Agents for fast game length estimates

Fabio Barbero
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—This paper discusses different agents that can be used for evaluating game metrics through self-play efficiently. The focus is on evaluating game length for a multitude of games, using the Ludii framework. This paper also introduces Heuristic Sampling, which bases decisions on heuristic evaluations of a subset of moves on the board. When compared to more rigorous methods like Alpha-Beta and UCT, Heuristic Sampling produces similar results up to thousands of times more efficiently.**

***Index Terms*—AI, Board Games, Game Length, Ludii**

I. Introduction

A. Overview

Ludii [10] is a complete general game system implemented in Java. Ludii comes with an interface for playing games written in the Ludii game description language. It also provides a framework for agents to return a move given a board state for a multitude of board games. These board games come from a variety of backgrounds and are played on different boards, with different pieces, rules, and end conditions. Ludii can also be used to generate new game syntax, which can then be compiled and used to evaluate if the game is playable. This framework is particularly useful for evaluating certain metrics of board games, by running a certain number of trials and averaging out the results.

The *Digital Ludeme Project* [8] focuses on the "Digital Archaeoludology" of ancient board games. The artefacts from these games are often incomplete, and several possible rule sets can be generated. These different rule sets can then be evaluated and can provide useful information for finding the most likely version of the original game, such as looking for rule sets that have similar metrics to known games in the same period.

Section II gives an overview of current techniques used for evaluating games and explains the issue of Monte Carlo resistance in games. Section III introduces three agents for game evaluation, including Heuristic Sampling and the concept of Same-Turn Continuation. The setup of the experiments is explained in Section IV, and the last three sections then go over the results, discussion, and conclusion.

B. Problem Statement

Ludii's current approach to evaluating game metrics is through self-play, by running multiple playouts and averaging out the metrics of each playout. At each iteration, Ludii lets the agent that is at play take a move with an enforced time limit. The move is then applied and the board state updated.

Ludii comes with a multitude of built-in agents. The Ludii AI selects the best-known agent for each given game and defaults to *Upper Confidence Tree* (UCT) if the game is unknown. Ludii has two main types of AI agents: those based on traditional tree-based methods (e.g., minimax, Alpha-Beta) and those based on Monte Carlo methods (e.g., Monte Carlo Tree Search, UCT, etc.). The Random agent can be used to quickly run a large number of playouts, but its results will likely be inaccurate as the choice of moves is significantly different from the one of a human player. A more advanced agent will give much better results but takes a much longer time to run, as choosing a move is more computationally expensive than at random. The goal is to achieve fast playouts that give plausible results.

Game metrics such as game length vary from player to player and are by no means universal. One of the goals of this paper is to reach results similar to the ones given by UCT and Alpha-Beta.

C. Research Questions

The following research questions addressed:

1) What is an efficient method for estimating plausible game lengths for general games in terms of accuracy?
2) Can we undersample the available move set for each state while retaining plausible results?
3) Is Heuristic Sampling without lookahead sufficiently strong for evaluating game metrics?

D. Motivation

Given the complexity of different rules for each game, a mathematical approach, such as the one used by Stanford's Game Definition Language [15], is unfeasible and would require too much time to run. Currently, all agents included in the Ludii framework were designed to win as many games as possible. The objective of the agent is different when designing an agent for evaluating game metrics.

One of the most commonly used approaches for evaluating moves is by running random playouts. This gives inaccurate results in multiple scenarios, such as for Monte Carlo resistant games, which will be defined in the next section. It is hence important to identify such games to avoid the risk of getting worse evaluations by increasing the number of playouts. More complex agents such as UCT or Alpha-Beta can also be used to evaluate games. Such agents will give more sensible results

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University. Supervisor: Cameron Browne

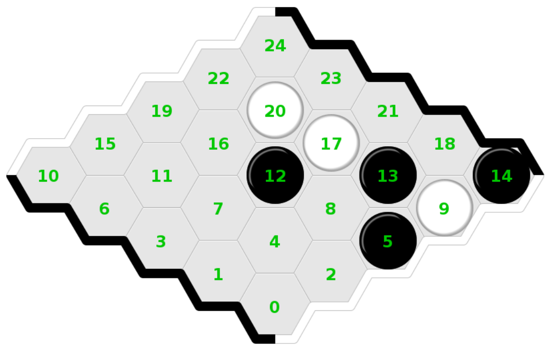

Fig. 1: A 5×5 hex board in a Monte Carlo resistant state. White's turn.

than random, but take too much time on a standard consumer machine.

II. Background

Writing programs that could play games has been one of the earliest goals in Artificial Intelligence. As such, the success of Deep Blue in 1997 and Alpha Go [22] in 2015 are still considered some of the biggest achievements in the field of Artificial Intelligence [12]. Despite being able to beat humans at Chess and Go respectively, these two agents are unable to generalize their knowledge to any other game. General Game Languages had as one of their main goals to provide a general framework that agents could use to play a multitude of different games. The AI then needed to be general enough to play many different games. Some early attempts include Zillions of Games (2003) [16] and Stanford's Game Definition Language [15]. The latter describes every game state as a series of facts, and the mechanics as logical rules. This makes it possible to mathematically prove some properties of the game, but has two major drawbacks: writing all the possible state transitions makes the game description long and hard to read, and computing all the logical values at every state is computationally expensive and makes running trials very slow.

Ludii offers a solution to the problems above: every game is described as a collection of *ludemes* and the framework for running games is considerably fast for running trials [17] [11].

Much research on these frameworks has been done to beat humans at games, or more generally to be used to compete against other agents. Recently, most agents have been designed for video-game playing [19].

A. Monte Carlo resistance in games

The use of Random playouts to evaluate metrics of games is still common in literature [5]. There exist certain games where, given a specific board states, increasing the number of random playouts will result in a worse evaluation of moves by looking at the winning rate. These games can be said to be *Monte-Carlo resistant*, a term provided to me by my supervisor [7].

To check if a game is Monte Carlo resistant, the following experiment can be conducted:

1) Generate a random state by applying a random number of random (legal) moves.
2) Letting an "expert" AI (chosen by Ludii AI) evaluate the moves (with a 10min thinking time) for the given state and subsequently make a ranking of every move.
3) Iterate through multiple random playouts for each legal move, storing and updating the win rate for that move after each iteration.
4) Compute and plot the error between the "expert" ranking and the ranking from random playouts after each iteration.

Fig. 1 shows a 5x5 Hex board in the Ludii General Game System. With the assumption that black plays with an optimal strategy, adding a piece to 1 or 8 will lead to a loss, and to 4 can lead to a win [7]. After running multiple random playouts, the winning rate of move 4 becomes much lower than the winning rate of move 8. Hex is hence a Monte Carlo resistant game.

Another game that is known to be Monte Carlo resistant is Yavalath. In Yavalath, players must place four pieces in a row without first making three in a row to win. [6]. Random playouts tend to disfavour making a line of 2 since it tends to lead to a loss (line of 3), although that is necessary to reach the winning state (line of 4) [5].

III. Methodology

As shown above, the biggest problem with random playouts is that it can often select losing moves that no human player would choose. The first naive approach to improve random without compromising speed is to only select non-losing moves. This results in a very "cautious" AI, or "Safe" AI.

A. Safe AI

Algorithm 1 Safe AI

```
1:  for each state $S_n$ do
2:      generate the set of available actions $A$
3:      loop $|A|$ times
4:          select an action $a$ from $A$ at random
5:          apply $a$ to $S_n$ to give $S_{n_a}$
6:          if $S_{n_a}$ is a losing state then
7:              remove $a$ from $A$
8:          else
9:              return $a$
10:         end if
11:     end loop
12:     return random action $a'$ from available actions
13: end for
```

This is the first, simplest improvement to the random agent. Instead of immediately returning a move after selecting the move at random, the move is first applied and if it leads to a loss another random move is selected.

B. Opportunistic AI

A more *opportunistic* behaviour can be implemented by looking for moves that lead to a win in addition to moves that lead to a loss. When given a board state, this *Opportunistic AI* avoids moves that lead to a loss, plays moves that lead to a win if they exist or plays a random move otherwise. Pseudocode of the Opportunistic AI can be seen in Algorithm 2.

Algorithm 2 Opportunistic AI

```
function FILTERACTIONS(actions $A$, depth $d$)
    for each action $a$ in $A$ do
        apply $a$ to $S_n$ to give $S_{n_a}$
        if $S_{n_a}$ is a winning state then
            return $a$
        else if $S_{n_a}$ is a losing state then
            remove $a$ from $A$
        else if $d > 1$ then
            generate set of actions $A'$ for next mover
            futureAction ← FILTERACTIONS($A'$, $d - 1$)
            if player is nextPlayer then
                if $A'$ is empty then
                    futureWinningAction ← $a$
                else if futureAction is not null then
                    remove $a$ from $A$
                end if
            else if futureAction is not null then
                return $a$
            end if
        end if
    end for
    return futureWinningAction
end function
for each state $S_n$ do
    generate the set of available actions $A$
    winningAction ← FILTERACTIONS($A$, $n$)
    if winningAction is not null then
        return winningAction
    else
        return random action $a'$ from $A$
    end if
end for
```

For games that have a very large branching factor (such as Go), evaluating all moves with depth greater than one takes a long time to run. Sampling a fraction of moves is then necessary to preserve a fast runtime. Another issue with the Opportunistic AI is that for most games where winning and losing moves only appear after a certain number of moves, it will return random moves with a much slower runtime. This led to the idea of quickly evaluating a subset of moves with heuristic functions.

C. Heuristic Sampling

The following idea about *Heuristic Sampling* (HS) can be found in [9], a paper co-authored with my supervisor Cameron Browne currently submitted to IEEE COG 2021.

Algorithm 3 Heuristic Sampling without STC

```
for each state $S_n$ do
    generate the set of available actions $A$
    loop $max(2, |A|/n)$ times
        select an action $a$ from $A$ at random
        apply $a$ to $S_n$ to give $S_{n_a}$
        if $S_{n_a}$ is a winning state then
            return $a$
        else if $S_{n_a}$ is not a losing state then
            compute heuristic estimate of $S_{n_a}$
        end if
    end loop
    return action $a'$ with highest seen heuristic estimate
end for
```

Heuristic Sampling requires a heuristic function that evaluates a state S of a game for a certain player. Further description of HS can be found in [9]. $HS_{1/n}$ denotes the fraction of moves taken into account for evaluation of each board state.

The simplicity of Algorithm 3 is an attractive feature for implementing such an agent. It is as short as Algorithm 1 but has a more sophisticated move selection. It is worth noting that for $n = 1$ it acts like Alpha-Beta search of depth one.

D. Same-Turn Continuation

Same-Turn Continuation (STC) involves the current mover repeatedly moving again until it is another player's turn to move [17].

There exist many games that require STC. In some games of capture, for example Nine Men's Morris, making a line of three triggers a capture move by the same player. When evaluating a move, it is important to compute the heuristic evaluation of the board at the end of the player's turn instead of just after the immediate move, as the heuristic evaluation of the first move often gives poor results. This is the case of Nine Men's Morris, where strong moves are preceded by weaker dependant moves. A similar principle to Same-Turn Continuation is *quiescence search*, where the search continues until a "quiet" state is reached [3].

IV. EXPERIMENTS

A. Games

Experiments were run using the Ludii general game system[1] on the following 2-players, fully observable, deterministic games pictured in Figure 2:

1) *Tic-Tac-Toe*: Players alternate adding their distinctive piece on a 3×3 board. The first player to do a line of three wins [4]. Tic-Tac-Toe has a relatively small game tree complexity of $\approx 10^5$ with an average game length of 6 plies (both players play optimally, resulting in a draw).
2) *Connect4*: Players alternate dropping their coloured discs from the top of a 6×7 grid. The first player to

[1]https://ludii.games

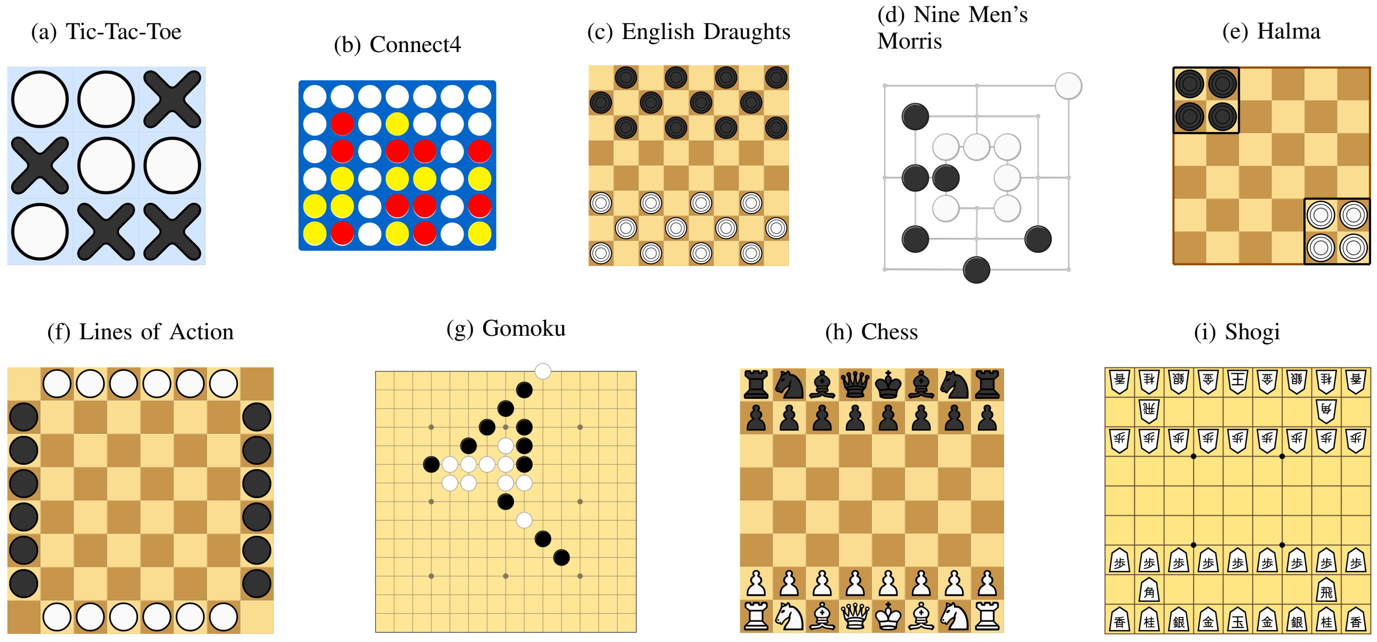

Fig. 2: Board states of tested game from the Ludii Player.

create a row of four disks in their colour wins [1]. The expected number of plies is 36. Connect4 has a game tree complexity of $\approx 10^{21}$ [2].

3) *English Draughts*: Each player has twelve pieces on a 8×8 board. Adjacent opponent's pieces must be captured when possible by hopping on them. Pieces that reach the opposite side of the board can move backwards. The first player without pieces loses [4]. The average number of plies is 70. The game tree complexity is $\approx 10^{31}$ [18].
4) *Nine Men's Morris*: Each player's goal is to eliminate all opponent's pieces from the board, by making a line of three and subsequently removing a piece of their choice that is not in a line. Each player starts by placing their 9 pieces and then moves them to empty orthogonal spots on the board [4]. The expected number of plies is 50, and the game tree complexity of $\approx 10^{50}$ [2].
5) *Halma*: Played on a reduced 6×6 board, each player's goal is to move their 4 pieces in the opponent's side of the board by either moving or jumping on any adjacent piece. Multiple jumps are allowed [13]. The game tree complexity and expected length are unknown.
6) *Lines of Action*: A player wins if they connect all of their pieces contiguously on a 8×8 board. Each piece can move by exactly the number of pieces that are present in the direction it is moving. Opponent's pieces can be captured [4]. The expected number of plies is 44, and the game tree complexity is $\approx 10^{64}$ [23].
7) *Gomoku*: Players alternate placing pieces of their own colour on a 15×15 Go board. First to line 5 in a row wins. The expected number of plies is $L_{exp} = 30$, and the game tree complexity is $\approx 10^{70}$ [2].
8) *Chess* Standard game on 8×8 board using FIDE rules. The expected number of plies is 70, and the game tree complexity is $\approx 10^{123}$ [20].
9) *Shogi*: Game of war played on a standard 9×9 board. The expected number of plies is 115, and the game tree complexity is $\approx 10^{226}$ [14].

B. Heuristics

Fifteen Ludii in-built heuristics from Ludii were used for Heuristic Sampling. These heuristics include [10]:

- *CentreProximity*: proximity of the pieces to the centre of the board. This is of particular use in Lines of Action, where a possible strategy is to connect pieces in the centre.
- *CurrentMoverHeuristic*: adds weight only for the player whose turn it is in any given game state.
- *LineCompletionHeuristic*: player's potential to complete lines up to a given target length. This heuristic is essential for Tic-Tac-Toe, Connect4, Nine Men's Morris and Gomoku.
- *PlayerRegionsProximity*: proximity of pieces to the regions owned by a particular player. This is of particular use for Lines of Action and Halma, where all pieces need to be connected.
- *Material*: material that a player has on the board and in their hand. This plays a major role in Chess, where each piece has a different score.
- *OwnRegionsCount*: sum of all counts of sites in a player's owned regions. In Nine Men's Morris and English Draughts the player with most pieces on the board is often in a better position.
- *SidesProximity*: proximity of pieces to the sides of a game's board.

- *CornerProximity*: proximity of pieces to the corners of a game's board. The goal of Halma, for example, is to bring all pieces to the Opponent's corner.
- *Influence*: total number of distinct spots that can be reached, divided by the total number of moves. In Halma, Chess and Shogi this is an important factor to determine if a state is promising.
- *MobilitySimple*: number of moves that a player has in a current game state.
- *RegionProximity*: proximity of pieces to a particular region. In Halma, pieces need to reach the opponent's region.

Each game has a predefined set of these heuristics in Ludii learnt per game through gradient descent. These learned weights are often competent, but not overly strong for all states.

C. Experiment setup

Each of the following agents ran 100 trails against themselves on the games described in subsection IV-A:

1) *Random*
2) *Heuristic Sampling*: $HS_{1/2}$, $HS_{1/4}$ and $H_{1/8}$ with Same-Turn Continuation.
3) *Standard Alpha-Beta*: AB_1, AB_2 and AB_3
4) UCT: Standard UCT search with a budget of 1,000 iterations per move using uniformly random playouts and a default exploration constant of $C = \sqrt{2}$.

A limit of 1,000 turns was applied for all trials except for $Random$ search; any trial that exceeded this turn limit was abandoned as a draw and excluded from the sample.

Timings were taken on a standard consumer machine with six 2.9 GHz $i9$ cores, similar to machines that Ludii users performing reconstruction tasks will typically use.

V. Results

Early results showed that the Opportunistic AI took on average more than 3 times the amount of time it took Heuristic Sampling $1/2$ with Same-Turn Continuation to run a playout, without giving better results. Safe AI returned results in a much shorter period of time, but with results almost identical to Random playouts. They were hence excluded from every further experiment.

Results from experiments conducted to evaluate Heuristic Sampling can be found in Table I. $AB1, AB2$ and $AB3$ refer to Alpha-Beta search with depth $1, 2$ and 3. For each game, the expected length L_{exp} (in plies) is given as well as the observed length L_{obs} (in turns) of each agent, computed by the average of N completed trials N. The table also contains information on minimum and maximum observed lengths, standard deviation (SD), standard error (SE) and average time per trial in seconds. The observed length for each game is plotted in Fig. 3. The dotted red line represents the expected length of each game, when available.

Agents are ordered by least to most computations required to return a move, from Random to UCT. Fig. 5 shows the cumulative time it took in milliseconds to perform a playout for all games, on a logarithmic scale. As expected, the values are in increasing order. Adding a level of depth search in Alpha-Beta increases the runtime by a factor of around 11, and increasing the samples by a factor of two approximately doubles the runtime of Heuristic Sampling.

Random Playouts return the highest estimate for six out of nine games, with values disproportionately higher for Shogi and Halma. For the games of Tic-Tac-Toe, Chess, Connect4 and Shogi, more sophisticated agents return a value closer to the expected value. That is not the case for the game of English Draughts, where UCT gives an estimate more than twice as high as the expected value, and Nine Men's Morris, where Alpha-Beta of depth three returns results that are much worse than Alpha-Beta of depth two.

Fig. 6 and Fig. 3 reveal that HS without STC (SHS) performs almost no better than Random for the game of Nine Men's Morris with chained action sequences per turn. Without STC, increasing the subset of moves taken at each turn does not improve the observed value.

A sum of absolute errors ($|L_{obs} - L_{exp}|$) can be seen in Fig. 4. Results for English Draughts and Lines of Action where UCT returns an uncommonly high value result in UCT having a worse error than all Alpha-Beta agents and Heuristic Sampling of size $\frac{1}{2}$ and $\frac{1}{4}$.

VI. Discussion

Safe AI gave fast results, but returned estimates close to Random Playouts. Opportunistic Playouts did not meet the requirement of giving results in a short time. The main issue with Opportunistic Playouts is that their evaluation is useless for most of the states, in games where the winning or losing moves can only be reached after a certain number of moves. For board states where such moves do not exist, the agent wastes time iterating through all moves only to return a purely random one.

Heuristic Sampling on the other hand can run trials in less than half the amount of time than Alpha-Beta and more than two thousands times faster than UCT. It is no surprise that UCT and Alpha-Beta return better results than Heuristic Sampling for most games, although Heuristic Sampling with STC outperforms Alpha-Beta for Nine Men's Morris, and returned values are not too far apart.

Fig. 4 shows misleading results, as it makes it look like UCT performs much worse than Alpha-Beta. As explained in the section above, this is only due to two games where UCT returns an extremely large number: English Draughts and Lines of Action. A better metric should be found to better compare agents.

With a few exceptions, larger samples of moves give better results than smaller samples for Heuristic Sampling. The distance between estimates is generally fairly small, and smaller sampling ratios can be preferred when focusing on speed.

What is interesting to note is that Heuristic Sampling sometimes returns results faster than Random playouts. It is the case for Halma and Nine Men's Morris, where $Random$ playouts take a longer time to run as their expected number of

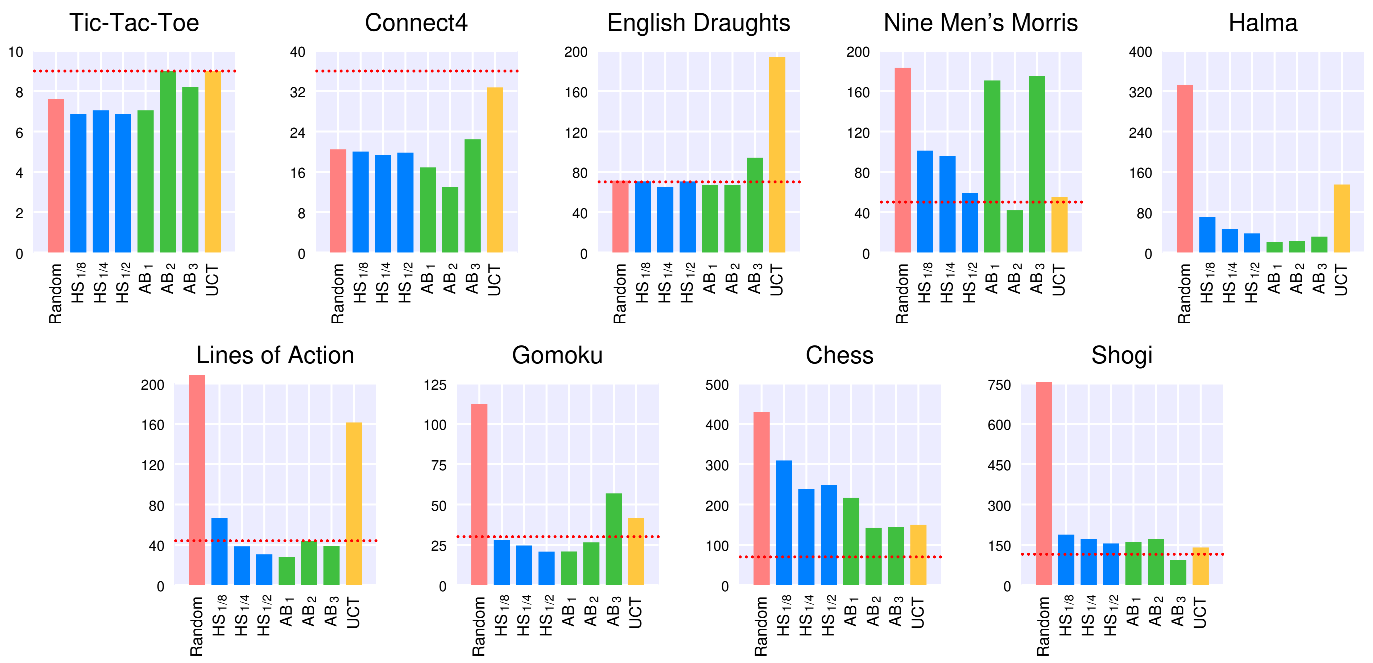

Fig. 3: Mean number of turns per playout for each method applied to each game [9].

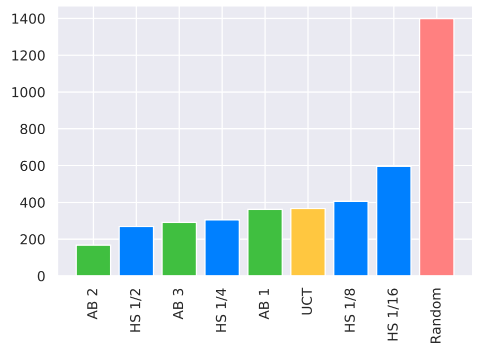

Fig. 4: Cumulative error (in moves) of each agent for all games.

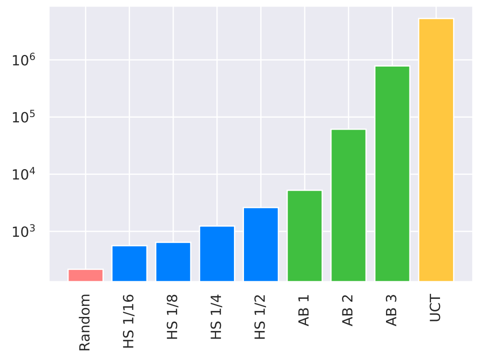

Fig. 5: Cumulative time (in ms) of each agent for all games.

moves is much higher. In general, Heuristic Playouts converge to a result more quickly than Random. Surprisingly, increasing the depth of Alpha-Beta search produces worse game length estimates for many games. A possible explanation for this is the "Odd-Even Effect" in Alpha-Beta search, in which odd search depths can give overly optimistic results, while even search depths can give overly pessimistic results.[2]

Same-Turn Continuation seems to give major improvements for games that have consecutive moves of the same player. As can be seen in Fig. 6 and 3, increasing the proportion of sampled moves does not improve the results of Heuristic Samplings without STC for Nine Men's Morris. The observed value without Same-Turn Continuation is very close to the one from Random Playouts.

[2]https://www.chessprogramming.org/Odd-Even_Effect

VII. Conclusion

While Safe AI and Opportunistic Playouts did not show any benefits for estimating game length, they did prove to be important stepping stones for developing Heuristic Sampling. Results suggest that Heuristic Sampling with Same-Turn Continuation produces game length estimates similar to UCT and Alpha-Beta, with speed comparable to Random Playouts.

Undersampling plays a major role in improving the speed of playouts but leads to overall slightly worse results. $HS_{1/2}$ and $HS_{1/4}$ should be considered for fast evaluation of games similar to the ones tested in the experiments, as they give similar results and can be picked depending on whether the main focus is speed or accuracy.

Same-Turn Continuation leads to stronger heuristic evaluations of boards for games such as Nine Men's Morris, with stronger moves contingent on sequence of weaker moves,

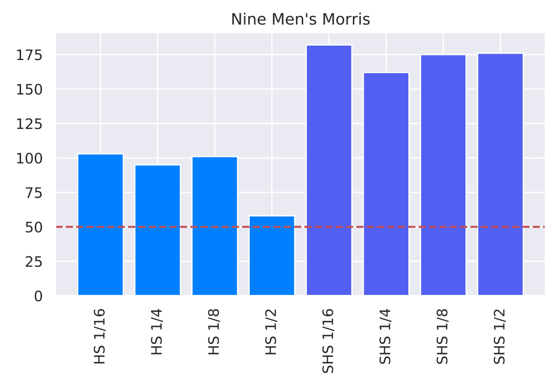

Fig. 6: Effect of STC on Heuristic Sampling for Nine Men's Morris.

providing better results to Heuristic Sampling than Alpha-Beta of depth 3.

A. *Future Work*

Although initial results seemed to disfavour Opportunistic Playouts for game evaluation, they should be properly compared to Heuristic Sampling for other game estimates than game length. What also turned out to be interesting is that given a one-second time limit, Monte Carlo Tree Search (MCTS) with Opportunistic Playouts of depth seems to outperform other agents including UCT for certain games, which suggests that MCTS with HS playouts could give even stronger results.

It is also important to evaluate the impact of training heuristic functions for a new game on the total time it takes to evaluate a game. This is expected to take a short period of time, and similar heuristics are likely to be shared by games with similar concepts [21]. Heuristic Sampling should also be tested on a wider range of games, with the focus on evaluating flawed rule sets.

Acknowledgement

Special thanks to my supervisor Cameron Browne for inspiring and guiding me through my thesis. Thanks to the Ludii team (Éric Piette, Matthew Stephenson and especially Dennis Soemers) for help developing the Ludii framework in which these experiments were run. The Ludii source code is provided as open-source under a CC BY-NC-ND 4.0 licence (https://github.com/Ludeme/Ludii).

References

[1] L. V. Allis. A knowledge-based approach of connect-four. *J. Int. Comput. Games Assoc.*, 11(4):165, 1988.

[2] L. V. Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.

[3] D. Beal. A Generalised Quiescence Search Algorithm. *Artificial Intelligence*, 43:85–98, 1990.

[4] W. Brice. A history of board-games other than chess. *The Journal of Hellenic Studies*, 74:219–219, 1954.

[5] C. Browne. The dangers of random playouts. *ICGA Journal-International Computer Games Association*, 34(1):25, 2011.

[6] C. Browne. Yavalath. In *Evolutionary Game Design*, pages 75–85. Springer, 2011.

[7] C. Browne. A problem case for uct. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(1):69–74, 2012.

[8] C. Browne. Ai for ancient games: Report on the digital ludeme project. *Künstliche Intelligenz*, 34(1):89–93, Mar. 2020.

[9] C. Browne and F. Barbero. Heuristic sampling for fast plausible playouts. *IEEE Conference on Games (CoG 2021)*, 2021 (submitted).

[10] C. Browne, D. Soemers, E. Piette, M. Stephenson, and W. Crist III. *Ludii Language Reference*. Dec. 2020.

[11] C. Browne, J. Togelius, and N. Sturtevant. Guest editorial: General games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(04):317–319, oct 2014.

[12] M. Campbell, A. Hoane, and F. hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57–83, 2002.

[13] D. Group. *The Way to Play: The Illustrated Encyclopedia of the Games of the World*. New York: Paddington Press, 1975.

[14] H. Iida, M. Sakuta, and J. Rollason. Computer shogi. *Artificial Intelligence*, 134(1-2):121–144, 2002.

[15] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General game playing: Game description language specification, 2008.

[16] J. Mallett and M. Lefler. Zillions of games: Unlimited board games & puzzles, 1998. Available at https://www.zillions-of-games.com/.

[17] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – The Ludemic General Game System. In *Proc. ECAI 2020*, pages 411–418. IOS Press, 2020.

[18] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[19] T. Schaul. An extensible description language for video games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):325–331, 2014.

[20] C. E. Shannon. Programming a computer for playing chess. pages 637–656. IEEE Press, 1993.

[21] M. Stephenson, D. J. Soemers, E. Piette, and C. Browne. General game heuristic prediction based on ludeme descriptions. *arXiv preprint arXiv:2105.12846*, 2021.

[22] F.-Y. Wang, J. J. Zhang, X. Zheng, X. Wang, Y. Yuan, X. Dai, J. Zhang, and L. Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[23] M. H. M. Winands. Informed search in complex games. 2004.

TABLE I: Mean number of turns per game, per playout method (reprinted from [9]).

Game	**Method**	L_{obs}	N	**Min**	**Max**	**SD**	**SE**	**Time**
	$Random$	7.62	100	5	9	0.142	0.278	0.000148s
	$HS_{1/8}$	6.88	100	5	9	0.132	0.259	0.000621s
Tic-Tac-Toe	$HS_{1/4}$	7.05	100	5	9	0.139	0.272	0.000640s
	$HS_{1/2}$	6.88	100	5	9	0.155	0.303	0.000769s
$L_{exp} = 9$	AB_1	7.05	100	5	9	0.185	0.363	0.000583s
	AB_2	9	100	9	9	0.0	0.0	0.000929s
	AB_3	8.22	100	7	9	0.098	0.192	0.00312s
	UCT	9	100	9	9	0.0	0.0	0.0294s
	$Random$	20.46	100	7	37	0.729	1.428	0.000226s
	$HS_{1/8}$	20.02	100	7	42	0.719	1.41	0.000736s
Connect4	$HS_{1/4}$	19.29	100	7	42	0.697	1.37	0.000709s
	$HS_{1/2}$	19.81	100	7	42	0.97	1.90	0.000996s
$L_{exp} = 36$	AB_1	16.88	100	11	29	0.564	1.11	0.00109s
	AB_2	13	100	13	13	0.0	0.0	0.00250s
	AB_3	22.45	100	7	42	1.204	2.36	0.190s
	UCT	32.75	100	17	42	0.601	1.178	0.403s
	$Random$	71.3	100	37	161	3.12	6.12	0.00192s
	$HS_{1/8}$	70.5	100	35	145	2.54	4.99	0.00572s
English Draughts	$HS_{1/4}$	65.2	100	33	197	2.41	4.71	0.00606s
	$HS_{1/2}$	70.5	100	37	139	2.25	4.4	0.00943s
$L_{exp} = 70$	AB_1	67.2	100	43	105	1.76	3.454	0.00948s
	AB_2	67.0	93	47	87	1.64	3.22	0.00784s
	AB_3	94.0	63	74	158	2.042	4.002	0.418s
	UCT	194.0	92	53	879	16.424	32.19	13.9s
	$Random$	183.2	100	56	790	11.801	23.13	0.00432s
	$HS_{1/8}$	101.0	100	36	431	6.22	12.2	0.00176s
Nine Men's Morris	$HS_{1/4}$	95.9	100	36	273	4.57	8.96	0.00215s
	$HS_{1/2}$	58.9	100	30	173	2.44	4.78	0.00349s
$L_{exp} = 50$	AB_1	170.6	100	43	532	10.2	19.9	0.00491s
	AB_2	41.9	100	20	65	0.909	1.78	0.00481s
	AB_3	175.2	84	44	963	19.9	39.1	0.377s
	UCT	54.8	100	31	149	2.22	4.34	6.86s
	$Random$	333.0	100	77	923	15.22	29.83	0.00435s
	$HS_{1/8}$	70.84	100	36	140	2.09	4.10	0.00163s
Halma	$HS_{1/4}$	46.1	100	26	97	1.19	2.33	0.00135s
	$HS_{1/2}$	37.8	100	20	71	1.011	1.98	0.00133s
$L_{exp} = ?$	AB_1	20.9	100	14	26	0.272	0.532	0.00116s
	AB_2	23.2	45	17	36	0.653	1.28	0.0463s
	AB_3	31.4	23	20	70	2.69	5.27	0.470s
	UCT	134.83	88	29	922	16.9	33.1	45.5s
	$Random$	208.3	100	49	417	7.734	15.158	0.00896s
	$HS_{1/8}$	66.6	100	32	150	2.518	4.94	0.00424s
Lines of Action	$HS_{1/4}$	38.5	100	23	77	1.02	2.00	0.00353s
	$HS_{1/2}$	30.6	100	23	43	0.418	0.82	0.00440s
$L_{exp} = 44$	AB_1	28.1	100	22	38	0.307	0.602	0.00762s
	AB_2	43.8	97	22	163	2.13	4.181	0.200s
	AB_3	38.8	98	23	73	0.778	1.53	1.24s
	UCT	161.0	100	30	518	10.6	20.7	184.4s
	$Random$	112.2	100	52	171	2.64	5.17	0.000353s
	$HS_{1/8}$	28.1	100	10	57	0.862	1.69	0.00360s
Gomoku	$HS_{1/4}$	24.6	100	10	43	0.708	1.39	0.00736s
	$HS_{1/2}$	20.8	100	10	43	0.608	1.19	0.0105s
$L_{exp} = 30$	AB_1	20.9	100	13	45	0.653	1.28	0.0125s
	AB_2	26.5	100	13	81	1.35	2.65	0.141s
	AB_3	56.9	100	11	113	2.73	5.36	2.28s
	UCT	41.47	100	20	70	1.12	2.20	0.904s
	$Random$	429.5	100	54	739	13.2	25.9	0.0243s
	$HS_{1/8}$	309.1	100	5	516	10.7	21.0	0.0317s
Chess	$HS_{1/4}$	238.0	100	16	471	11.1	21.7	0.0376s
	$HS_{1/2}$	248.6	100	16	435	10.8	21.2	0.0800s
$L_{exp} = 70$	AB_1	216.8	100	12	459	10.2	20.1	0.020s
	AB_2	142.3	100	22	501	9.58	18.8	0.564s
	AB_3	144.8	100	24	402	7.92	15.5	6.93s
	UCT	149.8	100	9	532	11.4	22.3	25.7s
	$Random$	756.4	100	100	2501	68.5	134.3	0.172s
	$HS_{1/8}$	187.9	100	75	587	9.5	18.6	0.593s
Shogi	$HS_{1/4}$	171.4	100	31	459	8.80	17.3	1.18s
	$HS_{1/2}$	155.0	100	22	533	9.14	17.9	2.50s
$L_{exp} = 115$	AB_1	161.1	100	38	871	10.8	21.2	5.17s
	AB_2	172.3	96	30	728	10.9	21.3	60.1s
	AB_3	94.0	10	38	114	8.26	16.194	751.5s
	UCT	140.1	10	77	218	14.2	27.7	4,790.9s