem as a satisfiability problem. Note that it suffices to only consider games where the distribution of hand cards is known (which is a game of perfect information). Hence, all games in this section will have known hand cards.

We will first analyze games of The Crew with *fixed objectives*, where the objectives are already distributed among the players. This will then be extended to games with *free objectives*, where the distribution of the objectives is yet to be determined.

Our new goal is to create an expression in conjunctive normal form such that an assignment of its Booleans that makes it `True`, also gives a solution to our game of The Crew. Furthermore, if no

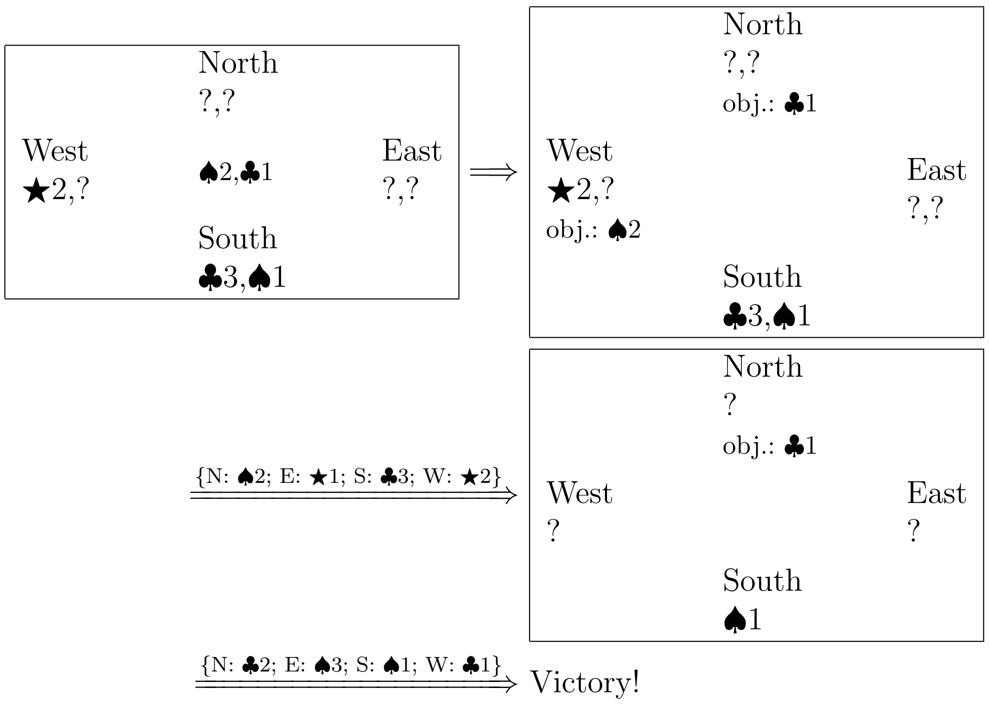

Figure 2: Example game of THE CREW.

such Boolean assignment exists, there must not be a solution to our game of THE CREW. We will now discuss how such an expression can be created.

4.1 Definitions

Consider a game of THE CREW with a set of players P, set of suits S and set of tricks $T = \{1, \ldots, \min_{p \in P} \#H_p\}$, where for every $p \in P$, the set H_p contains all hand cards of player p. In addition to the restrictions explicitly mentioned, the variables in this and the following sections on winnability are always restricted or iterated as follows:

$$p, q \in P \qquad t, t' \in T \qquad s \in S \qquad i \in \{1, \ldots, \#H_p\} \qquad j \in \{1, \ldots, (\#H_p \text{ or } \#H_q)\} \tag{1}$$

where the upper bound of j can be derived from the context (always either H_p or H_q).

We will use the following Booleans to create our expression

$$\begin{aligned} x^p_{t,i} &= \text{“player } p \text{ plays their } i\text{-th card on trick } t\text{”} \\ r^s_t &= \text{“the requested suit of trick } t \text{ is } s\text{”} \\ w^p_t &= \text{“player } p \text{ wins trick } t\text{”} \end{aligned} \tag{2}$$

For every player p, we also define the functions

$$\begin{aligned} V_p(i) &= \text{“value } v \in \mathbb{N} \text{ of the } i\text{-th card of player } p\text{”} \\ S_p(i) &= \text{“suit } s \in S \text{ of the } i\text{-th card of player } p\text{”} \end{aligned} \tag{3}$$

4.2 Construction of the Expression

We must ensure that in the expression any assignment which satisfies the Booleans, also gives a playout of our game which follows the rules of The Crew. Thus we must, among other things, ensure that every player plays exactly one card on each trick and that no card is played twice. This is realized by the following:

$$\bigwedge_{p,t} \bigvee_{i} x^p_{t,i} \quad \wedge \quad \bigwedge_{p,t} \bigwedge_{\substack{i,j \\ i \neq j}} \neg x^p_{t,i} \vee \neg x^p_{t,j} \quad \wedge \quad \bigwedge_{p,i} \bigwedge_{\substack{t,t' \\ t \neq t'}} \neg x^p_{t,i} \vee \neg x^p_{t',i} \tag{4}$$

We must also ensure w^p_t is `True` if and only if player p wins trick t, i.e., when no card "higher" than the card played by player p is played. This is the same as saying, when the j-th card of player q is greater than the i-th card of player p (in the requested suit s), it cannot be that both are played. This first expression of (5) can now also be made true by setting all w^p_t to `False`. To fix this, a second expression is added, requiring that every trick has at least one winner. Since in every trick, there is exactly one "highest" card played, the first expression already ensures that at most one w^p_t is true for each trick. This now yields

$$\bigwedge_{\substack{p,q,t,s \\ p \neq q}} \bigwedge_{\substack{i,j \\ (p,i) \prec_s (q,j)}} \neg w^p_t \vee \neg r^s_t \vee \neg x^p_{t,i} \vee \neg x^q_{t,j} \quad \wedge \quad \bigwedge_{t} \bigvee_{p} w^p_t \tag{5}$$

where $(p,i) \prec_s (q,j)$ if and only if the i-th card of player p would lose to the j-th card of player q in a trick of suit s. This happens if players p and q both played the same suit (either the requested suit or trumps) but player p's card is lower; or player p did not play trumps but player q did; or player p played neither the requested suit, nor a trump card and player q did play the requested suit. This is captured by

$$\begin{aligned} (p,i) \prec_s (q,j) \iff & (S_q(j) = S_p(i) \in \{s, \bigstar\} \wedge V_p(i) < V_q(j)) \\ & \vee (S_q(j) = \bigstar \wedge S_p(i) \neq \bigstar) \\ & \vee (S_q(j) = s \wedge S_p(i) \notin \{s, \bigstar\}) \end{aligned} \tag{6}$$

Notice r^s_t must be true if and only if suit s was played by the winner of the previous trick $(t-1)$ or, on the first trick, by the commander. This gives

$$\bigwedge_{\substack{t,p,i \\ t \neq 1}} \neg w^p_{t-1} \vee \neg x^p_{t,i} \vee r^{S_p(i)}_t \quad \wedge \quad \bigwedge_{i} \neg x^c_{1,i} \vee r^{S_c(i)}_1 \quad \wedge \quad \bigwedge_{\substack{t,s,s' \\ s \neq s'}} \neg r^s_t \vee \neg r^{s'}_t \tag{7}$$

where $1 \leq c \leq P$ is the commander. Now to ensure all players follow suit, we check that if a card not of the requested suit is played, all cards of the requested suit have been played already, resulting in

$$\bigwedge_{\substack{p,t,i,s \\ s \neq S_p(i)}} \neg x^p_{t,i} \vee \neg r^s_t \vee \left(\bigwedge_{j : S_p(j) = s} \bigvee_{t' < t} x^p_{t',j} \right) \iff \bigwedge_{\substack{p,t,i,s \\ s \neq S_p(i)}} \bigwedge_{\substack{j \\ S_p(j) = s}} \neg x^p_{t,i} \vee \neg r^s_t \vee \left(\bigvee_{t' < t} x^p_{t',j} \right) \tag{8}$$

Lastly, to ensure all objectives are achieved, we use

$$\bigwedge_{\substack{p,t\\(q,i)\in O_p}} \neg x^q_{t,i} \vee w^p_t \qquad \wedge \qquad \bigwedge_{\substack{p\\(q,i)\in O_p}} \bigvee_t x^q_{t,i} \tag{9}$$

with

$$O_p = \{(q,i) : \text{the } i\text{-th card of player } q \text{ is an objective of player } p\} \tag{10}$$

In (9), the second term is only required if not all players have the same number of cards. In that case, it can happen that a card is never played, thus we must check that all objective cards are actually played at some point.

4.3 The Expression

Combining equations (4), (5), (7), (8) and (9) we get equation (11), which can be used to solve games of THE CREW with fixed objectives. If there exists an assignment such that the system is `True`, the assignment of the $x^p_{t,i}$ variables gives a way to play the cards such that the game is won. If the system cannot be made `True`, the corresponding game of THE CREW cannot be won.

4.4 Extending to Games with Free Objectives

As discussed earlier, we also want to solve games with free objectives, i.e., games where the objectives have not yet been distributed. In order to allow the SAT-solver to assign objectives, we add a new variable

$$o^p_{(q,i)} = \text{“player } p \text{ has the } i\text{-th card of player } q \text{ as an objective”} \tag{12}$$

In a game of THE CREW, the objectives are picked one-by-one, going clockwise and starting at the commander (the starting player) until all objectives are picked. Hence, the number of objectives a player will be assigned is known and fixed. For player p we will denote this number by n_p. To ensure that every player has n_p objectives assigned, it suffices to require that every player is assigned at most n_p objectives and that every objective is assigned to at least one player. This holds because $\sum_p n_p = \#O$ with O the set of objectives. This is captured by

$$\bigwedge_p \bigwedge_{\substack{C\subseteq O\\ \#C=n_p+1}} \bigvee_{c\in C} \neg o^p_c \qquad \wedge \qquad \bigwedge_{c\in O} \bigvee_p o^p_c \tag{13}$$

with

$$O = \{(q,i) : \text{the } i\text{-th card of player } q \text{ can be chosen as an objective}\}. \tag{14}$$

Note that more efficient ways to encode “at most k out of n literals true” exist, most of which introduce new variables to speed up the SAT-solver [FG10]. We have opted for the “binomial” encoding because of its relative simplicity compared to other, more efficient encodings. Equation (13) can now be tied into (11) by changing (9) to

$$\bigwedge_{\substack{p,t\\(q,i)\in O}} \neg o^p_{(q,i)} \vee \neg x^q_{t,i} \vee w^p_t \qquad \wedge \qquad \bigwedge_{(q,i)\in O} \bigvee_t x^q_{t,i} \tag{15}$$

$$\begin{cases} \bigwedge_{p,t} \bigvee_{i} x^p_{t,i} \\ \bigwedge_{p,t} \bigwedge_{\substack{i,j \\ i \neq j}} \neg x^p_{t,i} \vee \neg x^p_{t,j} \\ \bigwedge_{p,i} \bigwedge_{\substack{t,t' \\ t \neq t'}} \neg x^p_{t,i} \vee \neg x^p_{t',i} \\ \bigwedge_{\substack{p,q,t,s \\ p \neq q}} \bigwedge_{\substack{i,j \\ (p,i) \prec_s (q,j)}} \neg w^p_t \vee \neg r^s_t \vee \neg x^p_{t,i} \vee \neg x^q_{t,j} \\ \bigwedge_{t} \bigvee_{p} w^p_t \\ \bigwedge_{\substack{t,p,i \\ t \neq 1}} \neg w^p_{t-1} \vee \neg x^p_{t,i} \vee r^{S_p(i)}_t \\ \bigwedge_{i} \neg x^c_{1,i} \vee r^{S_c(i)}_1 \\ \bigwedge_{\substack{t,s,s' \\ s \neq s'}} \neg r^s_t \vee \neg r^{s'}_t \\ \bigwedge_{\substack{p,t,i,s \\ s \neq S_p(i)}} \bigwedge_{\substack{j \\ S_p(j) = s}} \neg x^p_{t,i} \vee \neg r^s_t \vee \left(\bigvee_{t' < t} x^p_{t',j} \right) \\ \bigwedge_{\substack{p,t \\ (q,i) \in O_p}} \neg x^q_{t,i} \vee w^p_t \\ \bigwedge_{\substack{p \\ (q,i) \in O_p}} \bigvee_{t} x^q_{t,i} \end{cases} \tag{11}$$

4.5 Results

Using (11) and Section 4.4, we can determine the fraction of games that is winnable, both with fixed and free objectives. In order to do this, we have written a `C++` program capable of generating games and solving winnability for games of THE CREW, making use of the Kissat SAT Solver [Kis]. This program can be found on our GitHub [Jon]. Many different SAT solvers exist but we have chosen the Kissat SAT Solver because it got first place on the main track and first place on unsatisfiable instances in the 2020 SAT Competition.

For every possible number of missions, 10,000 games of THE CREW with four trump cards and nine cards in all of the other four suits (just like CLASSIC CREW) have been generated and we have determined whether they are winnable. This has been done for both games with fixed and free objectives. The results have been plotted in Figure 3.

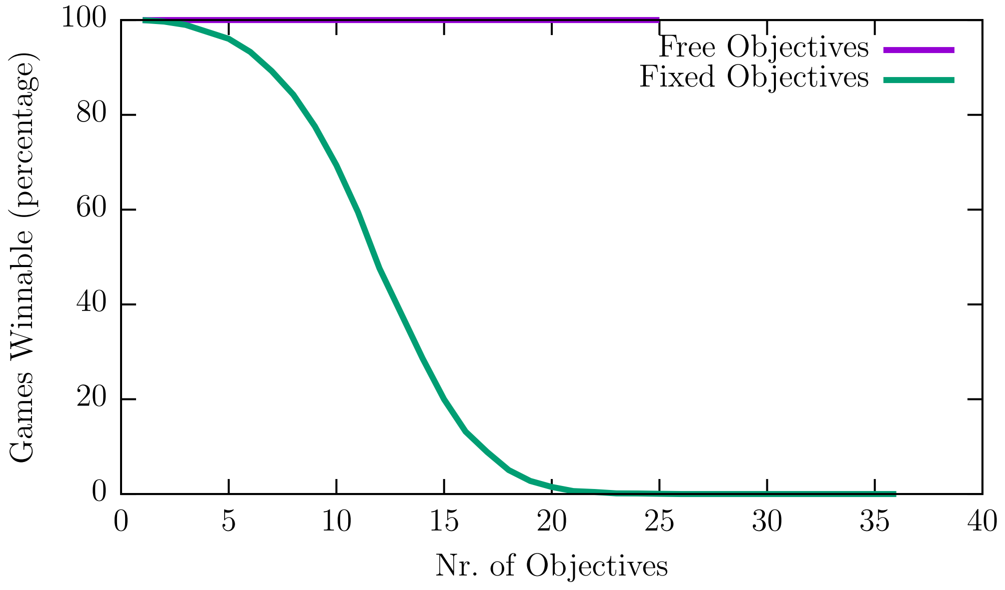

Figure 3: Percentage of CLASSIC CREW games winnable with both fixed and free objectives.

As can be seen in Figure 3, allowing players to divide the objectives makes almost every game winnable. Interestingly, all of the 220,000 generated games with 2 up to and including 23 objectives turned out to be winnable. Yet we know not all such games are winnable. Take, for instance, a game where the commander has all trump cards and all their other cards are highest in their suit. The commander is then unable to lose a trick and hence will win all tricks. If then any player other than the commander has an objective (which happens if there is more than one objective), the mission will fail. We conclude that the fraction of games with free objectives that is winnable is close to 1, irrelevant of the number of objectives. That is to say, THE CREW is a well-designed game because it is almost always winnable.

5 Monte Carlo Players

A Monte Carlo player works using *random playouts*, the process playing or continuing a game but with all players selecting random moves. For every possible move in a game state, the Monte Carlo player will perform many random playouts on a game with this move performed (more on this later), assigning scores to each playout (based on the state of the game after it is either won or lost). The move whose playouts have the best average score is then selected. If two moves can have equal average scores, a tiebreaker must be given. This can be as simple as randomly choosing one of the moves with equal best average score but can also be more complicated in order to create secondary priorities (for instance, if the player knows they will fail, they can still try to play as long as possible before failing).

For The Crew, we might score a random playout by the number of objectives left to complete (where a lower score is better). If two playouts have the same number of objectives left but one has more tricks completed before failing, we will prefer the playout with more tricks. If the number of tricks is also equal, we may choose any of the equally "good" moves. In such cases, we will always choose the first such move we encounter.

In the following sections, we will discuss a cheating Monte Carlo player, create a non-cheating ignorant and a non-cheating smart Monte Carlo player (the latter of which requires determinization) and proceed to compare the performance of the different player types.

5.1 Types of Monte Carlo Players

A *cheating Monte Carlo player* would take the current game state, perform a move on it and then start random playouts (as explained above) on this updated game state. This is cheating, because in The Crew, the hands of other players are (most of the time) unknown. Thus continuing from the current game state will use information that should be unknown to the player. One way to remedy this would be to randomly generate the hands of other players instead of taking them from the current game state.

One way to generate a distribution of the remaining cards over the hands of the other players would be to distribute these cards at random. We will call a Monte Carlo player which uses this approach for creating a game state an *ignorant Monte Carlo player*. This player is called "ignorant" because it does not use all information that is available. Whenever a player p does not follow the requested suit s, this tells us that player p has no cards of suit s. The ignorant Monte Carlo player can however still generate a card distribution in which player p has cards of suit s.

To create a *smart Monte Carlo player* (which does not cheat), we would like to generate card distributions satisfying all our current information. For our purposes, "all our current information" will be a collection of supersets C_p for each player p such that the hand of player p is known to be a subset of C_p. This is further discussed in Section 5.2, where this problem of generating a card distribution satisfying our current information is covered.

5.2 Determinization

"For an instance of a stochastic game with imperfect information, a *determinization* is an instance of the equivalent deterministic game of perfect information, in which the current state is chosen from the AI agent's current information set, and the outcomes of all future chances are fixed and known" [WPC11]. Thus, for our game of THE CREW, a determinization is an instance of the game where all players' cards are known to all players. We will discuss multiple strategies for determinization, only one of which, determinization using probability, actually works.

We can represent the information our Monte Carlo player has as a set C_p for every player p such that the hand H_p of player p is known to be a subset of C_p. Then if player p does not follow requested suit s, we can remove all cards of suit s from their superset C_p. Furthermore, all cards played are removed from all supersets. Since we also know that the commander c has the highest trump card, we can remove the highest trump card from all supersets C_p with $p \neq c$. Also notice that the number of cards a player has, $\#H_p$, is known.

We would like to generate any determinization from the set of possible determinizations with (approximately) equal probability.

5.2.1 Determinization using Probability

For this section, we consider general card games of imperfect information where all cards are dealt at the start of the game. We will first introduce our strategy using a game with an arbitrary number of players and then work out the specifics for 3– and 4–player games.

Since all unknown cards are hand cards, we have

$$\bigcup_{p \in P'} H_p = \bigcup_{p \in P'} C_p \tag{16}$$

where P' is the set of players whose hand cards are unknown/yet to be determined.

Using the following algorithm, we can create a card distribution using simple probabilities. Here, we construct the sets H_p such that $H_p \subseteq C_p$ for all p and such that all such distributions are equiprobable. We do this by narrowing down the C_p until they are disjoint.

Algorithm 1. Given players P' and a collection of supersets $C_p \supseteq H_p$ for every player $p \in P'$, execute:

```
1. For every card $c \in \bigcup_{p \in P'} C_p$, in any order, do
    (a) Calculate probability $P_p = \mathbb{P}(c \in H_p \mid \forall q \in P' : H_q \subseteq C_q)$ for every player $p \in P'$.
    (b) Choose one player $q \in P'$, each with probability $P_q$.
    (c) For every player $p \in P' \setminus \{q\}$, set $C_p = C_p \setminus \{c\}$.
2. The generated hands are $H_p = C_p$ for all players $p \in P'$.
```

We will now prove that this algorithm has the desired result.

Lemma 1. *Algorithm 1 uniformly generates a determinization with $H_q \subseteq C_q$ for all players $q \in P'$.*

Proof. We prove this using induction on the number of cards divided, that is, the iteration in step 1 of the algorithm. Let supersets C_p for all players $p \in P'$ be given. Number all cards $\{c_1, c_2, \dots, c_n\}$ in the order of step 1. We will now prove that the partial distribution $D_i := \{C_p\}_{p \in P'}$ obtained on the i-th iteration satisfies our induction hypothesis, which is

$$\mathbb{P}(\text{algorithm generates } D_i) = \mathbb{P}(D_i \supseteq H) \tag{17}$$

where we write $D_i \supseteq H$ for D_i being supersets of a uniformly chosen valid distribution.

For $i = 1$ we have D_1 the localization of card c_1 to a single set C_p with $p \in P'$. Thus the algorithm generates this D_1 with probability $P_p = \mathbb{P}(c_1 \in H_p \mid \forall q \in P' : H_q \subseteq C_q)$ which is exactly the probability of player p having card c_1 given our restrictions. This is the same as saying it is the probability of player p having card c_1 in a uniformly chosen valid distribution, i.e., $c_1 \in C_p$ and $c_1 \notin C_q$ for $q \neq p$.

Now let $1 < i \leq n$ be given and assume (17) holds for $i - 1$. Let c_i be assigned to player p on iteration i. We now have

$$\begin{aligned}
\mathbb{P}(D_i \supseteq H) &= \mathbb{P}(D_{i-1} \supseteq H \text{ and } c_i \in H_p) \\
&= \mathbb{P}(D_{i-1} \supseteq H) \cdot \mathbb{P}(c_i \in H_p \mid D_{i-1} \supseteq H) \\
&= \mathbb{P}(\text{algorithm generates } D_{i-1}) \cdot P_p \\
&= \mathbb{P}(\text{algorithm generates } D_i)
\end{aligned}$$

Hence we conclude (17) holds for all $1 \leq i \leq n$ and specifically for $i = n$. Thus from (17) we conclude that for any collection of supersets D we have

$$\mathbb{P}(\text{algorithm generates } D) = \mathbb{P}(D \text{ is superset of a uniformly chosen valid distribution}) \tag{18}$$

and since the different supersets of the generated D are disjoint, it also describes a determinization of the hand cards with $H_q = C_q$ for player $q \in P'$ and $D = \{C_p\} p \in P'$.

Since a collection of superset D is only generated if it is a superset of a uniformly chosen valid distribution, the algorithm only generates valid distributions. □

It remains to calculate the probabilities $P_p = \mathbb{P}(c \in H_p \mid \forall q \in P' : H_q \subseteq C_q)$ for arbitrary player p, card c and sets H_q, C_q with $q \in P'$. We will do this by considering 3 and 4–player games separately.

3–Player Games) Since our own hand is known, we only need to create a determinization of the hands of the remaining 2 players. Let players p and q be the players with unknown hands, i.e., $P' = \{p, q\}$. Notice all cards of $C_p \setminus C_q$ are guaranteed to be owned by player p and vice versa. We conclude that player p must select $\#H_p - \#(C_p \setminus C_q)$ cards from $C_p \cap C_q$, thus for any card c, we have

$$\mathbb{P}(c \in H_p \mid H_p \subseteq C_p, H_q \subseteq C_q) = \begin{cases} \frac{\#H_p - \#(C_p \setminus C_q)}{\#(C_p \cap C_q)}, & c \in (C_p \cap C_q) \\ 1, & c \in C_p \setminus C_q \\ 0, & c \notin C_p \end{cases} \tag{19}$$

4–Player Games) Analogous to the 3-player games, we only need to create a determinization of the remaining 3 players. Let p, q, r be the players with unknown hands, i.e., $P' = \{p, q, r\}$. For given supersets C_p, C_q, C_r, the number of valid distributions is given by

$$T_v(C_p, C_q, C_r) \coloneqq \sum_{i,j,k} \binom{\#((C_p \cap C_q) \setminus C_r)}{i} \binom{\#(C_p \cap C_q \cap C_r)}{j} \binom{\#((C_p \cap C_r) \setminus C_q)}{\#H_p - i - j - \#(C_p \setminus (C_q \cup C_r))} \binom{\#((C_q \cap C_r) \setminus C_p)}{k} \binom{\#(C_p \cap C_q \cap C_r) - j}{\#H_q - \#((C_p \cap C_q) \setminus C_r) + i - k} \tag{20}$$

with $i, j, k \in \mathbb{Z}_{\geq 0}$ such that none of the binomial coefficients contain negative numbers or have the bottom number be greater than the top number.

Here, the first 3 terms determine how the cards of player p are distributed over the different superset intersections. That is, player p has i cards from $(C_p \cap C_q) \setminus C_r$ and has j cards from $C_p \cap C_q \cap C_r$. Knowing i and j, the remaining cards that player p needs to take from $(C_p \cap C_r) \setminus C_q$ can be calculated (which is used in the third term). The last 2 terms determine how player q's cards are distributed. This is done by letting player q take k cards from $(C_q \cap C_r) \setminus C_p$ and take the rest of the cards they need from $C_p \cap C_q \cap C_r$ (from which player p has already taken j cards). Having both the cards of p and q distributed leaves the remaining cards for player r. This distribution has been visualized in Figure 4, with the subscripts indicating to which player the number of cards belong.

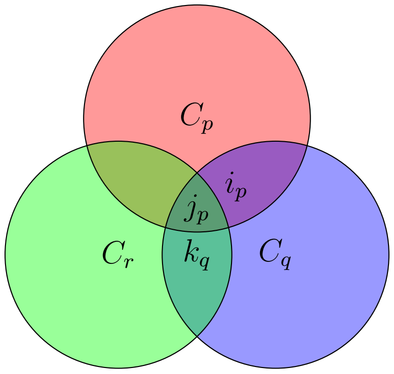

Figure 4: Card distribution visualized.

The number of valid distributions with $c \in H_p$ can be seen as the total number of distributions with supersets $C'_p = C_p$, $C'_q = C_q \setminus \{c\}$ and $C'_r = C_r \setminus \{c\}$ since this ensures card c can only be assigned to player p. Hence, we have

$$\mathbb{P}(c \in H_p \mid \forall q \in P' : H_q \subseteq C_q) = \frac{T_v(C_p, C_q \setminus \{c\}, C_r \setminus \{c\})}{T_v(C_p, C_q, C_r)} \tag{21}$$

with T_v as in (20).

***n*-Player Games)** The strategy for calculating $\mathbb{P}(c \in H_p \mid \forall q \in P' : H_q \subseteq C_q)$ used for 3-player games, that is, calculating the number of games which satisfy the conditions, can be extended to more players. We calculated (20) by assigning cards to players in order. Every player p required a variable for every intersection $C = (\bigcap_{i \in I} C_i) \setminus (\bigcup_{j \in J} C_j)$ (with $I \cup J = P'$) of supersets with $p \in I$ such that there is another player $q \in I$ with $p \neq q$ whose cards have not yet been fixed.

This variable then indicates how many cards player p takes from this intersection. We will now determine the number of variables required if we want to do the same for n players.

Let $P' = \{1, \ldots, n\}$. We will assign cards to players in order, hence, when we get to assigning cards to player i, the cards of all players $j < i$ have already been fixed. The cards of player n are fixed by the distribution of all other cards. The cards of player $n-1$ can still be undetermined in every intersection containing both C_n and C_{n-1}, thus requiring $2^{n-2} - 1$ variables. In general, the cards of player $i \neq n$ can still be undetermined in every intersection containing both C_i and any of $C_{i+1}, C_{i+2}, \ldots, C_n$. There are $2^{n-1} - 2^{i-1}$ such intersections because there are 2^{n-1} intersections containing C_i, with 2^{i-1} of those not containing any of the $C_{i+1}, C_{i+2}, \ldots, C_n$. Since determining the number of cards in all but one of these intersections also determines the number of cards in the last intersection, $2^{n-1} - 2^{i-1} - 1$ variables are required for player $i \neq n$ and player n requires 0 variables. Hence, the total number of variables required is

$$\begin{aligned}\sum_{i=1}^{n-1} \left(2^{n-1} - 2^{i-1} - 1\right) &= (n-1)(2^{n-1} - 1) - \sum_{i=1}^{n-1} 2^{i-1} \\ &= (n-1)(2^{n-1} - 1) - (2^{n-1} - 1) \\ &= (2^{n-1} - 1)(n-2) \end{aligned} \tag{22}$$

A few examples of n and the corresponding number of variables required can be found in Table 1.

n	1	2	3	4	5	6	7	8	9	10
nr. of variables	0	0	3	14	45	124	315	762	1785	4088

Table 1: Amount of summation variables required for n player games.

5.2.2 Determinization using Switch Markov Chains

Markov Chains have previously been used to model the shuffling of cards [Ham05] and seem to be a good approach for uniform determinization. This Markov Chain must then have the valid determinizations as a subset of the state space and must have a stationary distribution in which all valid determinizations are equiprobable. Algorithm 1 is, in essence, a Markov Chain and has been proven to uniformly generate a valid determinization.

We will call Markov Chains which traverse the set of determinizations by first selecting cards with a given probability distribution and then transposing the positions of these cards *switch Markov Chains* (analogous to switch Markov Chains used for selecting graphs with given degree sequence approximately uniformly at random [FGMS06]). The "simplest" form of such a switch Markov Chain would have the set of all valid determinizations as its state space. It turns out such a "simple" switch Markov Chain would not be simple at all. The following example gives a game of The Crew with P players such that any switch Markov Chain which has the set of valid distributions as its state space and is capable of randomly generating any of the valid determinizations would have to be capable of switching P cards at once.

Example 1. *Consider a game of* The Crew *with players* $1, \ldots, P$ *and suits* $1, \ldots, P$ *where every suit has exactly one card. Assume player* p *can only have cards of suit* p *or* $(p+1) \bmod P$*. Notice* *that there are exactly 2 valid card assignments: either every player p has the card of suit p or every player p has the card of suit $(p+1) \bmod P$. Thus any switch Markov Chain capable of generating both assignments must be capable of transforming one into the other, requiring the shifting of all cards to the next/previous player. Thus requiring the switching of P cards at once.*

We conclude that determinization using switch Markov Chains, although theoretically feasible, is most likely not practical.

5.2.3 Determinization using Backtracking

A standard approach for generating solutions to a problem is backtracking. Backtracking, the process of changing previous decisions until the solution is valid, will walk through the space of possible (but not always valid) distributions in a specific order until it finds a distribution that is valid. Although this generates a valid distribution, it does not always pick a valid distribution uniformly at random.

Consider the game in Figure 5 with cards $\{\spadesuit, \clubsuit, \heartsuit\}$. Backtracking would visit the possible solutions in a predetermined order. Say this is lexicographical. The order used in Figure 6 is lexicographical from high to low with $\spadesuit > \clubsuit > \heartsuit$. The only valid assignments are assignments 4 and 5 of Figure 6, thus a backtracking algorithm starting on 5 would generate assignment 5 and for all other starting positions it would generate assignment 4. Therefore backtracking would not uniformly generating a valid card distribution.

	North Has no $\spadesuit$	
West Has no $\heartsuit$		East Has no $\clubsuit$

Figure 5: An example game for backtracking difficulties.

1. $\{$N: $\spadesuit$; E: $\clubsuit$; W: $\heartsuit\}$
2. $\{$N: $\spadesuit$; E: $\heartsuit$; W: $\clubsuit\}$
3. $\{$N: $\clubsuit$; E: $\spadesuit$; W: $\heartsuit\}$
4. $\{$N: $\clubsuit$; E: $\heartsuit$; W: $\spadesuit\}$
5. $\{$N: $\heartsuit$; E: $\spadesuit$; W: $\clubsuit\}$
6. $\{$N: $\heartsuit$; E: $\clubsuit$; W: $\spadesuit\}$

Figure 6: All possible card assignments for 3 player 3 card games.

5.2.4 Determinization using Most Restricted First

One of the more intuitive strategies for determinization is to first deal cards to the player with the "most restrictions" on their hand, then going to less restricted players in order. We can define "most restricted" as having the least number of possible cards, i.e., player $p \in P'$ is most restricted if $\#C_p \leq \#C_q$ for all $q \in P'$. This strategy can be formalized as the following algorithm

Algorithm 2. Given players P' and a collection of supersets $C_p \supseteq H_p$ for ever player $p \in P'$.

1. While $P' \neq \emptyset$:
   (a) Choose any $p \in P'$ such that $\#C_p \leq \#C_i$ for all $i \in P'$.
   (b) With a uniform random distribution, choose a set C of $\#H_p$ cards from C_p.
   (c) Set $C_p = C$ and $C_q = C_q \setminus C$ for every player $q \in P' \setminus \{p\}$.
   (d) Set $P' = P' \setminus \{p\}$.
2. The generated hands are $H_p = C_p$ for all players $p \in P'$.

This approach does not work for all games, as is shown in the following example.

Example 2. *Consider the game in Figure 7 with cards* $\{\spadesuit, \clubsuit, \heartsuit\}$*. Only assignments 4, 5 and 6 of Figure 6 are valid card distributions for this game. Notice players North and West are equally restricted, i.e.,* $\#C_{North} = \#C_{West} < \#C_{East}$*. Uniformly assigning a card to North would result in* $\spadesuit$ *and* $\heartsuit$ *having the same probability, even though north has* $\heartsuit$ *in 2 of the 3 possible distributions and* $\spadesuit$ *in only 1 of the 3. By symmetry reasons, this also does not work if West is assigned cards first.*

	North	
	Has no $\spadesuit$	
West		East
Has no $\heartsuit$		-

Figure 7: An example game for Most Restricted First.

Interestingly, if all card supersets form a chain, determinization using Most Restricted First does uniformly generate a valid card distribution.

Lemma 2. *Let an undetermined game of The Crew with players* $P' = \{p_1, p_2, \ldots, p_n\}$ *such that*

$$C_{p_1} \subseteq C_{p_2} \subseteq C_{p_3} \subseteq \ldots \subseteq C_{p_n} \tag{23}$$

be given. Then determinization using Most Restricted First (Algorithm 2) generates any of the valid card distributions uniformly at random.

Proof. Note that any valid distribution can only be reached by assigning exactly the right hand cards on each iteration of step 1. Hence there is exactly one way to reach a given valid determinization. Also notice that the cards assigned to player $p_i \in P'$ do not affect the number of cards available to other players p_j with $j > i$ since $C_{p_i} \subseteq C_{p_j}$. Hence, the number of valid games that can be selected is independent of the hand selected for player p_i. Thus all hands available to player p_i should be equiprobable, which is exactly what the algorithm does by uniformly selecting $\#H_{p_i}$ cards from C_{p_i} in step 1b. □

5.2.5 Determinization using Uniform Random Selection

Lastly, one might attempt to distribute cards by randomly selecting a player and randomly assigning them a valid card. This would fail too. Take, for instance, the game in Figure 7 and consider the probability of North having ♡. This should be $\frac{2}{3}$. The possible constructions in which North would get ♡ with their probabilities are listed in Figure 8. We conclude the total probability of North getting ♡ using distribution by uniform random selection is $\frac{1}{6} + \frac{1}{12} + \frac{1}{9} + \frac{1}{8} + \frac{1}{12} = \frac{41}{72} \neq \frac{2}{3}$.

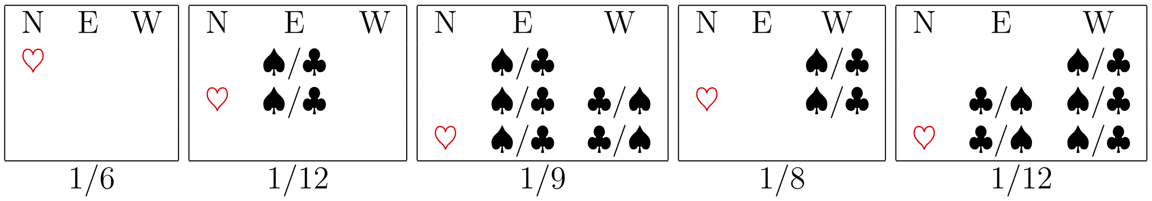

Figure 8: Possible ways of assigning ♡ to north with associated probabilities.

5.3 Results

To compare the different Monte Carlo players defined in Section 5.1, we generate 50 random 4–player games of Classic Crew with fixed objectives. Every game is played three times, one for each type of player. That is to say, first we try with four ignorant Monte Carlo players, then with four true Monte Carlo players and finally with four cheating Monte Carlo players. This is done both with a restriction on the number of playouts and with a restriction on the amount of computation time per playable card. We do not allow hinting in these games. The results can be seen in Figures 9 and 10.

It is important to mention that, as can be seen in Figure 3, not all games with fixed objectives are winnable. Hence we should expect our players to perform worse when there are more objectives. This is also what we observe.

Because of the amount of work required to generate a single determinization using probability (see Section 5.2.1), it is expected that, when restricted using time, the smart Monte Carlo player will be able to perform fewer playouts than both the ignorant and cheating Monte Carlo player. This is exactly what happens and could be the reason for the smart Monte Carlo player (on average) performing worse than both other Monte Carlo players (as seen in Figure 10). This hypothesis is however quickly rejected when restricting the players to playouts. As can be seen in Figure 9, on average, the smart Monte Carlo player performs worse than both other types of players. Note that the bound of 100 playouts was chosen for performance reasons. With the available hardware, performing more playouts would take unreasonably long.

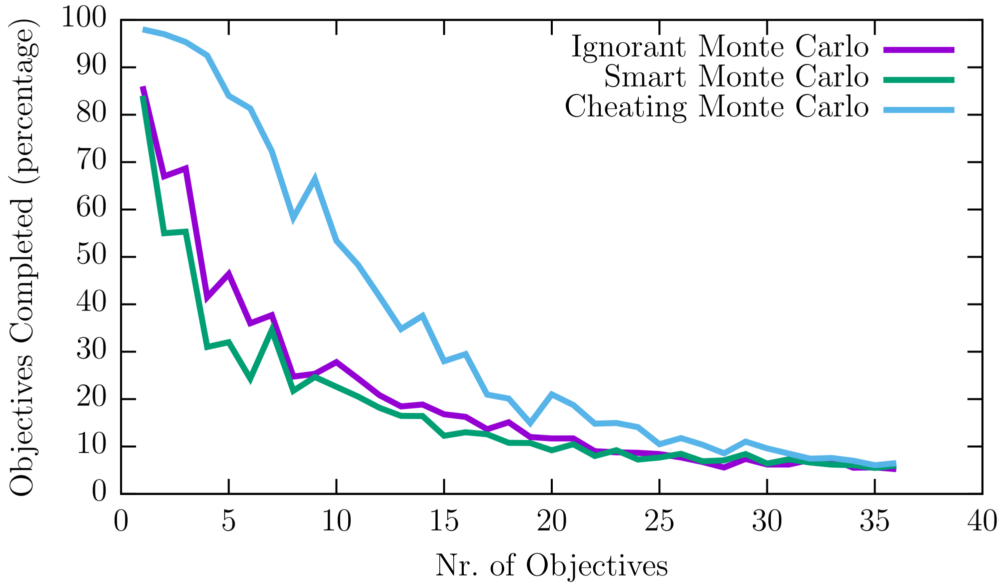

Figure 9: Percentage of objectives completed with 100 playouts per card

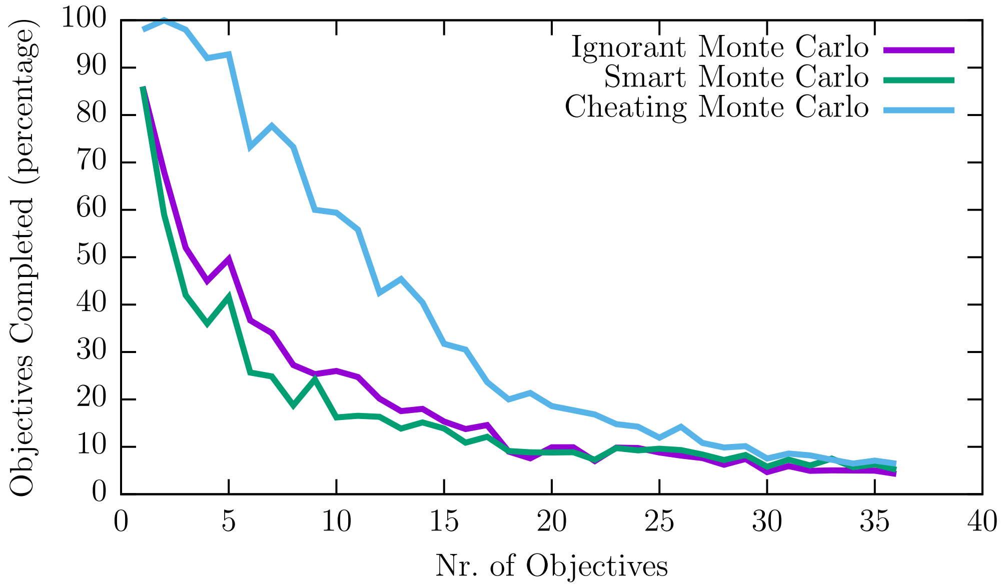

Figure 10: Percentage of objectives completed with .5 seconds thinking time per card

6 Conclusion and Further Research

We have analyzed The Crew, a cooperative trick taking game of imperfect information. We constructed a way of translating an instance of the winnability problem of The Crew into an instance of the satisfiability problem. A solution of the latter can then be translated back into a solution of the former. This allowed us to determine the percentage of games of The Crew that is winnable using brute-force, the results of which can be seen in Figure 3. From these results we conclude that the game The Crew with free objectives (as it was designed) is close to always winnable. This can be compared to a game like, for instance, Klondike Solitaire, where between 82% and 91.44% of the games is winnable [BTF07]. Hence, one could say that, in some way, The Crew is better designed than Klondike Solitaire.

We have constructed Algorithm 1 in order to generate a card distribution conforming to what we currently know about other players' hands uniformly at random (also known as determinization). This was then used to generate games for the random playouts of our smart Monte Carlo player, which we compared to a cheating and ignorant Monte Carlo player. The cheating Monte Carlo player used the current game state for random playouts and the ignorant Monte Carlo player distributes all remaining cards over the unknown hands for random playouts. The results of this comparison can be found in Figures 9 and 10. We concluded that, even though one might expect the smart Monte Carlo player to perform better than the ignorant Monte Carlo player, this is not the case. We do not know what the reason for this is. As expected, the cheating Monte Carlo player did perform best.

We would still like to compare the Monte Carlo player again but with hinting implemented. This could be done by having a random player select a future trick in which it will hint. Hinting a card can then be added as a possible move in between tricks. We expect adding hints to benefit the smart Monte Carlo player since it will then have more information and can hence generate more accurate determinizations.

We would still like to implement an agent using Monte Carlo Tree Search and compare it against the other agents.

We have not yet compared the Monte Carlo player to a human (inspired) player. We would like to formulate a human strategy (based on experience playing the game) and compare it against the Monte Carlo players. Such a strategy would consist of, for example, ranking the available objectives based on our hand cards and picking the "best" objective. A high card which we have in our own hand would be an easy objective while a low card from our own hand would be hard. Furthermore, we would try to eliminate the harder objectives so that others are not forced into taking objectives that would be even harder for them to complete. Knowing which objectives were picked first also gives insight into which objectives will be easily to complete, hence their suits can probably be played in the first few tricks.

References

[Ber15] Mark van den Bergh. Hanabi: A Co-operative Game of Fireworks. Bachelor's thesis, Leiden University, 2015.

[BTF07] Ronald Bjarnason, Prasad Tadepalli, and Alan Fern. Searching solitaire in real time. *ICGA Journal*, 30:131–142, 2007.

[FG10] Alan M. Frisch and Paul A. Giannaros. SAT Encodings of the At-most-k Constraint; Some Old, Some New, Some Fast, Some Slow. In *Proceedings of the Tenth International Workshop of Constraint Modelling and Reformulation*, page 36, 2010.

[FGMS06] Tomas Feder, Adam Guetz, Milena Mihail, and Amin Saberi. A Local Switch Markov Chain on Given Degree Graphs with Application in Connectivity of Peer-to-Peer Networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, page 69–76, 2006.

[Gin01] Matthew L. Ginsberg. GIB: Imperfect Information in a Computationally Challenging Game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

[Ham05] Harald Hammarström. Card-Shuffling Analysis with Markov Chains. `http://www.math.chalmers.se/~olleh/Markov_Hammarstrom.pdf`, 2005. Retrieved 2021-07-14.

[Jon] Aron de Jong. An analysis of The Crew GitHub. `https://github.com/Thyrum/an-analysis-of-the-crew`. Retrieved 2021-07-14.

[Kis] Kissat SAT Solver. `http://fmv.jku.at/kissat/`. Retrieved 2021-07-14.

[SHM+16] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529:484–489, 2016.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the Game of Go without Human Knowledge. *Nature*, 550:354–359, 2017.

[WPC11] Daniel Whitehouse, Edward Powley, and Peter Cowling. Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu. *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 87–94, 2011.