Multigame playing by means of UCT enhanced with automatically generated evaluation functions

Karol Walędzik and Jacek Mańdziuk

Faculty of Mathematic and Information Science,
Warsaw University of Technology,
Pl. Politechniki 1, 00-661 Warsaw, Poland,
{k.waledzik,j.mandziuk}@mini.pw.edu.pl

Abstract. General Game Playing (GGP) contest provides a research framework suitable for developing and testing AGI approaches in game domain. In this paper, we propose and validate a new modification of UCT game-tree analysis algorithm working in cooperation with a knowledge-free method of building approximate evaluation functions for GGP games. The process of function development consists of two, autonomously performed, stages: *generalization* and *specification*.

1 Introduction

Games have long been a fascinating topic for Artificial Intelligence (AI) and Computational Intelligence (CI) research. Majority of spectacular accomplishments of AI in games, however, lacked *universal learning mechanisms* (most of the top playing programs in classical board games do not apply any learning whatsoever) and *generality of approach* also known as *multigame playing ability*.

In this paper we adopt autonomous learning approach to building evaluation function for the General Game Playing (GGP) contest [6]. Our approach interweaves generalization mechanism, which allows building a large pool of candidate features, with specification stage (which selects a reasonable subset of pertinent features). Both stages are performed without human intervention as they are based on generally applicable heuristical meta-rules. What is more, contrary to approach of many GGP competitors [16, 9], our method operates strictly on game descriptions only, without any implicit expectations about their structure or rules, such as expecting them to be played on boards, use pieces and so on.

The evaluation function devised based on the above described subset of features is subsequently employed by what we call a Guided UCT algorithm - our modification of the state-of-the-art UCT tree search method described in section 3. Results of simulations performed in three game domains: chess, checkers and connect-4 prove a clear upper hand of the enhanced UCT method over its plain version, even in the case of not restrictive time regime.

While, due to space limitations, the description of some parts of our solution must remain very brief and omit all non-crucial details, we invite users who

find our approach interesting to acquaint themselves with the full version of this article available at AGI'2011 conference webpage.

2 General Game Playing Competition

GGP, one of the latest and most popular approaches to the multigame playing topic, was proposed at Stanford University in 2005 in the form of General Game Playing Competition [8]. General Game Playing applications are able to interpret game rules encoded in Game Description Language (GDL) [11] statements and devise a strategy allowing them to play those games effectively without human intervention. Game states are represented by sets of facts while algorithms for computing legal moves, subsequent game states, termination conditions and final scores for players are defined by logical rules.

3 UCT

Upper Confidence bounds for Trees (UCT) is a simulation-based game playing algorithm that proved to be quite successful in case of some difficult game-based tasks, including Go [4] and GGP tournament (being employed by two-times-in-a-row champion CadiaPlayer [3]). In each game state UCT advises to first try each action once and then, whenever the same position in encountered again, choose action according to the following formula: $a* = argmax_{a \in A(s)}\{Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}}\}$, where $A(s)$ denotes the set of all actions possible in state s, $Q(s,a)$ – average return of the state-action pair so far, $N(s)$ – number of times state s has been visited by the algorithm and $N(s,a)$ - number of times action a has been selected in state s. In realistic cases it is of course impossible to store information about all game states in memory at the same time, therefore the in-memory tree is actually expanded according to a kind of best-first strategy and paths below it are sampled via traditional Monte-Carlo simulations only.

4 Guided UCT

UCT algorithm in its basic form requires no expert game-specific knowledge. We, however, investigate the possibility of augmenting UCT with **automatically inferred** game-specific state evaluation function. Approaches similar in idea, but very different in realization, have been employed by several programs, e.g. aforementioned CadiaPlayer [3] and MoGo [5]. For the sake of clarity, we will refer to any version of our augmented UCT algorithm as Guided UCT (GUCT).

Once defined, the evaluation function $F(s)$ can be employed by the GUCT method in several ways, both in strict UCT and Monte-Carlo simulation phases. $Q(s,a)$ can be redefined as a weighted average of the evaluation function value and the current simulation results or, alternatively, F can be used to pre-initialize the data stored in the game tree built by the UCT whenever a new node is

added to it. F can also be used to influence the probability of selecting possible actions in the Monte-Carlo phase so that it is (in some way) proportional to their estimated value. In yet another approach the routine can be modified so that in each and every state there is a (relatively low) probability that the simulation will be stopped and the evaluation function's value returned instead. This approach, relying on the expectation that the $F(s)$ values are more reliable for positions closer to the end of game, can lead to improvements in algorithm speed and, thus, allows for increasing the number of simulations.

5 Evaluation function

Due to the nature of GGP environment, there is virtually no practical way of including significant expert domain-specific knowledge in the program itself. The evaluation function must be automatically generated by some kind of AI-based routine. Still, some GGP agents' developers choose to specifically tune their application towards certain classes of problems, expecting tournament organizers to be inspired by real-world human games incorporating concepts such as boards, pieces and counters [10, 16, 9]. In our application, however, we decided to concentrate on developing the evaluation function in as knowledge-free a manner as possible and with as few preconceptions as possible. We construct the function as a linear combination of a number of numerical characteristics of game states called *features*. Features are by their nature game-specific and are inferred from the game rules by a set of procedures described in the following sections.

5.1 Features generation

Our approach to game state features identification was inspired by prior work in the GGP area, most notably [10], [2] and [16]. We aim to obtain features represented by expressions similar to those in GDL, e.g. (*cell* ?x ?y b). In order to find the value of such a feature in a given game state, we would attempt to find all values of ?x and ?y variables for which this expression would be true. The number of solutions to the expression is considered the feature value.

Finding the initial set of possible features consists in simple analysis of the game definition (in GDL) and extraction of all suitable statements directly from it. Afterwards, we proceed to the generalization phase, i.e. generate new features by replacing all constants in existing expressions with variables, generating all possible combinations of variables and constants. Next, we want to specialize the features, i.e. generate features containing less variables than those in the original set. In order to do this in a reasonable way, without generating a huge number of features that would by definition always have zero value, we need to identify valid domains of each and every argument of the predicates we try to specialize. We do it in a simplified and approximate way, according to a routine inspired by [16], relying on identification of how variables are shared between predicates.

Once a set of potential features has been generated, we perform some simple simulations in order to analyze them and compute a number of statistics.

Two of the statistics gathered at this point require more attention. Firstly, we calculate each feature's correlation with the expected final score for each player. Secondly, we calculate a characteristic called stability. Stability reflects the ratio of feature's variation measured across all game states to average variation within randomly generated game sequences. The idea is that more stable features are more promising components of the evaluation function.

5.2 Evaluation function generation

Having identified a set of potential game state features, the last step in building a linear evaluation function is selection of the most useful of them and assigning a weight to each of them. While we plan to employ more advanced CI-based approaches to this problem, as the first phase of our research we decided to employ for the task a very simplistic heuristical approach in order to validate the feasibility of our ideas. The actual procedure we use for building the evaluation function first orders the features by the minimum of their stability and absolute value of their correlation with the final score (we prefer both these characteristics to be as high as possible) and then rejects all but the first 30 features (out of several thousand available). The linear combination of those features is created by assigning them weights equal to the product of their stabilities and correlations with the final score.

6 Experiment

In order to test the quality of the GUCT algorithm in cooperation with the simple generated evaluation functions, we decided to run a small competition comparing players using GUCT and UCT in 3 games of various complexity: connect-4, checkers and chess. All game definitions have been downloaded from [7].

For the sake of fair comparison, both competing agents were based on the same single-threaded implementation of the UCT algorithm. GUCT player made use of the evaluation function only in the Monte-Carlo simulations phase, stopping the simulation and using the evaluation function's value as the result with the probability of 0.1 in each searched node. All in all, the tournament consisted of 60 matches in total, 20 matches for each game - 4 per time limit for move of 1s, 10s, 15s, 30s and 60s. Players swapped sides after each game. GUCT player regenerated its evaluation function from scratch before each and every match. Each player was rewarded 1 point for a victory, -1 point for a loss and 0 – for a draw.

The tournament results for GUCT player are presented in figure 1. Please keep in mind that any score above 0 indicates player's supremacy over plain UCT approach. First and most obvious observation here is that, considering the simplicity of the evaluation function generation procedure, GUCT player fares unexpectedly well, significantly outperforming its opponent in all games.

More detailed analysis of the results leads to two interesting observations regarding the dependence of the algorithm's performance on time limit per move.

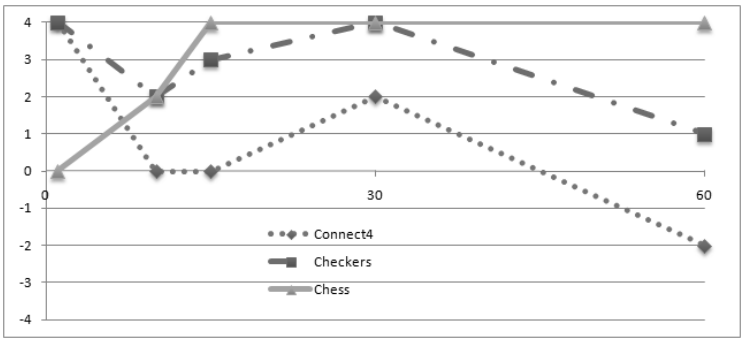

Fig. 1. *GUCT* player scores for each game depending on time allotted for a move (in seconds)

Both of them can only be treated as hypotheses considering limited experimental data but anecdotal data gathered during development and preliminary testing of the system strongly supports them as well.

Firstly, in case of very low time limits and sophisticated games, results of the games are often insignificant (typically being a draw), as neither player has enough time for analysis to play in a reasonable way. This effect is clearly visible in the case of chess. At the same time, as the time limit per move is increased, another effect can be observed – especially in the case of simpler games (e.g. connect-4). While the UCT player is able to perform more and more simulations, obtaining more and more precise results, GUCT-based agent still heavily relies on the very rudimentary evaluation function, whose quality remains constant.

7 Conclusions and future research plans

As presented above, the experiments we have performed so far, strongly suggest that our approaches to both modification of the UCT tree search algorithm and automated game-independent process of creating evaluation function have high potential. Our feature-building strategy follows two principles typical for human thinking: generalization and specialization. While the former process is useful for generating new concepts by ignoring certain details of the problem aspects, the latter allows applying the concepts to specific situations and finding special cases and exceptions to the rules. It is the unique synergy of the two approaches that facilitates solving even seemingly distant and unrelated tasks.

At the moment, our immediate research plans include two paths of further system development. Firstly, we intend to further enhance feature generation system by including compound features, defined as differences or ratios of related simple features. Secondly, we are working on more sophisticated, CI-based methods of evaluation functions generation. In the immediate future, we consider employing co-evolutionary and/or Layered Learning [13] schemes, as well as replacing linear evaluation functions with artificial neural networks.

References

1. P. Auer, N. Cesa-Bianchi, P. Fischer: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2/3), pages 235–256, 2002
2. J. Clune: Heuristic evaluation functions for General Game Playing. In Proceed- ings of the Twenty-Second AAAI Conference on Articial Intelligence (AAAI- 07), pages 1134–1139, Vancouver, BC, Canada, 2007. AAAI Press.
3. H. Finnsson, Y. Björnsson: Simulation-based approach to General Game Playing. In Proceedings of the Twenty-Third AAAI Conference on Articial Intelligence (AAAI-08), pages 259–264, Chicago, IL, 2008. AAAI Press.
4. S. Gelly, Y. Wang: Exploration exploitation in Go: UCT for Monte-Carlo Go. In Neural Information Processsing Systems 2006 Workshop on On-line trading of exploration and exploitation, 2006
5. S. Gelly, Y. Wang, R. Munos, O. Teytaud: Modification of UCT with patterns on Monte Carlo Go. Technical Report 6062, INRIA, 2006
6. General Game Playing website by Stanford University. http://games.stanford.edu/.
7. General Game Playing website by Dresden University of Technology. http://www.general-game-playing.de/.
8. M. Genesereth, N. Love: General Game Playing: Overview of the AAAI Competition. http://games.stanford.edu/competition/misc/aaai.pdf, 2005.
9. D. Kaiser: The Design and Implementation of a Successful General Game Playing Agent. In Proceedings of FLAIRS Conference, pages 110–115, 2007
10. G. Kuhlmann, K. Dresner, and P. Stone: Automatic heuristic construction in a complete General Game Player. In Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence (AAAI-06), pages 1457–1462, Boston, MA, 2006. AAAI Press.
11. N. Love, T. Hinrichs, D. Haley, E. Schkufza, M. Genesereth: General Game Playing: Game Description Language Specification. http://games.stanford.edu/language/spec/gdl_spec_2008_03.pdf, 2008.
12. J. Mańdziuk, *Knowledge-Free and Learning-Based Methods in Intelligenet Game Playing*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer-Verlag, 2010, vol. 276.
13. J. Mańdziuk, M. Kusiak, K. Walędzik: Evolutionary-based heuristic generators for checkers and give-away checkers. Expert Systems, 24(4): 189–211, Blackwell-Publishing, 2007.
14. J. Reisinger, E. Bahçeci, I. Karpov and R. Miikkulainen: Coevolving strategies for general game playing. In Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG'07), pages 320–327, Honolulu, Hawaii, 2007. IEEE Press.
15. J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317:1518–1522, 2007.
16. S. Schiffel, M.Thielscher: Automatic Construction of a Heuristic Search Function for General Game Playing. In Seventh IJCAI International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC07).