BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Blahoš Jan** Personal ID number: **498841**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Pruning for Best-Response Algorithms for Imperfect Information Extensive-Form Games

Bachelor’s thesis title in Czech:

Prořezávání pro výpočet nejlepší odpovědi v extensivních hrách s neúplnou informací

Guidelines:

Computing a best response to a fixed strategy of the opponent is a key method used in game theory either to evaluate the quality of the strategy or it is directly incorporated into many equilibrium-computation algorithms (for example Fictitious Play, Double-Oracle algorithm, etc.). Computing best response in imperfect information extensive-form games requires traversing through exponentially large game tree, hence the full search algorithm does not scale well. The goal of the student is:
1) to review existing methods on computing exact best response in imperfect-information extensive-form games
2) implement exact best response in imperfect-information extensive-form games based on [3] into OpenSpiel
3) extend the implementation to compute an error-bounded approximate variant of best response
4) compare the performance of newly implemented algorithms with full best response and other methods identified in step 1)

Bibliography / sources:

1. Marc Lanctot et al. 'OpenSpiel: A framework for reinforcement learning in games.' arXiv preprint arXiv:1908.09453 (2019).
2. Yoav Shoham and Kevin Leyton-Brown. 'Multiagent systems.' Cambridge Books (2009).
3. B. Bošanský, C. Kiekintveld, V. Lisý, and M. Pechoucek (2014). 'An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information.' Journal of Artificial Intelligence Research, 51, 829-866.

Name and workplace of bachelor’s thesis supervisor:

doc. Mgr. Branislav Bošanský, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **14.06.2023** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **16.02.2025**

doc. Mgr. Branislav Bošanský, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

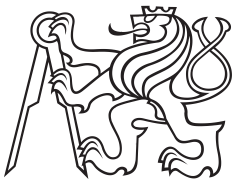
The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Pruning for Best-Response Algorithms for Imperfect Information Extensive-Form Games

Jan Blahoš

Supervisor: doc. Mgr. Branislav Bošanský, Ph.D.
Field of study: Open informatics
Subfield: Computer Games and Graphics
May 2024

Acknowledgements

I would like to thank doc. Mgr. Branislav Bošanský, Ph.D. for helping me better understand the theory behind the algorithm and his overall guidance. I would also like to express my gratitude towards OpenSpiel developers and contributors for maintaining this amazing framework. Finally I'd like to thank my family and friends for all of their support during my studies.

Declaration

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

In Prague, 27. May 2024

Abstract

This thesis aims to identify and review existing methods of computing exact best response. It provides description and implementation of exact best response algorithm with pruning inspired by [Bos14] extended to compute an approximate best response. Another algorithm based on Monte Carlo tree search similar to [Tim22] is described and implemented. All implementations are done in OpenSpiel [OS19] framework. Finally, we compare above mentioned algorithms and the current version of best response in OpenSpiel.

Keywords: game theory, artificial intelligence, sequential games, imperfect information games

Supervisor: doc. Mgr. Branislav Bošanský, Ph.D.
Katedra počítačů, FEL

Abstrakt

Tato práce se zabývá identifikováním a přezkoumáním existujících metod pro výpočet přesné nejlepší odpovědi. Je poskytnut popis a implementace algoritmu přesné nejlepší odpovědi s prořezáváním inspirovaný [Bos14] rozšířený o výpočet přibližné nejlepší odpovědi. Dále je popsán a implementován algoritmus založený na metodě Monte Carlo prohledávání, který vychází z [Tim22]. Všechny implementace jsou realizovány v prostředí OpenSpiel [OS19]. Nakonec porovnáváme zmíněné algoritmy a současnou verzi nejlepší odpovědi v prostředí OpenSpiel.

Klíčová slova: herní teorie, umělá inteligence, sekvenční hry, hry s neúplnou informací

Překlad názvu: Prořezávání pro výpočet nejlepší odpovědi v extensivních hrách s neúplnou informací

Contents

Figures

Tables

Chapter 1

Introduction

Finding a best response strategy is a common task in competitive games that aids game solving algorithms and agents. The best response sequence algorithm finds an optimal strategy for a player, given the opponent strategies, which is particularly useful for finding equilibria in games. It is mostly used as a part of oracle algorithms, such as [Bos14], but also has other uses, for example evaluating the exploitability of a strategy or as a pruning method to speed up algorithms such as counterfactual regret minimization [BS17].

OpenSpiel[1] provides a simple yet efficient implementation of the best response algorithm. While the algorithm is fast and in many cases sufficient, it requires enough memory to store the entire game tree. Since most of the games are exponential in size, storing the entire tree isn't a viable option for larger games.

The goal of this thesis is to implement exact best response algorithm with pruning inspired by Section 4.2 of [Bos14], which should be more memory efficient than the OpenSpiel version, since storing the tree in memory is not required for it to work. The original algorithm is extended by the computation of approximate best response. We also explore another algorithm (slightly altered version of [Tim22]) that takes the Monte Carlo approach, which converges to best response. Finally, we compare implemented algorithms together with OpenSpiel's best response algorithm on two games: Kuhn and Leduc poker.

[1] OpenSpiel[OS19] is an open source framework by Deepmind, which offers a variety of game theory related algorithms and games that these algorithms operate on.

Chapter 2

Background

In this section we discuss some important concepts from game theory, establish a few definitions and introduce notation in order to better understand the algorithm description in chapter 4. The definitions are derived from [Bos14], [SL09] and [Sch21] while the notation is kept similar to [Bos14] where applicable.

2.1 Extensive form game

Our algorithm operates on two player zero-sum extensive form games (EFGs) with imperfect information. An extensive form game is represented as a tree, where nodes represent the current game state and edges correspond to actions that may be taken by the acting player in that state. Each node has its unique history of actions. The leaf nodes are called terminal and assign rewards to each player. Zero-sum means the sum of utilities at each terminal node of both players equals zero, this models competition between the two players. They have perfect recall of their own actions, however are deciding with uncertainty, which can be either a result of working with stochastic environment or some of the opponent's actions not being observable.

Definition 2.1 (Extensive form game)**.** Two player EFG is formally defined as a tuple $(N, H, Z, A, p, u, \mathcal{C}, \mathcal{I})$, where N is a set of two players, typically $\{1, 2\}$, H represents all nodes in the game tree, $Z \subseteq H$ denotes the set of terminal states. A denotes the set of all actions, we use $A(h)$ to represent all actions available at node h. The function $p : H \to N \cup c$ assigns nodes to players that are acting in that state or to Nature player (c). Nature player picks an action based on a predetermined probability distribution known to each player. For each terminal node, the function $u : Z \to \mathbb{R}$ is defined, which assigns utility values to each player in this node. For each node, the value $\mathcal{C}(h)$ represents the probability of reaching this node based on Nature player's policy, assuming the other players play the necessary actions. Finally, $\mathcal{I}$ represents the set of information states (or sets) of acting players, which is explained in the following section.

For better understanding, let's have a look at Figure 2.1. In this case, N would be {Red player, Blue player}, Z are all 9 leaf nodes of depth 2 to which

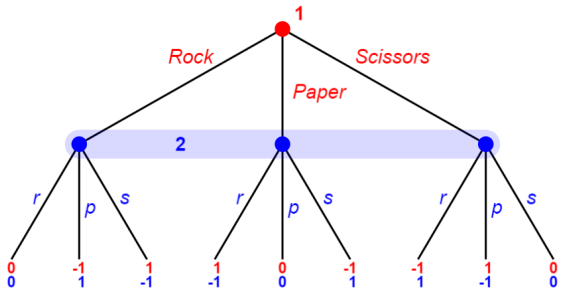


Figure 2.1: Rock-paper-scissors represented as a sequential game

function u assigns utility values for both players. A is the set {Rock, Paper, Scissors}. Function p is represented by colouring of vertices. We can see the root node assigned to the Red player while Blue player has 3 nodes in depth 1. There are no nodes assigned to the Nature player, yet the game can still be an imperfect information game by players not being able to observe the opponent's action.

2.2 Information set

Each player's acting nodes in imperfect information games can be partitioned into information sets. The players cannot distinguish between nodes in a given information set, since the observed actions leading to these nodes and available actions at them are the same.

As an example, please see the game tree representing the game rock-paper-scissors in Figure 2.1. All decision nodes of Blue player are in the same information set: the player doesn't know which action Red player had chosen to play and the actions available are the same in each of those nodes. Trivially, the single decision node of Red player is part of their only information set.

2.3 Best response

When talking about players, we will refer to the best responding player as *i* and other players as *-i*. Strategy or policy of a player can be either mixed or pure. Players decide what action to play based on probability distribution over available actions at each information state. If in every information state one action is assigned the probability 1.0, it is referred to as pure strategy, mixed otherwise [1].

Best response of player *i* to a fixed strategy of player *-i* can be understood as a mapping of actions and their probabilities of being played to each information

[1] Pure strategy is a special case of mixed strategy, meaning every pure strategy is mixed and a mixed strategy can but doesn't have to be pure

set of player *i* that maximizes their utility. There doesn't necessarily have to be one unique best response as explained in Section 3.3.2 of [SL09].

Definition 2.2 (Best response)**.** Let $u_i(\pi_i, \pi_{-i})$ be the expected utility gained by player i, when following strategy π_i against the opponents' strategies π_{-i} and Π_i be the set of all possible strategies of player i. Best response against policy π_{-i} is a mixed strategy π_i^* that satisfies:

$$u_i(\pi_i^*, \pi_{-i}) \geq u_i(\pi_i, \pi_{-i}) \; \forall \pi_i \in \Pi_i$$

Chapter 3

Related work

There are many ways to tackle the computation of best response in imperfect information games. They can be separated into two categories: domain-specific, meaning optimized for a given game or a set of similar games and general, which should work in any environment. In this thesis, we will focus on the latter. The simplest approach is performing full search of the game tree, as described in Section 4.1. However, the game tree is exponential in size, meaning we want to avoid traversing as many branches of the tree as possible.

An algorithm that applies pruning to this naive approach is introduced in [Bos14]. Unfortunately, pruning a percentage of the exponentially large tree isn't a feasible approach for larger games. A different approach based on Monte Carlo tree search is proposed in [Tim22]. Both of these algorithms are further described in the following chapters.

Another way of computing an approximation of best response is the Local Best Response algorithm [LB17], which relies on game-specific knowledge and heuristics.

Parallelization and avoiding game-specific isomorphisms can lead to significant speed-ups in computation [Joh11], where the former depends on hardware and the latter is a domain-specific method and are thus not explored further in this thesis.

Chapter 4

Best response with pruning

In this chapter we will discuss the basic logic behind the algorithm, the the ideas behind pruning and computing error bounded best response.

4.1 Basic logic

The algorithm aims to exploit a predetermined strategy of player $-i$ and find a best response policy together with returning the expected utility value.

The logic is based on a classic depth-first search (DFS) algorithm that differentiates between several options: we are either in a terminal node, chance node or a decision node of one of the players.

- **Terminal node**: returns the utility of best responding player weighted by the probability of reaching this node, assuming i plays the necessary actions to reach it.
- **Chance node and decision node of player *-i***: the value of this state is gained by accumulating the returns of subsequent calls of the DFS function for each of the available actions in current state
- **Decision node of player *i***: determines which action has the maximum expected utility, considering all nodes in current information set, caches this action in order to build a best response policy and returns the value of this action, but only the value gained from the current node

4.2 Pseudocode and pruning

In this section we explain the pseudocode of the algorithm shown in Figures 4.1 and 4.2 [1] Please see Table 4.1 for better understanding of the notation used. Before starting, we need to know the game's minimum and maximum utility values and set λ to minimum utility.

[1] Very similar pseudocode can be found in [Bos14, Section 4.2]

Expression	Meaning
BRS_i	recursive function
$\overline{r}_{-i}(seq_{-i}(h))$	probability of reaching node h based on $-i$'s strategy
$C(h)$	probability of reaching node based on Nature play
ha	node reached by playing action a at h
$A(h)$	actions available at node h
I_i	current information set of player i

Table 4.1: Expressions used in the pseudocode

4.2.1 Nodes of the other players

First, let's talk about processing nodes of the opponent and Nature player (Figure 4.1). The case of node h being terminal can be viewed as a separate part of the algorithm, since no player is acting there. Both here and in nodes of the best responding player, a weighted value is returned, where the weight is the probability of reaching this node, assuming player i picks the necessary actions to reach it. This can be easily calculated from policy of player -i and public knowledge of Nature player's policy.

1: $w \leftarrow \overline{r}_{-i}(seq_{-i}(h)) \cdot \mathcal{C}(h)$
2: **if** $h \in Z$ **then**
3: **return** $u_i(h) \cdot w$
4: **end if**
5: sort $a \in A(h)$ based on probability $w_a \leftarrow \overline{r}'_{-i}(seq_{-i}(ha)) \cdot \mathcal{C}(ha)$
6: $v^h \leftarrow 0$
7: **for** $a \in A(h)$, $w_a > 0$ **do**
8: $\lambda' \leftarrow \lambda - [v^h + (w - w_a) \cdot$ MaxUtility$]$
9: **if** $\lambda' \leq w_a \cdot$ MaxUtility **then**
10: $v' \leftarrow BRS_i(ha, \lambda')$
11: **if** $v' = -\infty$ **then**
12: **return** $-\infty$
13: **end if**
14: $v^h \leftarrow v^h + v'$
15: $w \leftarrow w - w_a$
16: **else**
17: **return** $-\infty$
18: **end if**
19: **end for**
20: **return** v^h

Figure 4.1: BRS in nodes of other players

To start, we set up variables w, which represents the probability of reaching the current node and set the expected utility v^h to 0. We iterate over all available actions sorted by their probability of play, which we know either from the fixed policy $\overline{r}_{-i}$ of the best responder's opponent or the publicly

known policy of Nature player C. Actions with 0 probability are omitted. First, a new lower bound (λ') for child node is calculated. This represents the minimal value that must be returned from the following call of BRS_i in order for this node to be relevant for the best response. If this value cannot be gained, a cut-off occurs (line 16.). The other possible cut-off happens by propagation from a child node (line 11.). If no cut-off occurs, the value v^h is incremented by the value returned from recursive call and w is decremented by the probability of reaching node ha. The value accumulated in v^h is returned at the very end.

4.2.2 Nodes of the responding player

Handling the terminal case is the same as explained in Section 4.2.1. Additionally, if the value of the current information state is already calculated, we simply return the cached value.

We begin by collecting all nodes from the information set of node h (line 7.), which are then sorted based on the probability of being reached. The variable w is set to represent the probability of reaching the current information set, which is the sum of probabilities of reaching each of its nodes and the value v_a is set to 0 for each action $a \in A(h)$. We then iterate over the sorted list of nodes and evaluate all actions for every node (line 11.).

A new lower bound λ' is calculated (lines 15. and 17.), which represents the minimum value that needs to be acquired from the recursive call in order for the current action to remain a potential best response action ("If the current action yields maximum value in all unprocessed nodes from this information set, can it still be the best action?"). If the current action can no longer be the best action, the recursive call is not performed, otherwise the current action value is updated. At line 26., a cut-off occurs when the original node h was evaluated and the greatest utility that can be achieved at this node is less than the lower bound λ. Finally at line 30., the individual expected utilities for found best action are stored for each node in current information set and the expected value of current node h is returned.

4.3 Error bounded approximation

This section describes error bounded best response which should return no more than ϵ worse value than the exact best response. There are approximation methods for computing best response ([Tim22], [LB17]), however, they do not guarantee any error bound. To the best of our knowledge, there is no research related to computing error bounded best response.

Regarding the algorithm described in this chapter, it accumulates return values from recursive calls for each action (and in the case of responding player fully evaluates best action at each information set). We could theoretically allow each node to return a value that is at most ϵ_1 lower. Unfortunately, it isn't that simple. Imagine a game where player i decides to play an action in root node, then player $-i$ plays and a terminal state is reached. If we

1: **if** $h \in Z$ **then**
2: **return** $u_i(h) \cdot \overline{r}'_{-i}(seq_{-i}(h)) \cdot \mathcal{C}(h)$
3: **end if**
4: **if** v^h is already calculated **then**
5: **return** v^h
6: **end if**
7: $H' \leftarrow \{h';\ h' \in I_i\}$
8: sort H' descending according to value $\overline{r}_{-i}(seq_{-i}(h')) \cdot \mathcal{C}(h')$
9: $w \leftarrow \sum_{h' \in H'} \overline{r}_{-i}(seq_{-i}(h')) \cdot \mathcal{C}(h')$
10: $v_a \leftarrow 0\ \forall a \in A(h)$; $maxAction \leftarrow \emptyset$
11: **for** $h' \in H$ **do**
12: $w_{h'} \leftarrow \overline{r}'_{-i}(seq_{-i}(h')) \cdot \mathcal{C}(h')$
13: **for** $a \in A(h')$ **do**
14: **if** $maxAction$ is empty **then**
15: $\lambda' \leftarrow w_{h'} \cdot$ MinUtility
16: **else**
17: $\lambda' \leftarrow (v_{maxAction} + w \cdot \text{MinUtility}) - (v_a + (w - w_{h'}) \cdot \text{MaxUtility})$
18: **end if**
19: **if** $\lambda' \leq w_{h'} \cdot$ MaxUtility **then**
20: $v_a^{h'} \leftarrow BRS_i(h'a,\ \lambda')$
21: $v_a \leftarrow v_a + v_a^{h'}$
22: **end if**
23: **end for**
24: $maxAction \leftarrow \arg\max_{a \in A(h')} v_a$
25: $w \leftarrow w - w_{h'}$
26: **if** h was evaluated $\wedge$ $(max_{a \in A(h)})\ v_a^h < \lambda$ **then**
27: **return** $-\infty$
28: **end if**
29: **end for**
30: store $v_{maxAction}^{h'}$ as $v^{h'}\ \forall h' \in H'$
31: **return** $v_{maxAction}^h$

Figure 4.2: BRS in nodes of best responding player

allow both states to return a value that is ϵ_1 lower, the final return value can suddenly be $2\epsilon_1$ lower and this would continue with increased depth. A similar problem is mentioned in [Atz18, page 12].

The games are not usually represented as a full tree, but we could allow each node to return a value (ϵ_1 / $normalizationFactor$) lower, where $normalizationFactor$ would be the greatest depth. We believe this would waste the potential of the allowed error, as many games have varying number of player actions before reaching a terminal node (e.g. in poker, a player may fold immediately or perhaps both players are opting to check to the very end) and in the low action count branches the allowed error would be very low compared to the original ϵ_1.

We instead propose a solution that allows one information state of player i

on each path to a terminal node to return at most η % worse value than the exact best response.

Lemma 4.1. *Given the algorithm described in Section 4.2; if we allow at most one node on each path from root to leaf to return a value at most η% lower, the final expected value is guaranteed to be at most η% lower.*

Proof. Let's mark nodes that are allowed to return η% lower value as E and those guaranteed to do so as G. We start with a graph that has no nodes marked as G and some amount marked as E with respect to the assumptions. We can mark all nodes that don't contain any node E in their subtree as G as they return an unchanged value. When all nodes are marked, then the root node is guaranteed to return a value at most η% lower and the statement is true (1).

As long as there are unmarked nodes, we have at least one unmarked node h which has all of its child nodes $\{h_1, ..., h_k\}$ marked as E and/or G. (Note that from the assumption of allowing only one node on each path to return less, node h or any of its ancestors can no longer be marked as E.)

Due to the nature of our algorithm, the nodes of player $-i$ and chance simply accumulate values from child nodes and return it or $-\infty$ when this branch is irrelevant and to be pruned. When the branch is still relevant, we are summing up the values of child nodes, which allows node h to be marked as G, since in the extreme case of each child node returning lowest possible value, we still get:

$$\sum_{i=1}^{k}[(1-\eta)\cdot u_i] = (1-\eta)\cdot\sum_{i=1}^{k} u_i \qquad (2)$$

Here u_i represents the original return values of nodes $\{h_1, ..., h_k\}$. Similarly, in the nodes of player i, the algorithm is guaranteed to fully explore the best action and return its value, which we get by summing up the value of this action in each node of the information set that is nothing else than the value of the child node given by the action taken. We can thus mark all nodes in a given information set H' of player i as G when each child node of nodes $h' \in H'$ had been marked, based on the same principle as (2).

The algorithm sometimes backpropagates the values $-\infty$ in no longer relevant branches and all such nodes can be ignored as they do not contribute to the final value.

By repeatedly labeling unmarked nodes h, we eventually label the entire tree and according to (1), the final return value is guaranteed to be at most η% lower than the original.

□

Chapter 5

MCTS based algorithm

In this chapter we discuss how to get from Monte Carlo variant for perfect information games to the best response information set Monte Carlo tree search for games with imperfect information.

5.1 MCTS

Monte Carlo tree search (MCTS) algorithms are a popular way to deal with large perfect information games and has been most successful in games like Go, where an algorithm such as minimax with α-β pruning or NegaScout struggles due to the extreme size of the game tree. Apart from scalability, MCTS is an anytime algorithm, meaning it can return a value even when the computational time is limited.

MCTS conctructs a subtree of the game by repeating these 4 steps:

- **Selection**: Repeatedly descend nodes in the tree using a bandit algorithm (UCB being the most popular) until a node that has not yet been added is reached or the state is terminal.
- **Expansion**: Add a new node to the tree.
- **Simulation**: Run a simulation from the newly added node to the end of the game. Popular methods to achieve this are random rollout evaluator or a learned value function.
- **Backpropagation**: Update node visit counts and utility based on the value returned from simulation.

MCTS is described in more detail in [Cow12] and there are many sources online.

5.2 ISMCTS

One way to apply MCTS to imperfect information games is using determinization, which means sampling states from information sets and thus analyzing games of perfect information. Some of the weaknesses of this approach are

addressed in [Cow12, Chapter 1], the main one is evaluating nodes that are shared between the sampled trees multiple times.

The other approach is using information set MCTS (ISMCTS) described in detail in [Cow12, Chapter 4], which eliminates some of the determinization weaknesses by instead constructing a tree where each node represents an information set, thus collecting all the information in one tree.

5.3 ISMCTS-BR

The first step to computing best response using ISMCTS is to use the policy of player $-i$ during action selection in states where the opponent is acting. With this modification, we still can't use the ISMCTS directly, since it doesn't sample from the belief distribution and the result doesn't converge to the exact best response [Tim22, Section 3.2].

When sampling a history at each information state of player i, we need to consider their belief and compute the probabilities $P(h|s)$ ("What is the probability of being in node h given that we're in information state s?") for each node in given information state. This can be calculated using Bayes' rule:

$$P(h|s) = \frac{P(s|h)P(h)}{P(s)}$$

It is easy to see that $P(s|h) = 1$ and the priors $P(h)$ and $P(s)$ are the reach probabilities derived from the opponent's strategy (and strategy of Nature player).

The full ISMCTS-BR algorithm can be found in [Tim22, Section 3.2].

Chapter 6

Implementation and Testing

In this section we discuss some changes made during the implementation to the previously described algorithms and forms of testing. BRS and ISMCTS-BR shorts are used for the two implemented algorithms.

6.1 Implementation

6.1.1 BRS

The structure of the final implementation was inspired by OpenSpiel's version. For the purpose of better code readability, OpenSpiel introduces a secondary recursive function that essentially makes the decision if we are in a terminal node, chance node or a decision node, and based on this calls a matching recursive function. The algorithm logic still follows the rules described in section 4.2.

One of the challenges was collecting all nodes in a given information set (can be seen in Figure 4.2, line 7). In order to achieve this in OpenSpiel without building the entire game tree, we have to follow the known history to player i, and add all nodes matching this partial history to the information set.

OpenSpiel doesn't offer a well defined observation vector (varies with each game) nor are there other ways of knowing which actions should be visible to player i, so we have to do more branching than actually needed. This could be avoided at the cost of additional memory to remember which chance and player nodes change the information state string, which the State method provides. This string varies between different game implementations, so it is not possible to extract the information directly from it. Our final solution includes a Leduc poker specific implementation, using the knowledge of which actions can be observed by player i.

The option to compute an approximate best response was added by modifying the algorithm according to Section 4.3.

6.1.2 ISMCTS-BR

Our implementation of ISMCTS-BR algorithm is a modification to OpenSpiel's ISMCTS, extended to utilize policy of player $-i$ and adjusting the sampling function in order to guarantee convergence to best response as explained in Section 5.3. The sampling function needs probabilities of reaching each state and should return a pointer to one of them based on chance. This task is similar to collecting all nodes in given information set and the same function is reused, again optimized for Leduc poker.

6.2 Testing

6.2.1 Games

The algorithms were tested on OpenSpiel's Kuhn poker and Leduc poker implementations. Kuhn poker is a very simplified version of poker: the deck contains only 3 cards of ascending value and the actions are either check/fold or bet/call. At the beginning of the game, the chance player deals one random card to each of the players and then each player picks an action to play; they are allowed to play the bet/call action once. The greatest depth of the game tree is reached by first dealing two cards and then the sequence of actions *check* $\rightarrow$ *bet* $\rightarrow$ *call*. The player with the higher card wins. If there is a called bet, the winner receives a positive reward of +2, +1 otherwise. Note that there is no draw scenario.

Leduc poker is also a greatly simplified version of poker, however has significantly more nodes in the game tree (around 9500 vs 58 for Kuhn). This is achieved by a deck of 3 pairs of cards in ascending value and a 2 round betting with a public card revealed in between rounds.

As may be obvious from the description above, Kuhn Poker is ideal for testing and debugging, since it is easy to follow and the values can even be computed by hand. Leduc Poker on the other hand is great for testing the speed, pruning capabilities and memory used by the algorithm.

6.2.2 BRS

During testing we made use of the existing OpenSpiel implementation and compared the return values and best response policies. A short mass testing script that can run a given amount of tests on target game with fully randomized opponent's policy was written.

The overall speed of the algorithm is worse than OpenSpiel's unless we implement game specific function to collect information states as mentioned in Section 6.1.1. The average run of 1 Leduc Poker game on our hardware[1] takes slightly under 18s, compared to OpenSpiel's 1.5s. However, after implementing the mentioned function, the results are much better. The algorithm runs on average 1.4 times faster than OpenSpiel's version on Leduc

[1] processor: AMD Ryzen 5 4500U with integrated graphics, 8GB RAM

Poker. Additionally, it consistently prunes between 600-1000 nodes out of 9457 game states (that is about 6-10%) and uses around 1/3 of OpenSpiel's memory. When using random deterministic policy for player $-i$, meaning random action in each information set gets assigned the probability 1.0, we explore about 1/4 of the game tree.

6.2.3 Approximate BRS

When allowed to compute an approximate best response, the algorithm doesn't run faster unless a large deviation is allowed (greater than 20%). At 10% deviation, although it prunes a few hundred nodes more than the exact best response, the times are comparable to exact best response likely due to more calculations needed in each information state of player i.

In practice, the algorithm prunes up to 1000 nodes more than the exact best response with pruning, however, at the cost of allowed deviation approaching 100%. The actual deviation doesn't usually surpass 5% even with 100% deviation allowed, however we can't guarantee this.

A parameter was added to allow returning a lower value starting at depth [param], which lead to slightly improved performance on Leduc when set to 6, as it allowed more nodes to return a lower value.

6.2.4 ISMCTS-BR

The algorithm was tested with UCB exploration constant set to 10. While the ISMCTS-BR algorithm should be scalable for larger games, it doesn't converge to the exact best response value fast enough to be usable on Leduc Poker. With low sample size, it is no better than a random player and when the sample size is approaching 50000, the results get better, but timewise gets outperformed by exact best response algorithm. The deviation to sample count dependency is shown in Figure 6.1. The flat deviation is an average flat deviation out of 100 tests with the given sample size.

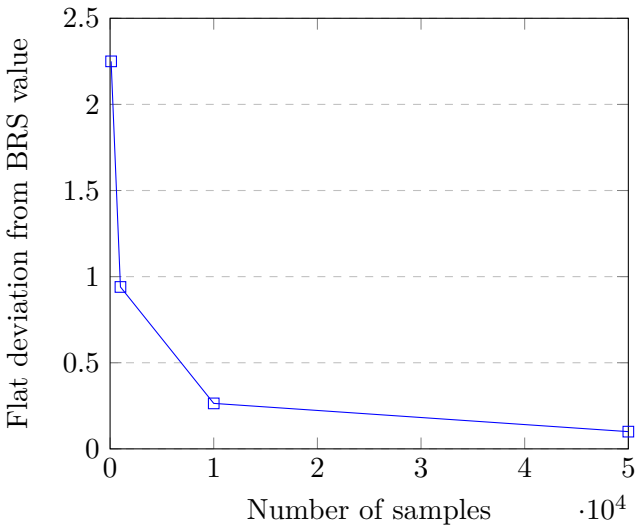


Figure 6.1: Convergence to BRS value based on number of samples

Chapter 7

Conclusion

Best response algorithm with pruning was implemented and extended to compute error bounded approximate best response. The algorithm has been tested with successful results on OpenSpiel's Kuhn poker and Leduc poker games. It appears to be slightly faster than the current OpenSpiel variant, given that a game specific function to collect nodes from target information set is implemented. Otherwise, the algorithm still uses less memory than OpenSpiel's, but timewise appears to be marginally worse. The ISMCTS-BR algorithm converges quite slowly to the actual best response value and actions, however, should be scalable for larger games where BRS would struggle to return any value due to exponential node count.

Bibliography

[Atz18] Atzmon, D. et al. (2018) *Bounded Suboptimal Game Tree Search.* https://ojs.aaai.org/index.php/SOCS/article/view/18448/18239

[Bos14] Bošanský, B. et al. (2014). *An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information.* Journal of Artificial Intelligence Research, 51, 829-866. URL: https://www.jair.org/index.php/jair/article/view/10924/26040

[BS17] Noam Brown and Tuomas Sandholm. *Reduced Space and Faster Convergence in Imperfect-Information Games via Pruning.* Proceedings of the 34th International Conference on Machine Learning, PMLR 70:596-604, 2017. URL: http://proceedings.mlr.press/v70/brown17a.html

[Cow12] Cowling, P. et al. (2012) *Information Set Monte Carlo Tree Search* URL: https://ieeexplore.ieee.org/abstract/document/6203567

[Joh11] Johanson, M. et al. (2011) *Accelerating Best Response Calculation in Large Extensive Games.* URL: http://johanson.ca/publications/poker/2011-ijcai-abr/2011-ijcai-abr.pdf

[LB17] Lisý, V. and Bowling, M. (2017) *Eqilibrium Approximation Quality of Current No-Limit Poker Bots.* URL: https://arxiv.org/abs/1612.07547

[OS19] Lanctot, M.et al. (2019). *OpenSpiel: A framework for reinforcement learning in games.* arXiv preprint arXiv:1908.09453. URL:https://arxiv.org/pdf/1908.09453.pdf

[Sch21] Schmid, M. *Search in Imperfect Information Games.* URL: https://arxiv.org/pdf/2111.05884.pdf

[SL09] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems.* Cambridge Books (2009). URL: http://www.masfoundations.org

[Tim22] Timbers, F. et al. (2022). *Approximate exploitability: Learning a best response in large games.* URL: https://www.ijcai.org/proceedings/2022/0484.pdf