Performance of a multi-agent greedy algorithm in a cooperative game with imperfect information

Sam Chan

Faculty of Humanities
Utrecht University
Heidelberglaan 8
3584 CS Utrecht
2 July 2021

Bachelor Thesis Artificial Intelligence
7,5 ECTS
First Supervisor: Krisztina Szilagyi
Second Supervisor: Tomas Klos

Abstract

Artificial Intelligence (AI) is slowly integrating into our everyday lives. Some of these AI applications must cooperate with humans and each other. Such cooperation is essential for the safety of people in contact with these AIs. This proves to be difficult, due to the sheer amount of uncertainty in the real world. The virtual environments in games are a safe starting place for the research and development of AI capabilities such as cooperation in imperfect information environments. However, cooperative games with imperfect information are an uncommon topic for AI, which is why we have researched such a game called The Crew. We implemented the card game into a program and made an AI that can play it using different algorithms. We experimented with a random algorithm, a cooperative multi-agent greedy algorithm called CoopGreedy, and a competitive multi-agent greedy algorithm called CompGreedy. These algorithms also incorporated heuristics to improve their performance. Our results showed that CoopGreedy outperformed the other algorithms considerably. CoopGreedy in combination with the Expected Value Hand Estimation Heuristic and the Longest Suit Higher Card Goal Selection Heuristic also noticeably improved performance. We also found that CoopGreedy performed better with perfect information than with imperfect information. Our findings indicate that cooperation and heuristics improved performance significantly, but that imperfect information impedes AI performance. We can derive from our experiment that CoopGreedy could be used as a benchmark algorithm for further research. We recommend future research on local search and learning algorithms for The Crew.

Keywords: multi-agent system; greedy algorithm; cooperative game; imperfect information

Introduction

One of many qualities that humans possess is their ability to work together, even when they are surrounded by many uncertain factors. This can be illustrated by traffic, where multiple individuals cooperate with minimal communication in an imperfect information environment. Many applications of AI revolve around an AI's potential to replicate human capabilities. For instance, the self-driving car is such an application. The goal of researching self-driving cars is to make traffic more autonomous. To achieve this, a self-driving car needs to be able to cooperate with other cars. Of course, just like we cannot replace all fuel-driven cars with electric cars in an instant, we cannot immediately make all cars autonomous. We will have to find a way to safely incorporate self-driving cars into traffic that is not yet void of human-driven cars. For this reason, self-driving cars highlight the importance of cooperation between man and machine. However, the real world can sometimes be a too unpredictable starting ground for research, due to many uncontrollable variables. These variables can impede experiments. In contrast, the virtual environment in games is not accompanied by these uncontrollable variables. Therefore, games can be a useful starting place to research AI capabilities.

Since AI's first usage in games, it has expanded and improved to a considerable degree. Several games feature AI players on the same level as human players of average skill. These games range from turn-based tabletop games to fast-paced 3D video games. Moreover, AI players have even defeated professional chess players (Campbell, Hoane Jr, & Hsu, 2002).

Despite the importance of cooperation between humans and AI, most common games are competitive zero-sum games where players compete against each other to achieve their goals (Russell, S., & Norvig, P., 2002). In comparison to competitive games, the amount of research conducted on AI in cooperative games is limited.

Besides the competitive aspect, perfect information occurs frequently in games with AI (Russell, S., & Norvig, P., 2002). Such games have all information about the game state available to everyone. An example is the game of chess, where both players have all the necessary knowledge about the rules, the position of the pieces, etc. In Poker, players do not have all knowledge of the game state available to them. Players do not know what cards other players have up until the end of the game, hence Poker is an imperfect information game. Most AI applications in games involve perfect information because imperfect information games are difficult to optimize (Blair, J. R., Mutchler, D., & Liu, C., 1993). The major challenge is the large state space that accompanies imperfect information games (Ganzfried, S., & Sandholm, T., 2013). Machine learning is an effective tool to tackle this problem (Charlesworth, H. 2018). Combining machine learning and traditional search algorithms has also proven to be useful. The results of these techniques on imperfect information games are AIs that can perform on a

super-human level (Brown, N., Bakhtin, A., Lerer, A., & Gong, Q., 2020).

Prior research on cooperative imperfect information games has been done on the tabletop game Hanabi. However, the reinforcement learning algorithms used were unable to surpass hand-coded AIs in this game (Bard, N. et al, 2020). We chose to research another cooperative imperfect information game called The Crew. For our experiment, we created three algorithms that could play this game: a random algorithm and two greedy algorithms that use heuristics. An AI that uses our random algorithm bases its decision-making process entirely on chance. With our greedy algorithm, the AI always chooses the most obvious decision, which is the decision that maximizes the AI's expected utility.

The random algorithm is not an effective way for an AI to play this game. However, since no prior research has been done on The Crew, the random algorithm's performance does form a clear baseline for determining how effective algorithms such as our greedy algorithms are in this game. Greedy algorithms have been used in studies on games such as the real-time strategy game StarCraft (Churchill, D., & Buro, M., 2013) and other real-time adversarial games (Moraes, R., Mariño, J., & Lelis, L., 2018). In these studies, they used greedy algorithms in combination with other algorithms, due to the high complexity of such games. In the algorithms section, we will discuss in detail the algorithms and heuristics of our experiment.

Our research question is: How well does a multi-agent greedy algorithm perform in a cooperative game with imperfect information? We will answer this question by answering these sub-questions:

- How much do the heuristics influence the greedy algorithm performance?
- How does the greedy algorithm compare to a random algorithm?
- How does the greedy algorithm compare to a competitive variant of the same algorithm?
- How does the greedy algorithm perform with perfect information?

We will compare test runs with different parameters to answer these sub-questions.

Definitions

The Crew

The Crew: Quest for Planet Nine is a trick-taking card game that can be played with two to five players. The rules of a two-player game are different compared to a game with three or more players. Since the complexity increases with the player count, we decided to implement the game with three players.

The Crew is played with a deck of 40 playing cards and a deck of 36 goal cards. The playing card deck contains four regular suits and one trump suit: pink, yellow, green, blue, and rocket (trump suit). Every regular suit has nine cards with values 1 through 9. The trump suit has four cards with values 1 through 4. The goal card deck contains the same cards as the playing card deck except that it does not contain rocket cards.

At the start of every mission, the playing deck and the goal deck are shuffled. Then a predetermined number of goal cards are dealt face up. The number of goals varies per mission configuration. After that, the playing card deck is distributed over all the players. Player 1 gets 14 playing cards and the other players get 13 playing cards each. Players cannot see the hands of others. The player with the highest rocket card (rocket 4) is the commander. The commander always leads the first trick. A trick is a single round of play in which every player gets one turn to play a card. Every trick starts with the leader playing a card of their choice. Then each player gets a turn to play a card of the same suit as the leader's card, this is called following suit. Players always take turns in clockwise order. When a player cannot follow suit, he can play any card in their hand instead. After every player has played a card, the player with the highest card that followed suit 'takes the trick'. If a rocket is played, then the player that played the highest rocket takes the trick. The player that takes the trick takes all the cards played in the trick and is the leader of the next trick. Cards taken from a trick are out of the game for the current mission.

After the commander has been determined, players take turns to choose a goal card from the available face-up goal cards, starting with the commander. This repeats until all face-up goal cards have been distributed. The goal of the game is that every player completes all their goal cards. A player's goal card is completed when they take the playing card that corresponds with their goal card. However, when a player takes a card that matches someone else's goal card, that goal has been failed and thus cannot be completed. In the original version of The Crew, the mission is over when a goal has been failed. In our implementation, the mission ends when no more goals can be completed or after the trick in which a player plays their last card.

At the start of each trick, before a card is played, players get the opportunity to communicate one of the cards in their hand that is not a rocket card. This card is revealed to everyone. The communicating player also must reveal whether this card is the highest, lowest, or only card in its suit that they have in hand. If no such revelation can be made, that card cannot be communicated. Each player is only allowed to communicate once per mission. All other forms of communication are prohibited.

Imperfect Information

In a perfect information environment, our AIs would only need a knowledge base to store true information about the environment. More specifically, the AI would store in their knowledge base every card and the corresponding player that has it in their hand. However, since The Crew has an imperfect information environment, we also need the AIs to be able to work with cards without knowing which player has them in their hand. In The Crew, players can only see the cards in their hand and the cards that are revealed by play or communication. Therefore, algorithms that normally work

with perfect information games are not as effective in this game. Especially at the beginning of a mission, there is limited information available to the players. Due to this, players would have to make a lot of random decisions. Furthermore, the state space in The Crew is large due to imperfect information. We attempted to solve the limited information problem and the large state space problem with hand estimation. We will discuss hand estimation in more detail in the algorithms section.

Decision Tree/The Problem

We represented the decision-making process of the active player with a tree structure. The active player is the player whose turn it is. To better illustrate how this decision-making process was represented, we give the informal definition of the tree structure and its components here:

- A *tree* is a recursively defined data structure that consists of a root node and a list of subtrees.
- A *root node* is the topmost node of a tree.
- A *node* is an element of a tree that consists of an actor and an action.
- An *actor* is a player that performs the action of a node.
- An *action* corresponds to a card that the actor can play according to the active player.
- A *subtree* is in itself a tree with a root node and a list of subtrees.
- A *leaf node* is the root node of a tree without subtrees. This node marks the end of a branch.
- A *branch* encompasses all the nodes from the root node of a tree to a leaf node.
- The *main root node* is the topmost root node of the entire tree structure.

Algorithms

In this section, we will first discuss the heuristics that the algorithms use. Afterward, we will discuss the algorithms themselves.

Hand Estimation Heuristics

We explored two ways of estimating the hands of players: the Random Hand Estimation Heuristic (RHH) and the Expected Value Hand Estimation Heuristic (EVHH).

With the RHH, all the cards that a player (Player X) knows nothing certain about are compiled into a deck. Player X then randomly assigns these cards to the other players (Player Y and Z), such that each player has an equal number of assigned cards. These assigned cards are called assumptions. Player X then combines the knowledge he does have of the other players' hands with the assumptions. The result is an estimated hand for Player Y and an estimated hand for Player Z. Player X can then use these estimated hands to make decisions the same way he would in a perfect information environment.

Using the EVHH, all the cards that Player X knows nothing certain about are compiled into a deck as well. Instead of randomly assigning these cards to other players, the cards are assigned based on a heuristic. The heuristic determines the expected number of cards Player Y and Z have of each suit. Since the playing card deck is shuffled before dealing the hands, the probability of having a card in hand of a specific non-trump suit is 9/40. The probability of having a rocket card is 4/40 = 1/10. This means the expected number of cards in hand of a specific non-trump suit is approximately three cards (13 * 9/40). And the expected number of rocket cards in hand is approximately one card (13 * 1/10). Player Y and Z are then assigned the appropriate amount of cards for each suit. If Player Y or Z is the player that starts with 14 cards, then that player gets assigned an additional card. If it is known to Player X that another player has more cards of a suit than is expected, that player is assigned fewer cards of other suits, and vice versa.

The RHH was the most straightforward way to estimate hands but was also prone to high estimation errors. The estimation error is the difference between how many cards a player is assumed to have and how many cards they actually have. When estimating a player's hand with the RHH, the worst-case estimation error is nine cards. For instance, when a player is assumed to have no cards in a certain suit, but in reality, that player has nine cards in that suit. Then the difference between the assumption and reality is nine cards, which is the highest possible estimation error. Our motivation for implementing the EVHH was to reduce the worst-case estimation error of the RHH. The worst-case estimation error of the EVHH is six cards. For example, when Player X has no knowledge of Player Y's hand, Player Y is assumed to have the expected three cards per suit. Then when Player Y's hand actually contains nine cards in a suit, the difference between the assumption and reality is only six cards instead of nine. There is one exception: if Player Y has a hand of 14 cards, then the worst-case estimation error is seven cards instead of six.

Apart from reducing the estimation error, we implemented these hand estimation heuristics to decrease the state space of The Crew. When Player X assumes Player Y has a certain Card Y, then Player X automatically assumes Player Z does not have Card Y. Therefore, all the states wherein Player Z has Card Y are not taken into consideration when making a decision. This decreases the state space and consequently speeds up Player X's decision-making process.

Goal Selection Heuristics

At the start of a mission, players must base their selection of goals on their hands. We explored three heuristics for choosing goals: the Random Goal Selection Heuristic (RGH), the Matching Cards Goal Selection Heuristic (MCGH), and the Longest Suit Higher Card Goal Selection Heuristic (LHGH). The RGH is simple: pick a goal at random. The other strategies are based around the choosing player's (Player C) limited knowledge, which is their hand.

For the MCGH, Player C looks at the available goals and their hand. If an available goal matches one of their cards in hand, the player picks that card. If more than one goal

satisfies this condition, the goal with the highest value that satisfies the condition is picked. We chose to implement this heuristic because if Player C has the card in hand that corresponds to the chosen goal (Card C), it automatically follows that Player C knows who has Card C. Therefore, it feels intuitive for Player C to pick the goal for which they have the corresponding card in hand.

For the LHGH, Player C determines their longest suit by looking at their hand and count how many cards they have of each suit. The suit with the most cards is Player C's longest suit. Then Player C looks at the available goals that have the same suit as Player C's longest suit. Player C will not select a goal that corresponds to one of their cards in hand. Also, Player C will only select a goal if they have a card in hand that is higher than that goal and is of the same suit. If more than one goal satisfies these conditions, Player C chooses one of these goals at random. If none of the goals satisfy this condition, the process is repeated with the next longest suit if possible. Otherwise, a goal is picked at random. We implemented this heuristic because it prioritizes goals that are easiest to complete for Player C.

CoopGreedy

The main algorithm of our experiment is CoopGreedy. It can use the heuristics we described to make decisions in the goal selection phase. This algorithm uses hand estimation to cope with imperfect information. The goal of CoopGreedy is to maximize goal completion.

In a trick, the active CoopGreedy player (Player A) utilizes the cards that have been played in the current trick to better determine which card to play. Of course, if Player A is the leader of the trick, no cards have been played in the trick yet. Consequently, the leader and the second player of the trick must estimate the actions of other actors. Only the last player of the trick determines which card to play without doing these estimations.

In the tree structure of Player A, each branch from the main root node to a leaf node represents a possible scenario of how the trick can play out according to Player A's knowledge and assumptions. Player A calculates the expected utility for each possible scenario. The utilities of scenarios in which Player A plays the same card are summed to form one value. Finally, Player A takes their turn by playing the card that has the highest summed expected utility. The utility of a scenario is calculated with the following formula:

$$c * r - f * p + \textit{Utility of Play}$$

c = Number of goals that will be completed in this scenario.
r = Reward per goal completed.
f = Number of goals that will be failed in this scenario.
p = Penalty per goal failed.g = Number of goals Player A will have left after this scenario plays out.
h = Number of cards still in the hands of other players that are higher than Player A's play in this scenario.
l = Number of cards still in the hands of other players that are lower than Player A's play in this scenario.

Play Points is the predetermined maximum utility of a play. *Utility of Play* represents how good of a choice it is for Player A to play this card in this scenario. The calculation of *Utility of Play* depends on two conditions: whether $c > 0$ and whether $g > 0$. Table 1 shows how *Utility of Play* is calculated.

Table 1: Utility of Play formulas.

$c > 0$	$g > 0$	formula
true	true	*Play Points* / (l + 1)
true	false	*Play Points* / (h + 1)
false	true	*Play Points* / (h + 1) * -1
false	false	*Play Points* / (l + 1) * -1

CompGreedy

CompGreedy is the competitive variant of the CoopGreedy algorithm. Like its cooperative counterpart, CompGreedy can use the same heuristics to select goals and uses hand estimation to handle imperfect information. The difference between these two versions of the greedy algorithm is that CompGreedy always strives to take a trick, whereas CoopGreedy only tries to take tricks that must be won by them. CompGreedy uses a formula for calculating the utility of a scenario similar to CoopGreedy:

$$c' * r - f' * p + \textit{Utility of Play}$$

c' = Number of goals of Player A that will be completed in this scenario.
f' = Number of goals of Player A that will be failed in this scenario.
Utility of Play is calculated the same way as with CoopGreedy, except that $c > 0$ is replaced with $c' > 0$.

Method

The experiment was conducted on a Lenovo Ideapad 320 laptop. We implemented The Crew in C# with Visual Studio 2019. The input variables we used for each algorithm and heuristic are listed in Table 2 and Table 3, respectively. For all our tests, we had three AI players who used the same algorithms and heuristics. The performance of every algorithm and heuristic was tested by running 1000 missions for each goals-per-mission configuration. There were ten configurations, each with a different number of goals per mission ranging from one to ten. So the first configuration had one goal per mission, the second configuration had two goals per mission, and so forth.

The number of goals per mission is the independent variable of our experiment. The Mission Completion Rate and the Goal Completion Rate are the dependent variables. The Mission Completion Rate was calculated by dividing the number of missions completed by the total number of missions. A mission was considered complete when all goals have been fulfilled. The Mission Completion Rate better reflected the official performance measure of The Crew. However, it only provided a binary view of the performance

of the algorithms and their heuristics. The Goal Completion Rate was a more detailed assessment of this performance. We calculated the Goal Completion Rate by dividing the average number of goals completed by the number of goals per mission. We used the LHGH for the algorithm tests because our experimental results indicated that this heuristic performs best in comparison to other goal heuristics. The same holds for our choice for the EVHH. After some experimenting, we have found that using an equal value for Goal Reward and Goal Penalty gave us the most consistent results. Initially, we set the value of these scoring parameters to 100. However, later we found that the performance was best with the Play Points scoring parameter set to a value 1/10 of the Goal Reward or Goal Penalty values. When we set Play Points to 10, inaccurate rounding formed a problem. We fixed this by setting the Goal Reward and Penalty values to 1000 and the Play Points to 100. All algorithms used the same heuristics and scoring parameters, but the random algorithm's performance was not influenced by these heuristics and parameters in contrast to other algorithms.

Table 2: The input variables used per algorithm.

	Random	Imperfect CoopGreedy	CompGreedy	Perfect CoopGreedy
Perfect Information	false	false	false	true
Goals Per Mission	varies between 1 and 10	varies between 1 and 10	varies between 1 and 10	varies between 1 and 10
Max Missions	1000	1000	1000	1000
Trick Algorithm	random	coopGreedy	compGreedy	coopGreedy
Goal Heuristic	LHGH	LHGH	LHGH	LHGH
Uses EVHH	true	true	true	true
Goal Reward	1000	1000	1000	1000
Goal Penalty	1000	1000	1000	1000
Play Points	100	100	100	100

Table 3: The input variables used per heuristic combination.

	EVHH & LHGH	RHH & LHGH	EVHH & RGH	EVHH & MCGH
Perfect Information	false	false	false	false
Goals Per Mission	varies between 1 and 10	varies between 1 and 10	varies between 1 and 10	varies between 1 and 10
Max Missions	1000	1000	1000	1000
Trick Algorithm	coopGreedy	coopGreedy	coopGreedy	coopGreedy
Goal Heuristic	LHGH	LHGH	RGH	MCGH
Uses EVHH	true	false	true	true
Goal Reward	1000	1000	1000	1000
Goal Penalty	1000	1000	1000	1000
Play Points	100	100	100	100

Results

Heuristics

The Mission Completion Rate of CoopGreedy with different setups of heuristics is shown in Figure 1. Figure 2 shows the Goal Completion Rate of CoopGreedy with different configurations of hand and goal heuristics. The difference between mission completion and goal completion was most apparent when there were ten goals per mission. Despite CoopGreedy with EVHH and MCGH completing almost no missions, it did complete 65% of the goals. We see that the EVHH consistently performed better than the RHH. The MCGH performed worse than CoopGreedy with the RGH in all cases. This is most noticeable when the amount of goals per mission is high. The LHGH did improve the performance of CoopGreedy significantly when compared to RGH, especially with three or more goals per mission.

Algorithms

In Figures 3 and 4 we compare for each goals-per-mission configuration the performance of the random algorithm, CoopGreedy, and CompGreedy. We can see that the performance of CoopGreedy was substantially better than the other algorithms. With ten goals per mission, CoopGreedy completed 115.7% more goals than Random and 94.4% more than CompGreedy. We can also see that the performance of CompGreedy was better than Random. With ten goals per mission, CompGreedy completed 11% more goals than Random. The performance of Random and CompGreedy from one goal to ten goals per mission decreases exponentially. In terms of mission completion, there are three outliers in the results of Random and CompGreedy. Random completed one mission with six goals per mission and one with eight goals per mission. CompGreedy did the same with eight goals per mission. Apart from these outliers, Random was unable to complete missions when the goals per mission rose above five goals. For CompGreedy, this was the case when the goals per mission were higher than six goals.

Imperfect Information and Perfect Information

In Figures 5 and 6 we do the same comparisons as before, except here we compare the performance of CoopGreedy with imperfect information to CoopGreedy with perfect information. In all cases, CoopGreedy with imperfect information performed worse than CoopGreedy with perfect information. Imperfect information CoopGreedy with ten goals per mission completed 4.2% fewer goals than perfect information CoopGreedy.

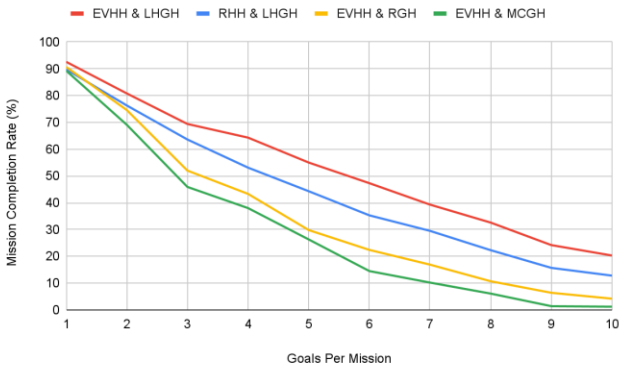

Figure 1: Mission Completion Rate comparison of the different heuristic combinations for CoopGreedy.

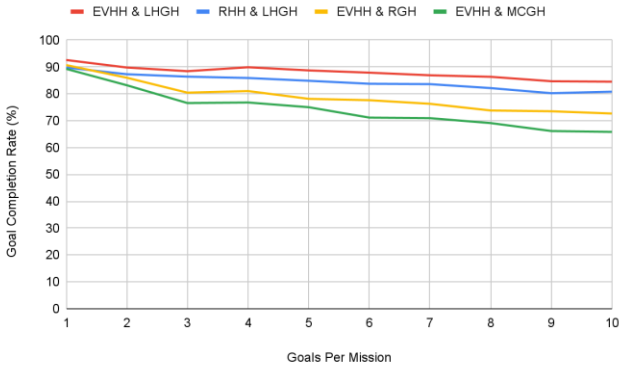

Figure 2: Goal Completion Rate comparison of the different heuristic combinations for CoopGreedy.

Figure 3: Mission Completion Rate comparison of Random, CoopGreedy, and CompGreedy, for each goals-per-mission configuration.

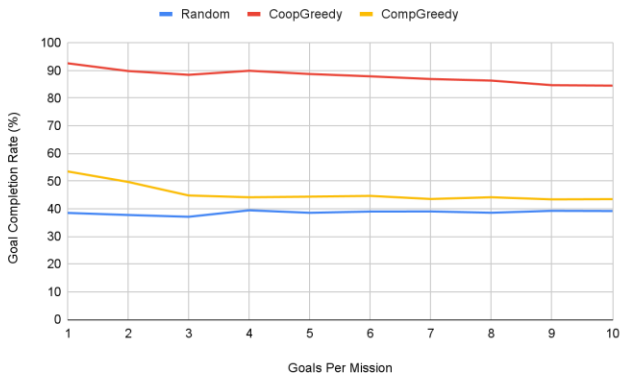

Figure 4: Goal Completion Rate comparison of Random, CoopGreedy, and CompGreedy, for each goals-per-mission configuration.

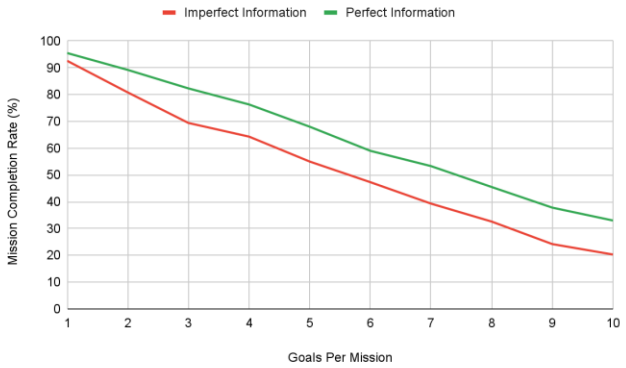

Figure 5: Mission Completion Rate comparison of CoopGreedy with imperfect information and CoopGreedy with perfect information.

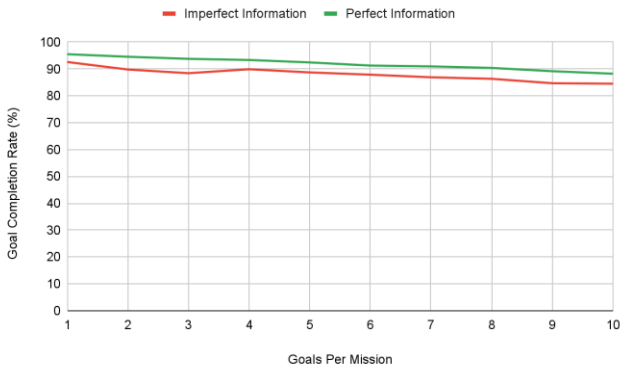

Figure 6: Goal Completion Rate comparison of CoopGreedy with imperfect information and CoopGreedy with perfect information.

Discussion

Our results show that both the Mission Completion Rate and the Goal Completion Rate decrease when the number of goals per mission increases. This means that the complexity of a mission is dependent on the number of goals. This negative relation between completion rate and goals per mission is most noticeable with the Mission Completion Rate because failing a goal automatically renders a mission impossible to complete. Consequently, the number of missions completed drops faster than the number of goals completed.

As anticipated, the EVHH performed better than the RHH. This can be explained by the fact that the worst-case estimation error of the EVHH is smaller than that of the RHH. The goal selection heuristics were based on decisions that were intuitive to us. Contrary to what we initially expected, the MCGH performed worse in all cases compared to randomly selecting goals with the RGH. The problem of the MCGH is that if the card that corresponds to the goal (Card X) selected by a player (Player X) is Player X's only card in that suit, the other players will have more cards of Card X's suit than expected. This increases the probability of Player X being forced to play Card X. The conditions for this problem to occur are easily met and could be the reason why MCGH performed worse than RGH. The LHGH performed as expected, surpassing all other heuristics in terms of absolute performance gain. The most likely reason for this performance gain is two properties of the LHGH. The first property is that the LHGH always prioritizes goals that allow the choosing player to complete the goal in the same suit as the goal. The second property is that the LHGH favors goals that are of the longest suit of the choosing player.

The exponential decay of Random and CompGreedy's performance as the number of goals increases suggests that the complexity of a mission increases exponentially with each goal added. The stark difference between the performance of CoopGreedy and CompGreedy exemplifies the importance of cooperation between players in The Crew. From our results, we can derive that CoopGreedy cannot solve The Crew, even with perfect information. This suggests that an optimal algorithm might require more than a utility function that maximizes goal completion. It also suggests that an algorithm must look further into the game than the current trick.

Conclusion

Our experiment showed that a well-chosen heuristic such as LHGH did improve CoopGreedy significantly. However, a poorly chosen heuristic like MCGH could do the contrary. The hand estimation heuristics seemed to influence the performance of CoopGreedy. Reducing the worst-case estimation error of a hand estimation resulted in improved performance. However, without an algorithm that does not use hand estimation, we cannot speak of the influence of hand estimation as a whole on performance. Further research is required to answer this. CoopGreedy performed substantially better than both Random and CompGreedy. Since neither Random nor CompGreedy actively encouraged cooperation like CoopGreedy, this result showed that cooperation is an essential aspect of The Crew. Comparing imperfect information to perfect information revealed that even with perfect information, CoopGreedy was not optimal. Despite this, CoopGreedy with perfect information did show an improvement of performance compared to CoopGreedy with imperfect information. All in all, our experiment showed that

CoopGreedy is a decent replacement for Random as a benchmark algorithm for further research on The Crew.

For future work on The Crew, it would be interesting to see the performance of an algorithm that looks further than the current trick to make its decision. Local search algorithms such as Hill Climbing or Simulated Annealing could be effective approaches for this. We also recommend looking into the Monte Carlo Tree Search algorithm. It has been used for other trick-taking games in the past and proved to be an effective method for improving the performance of AI in such games. Machine learning techniques applied on The Crew are also interesting topics for future research. For example, an AI that uses reinforcement learning could learn to play The Crew from experience. Since there are no training data sets available for The Crew, especially reinforcement learning could be a viable machine learning approach.

References

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., ... & Bowling, M. (2020). The Hanabi challenge: A new frontier for ai research. Artificial Intelligence, 280, 103216.

Bax, F. (2020). Determinization with Monte Carlo Tree Search for the card game Hearts (Bachelor's thesis).

Blair, J. R., Mutchler, D., & Liu, C. (1993). Games with imperfect information. In Proceedings of the AAAI Fall Symposium on Games: Planning and Learning, AAAI Press Technical Report FS93-02, Menlo Park CA (pp. 59-67).

Brown, N., Bakhtin, A., Lerer, A., & Gong, Q. (2020). Combining deep reinforcement learning and search for imperfect-information games. arXiv preprint arXiv:2007.13544.

Campbell, M., Hoane Jr, A. J., & Hsu, F. H. (2002). Deep blue. Artificial intelligence, 134(1-2), 57-83.

Charlesworth, H. (2018). Application of self-play reinforcement learning to a four-player game of imperfect information. arXiv preprint arXiv:1808.10442.

Churchill, D., & Buro, M. (2013, August). Portfolio greedy search and simulation for large-scale combat in StarCraft. In 2013 IEEE Conference on Computational Intelligence in Games (CIG) (pp. 1-8). IEEE.

Ganzfried, S., & Sandholm, T. (2013, June). Improving performance in imperfect-information games with large state and action spaces by solving endgames. In Workshops at the twenty-seventh AAAI conference on artificial intelligence.

Moraes, R., Mariño, J., & Lelis, L. (2018, September). Nested-greedy search for adversarial real-time games. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 14, No. 1).

Mutchler, D., & van Lent, M. (1993). A pruning algorithm for imperfect information games. Technical report, Dept. of Computer Science, University of Tennessee.

Osawa, H. (2015, April). Solving Hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information. In Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.

Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

Sturtevant, N. (2002, July). A comparison of algorithms for multi-player games. In International Conference on Computers and Games (pp. 108-122). Springer, Berlin, Heidelberg.