Heuristic Function Evaluation Framework

Nera Nešić and Stephan Schiffel(✉)

School of Computer Science, Reykjavik University, Reykjavik, Iceland
{nera13,stephans}@ru.is

Abstract. We present a heuristic function evaluation framework that allows to quickly compare a heuristic function's output to benchmark values that are precomputed for a subset of the state space of the game. Our framework reduces the time to evaluate a heuristic function drastically while also providing some insight into where the heuristic is performing well or below par. We analyze the feasibility of using Monte-Carlo Tree Search to compute benchmark values instead of relying on game theoretic values that are hard to obtain in many cases. We also propose several metrics for comparing heuristic evaluations to benchmark values and discuss the feasibility of using MCTS benchmarks with those metrics.

1 Introduction

Developing heuristics for games or other search problems typically involves a great deal of testing by playing the game repeatedly under realistic time constraints against several different opponents. This process is very time consuming and slows down the development of heuristics. In addition, this form of testing also gives little insight into where the heuristic has deficits.

We propose HEF, a heuristic function evaluation framework that is based on comparing the heuristic values of a sample of the state space to so-called ground truth values. Once the ground truth values for the samples are computed, HEF allows us to quickly evaluate different heuristics. Given sufficient samples and meta-data about those samples, it also allows to analyze in which position in the game the heuristics are accurate or in-accurate. We propose different metrics for this comparison that allow to focus on different aspects of the heuristic, e.g., whether it is more important to find the best move or the exact values of the moves. The source code of HEF is available online[1].

In this paper, we present HEF, as well as several metrics for comparing heuristics to ground truth values. We further analyze to what extent Monte-Carlo Tree Search (MCTS) can be used to compute the ground truth values in absence of game theoretic values. Finally, we show some results we obtained from analysis of a specific heuristic in the General Game Playing [5] domain using HEF.

[1] https://github.com/nnesic/HEF.

A. Plaat et al. (Eds.): CG 2016, LNCS 10068, pp. 71–80, 2016.
DOI: 10.1007/978-3-319-50935-8_7

2 Preliminaries

The development of HEF was driven by the need of evaluating heuristic functions for General Game Playing (GGP) [5]. As such, the implementation of the frameworks is aimed at deterministic, finite, perfect information games with an arbitrary number of players. However, the principles we used naturally extend to non-deterministic games or games with imperfect information. Only a few changes in the framework would be required to extend it to such games.

In particular, we assume the following three properties of games.

- **Finiteness:** The game has finitely many reachable states, each player has finitely many legal moves in each state, and the game ends after finitely many steps.
- **Determinacy:** The successor state is fully determined by the actions of the players and the predecessor state.
- **Perfect information:** All players have always sufficient information to infer the current state of the game.

In other words, we think of games as deterministic acyclic Markov decision processes.

We define a heuristic to be a function $h\colon S \times P \times M \to \mathbb{R}$ associating a real value with every legal move $m \in M$ for player $p \in P$ in state $s \in S$. It means that our assumption is that heuristics provide a value for each move as opposed to evaluating states of the game. In a turn-taking game, the evaluation of a move is the same as the evaluation of the successor state reached by that move. However, in games of simultaneous moves – which are often encountered in GGP – evaluating moves directly is often more convenient.

3 Heuristic Function Evaluation Framework

We propose an evaluation paradigm that examines a heuristic function's performance at a per-state level. This is done by identifying some features that a good heuristic should exhibit in each state (for example, the ability to accurately identify good moves and traps) and defining metrics which specify evaluation criteria for individual features. The heuristic function is then evaluated by comparing its output on a state to some ground-truth value for that state. To ensure a fast and flexible evaluation at development time, we pre-compute benchmarks of state-action pairs and their ground-truth values, and use these values for our metric evaluation. Below we discuss: structure (Sect. 3.1), proposed metrics (Sect. 3.2), and benchmark ground-truth value computation (Sect. 3.3).

3.1 Structure

We implemented our paradigm in the Heuristic Function Evaluation Framework (HEF), which provides all the necessary infrastructure to facilitate working with the paradigm. HEF offers utilities for benchmark generation and management,

data access, metric analysis, and data visualization, allowing users to focus only on defining the metrics that fit their study. HEF services are divided into three layers: benchmark management, metric analysis, and visualization.

The first layer stores and provides access to the benchmark datasets. New datasets can be imported from XML files containing benchmark information of states (such as state description, depth at which the state is found in the game, values of all available actions). Generating benchmarks requires two operations: selecting states to include in the benchmark, and computing the ground-truth values for each state.

HEF comes with a default implementation of a benchmark state selector and a Monte-Carlo tree search based ground-truth evaluator, both of which are based on the Game Definition Language (GDL) [7], allowing them to be used seamlessly on many different games, as long as they are encoded in GDL. The default state selector will somewhat randomly choose a specified number of states on each depth level of the game. The ground-truth evaluator then runs the MCTS algorithm on each selected benchmark state for a specified amount of time, outputting the results in the HEF benchmark format.

The second layer provides an analysis pipeline that is used by HEF metrics. This pipeline filters benchmark states according to player, depth, or other specifications, and passes them to the metric one by one. The heuristic function is then evaluated according to criteria specified by the metric, and it is assigned a score, which is then collected, aggregated across all examined states, and exported to the visualization layer. HEF is designed to allow users to easily define and use custom metrics.

The third layer provides a GUI, allowing the user to select metrics, games, and datasets, and to specify aggregation and visualization options. Since the topology of a game can change significantly throughout a game - for instance, the significance of some board properties or strategies can be more relevant in the endgame - all metric data is aggregated by depth level, allowing users to see how the heuristic's performance adapts to the progress of the game.

3.2 Proposed Metrics

We propose a set of basic metrics to be used with HEF, aimed mainly at Minimax and MCTS heuristics. Each metric specifies a *metric score function* that is used to evaluate different features of a heuristic's performance.

Definition 1 *(metric score function). Given an evaluation metric E, we define the metric score function, MS_E, as a function mapping a game state S and a role R according to policy specified by E to some value v.*

In this section we will briefly describe categories of our proposed metrics, and define the most prominent ones. As many of the metrics operate on game-theoretic values of benchmark states, in our definitions of metric score functions we will use $GTmax(S, R, M)$ to indicate the game-theoretical value of move

M for role R in state S, and $GTmax(S, R)$ and $GTmin(S, R)$, respectively, to indicate the maximum and minimum game-theoretical values of moves available to role R in state S.

Game Property Metrics are a set of measurements not bound to the heuristic functions, which instead keep track of how some properties of the game itself are changing at different depth levels. Of these, we find the *maximum score difference* metric particularly useful. This metric measures the ground-truth score difference between the best and the worst moves available to each player in a state, and helps us identify the "critical zones" of a game where a different choice of moves can lead to very different outcomes.

Definition 2 *(maximum score difference metric). For every state S and role R, we define the maximum score difference metric (DIFFMAX) as*

$$MS_{DIFFMAX}(S, R) = GTmax(S, R) - GTmin(S, R)$$

Best-Only Move Accuracy Metrics measure how accurately a heuristic can identify good moves in a state. They include the *K-best* metric, which requires a heuristic to identify at least one optimal move within the K moves it scores the highest, and *strict-best* metric, which also penalizes a heuristic for assigning a high score to bad moves.

Definition 3 *(k-best metric). Given a state S, role R, a heuristic function H, parameter $K \in \mathbb{N}$, and a set of moves for role R $HM_K = \{hm_1...hm_K\}$ containing K highest-scored moves according to H, we define the K-best metric (KBEST) as*

$$MS_{KBEST}(S, R) = \begin{cases} 1 \text{ if } \exists\ hm\ \text{—}\ hm \in HM_K \wedge GT(S, R, hm) = GTmax(S, R) \\ 0 \text{ otherwise} \end{cases}$$

Definition 4 *(strict best metric). Given a state S, role R, a heuristic function H, and a set of moves for role R $HM_K = \{hm_1...hm_n\}$ containing all moves hm_i such that $H(S, R, hm_i) = H(S, R, hm_1)$, where hm_1 is the highest-scored move according to H, we define the strict best metric (STRICT) as*

$$MS_{STRICT}(S, R) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1 \text{ if } GT(S, R, hm_i) = GTmax(S, R) \\ 0 \text{ otherwise} \end{cases}$$

We usually use the best-only metrics in conjunction with their random baselines (showing the optimal move selection accuracy that a heuristic which samples moves randomly would achieve) to identify areas of a game where the accuracy score is influenced by the position configuration (e.g., if all moves result in victory, random selection is 100% accurate).

Expected Score Metrics are used in applications where moves are chosen by sampling moves based on their heuristic scores according to some sampling function. As the name suggests, they calculate the expected score of employing the heuristic and sampling function together.

Definition 5 *(expected score metric). Let F be a move sampling function that maps a heuristic value of a move to the probability of choosing that move. Given a role R, a state S and a set $M = \{m_1...m_n\}$ of all moves available to R, and a heuristic function H, we define the expected score metric (EF) for function F as*

$$MS_{EF}(S, R) = \sum_{i=1}^{n} GT(S, R, m_i) \times F(H(S, R, m_i))$$

Move Ordering Metrics measure a heuristic's ability to not only identify the best moves, but also correctly distinguish between the quality of remaining moves. They do so by comparing the ordering of moves according to their heuristic scores to the ordering according to ground-truth values.

Move Categorization Metrics evaluate the heuristic's ability to produce scores that are similar to the ground-truth scores. These metrics are useful for applications which rely on the absolute value of the heuristic, e.g., using a heuristic as a state evaluation function for Minimax.

3.3 Benchmark Ground-Truth Value Computation

The metrics we proposed are intended to work with a benchmark of game-theoretical values for moves. Generating such datasets, however, may not always be feasible. Instead, we investigated the possibility of using Monte-Carlo tree search [2] (MCTS), in particular UCT, for benchmark generation. The algorithm uses random simulations and an exploitation-exploration policy to drive the exploration of the search tree towards most promising areas. MCTS has seen very successful implementations in game playing (e.g., in GGP [3] or Go [4]), where it is used to identify the best move in a state.

However, we can typically not run MCTS to convergence, which means we will not have the game theoretic values for the moves. Instead, we get q, the average score achieved by taking a move over all simulations, and n, number of times the move was included on the simulation path. Furthermore, MCTS focuses its simulations on the best moves. Thus, the average score q of a bad move is likely an unreliable estimate of the move's game theoretic value.

4 Using MCTS as a Benchmark

We investigated the viability of MCTS benchmarks for HEF analysis by comparing the output of our proposed metrics on a dataset with game-theoretic(GT) move values to their output on the MCTS benchmark for the same heuristic

function. We focused on the game of Connect Four, which has a sufficient large search space not to be trivial, while still being solvable in relatively little time. Both GT and MCTS datasets are composed of the same set of 498 game states. Values in the GT dataset were computed with an optimized game solver for Connect Four [9]. We compared the performance of two MCTS datasets, MCTS1H and MCTS3H, obtained through running respectively one and three hours of MCTS simulations per state. The number of simulations per state ranges from approximately 1.5 million (at depth 0) to 54 million (at depth 39) for MCTS1H, and 2 million to 151 million simulations for MCTS3H. Below we discuss the depth score differential (Sect. 4.1), the best-only metrics (Sect. 4.2), the expected score metrics (Sect. 4.3), and the move categorization and move ordering metrics (Sect. 4.4).

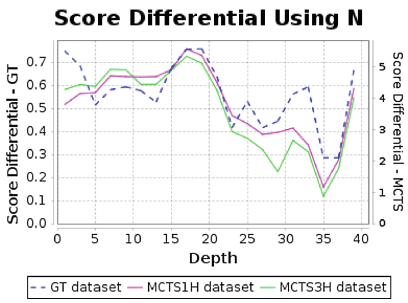
4.1 Depth Score Differential

In our first attempt of applying the depth score metric to MCTS datasets we used the q value in the benchmark directly for computing the score differential between best and worst moves. We observed that the result vaguely resembled the GT dataset results for the MCTS1H dataset (having identified the area of greatest score difference, although losing the amplitude of the difference), while the MCTS3H dataset preserved less information revealed by the GT dataset. We concluded that the q value is not reliable for this kind of metric, since its value gets diluted through simulations and it can stray far from its GT value for less explored moves. We had more success using n to calculate the differential by taking the logarithm of the ratio of highest to lowest n value in a state. As we see in Fig. 1a, the two MCTS curves follow the GT one closely, and the extra simulations in MCTS3H pay off in terms of capturing the behavior of the metric on a GT dataset. MCTS is driven by the intention to exploit the best available move as much as possible, so it is reasonable that the n value (counting the number of times a moves has been exploited) would be a good indicator of good and bad moves.

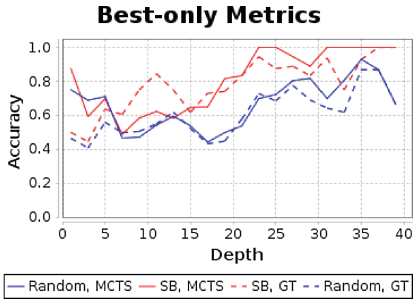
4.2 Best-Only Metrics

Best-only metrics rely on identifying a set of best moves in a benchmark state to compare the heuristic's choice of best moves to. In GT datasets the best moves are easily identifiable by their score, but with MCTS datasets identifying best moves becomes more challenging because of the unreliability of q as an estimator of the game-theoretic value. Therefore, we investigated a method of choosing best moves based on the n values: We order the moves according to n, and choose a "breaking point" that distinguishes the good moves where the difference between consecutive moves' n values is the highest.

Overall, we found the strict-best metrics to adapt to MCTS datasets quite well; Fig. 1b shows the benchmark comparison for the strict-best metric and its baseline, and we can see that both MCTS metrics follow closely the trends exhibited by the GT dataset.



(a) Performance of score differential metrics using the N value on MCTS1H and MCTS3H datasets.



(b) Performance of strict-best (SB) and random baseline metrics on MCTS3H dataset.

Fig. 1. Adapting HEF metrics to MCTS datasets

4.3 Expected Score Metrics

Expected score metrics proved more challenging to adapt to MCTS due to their strong reliance on values of moves. Using q values once again proved inadequate. The choice of sampling function had little impact on the expected score, and the MCTS metric failed to capture the information displayed by the GT metric. We then limited the sampling to the three moves with the highest q value, that is, we only selected among the best three moves based on their score and never selected any of the other moves. While the absolute values of the expected score were much lower in the MCTS dataset compared to the GT one, we observe the same trends and ratios between different sampling functions as we did in the GT dataset. Thus, useability of MCTS datasets for expected score metrics seems limited to cases where we only care about the general trends, but not the exact scores.

4.4 Move Categorization and Move Ordering Metrics

In order for the move categorization metrics to work on MCTS datasets, there needs to exist a mapping from benchmark MCTS values to the GT scores. We used several classifiers offered by Weka [6] to train a model which predicts the GT dataset score based on the q, n, total n, and depth values of a move in a state. The best accuracy of categorizing moves was of 67%, although the performance varied depending on players and depths – with the second player performing as low as 25% accuracy in certain states. With this much inaccuracy in the benchmark itself, we do not believe the move categorization metrics to be viable with MCTS datasets. Likewise, the move ordering metrics rely on the absolute ordering of moves in the benchmark – which, as we saw, is not the case with MCTS – and only achieved an accuracy of 57% in matching GT results.

5 Case Study: Analyzing Heuristics with HEF

Heuristics for General Game Playing need to be adaptable and applicable to many different games. Action heuristic [10], for example, is automatically constructed at the beginning of a match by regression on GDL rules of the game, and used in conjunction with MCTS to steer the random simulations towards more significant moves. The impact of the heuristic was evaluated on several games by pitching the heuristic-MCTS player against the pure MCTS one. In many games the use of heuristic produced in improvement in score for the heuristic player; in some cases it did not produce a difference, and in two cases it made the performance drop.

We used HEF to investigate the behavior of the Action heuristic on various games and to obtain a better insight into the results of the matches. Of particular interest are games of Pentago, where the heuristic player performed the best, winning approximately 73% of 400 test matches against the vanilla player, and Checkers-small, in which the heuristic player lost 90% of matches. To analyze these games, we precomputed benchmarks of 200 states for Pentago and 1100 for Checkers-small, using one hour of MCTS simulations per state. In Fig. 2 we show the score differential, strict-best, and random baseline metrics for the Action heuristic on these two games.

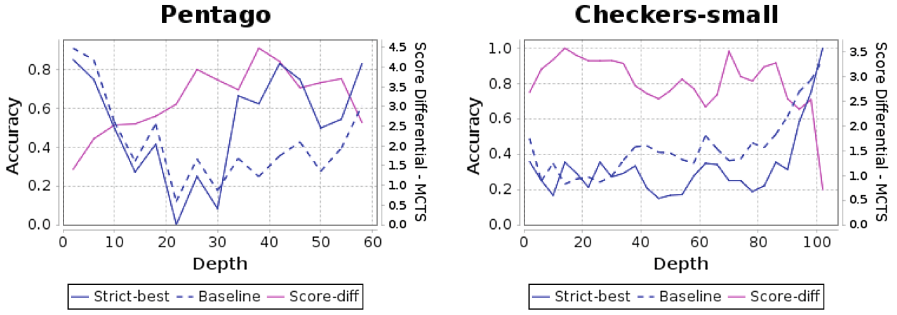


Fig. 2. HEF analysis of Action heuristic

In case of Pentago, we see that strict-best metric overtakes its random baseline strongly in the second half of the game. We also see that the accuracy of the heuristic peaks in sync with the score differential; this is a good thing to observe, as it tells us that the heuristic performs well in the critical areas of the game. Moreover, Pentago is a game with a large branching factor, reaching 38 moves available in some state. As such, the ability of the heuristic to identify good moves and steer the MCTS search in relevant directions has helped the heuristic player achieve much better results.

In the case of Checkers-small, we see that the strict-best metric consistently lags behind its baseline – telling us that the heuristic is actually preferring suboptimal moves. Moreover, we see that the accuracy drops in states at depths

65–85 where the depth differential metric exhibits the highest peak, meaning that at the critical points in the game the heuristic performs on its worst.

This kind of analysis gives us an insight into how the heuristic behaves at different points of the game, and lets us ask targeted questions, expressed in the form of metrics, to investigate various aspects of the heuristic. Having already pre-computed the benchmark, we can change and refine the heuristic and change the metrics as needed, and have the results of our analysis in a matter of minutes.

More case studies and more detailed results can be found in [8].

6 Related Work

In [1], Anantharaman proposes a position-based evaluation scheme for chess heuristics in which he compares the performance of the heuristic coupled with minimax search against the same heuristic and minimax search, but allowing much longer search time in the second case. Anantharaman then proposes several metrics to compare the performance of the heuristic relative to the reference program, and measures the correlation between these metrics and the USCF rating system, concluding that some of the metrics can be used reliably to evaluate heuristic functions, with only one day of computation needed.

Our approach to evaluating the heuristics is similar to Anantharaman's in that we are focusing on evaluating the heuristic's performance on individual states using some user defined metrics. Our emphasis is, however, on producing a paradigm which can recycle computation, while the approach proposed by Anantharaman still requires substantial search time to compute the reference for each specific heuristic. Moreover, we aim for generality of use – we would like our paradigm to be applicable to functions designed to be used with a variety of search algorithms (we can, for example, analyze functions intended for both minimax and MCTS random search guidance) and a large variety of games.

7 Conclusion

We proposed a paradigm for evaluating heuristics in games and we presented a framework that implements it. Our main goal was to provide a fast and focused evaluation method for intermediate stages of heuristic function design. We achieve this by pre-calculating a benchmark dataset containing a set of states that have a value assigned to each action, and using this dataset to answer questions about the heuristic's performance – such as evaluating the accuracy with which it identifies the best move. The Heuristic function Evaluation Framework provides all the infrastructure needed to work with this paradigm, from benchmark generation to data visualization; the users only need to formulate their questions in the form of metrics which verify the heuristics' answers against the benchmark values.

For cases where game-theoretic values are not available, we propose using Monte Carlo Tree Search for generating benchmark datasets. These are less accurate, but easily obtainable. We found that these datasets are usable when we

are interested in identifying the best moves in a state, while we do not recommend using them if the correct ordering of moves or estimating the value of a move is of importance. So far, these conclusions are based only on the results in Connect Four. Studying more games is necessary to confirm these findings.

Finally, we presented a case study showing how HEF can provide useful insights into the behavior of heuristics. The conclusions are again not definitive.

Future Work. Currently, we leave the responsibility of deciding the significance of various metrics for given applications to the user of the paradigm. For example, the move categorization metrics are not overly relevant for the heuristics intended to find best moves for MCTS random simulations, so achieving a high score under these metrics does not mean that the heuristic will perform well in the actual MCTS search. In the future, we would like to investigate methods for determining the correlation between scores assigned by a metric to a heuristic and the actual performance of the program on relevant problems.

References

1. Anantharaman, T.: Confidently selecting a search heuristic. ICCA J. **14**(1), 3–16 (1991)
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
3. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence. AAAI Press (2008)
4. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Research report RR-6062, INRIA (2006)
5. Genesereth, M.R., Love, N., Pell, B.: General game playing: overview of the AAAI competition. AI Mag. **26**(2), 62–72 (2005)
6. Holmes, G., Donkin, A., Witten, I.H.: Weka: a machine learning workbench. In: Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 357–361. IEEE (1994)
7. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General game playing: Game description language specification. Technical report, Stanford University, Recent Version, March 2008. http://games.stanford.edu/
8. Nešić, N.: Introducing heuristic function evaluation framework. Master's thesis, Reykjavik University (2016). http://hdl.handle.net/1946/23743
9. Tromp, J.: John's connect four playground. https://tromp.github.io/c4/c4.html. Accessed 24 Nov 2015
10. Trutman, M., Schiffel, S.: Creating action heuristics for general game playing agents. In: The IJCAI 2015 Workshop on General Game Playing, pp. 39–47 (2015)