



Czech
Technical
University
in Prague

A Heuristic Algorithm for Phantom Go

Petr Syrovátka

Supervisor: Ing. Michal Šustr
May 2022

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Syrovátka** Jméno: **Petr** Osobní číslo: **492281**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Otevřená informatika**
Specializace: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Heuristický algoritmus pro Fantomové Go

Název bakalářské práce anglicky:

A heuristic algorithm for Phantom Go

Pokyny pro vypracování:

1. Make a literature overview of algorithms for solving large games of imperfect information, with the emphasis on Phantom Go.
2. Implement the game and reimplement a state-of-the art algorithm as a baseline in OpenSpiel.
3. Implement a new algorithm, based on sampling compatible boards and evaluating them with a Go engine as a heuristic for the value function.
4. Evaluate the algorithms in pair-wise matches to compute a statistically significant result.

Seznam doporučené literatury:

Saito, Jahn-takeshi, Guillaume Chaslot, and Jos WHM Uiterwijk Joris Borsboom. "A comparison of monte-carlo methods for phantom go." In Proc. 19th Belgian-Dutch Conference on Artificial Intelligence-BNAIC. 2007.
Wang, Jiao, Tan Zhu, Hongye Li, Chu-Husan Hsueh, and I-Chen Wu. "Belief-state Monte Carlo tree search for phantom go." IEEE Transactions on Games 10, no. 2 (2017): 139-154.
Fei, Li, Ouyang Li, Wang Yajie, and Dong Yanqiu. "Study of strategy selection based on Phantom Go." In The 26th Chinese Control and Decision Conference (2014 CCDC), pp. 3460-3462. IEEE, 2014.
Cowling, Peter I., Edward J. Powley, and Daniel Whitehouse. "Information set monte carlo tree search." IEEE Transactions on Computational Intelligence and AI in Games 4, no. 2 (2012): 120-143.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Michal Šustr katedra počítačů FEL

Jméno a pracoviště druhého(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2022**

Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Michal Šustr
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank the supervisor Ing. Michal Šustr for the time he gave me, and the advice that made it easier for me to write my thesis.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 20, 2022

Abstract

This work will describe the implementation of Phantom Go and algorithms to sample compatible histories and will show and discuss the results of pairwise evaluation of different algorithms for playing Phantom Go and the efficiency of IS-MCTS.

Keywords: Phantom Go, IS-MCTS, Imperfect-information games

Supervisor: Ing. Michal Šustr

Abstrakt

V této práci bude popsána implementace hry Phantom Go a algoritmů pro vytváření kompatibilních historií, a budou popsány a prodiskutovány výsledky párového vyhodnocení různých algoritmů pro hraní hry Phantom Go a efektivita IS-MCTS.

Klíčová slova: Phantomové Go, IS-MCTS, Hry s neúplnou informací

Contents

1 Introduction	1
1.1 Motivation	1
2 Background	3
2.1 Game formalism	3
2.2 Literature	4
2.2.1 A Comparison of Monte-Carlo Methods for Phantom Go	4
2.2.2 Belief-state Monte Carlo tree search for phantom go	5
2.2.3 Study of strategy selection based on Phantom Go	6
3 Go and Phantom Go	9
3.1 Go	9
3.2 Phantom Go	10
3.3 Implementation	10
4 Search Algorithms	13
4.1 Monte Carlo evaluation	13
4.2 Monte Carlo tree search	13
4.3 UCT	14
5 State sampling	15
5.1 Information State	15
5.2 Metaposition	16
6 Using Monte Carlo Tree Search on Phantom Go	19
6.1 Information State MCTS	19
6.2 ISMCTS against Random player	19
6.3 MaxValue or MaxVisitCount policy	19
6.4 Different efficiency with different number of simulations	20
6.5 Go engine as a heuristic	20
6.6 Go engine evaluation against Random Rollout evaluation	21
7 Conclusion	23
7.1 Future work	23
Bibliography	25

Figures

Tables

2.1 Table with game statistics of different algorithms [BSCU08]	5
2.2 Example of a game with Specify Gambit [FLYY14]	7
3.1 Example of territories on board . .	9
3.2 Example of Phantom Go game with player's observations [BSCU08]	10

Chapter 1

Introduction

In this work, I am going to define the game Phantom Go, describe how the game differs from its variant Go, and describe algorithms widely used for games with imperfect information, which are games where players do not have full observation of the state of the game, especially Phantom go, and I will use these algorithms on Phantom Go and discuss their effectiveness.

Phantom Go is an interesting domain to study because of its large belief space, but it is very easy to sample compatible histories for information/public states.

The goals of this project are to implement the game Phantom Go into the OpenSpiel framework[LLL⁺19], implement sampling method using OR-Tools, implement an evaluation algorithm using a Go engine and to evaluate this algorithm against already implemented algorithms in OpenSpiel and all of the algorithms against a player playing random actions. Lastly, the efficiency of IS-MCTS will be discussed according to the results of evaluations.

In Chapter 2 I will describe the game Phantom Go using a game formalism, describe algorithms usually used on games of similar complexity and summarize the literature about algorithms for solving large games of imperfect information, with the emphasis on Phantom Go. Chapter 3 will cover Go, and its imperfect information version Phantom Go as a board game and describe its implementation. Chapter 4 describes the algorithms used in perfect information games. Chapter 5 provides an overview of Metapositions and Infostates, state set quantifiers, and an approach to sampling. Chapter 6 gives an outline of the usage of previously described algorithms, their efficiency, and a discussion about the results of the experiment. In the last Chapter 7 I will make an overview of the whole project and achieved goals.

1.1 Motivation

Subgame solving is the standard technique for playing perfect-information games that has been used by strong agents in a wide variety of games including go. Methods for subgame solving in perfect-information games exploit the fact that a solution to a subgame can be computed independently of the rest of the game. However this approach fails in imperfect-information games, where the optimal strategy in a subgame can depend on strategies outside

that subgame. [ZS21]

Several problems have been identified with techniques based on determinization such as Perfect Information Monte Carlo, , when used on imperfect-information games. Two commonly reported problems are strategy fusion and non-locality. Strategy fusion occurs when distinct actions are recommended in states that the player is not able to distinguish due to the imperfect information. Non-locality occurs due to optimal payoffs not being recursively defined over subgames as in perfect information games. As a result, guarantees normally provided by search algorithms built on subgame decomposition no longer hold. [LLB15]

Recently, subgame solving techniques have been extended to imperfect-information games. Some of those techniques are provably safe in the sense that, under reasonable conditions, incorporating them into an agent cannot make the agent more exploitable. However, all of the prior techniques have a shared weakness that limits their applicability: as a first step, they enumerate the entire common-knowledge closure of the player's current infoset, which is the smallest set of states within which it is common knowledge that the current node lies. This is a problem in imperfect-information games like Phantom Go, because the set is unmanageably large. [ZS21]

Because of the size, an approach with Infostate sampling, covered in chapter 5, using Infostate Monte Carlo 6, has been chosen. It is shown that this heuristic approach does not converge to an ideal solution, because of the mentioned problems of fusion and non-locality. However it has been shown, that this approach can find in short time a solutions, that can play well. [LLB15]

As one of the experiments for this work, I will implement a method for sampling states the same Infostate for Phantom Go and evaluate the Infostate Monte Carlo Tree Search against a random player.

Chapter 2

Background

To begin, I would like to describe the whole problem fully. Game Phantom Go, which I will describe in chapter 3, is a deterministic game with imperfect information. The term Imperfect information suggests that, unlike in a perfect information game, the game's current state is only partially observable. A game is said to be deterministic, if in a state taking an action always leads to the same next state.

2.1 Game formalism

Phantom Go can be modeled as a factored-observation stochastic game which is defined by following tuple:

- Set on finite players $N = \{1, 2\}$.
- Set of world states W and an initial state w^{init} .
- Player function $p : W \rightarrow 2^N$.
- Space of joint actions A . This game is a sequential, which means that player take turns. One player plays an action, while the other waits.
- Transition function $T : W \times A \rightarrow \Delta W$. This function defines the stochasticity of this game, which means that by applying an action A on a state W , we get probability distribution ΔW of into which states we will get
- Reward function $R : W \times A \rightarrow \mathbb{R}^N$. This zero-sum function which value describes the result of the game at a terminal state, by assigning 1 to the winning player and -1 to the player that lost, or 0 to both players in case of a tie.
- Observation function $O : W \times A \times W \rightarrow \mathbb{O}$. This function returns an observation after transitioning from a state by applying and action into the following state.

[KSB⁺21]

2.2 Literature

2.2.1 A Comparison of Monte-Carlo Methods for Phantom Go

[BSCU08]

Monte-Carlo methods for Phantom Go

In the following sections, first I will describe Monte-Carlo evaluation, which is a evaluation function for game-tree search, and then a way of applying UCT, a move selection algorithm, to Phantom Go with usage of two heuristics for guessing the locations of opponent stones. These heuristics are important, because to be able to apply UCT to the move selection task, the stones have to be placed for more precise results.

Monte-Carlo evaluation for Phantom Go

In this article, a two-step approach was chosen. The first step consisted of placing unknown stones randomly onto the board according to the rules. In the second step, Monte-Carlo evaluation using one of the following two sampling techniques is used to find the best move.

In order to decide which game state S_i of a set of game states has the best Monte-Carlo evaluation, a number of Monte-Carlo evaluations are played out. A crucial distinction for computing the best evaluation refers to two sampling techniques: standard sampling and all-as-first sampling can be distinguished.[BSCU08]

In all-as-first sampling each sample game contributes to the evaluation of multiple game states S_i simultaneously. Each move random sample which results in a victory for the player to move first contributes to a statistic maintained for each of the S_i occurring in the random sample. The other used approach to sampling is Standard sampling, in which A number of Monte-Carlo evaluations are made for each S_i . The best S_i is the game state with the best evaluation. In order to decide which game state is the best, multiple samples are made succinctly.[BSCU08]

Monte-Carlo Tree Search for Phantom Go

To be able to use UCT to select a move in Phantom Go, a heuristic for placing the unknown opponent stones is required. The two used move guessing heuristics are (1) late random opponent-move guessing, and (2) early probabilistic opponent-move guessing. Both of these heuristics are used in leaf nodes of the game tree, which action were not all explored. *Late random opponent-move guessing* places unknown opponents stones randomly after a leaf node is reached during the exploration of the tree, whereas *Early probabilistic opponent-move guessing* places the unknown stones right after

the first own move is played. To better understand the exploration of the game tree using MCTS, see Subsection 4.2.

■ Experiment and Results

An experiment using the four Monte-Carlo based methods described above was conducted. The exact methods are : (1) Monte-Carlo evaluation with standard sampling (MCE_{std}), (2) MonteCarlo evaluation with all-as-first sampling (MCE_{all}), (3) UCT with random late opponent-move guessing (UCT_{rand}), and (4) UCT with early probabilistic opponent-move guessing (UCT_{prob}). Each method played 50 games against each other, playing half of the games as Black and the other half as White.[BSCU08]

The following table 2.1 contains the comparison of results of the methods. The rows show the number of victories the player in the leftmost column scored against the players in the other columns. The results of the experiment allow

	MCE_{std}	MCE_{all}	UCT_{rand}	UCT_{prob}	total wins
MCE_{std}	×	14	4	14	32
MCE_{all}	36	×	28	44	108
UCT_{rand}	46	22	×	42	110
UCT_{prob}	36	6	8	×	50

Figure 2.1: Table with game statistics of different algorithms [BSCU08]

the four following conclusions: (1) All-as-first sampling is a better choice for Monte-Carlo evaluation than standard sampling. (2) Late random opponent-move guessing is a better choice for UCT than early probabilistic opponent-move guessing. (3) The best player based on UCT does not outperform the best player based on Monte-Carlo evaluation but plays similarly strong. (4) Monte-Carlo evaluation with standard sampling yielded the weakest player. [BSCU08]

■ 2.2.2 Belief-state Monte Carlo tree search for phantom go

[WZL⁺18]

In this article, a general search framework named beliefstate Monte Carlo tree search (BS-MCTS) is put forward. BS-MCTS incorporates belief-states into Monte Carlo Tree Search, where belief-state is a notation derived from philosophy to represent the probability that speculation is in accordance with reality. In BS-MCTS, a belief-state tree, in which each node is a belief-state, is constructed and search proceeds in accordance with beliefs. Then, Opponent Guessing and Opponent Predicting are proposed to illuminate the learning mechanism of beliefs with heuristic information. The beliefs are learned by heuristic information during search by specific methods, and we propose Opponent Guessing and Opponent Predicting to illuminate the learning mechanism. [WZL⁺18]

■ Belief-State

The term of belief is derived from philosophy and used in game theory especially imperfect information games to represent the probability that speculation is in accordance with reality associated with various aspects, such as the opponent strategy, a possible board configuration or the most likely response of the opponent. [WZL⁺18]

■ Belief-State MCTS

The search method named as BS-MCTS is proposed to deal with imperfect information games with large state-space. During the evaluation, a Belief-State Tree is being built, which is a directed graph with nodes being represented by Belief-States and edges by actions. The whole tree is then being explored using MCTS 4.2.

■ Computation of Beliefs

In the former section, we assume that the players always have the belief that all the physical states within each beliefstate share the uniform distributions. It is also assumed that the opponent tends to select the action with the maximum expected reward according to the belief-state from player's point of view. For simplicity, pure strategy can be applied to evaluate beliefs that assumes they are equally likely, or otherwise mixed strategy is applied instead by using opponent modeling. Obviously, beliefs should be updated with the progress of the game as more information is revealed or can be deduced. Consequently, opponent modeling methods have been suggested to determine the beliefs of opponents in Phantom games. Deviation-based best response (DBBR) learns beliefs from previous actions and played games of opponents, and has obtained good performance in Texas Hold'em poker. [WZL⁺18]

■ Summary

This paper proposes a novel algorithm, BS-MCTS, to deal with Phantom Go, a challenging imperfect information game. In the algorithm, we incorporate belief-state in imperfect information game. By considering belief-states, we can make full use of opponent modeling to guess and predict the opponent's strategy. Therefore, the tree with belief-state nodes can be constructed and an MCTS-based method is applied in searching. [WZL⁺18]

■ 2.2.3 Study of strategy selection based on Phantom Go

[FLYY14] In this article, in addition to usage of Monte Carlo algorithms and UCT, usage of Opening Gambits, which are predefined series of moves used in Go, that help mainly in the beginning stages of the game, when the discoverable game tree is too large to exploit the efficiency of Monte Carlo and UCT.

■ Specify Gambit

As described in part of the article describing advantages and disadvantages of using Gambits in the beginning of a Phantom Go game, another advantage pointed out, is that the observational moves are played less during the early phases of the game, which means that a large portion of opponent's stones is undiscovered, which makes the mentioned algorithms even less effective.

Gambits are used in games of Go to capture favorable points on the board. This continues until a stone of opponent is detected. At this time, UCT is called in game.

■ Experiment

As an experiment, 10 games of Phantom Go were played against plain Monte Carlo. 9 of these games were won by player with Specify Gambit, and 4 of the wins the Gambit played an important role. In the summation of the experiment, the four mentioned game's beginnings were analyzed.

From the picture 2.2 we can see that the White stones have a scattered distribution. It is difficult for the White to form a situation, while the Black has occupied the important terrain which is easy to connect his stones and segment battlefield and to be in a good opening condition. [FLYY14]

■ Summary

Different strategies were used in different stages of the game, according to the characteristics of phantom go. Each strategy maximizes its advantages and avoids the defect. It is limited that only changing strategy to improve the ability of a game, however, the degree of strategy distribution and intelligent algorithm are more important for a game programming. [FLYY14]

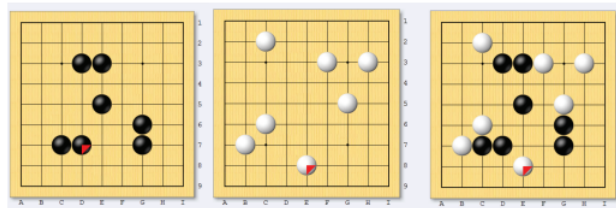


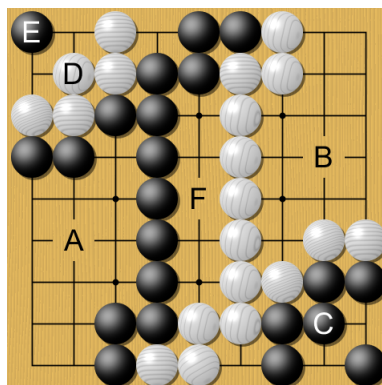
Figure 2.2: Example of a game with Specify Gambit [FLYY14]

Chapter 3

Go and Phantom Go

3.1 Go

Go is a Chinese strategic board game played by two players. They aim to gain more points than the opponent by surrounding territories with their stones. Players take turns in placing the stones on the board. The usual board size is 19x19, but smaller boards of sizes 13x13 or even 9x9 are used for quicker games or beginners. The stones cannot be moved once they are placed, but they can be removed by an opponent by occupying all orthogonally adjacent points of a group of stones. Then the stones are considered captured and removed from the board. One of the fundamental terms for Go is an Eye. A point on board is considered an Eye if it is an empty point inside a group of same color stones that acts as an adjacent point to the mentioned group, preventing the enemy player from capturing the group. Figure 3.1 shows an example of a game on a board of size 9x9. Black's territories are A and C, and a prisoners group D. Prisoner groups are those groups that could not defend their territory, meaning that the opponent would surely capture them. White has only one territory B. The territory F is dame, meaning it belongs to neither player.



[Scs11]

Figure 3.1: Example of territories on board

3.2 Phantom Go

Phantom Go is a version of Go with imperfect information. Unlike Go, Phantom go is played on three boards, one for each player and the third one for the judge. The players see only their boards, unlike the judge, who can see all the three boards. The judge's board contains all the stones played, and the player's board contains their stones and the opponent's stones they have observed in their turns. Players take turns as in Go, but instead of directly placing the stones, they point at a location on their board, suggesting their move. The judge then decides if the move is legal and proceeds to place a stone on the desired point or tells the player that the move was illegal but does not comment on why. The first reason for this can be that an opponent's stone occupies the point. In this case, the player observed the opponent's stone, and the judge adds the stone to the player's board. The other reason is that the move would be against the rules, such as a move inside the opponent's Eye. In both scenarios of an illegal move, the player is still on the move. One of the scenarios that can happen during a legal move is a capture of enemy stones. In Phantom Go, the judge points out to both players which stones were captured.

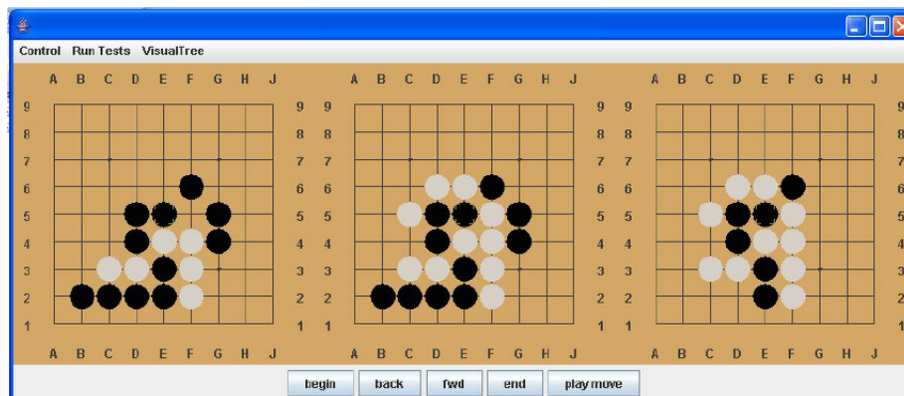


Figure 3.2: Example of Phantom Go game with player's observations [BSCU08]

In Figure 3.2 is shown an example of a Phantom Go game, with the judge board in the middle, white player's board on the right and black player's board on the left.

3.3 Implementation

As in the real board game, three objects represent the three boards. The boards are represented as an array of Colors, which can be black, white or empty. There are also auxiliary structures with the board to help with the game logic, such as an array of groups, to simplify checking for captures and counting the final score. The player boards are simplified because the only game logic that depends on them is getting moves for the player, which is done by searching for empty points. Finally, the last structure that was added

with Phantom Go is an integer array of size two, which helps to keep count of the stones on the judge board. This structure was implemented to simplify sampling, which is described in Chapter 5.

Chapter 4

Search Algorithms

Monte-Carlo Tree Search is a widely used algorithm for games with perfect information like Go. Most of the published literature uses Monte-Carlo Tree Search and its variants on Phantom Go. However, using MCTS naively in perfect-information games leads to highly exploitable strategies. In chapter 6 I will describe an adaptation of MCTS to imperfect-information games. It has been shown to not have convergence guarantees to minimax optimal solutions but often empirically performs well in many games.

4.1 Monte Carlo evaluation

Monte-Carlo evaluation is an evaluation function for game-tree search based on randomly sampled games. An evaluation for a game state A is computed in three steps: (1) A defined number of playthroughs is played from state S to a terminal state, (2) Each randomly played game is rated according to the rules of the game, (3) A statistic is computed based on the results of the randomly played games. By this evaluation of the game state, we can decide which sub-tree is the player most likely to win.[BSCU08]

4.2 Monte Carlo tree search

Monte Carlo tree search is a search algorithm that gradually creates a tree structure based on the game playthroughs using the Monte Carlo evaluation [BSCU08]. Each node N in the tree contains at least three different tokens of information: (i) a move representing the game-state transition associated with this node, (ii) the number $t(N)$ of times the node has been played during all previous iterations, and (iii) a value $v(N)$ representing an estimate of the nodes' game value. During each iteration, four stages are consecutively applied:

- Move selection - Using the tree policy, for instance, the UCT 4.3 function, to select the most promising node N , which has an unexplored child node.
- Expansion - Randomly, a child node of the node N is picked

- Simulation - One or more games are played randomly or using a simple strategy to a terminal state, and we rate the game with a reward. 1 for a win, 0 for a draw, and -1 for a loss. This stage is sometimes also called Rollout.
- Back-propagation - After evaluating the newly explored node, the tree leading to the node N is updated. According to the simulation results, the number of visits increases, and the value $v(N)$ is updated.

4.3 UCT

UCT is a move-selection algorithm used in many successful Go programs, such as ¹Crazy Stone, ²Mogo, and ³Mango[BSCU08]. At start of each evaluation on a state using Monte Carlo exploration, a new game tree is incrementally built, and each node's UCT value is being updated with each exploration playthrough, which visits the node. Given a node N with N_i representing its group of child nodes, which can be visited by playing an action on node N , UCT chooses the child node N_i , which maximizes the following function

$$f(i) = \overline{X}_i + C \sqrt{\frac{\ln t(N)}{t(N_i)}}$$

In this function, $t(N)$ represents how many times the node N was visited, $t(N_i)$ represents how many times the child node N_i was explored, and \overline{X}_i is the average evaluation of the child node. The constant C controls the ratio between exploration and exploitation. It prescribes how often a move represented by child node N_i with high confidence (large $t(N_i)$) and good average game value (large \overline{X}_i) is preferred to a move with lower confidence or lower average game value. The updating function used together with UCT sets the value X of a node N to the average of all the values of the previously selected children including the latest selected child's value X_i . [BSCU08]

¹<https://www.remi-coulom.fr/CrazyStone/>

²<https://senseis.xmp.net/?MoGo>

³<https://web.archive.org/web/20071103202224/http://www.cs.unimaas.nl/go4go/mango/>

Chapter 5

State sampling

To be able to run algorithms on imperfect information games like Phantom Go, we have to implement methods to sample the game state, that appear the same from the point of view of the player, but the actual state is different from the original state. The game can be resampled into a Metaposition or into an Information State.

5.1 Information State

Two states are in the same Information State, if they are indistinguishable in each step by the player's observation.

As a solution to this rather difficult problem, I have decided to use OR-Tools. OR-Tools is an open-source software suite for optimization, vehicle routing, flows, integer and linear programming, and constraint programming.

The whole problem can be described as a Constraint satisfaction problem. A model in this case is a structure of variables and constraints applied to these variables. Each point of the board is modelled as a structure of three BoolVars, each describing if the point is occupied by black stone, white stone or if the point is empty. BoolVar is a variable, which represents true or false. This structure was chosen, because it can be used as a condition in enforcing constraints, such as "enforce a constraint only if a certain point is white".

On each point of the board, a constraint is created, using a XOR operation between those three Booleans to ensure that only one of them will be true in the generated solution. This corresponds to the fact, that a point on a board can be only white or only black or only empty. To sample a game state in the information state, we need to ensure the observations the player receives are compatible with the information state observations at all time steps.

When creating the model, each board, excluding the first empty board, is based on the previous board and player's observation at the current time. Based on player's observations, I add additional constraints which specify what minimal changes between two successive boards are allowed

The new resampled state is then created by applying moves, which are generated as differences between the boards from the solved model. If there is no difference between the boards in the successive boards, there are 3 possibilities what type of move has happened. 1) In the case of a pass move,

in the resampled state, a pass move is also played. The other move that could cause no difference in the generated boards is either II) an observational move from the opponent, or III) player's move into opponent's eye, which results in no difference in player's observation. In case of opponent's observational move, an observational move onto any player's stone is played, so the count of the stones remains the same and the right player is on the move. In the other case, an observational move onto any of the discovered opponent's stones is played, so that as in the previous case, count of stones and the player on move remains the same as in the original state. If no opponent's stone was discovered, a move is played onto own stone.

For shorter games with 20 actions played take from 200 to 500 milliseconds to resample, medium length games with length of 150 resample in range between 1200 and 3000 milliseconds, and games longer than 300 actions take from 3500 milliseconds up to 6000 milliseconds. These values were measured on Intel i5-7300HQ 2.50GHz. The reason for almost exponentially rising times of resampling is affected by few factors. The first factor is captures. To correctly resample with a capture, two extra boards to constrain boards at time before and after the capture are created to ensure, that certain groups of stones are present at the right time to be captured, when the boards computed as Constraint satisfaction problem are converted into a history of a state that is in the same Infostate as the state that is being resampled. The other factor is the increasing count of stones on the boards at the later states in histories, which create more constraints between the boards that need to be satisfied.

Because OR-Tools solve Constraint satisfaction problems using an algorithm, for every state in the same Infostate, the OR-Tools resampling outputs the same state every time. Since this would reduce efficiency, a simple but effective algorithm has been implemented. From the state, that is being resampled, all empty points are inserted into a vector. The vector is then shuffled and according to the length of the sampled game, n points are used as a constraint, to make the point empty through the whole history. The constant n varies from 15 for games of length 50 or less, to 2 for games longer than 300 actions. With this algorithm, more than 98% models are correct, which is a sufficient percentage whereas each evaluation uses 10 word samples. If this algorithm makes the model unsolvable, because the chosen point can not be empty through the whole game, the original state is returned.

5.2 Metaposition

Two states are in the same Metaposition, if they are indistinguishable by the player in the last observation of the Information State.

Because only the observation and the count of the stones have to be the same, sampling a state from a Metaposition can be done easily. By playing moves for all the player's stones, each excluding the last move followed by opponent's pass, then playing moves for all the known enemy stones, followed by same move by player to add the stone to his observation and a pass.

Finally for the remaining stones, the stones are placed randomly in a way that does not cause changes on the board, such as captures.

Chapter 6

Using Monte Carlo Tree Search on Phantom Go

One of the goals of this project was to test the effectivity of MCTS algorithms against a random player. There are three final policies for MCTS algorithms implemented in OpenSpiel framework. I first tested their effectivity against a random player, then after considering the result, I tested the two most effective policies against each other.

6.1 Information State MCTS

Information State MCTS implementation is different from normal MCTS. Instead of representing the minmax tree with state nodes, we represent the tree with information set nodes, which uses the computational budget more efficiently, because the statistics about the moves are stored in one tree. Furthermore, the model is able to exploit moves that are good in many states in the information state. [CPW12]

6.2 ISMCTS against Random player

As anticipated, ISMCST has shown great results against a random player. ISMCTS with MaxValue final policy has won 96 out of 100 games, similarly MaxVisitCount policy won 98 out of 100 games. The NormalizedVisitCount has shown its inapplicability in Phantom go, winning only 68 out of 100 games.

Each of those 100 game simulations were played in a balanced way, 50 games played by the IS MCTS bot as a white player, 50 as a black player.

6.3 MaxValue or MaxVisitCount policy

Because both of these policies had great results against the random player, 1000 simulated games were played between those two policies. As in the previous simulations, the colors were switched after the half of the games.

Bot with MaxVisitCount policy has won 521 games out of the 100 simulated, bot with MaxValue policy won 474 games and 5 games resulted in a tie.

6.4 Different efficiency with different number of simulations

The most effective final policy MaxVisitCount was also tested against random player with different number of simulations per step. Games with 200 simulated games were run, with 200, 500, 1000 and 2000 simulations done for one step of the bot. As expected, the win-loss ratio was higher with each increase of simulations.

Number of simulations per step	Number of won games out of 200
200	174
500	182
1000	196
2000	199

6.5 Go engine as a heuristic

One of the main goals of this project is to implement an evaluation function using a Go Engine. The selected engine is ¹ GNU Go. This engine was picked due to its availability and its text based mode, that was suitable for this application.

To use an external program, a text-based communication environment was created. The program is ran as a sub-process using forking, and the communication with the process is done using pipes. To make the communication easier, Write and Read functions are implemented, which use the functions read and write from unistd, and make handling potential errors easier. To evaluate a state, after resampling, the judges board is passed to the engine using the implemented Write function. After, a Write with GNU Go command "experimental_score", followed by each color, is used to get the value of the state for each player. These values are then get using the implemented Read function, which parses the text into a number and returns the obtained values as a vector of two values. The method was picked because it is the only method in the GNUGO engine that can evaluate state score based on the players color.

¹<https://www.gnu.org/software/gnugo/>


6.6 Go engine evaluation against Random Rollout evaluation

As described in section 4.2, classical MCTS uses rollout evaluation to explore the game tree and select the best action. MCTS using the Go engine as evaluation function substitutes the Simulation stage by evaluating the state picked by Expansion stage. Both of the IS MCTS bots used the most effective MaxVisitCount final policy.

Since the evaluation using GNUGO is much slower and the evaluation replaces many simulations during the Simulation stage, the number of evaluations was lowered to 200 for the IS MCTS bot using GNUGO. Out of 400 simulated games, 284 were won by Bot with Random Rollout evaluation, 109 were won by Bot with GNUGO evaluation and 7 resulted in a tie. The lower count of the simulations is caused by the fact, that my implementation of GNUGO environment was not flawless, and many simulations did not reach terminal state.

The Inefficiency of the GNUGO evaluation could be caused by one of these factors: (1) the number of the simulations was picked too low, but higher number would make an imbalance between time taken to generate an action by each bot, (2) the GNUGO program may not be intended to be used as a state evaluator, but as a move generator, which can be assumed by its number of participations in online tournaments ², (3) the used method "experimental_score" may not have been suitable for this application.

²<https://www.gnu.org/software/gnugo/tournaments.html>



Chapter 7

Conclusion

In this work I have shown the difference between Go and Phantom Go and its implementation, described and implemented an algorithm to sample states that belong into the same Information State.

Using the implemented sampling algorithm, I have evaluated Information-State Monte Carlo Tree Search algorithm against a random player. In the experiment I have proved the statement, which is proposed in the Introduction, that IS-MCTS Search shows promising results in the domain of imperfect-information games.



7.1 Future work

As a possible future work, I would re-implement the evaluation function using a different Go engine, that would be more suitable for its usage as a heuristic for the value function.



Bibliography

- [BSCU08] Joris Borsboom, Jahn-Takeshi Saito, Guillaume Chaslot, and Jos Uiterwijk, *A comparison of monte-carlo methods for phantom go*.
- [CPW12] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse, *Information set monte carlo tree search*, IEEE Transactions on Computational Intelligence and AI in Games **4** (2012), no. 2, 120–143.
- [FLYY14] Li Fei, Ouyang Li, Wang Yajie, and Dong Yanqiu, *Study of strategy selection based on phantom go*, The 26th Chinese Control and Decision Conference (2014 CCDC), 2014, pp. 3460–3462.
- [KSB⁺21] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý, *Rethinking formal models of partially observable multiagent decision making*.
- [LLB15] Viliam Lisý, Marc Lanctot, and Michael Bowling, *Online monte carlo counterfactual regret minimization for search in imperfect information games*, vol. 1, 05 2015.
- [LLL⁺19] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Viničius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis, *OpenSpiel: A framework for reinforcement learning in games*, CoRR **abs/1908.09453** (2019).
- [Scs11] Scsc, *Gofin*, <https://commons.wikimedia.org/w/index.php?curid=15649052>, 2011.
- [WZL⁺18] Jiao Wang, Tan Zhu, Hongye Li, Chu-Husan Hsueh, and I.-Chen Wu, *Belief-state monte carlo tree search for phantom go*, IEEE Transactions on Games **10** (2018), no. 2, 139–154.

- [ZS21] Brian Hu Zhang and Tuomas Sandholm, *Subgame solving without common knowledge*, CoRR **abs/2106.06068** (2021).