Multi-Game Playing - a Challenge for Computational Intelligence

Jacek Mańdziuk
Faculty of Mathematics
and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw, POLAND
Email: mandziuk@mini.pw.edu.pl

Yew Soon Ong
School of Computer Engineering
Nanyang Technological University,
Singapore
Email: asysong@ntu.edu.sg

Karol Walędzik
Faculty of Mathematics
and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warsaw, POLAND
Email: k.waledzik@mini.pw.edu.pl

***Abstract*—In this paper, we deal with the topic of multi-game playing, i.e. creating agents able to autonomously learn to play any game within some arbitrarily defined genre. We describe the motivation for research in this particular area and briefly summarize the most important undertakings. Subsequently, we present a relatively young multi-game playing platform, called the General Game Playing (GGP) and our approach to development of GGP agent followed by some experimental results.**

***Index Terms*—Multi-game playing, General Game Playing, Human-like playing, Challenges to Computational Intelligence.**

I. Introduction

Early approaches of Artificial Intelligence (AI) to game playing were mainly focused on the development of universal playing capabilities, to a large extent similar to those exhibited by human players. Hence, the general idea was to develop learning algorithms and search mechanisms which could be widely applicable to the whole genres of games rather than particular games [1], [2]. Having been provided with the rules of the game, the artificial playing system was supposed to master the game with no need for human intervention of any kind (perhaps except for having human players as external opponents/trainers). This approach was stimulated by the observation that many individuals are capable of playing multiple games efficiently, some of them at the master level. In other words, mastering one game does not prevent humans from learning other games, even at a very high competency level. On the contrary, having some experience in playing games belonging to a particular genre (e.g. two-player board games or card games) makes learning subsequent games easier, since the basic notions or "game equipment" (boards, pieces, cards, suits, etc.) are already acquainted by the learner [3], [4].

While the multi-game approach seems to have been successful in some aspects (e.g. universal search tree algorithms) it did not prove to be effective for autonomous construction of the evaluation function. Potentially infinite pool of possible playable games based on various equipment (various types of cards, boards, pieces, game goals, etc.) makes the task of generating the evaluation function - in a universal form - extremely demanding. This was one of the reasons why early attempts in 1950s and 1960s were generally unsuccessful. In fact, the first truly multi-game playing systems originated around 1990s and their appearance have brought about a new resurgence of interest in the field. Due to the wide scope of the problem, these systems were usually devoted to games of a certain genre, most often two-player, deterministic, zero-sum, perfect-information ones.

The goal of this paper is threefold: first, we provide a brief summary of past AI attempts in the area of multi-game playing and then present a new, challenging research area named General Game Playing (GGP). From our survey, we point out several challenges for Computational Intelligence (CI) methods within the GGP field. In particular, we highlight the potentials in exploiting the abundant, readily available computational power for advancing search/simulations with the long-term goal of building a strong assessment function for the game of interest, in contrast to the typical practice of expending it on pure brute-force search/simulations. We hope that our brief survey of GGP and the challenges discussed will help crystallized some of the interesting issues for the field of CI. Last but not least, we propose a new approach to GGP, which addresses one of the pointed challenges, by employing simulations to the process of autonomous construction of the evaluation function for a given GGP game. Promising results are presented for five tested games.

In the next section, some renowned AI approaches to multi-game playing dating back to the 1990s are briefly summarized. These include the SAL system [5], Hoyle [6], [7], Morph II [8] and METAGAMER [9], [10]. Section III provides an overview of the General Game Playing competition and introduces some approaches used for the construction of GGP agents. In section IV, several challenging issues for CI methods within the framework of GGP are proposed and discussed. Moreover, we propose and experimentally verify a new approach to building the GGP agent, which relies on simulation-based construction of the evaluation function devoted to a particular game being played. The last section presents the conclusions and outlines directions for future research.

II. Early AI approaches

In this section, four interesting approaches to multi-game playing, which originated in the 1990s, are briefly summarized.

Special emphasis is put on aspects of generating the evaluation function in a fully automated process.

A. SAL

SAL (Search And Learning) system [5] was proposed by Michael Gherrity in 1993. SAL was able to learn how to play any deterministic, two-player game with perfect information. The general knowledge about the above-mentioned genre of games was represented in a game-independent kernel (module), which was pre-defined and remained unchanged during learning. The rules of a particular game to be learned/played were provided by a game-specific module.

The system generated two evaluation functions (separately for each of the playing sides) and allowed handling of non-symmetric (non zero-sum) games, as well. Both functions were represented by Multilayer Perceptrons (MLPs) with one hidden layer.

The evaluation functions were trained based on some pre-defined, generally-applicable binary features defined in the kernel. The learning algorithm was a combination of backpropagation [11] and TD-learning [12], [13]. Feature set included mainly three types of elements: positional features (related to the location of particular pieces on the board, their quantities, etc.), dynamic (non-positional) features (related to the last move that was performed, i.e. capturing/non-capturing move, type of a piece that was moved/captured, etc.), and rule-based features (related to potential strengths and weaknesses of a given position, e.g. threatened pieces, controlled squares, etc.).

The size of the MLP input layer was equal to the number of binary features defining game positions. The hidden layer was 10-times smaller than the input counterpart. Each move made during the training game defined one learning example, with the target evaluation function value calculated using the TD(λ) algorithm. The MLPs were trained in a batch manner, i.e. after completion of the whole game.

Although, in principle, SAL project could have been considered successful (the system was actually capable of playing many games of a certain genre), in practice its accomplishments were strongly hindered by extremely slow learning. For example, it required as many as $20\,000$ training games to learn tic-tac-toe and $100\,000$ games to reach a decent play level in connect-4. SAL also learned to play chess, though at a very low level (not much better than random). Due to the extremely long learning process, SAL is unlikely to play any non-trivial game even at a moderate level.

B. Hoyle

Another example of universal, game-independent learning is Hoyle [6], [7], [14] - a system devised by Susan Epstein. Similarly to SAL, Hoyle was devised to learn any deterministic, two-player game of perfect information played on a finite board, assuming the rules of the game had been provided. The system used a specific training scheme known as *lesson and practice* [7], in which lessons (i.e., games played against the expert) were interleaved with practicing periods (self-playing).

At the heart of Hoyle there was a set of universal (i.e., game-independent) *Advisors* specializing in specific positional or material aspects (e.g. finding the winning sequence of moves, calculating material advantage, etc.). Advisors had the right to recommend particular moves to be played, which were subsequently commented by other Advisors from the points of view of their specializations. The Advisors were arranged in three tiers [14].

First-tier ones performed shallow search so as to provide their definite opinions about individual moves (whether they should be played or not). For instance, the role of Sadder Advisor was to avoid immediately losing moves. The second-tier Advisors assessed particular plans of play, i.e. advocated sequences of moves leading to accomplishing certain goals. The Advisors belonging to the third layer voted for or against individual moves on the basis of their expert heuristic assessment. The Advisors in the first two tiers were ordered and made decisions in a sequential manner. In case any one of them had selected the next move, its decision was final and could not be modified or canceled by the remaining Advisors. The last-tier Advisors made their decision in parallel and applied a weighted voting scheme. Generally speaking, the diversity of Advisors was a crucial aspect in efficient learning of a new game.

Another crucial issue in the Epstein's system was assignment of weights in the third tier, which were generated with the help of so-called Probabilistic Weight Learning (PWL) algorithm [14]. In short, a weight learned by PWL reflected the estimated probability that the opinion provided by the respective Advisor was correct.

Hoyle's efficacy was demonstrated in 18 two-player board games, including tic-tac-toe, lose tic-tac-toe and nine men's morris [7], [14], though its potential in complex games has never been proven.

It is worth noting that the decision making processes implemented by Hoyle was, in some aspects, similar to that of humans. Hoyle analyzed the obviously strong/weak moves (in the first tier), before considering the remaining ones. Furthermore, the final decision made in the third tier reflected the result of a negotiations process among the Advisors, each of which represent a specific game concept that potentially conflict or support one another.

C. Morph II

Yet another interesting example of game-independent learning system is Robert Levinson's Morph II [8]. The system was an extension of Morph [15] - an adaptive pattern-based chess learning/playing system. Morph II employed several CI techniques (e.g. neural network-like backpropagation learning and genetic evolution) combined with symbolic learning.

The system maintained a database of patterns related to learned games and autonomously derived new features and patterns from the game rules by means of a small set of universally-defined mathematical transformations. It also autonomously determined the weights assigned to these "discovered" features (patterns) with the use of specifically de-

signed learning procedure. Learning parameters were adjusted dynamically by the system, based on its experience [8].

Morph II demonstrated intermediate-level playing abilities in chess, which was - in some sense - treated by its author as the main target. However, even though the project was directly related to designing a domain-independent, multi-task learning system, its performance in domains other than chess was not proven.

D. METAGAMER

METAGAMER as developed by Barney Pell [9], [10], [16] is another example of multi-game playing system. The system was applicable to a class of "symmetric chess-like (SCL)" games, specifically defined in Pell's papers. Popular examples of SCL games are chess, checkers or shogi.

METAGAMER was equipped with game-independent, universal evaluation function in the form of a linear combination of simple, pre-defined features (partial goals). Basing on the, externally provided, rules of the game to be played, the system was capable of building an efficient representation of game's positions and constructing game-related evaluation function, which was then used by a generic search engine (implementing alpha-beta search with iterative deepening). Domain knowledge was restricted to general characteristics of SCL games. Playing a particular game involved game-specific optimization, which was performed without human intervention. Similarly to Hoyle, METAGAMER employed some number of advisors which decided about potential usefulness of a given game aspect (feature) for one of the playing sides. Contrary to Hoyle, however, METAGAMER's advisors could formulate only positive opinions (a negative one was achievable by stating a favorable comment for the opposite side). Advisors were defined in the form of heuristic rules which returned numerical estimations of the worth of the respective game aspects (features). These values were ultimately combined into the overall estimation of a given position. Among the 23 advisors, there were 4 that focus on *mobility* issues, 4 related to *threats and capturing*, 4 responsible for *goals and step functions*, and 11 devoted to *material* values, respectively [17].

METAGAMER was used by Pell mainly for material analysis, that involved estimations on the relative strength of checkers (man vs. king) and chess pieces. Some preliminary experiments with playing chess and checkers were also performed. The system was pitted against GNU Chess and Chinook [18], respectively (both versions were of the year 1992). Pell reported [17] that METAGAMER was about even to the easiest level of Chinook when given a one man handicap.

In chess, METAGAMER appeared to play evenly to the GNU Chess playing at level 1 when given a knight handicap (level 1 means 1 ply search with possible extensions in non-quiescent positions). Likewise METAGAMER's search was 1 ply deep with occasional 1 ply extensions.

One of the main unresolved problems of METAGAMER lies in the manual assignment of weights to advisors' opinions and the lack of autonomous mechanism to perform negotiations among them (as was the case for Hoyle). This issue was left for further research, but to the best of our knowledge was never satisfactorily addressed. In the above-mentioned preliminary tests all weights were simply set to 1, but this is clearly not an optimal choice for all games. In our opinion, this issue could be approached based on some kind of self-playing procedure similarly to Samuel's [19] and Tesauro's [20], [21] experiments with checkers and backgammon, respectively.

Another missing information is related to METAGAMER's performance when playing some *ad-hoc* defined SCL games. Positive conclusions in this area would have ultimately proven the quality of this concept. Some preliminary experiments of this type against random opponents and against METAGAMER's clones were reported by Pell [10], [17], but overall they have been inconclusive.

E. Summary of previous approaches

Two of the four above-described systems, SAL and Morph II, did not prove to be truly universal multi-game playing systems, except for very simple games (especially in the case of SAL). Likewise, METAGAMER has not been convincingly verified in domains other than chess and checkers, where it presented itself as a rather weak player. Only Hoyle's construction was universal enough to be applicable to a range of 18 games, though most of them are relatively simple.

In spite of the substantial efforts devoted to the development of multi-game playing agents in the 1950s-1960s and then in the 1990s, their performance has not been impressive. Actually, brute-force search methods supported by hand-crafted evaluation functions, large opening and end-game databases proved to be more successful and capable of convincingly defeating human world champions in most of the popular board games (e.g. chess, checkers, Othello, backgammon, and others), with Go being one of a few notable exceptions. The 1995-2005 decade was definitely a time of single-purpose game playing programs with the exceptional achievements of Deep Blue [22] in chess, Chinook [18] in checkers, Logistello [23] in Othello, Maven [24] in Scrabble, and many other game playing systems.

Despite the great admirations for the world human-computer champion programs specialized for individual games, it should be underlined that majority of them lack *universal learning abilities* or, actually, do not employ *any form of learning*. These champion programs represent a "narrow AI" approach and, as such, are not applicable to any other game than the one they have been designated for. These two facets (*improvement by learning* and *general applicability*) are, on the other hand, clear hallmarks of human game playing and human intelligence in general.

III. General Game Playing Competition

In the above-described perspective of the success of narrow AI paradigm, quite surprisingly "the old idea" of early AI was resurrected in 2005 by means of the annual General Game Playing (GGP) competition proposed at Stanford University [25], [26]. Technically, as stated in [25] "General game players are systems able to accept declarative descriptions of

arbitrary games at runtime and able to use such descriptions to play those games effectively (without any human intervention)." Games (their rules and goals) are defined in Game Description Language (GDL) [27] (which is based on a subset of Prolog) in the form of logical sentences that must be true in every state of the game.

The capacity of GDL allows for the definition of *finite, discrete, synchronous, multi-player, perfect-information* games. Following [26], an instructive, condense example of GDL-based tic-tac-toe coding is presented in figure 1, where **role(**x**)** defines x to be a player; **init(**f**)** states that f holds in the initial state; **true(**f**)** states that f holds in the current state; **does(**x**,** m**)** states that x performs move m; **next(**f**)** states that f will be true in the next state (reached after all actions defined in $does$ relations are performed); **legal(**x**,** m**)** states that in the current state move m is legal for x; **goal(**x**,** v**)** states that, should the current state be terminal, the score of x would be v; **terminal** states that the current state is terminal.

;Roles:
(role x) (role y)
;Initial state:
(init (cell 1 1 b)) (init (cell 1 2 b)) ... (init (control x))
;Rules:
(<= (next (cell ?x ?y ?player)) does (?player (mark ?x ?y)))
(<= (next (control x)) (true (control o))) ... (<= (line ?player)
(row ?x ?player))
;Legal moves:
(<= (legal ?player (mark ?x ?y)) (true (cell ?x ?y b)) (true
(control ?player))) ...
;Goals:
(<= (goal ?player 100) (line ?player)) ...
;Terminal:
(<= terminal (line ?player)) ...

Fig. 1. A condensed Tic-Tac-Toe game description in GDL.

GGP games "take place in an environment with finitely many states, with one distinguished initial state and one or more terminal states. In addition, each game has a fixed, finite number of players; each player has finitely many possible actions in any game state, and each terminal state has an associated goal value for each player. The dynamic model for general games is synchronous update: all players move on all steps (although some moves could be "no-ops"), and the environment updates only in response to the moves taken by the players" [25].

Despite its specificity, GGP poses many great general challenges to AI/CI and addressing them requires the development and application of innovative solutions in various tracks. The core areas include example-based learning and generalization (with the autonomous construction of an evaluation function being one of the subgoals), efficient knowledge transfer between games, automated reasoning and opponent modeling in multi-agent environment, evolution/coevolution of knowledge, and the development of universally-applicable, domain-free heuristic search methods.

The systems winning in the first two GGP editions have relied on some kind of automatic generation of game-specific features based on a set of pre-defined generic features [28], [29] that are applicable to a broad class of games. These generated features were then linearly combined to form an evaluation function, which well reflected the specificity of the game currently played.

A. Cluneplayer

The first GGP winner (2005) was *Cluneplayer* [28], which performed an automated analysis of the game description (game rules) in order to specify features related to the following three core game aspects: *expected payoff*, *control* (mobility) and *expected game termination*. Each identified feature was assessed for its stability and correlation to the game score. The features finally selected were linearly combined to serve as a game-specific evaluation function. This function, along with min-max tree search method then formed the core of Cluneplayer's move selection mechanism.

Depending on the particular game of interest and time allotted for the analysis of its rules, some aspects of a game could dominate over others in the constructed evaluation function. For instance, in more complex games (Othello, chess, or six-player Chinese checkers) no control or termination features have been discovered, reducing the evaluation function to a weighted combination of payoff-related terms only. On the other hand, in simple games (e.g. racetrack corridor - invented for the purpose of the GGP Competition - see [28]), all the three major aspects of the game were identified and properly quantified by the system in the pre-game analysis. As reported in [28], one of the main problems encountered by Cluneplayer is the complete lack of stable features in some games, which questioned the applicability of the method in such cases. In summary, although Cluneplayer never repeated its 2005 success, its underlying idea has been inspiring and, as described in section IV-A, is, to some extent, followed and continued in our research.

B. Fluxplayer

Fluxplayer [29], the 2006 winner, assumes the existence of several elements common across many GGP games (boards, pieces, their quantities, etc.), which are included in the state evaluation function. Furthermore, the system applies fuzzy logic to determine the degree of fulfillment of *goal* and *terminal* predicates in a given state. Atomic features are assessed directly and more complex structures (formulas) are composed with the use of T-norm and associated S-norm functions (see [29] for details). Generally speaking, Fluxplayer's playing policy generally avoids terminal states unless high goal value is expected. Consequently, the predicate *terminal* contributes positively to the state assessment if and only if the value of *goal* is high. Otherwise its contribution is negative. Unlike Cluneplayer, which has not been a GGP participant for several years, Fluxplayer has been actively developed for confrontation with the other GGP players each year.

C. CadiaPlayer and UCT method

All winners in the subsequent tournaments (CadiaPlayer [30] - 2007, 2008, 2012, Ary [31] - 2009, 2010 and Turbo Turtle - 2011) adopted a radically different approach. All of them rely on Monte Carlo (MC) simulations, enhanced by the UCT algorithm (Upper Confidence bounds applied to Trees) [32], [33]. In short, in MC simulations, instead of using domain knowledge to construct heuristic evaluation function, programs play (as many as possible) random games from the current game state to the leaf (terminal) state and back up the encountered results along the respective playing paths. In GGP programs, the above-mentioned random MC simulations are governed by the UCT method, which keeps track of the average return of each state-action combination $Q(s,a)$ that has been played and chooses action a^* to be explored in state s according to equation (1) [30].

$$a^* = argmax_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}, \quad (1)$$

where $Q(s,a)$ is the state-action function value, $A(s)$ represents the set of all possible actions in state s, $N(s)$ is the number of times state s was selected in previous simulations, and $N(s,a)$ denotes the number of times action a was explored in state s.

UCT builds up a game tree gradually (usually one or two nodes per move) and the state-action pairs are labeled with the average historical payoff obtained by the simulations conducted. Lines of play which appear not to be rewarding are gradually explored less frequently than promising playing paths. In effect, the promising paths develop much faster and to greater depths in the game tree, as maintained by the UCT algorithm. Equation (1) thus establishes the appropriate balance between exploration and exploitation, which is a common dilemma of many tree search methods.

Certainly, besides MC-UCT simulations, the world best GGP programs are also equipped with other heuristic algorithms, which help them gain advantage over their competitors. For instance, CadiaPlayer - the most renowned GGP agent - employs the opponent modeling technique and assigns a separate model to each player in multi-player games.

On a general note, the MC-UCT method has recently dominated the scene of multi-game playing programs. The main advantage of this simulation-based approach lies in the ability to capture implicit game properties. Furthermore, MC-UCT simulations are easy to parallelize and scale well on multi-core parallel computing environments.

On the downside, MC-UCT agents have a tendency to play too optimistically, relying on potential opponents' faults (especially when the number of simulations is too low compared to the game complexity). Moreover, despite being guided by hitherto quality of considered moves, MC-UCT, at its core, heavily relies on extensive simulations and therefore is limited by the availablility of computing resources and time (which is especially visible in the case of more complex games or on games with high branching factors).

IV. Challenges within GGP

General Game Playing Competition poses several challenges for both symbolic, logical AI approaches and evolutionary or neural-based CI methods. In order to efficiently play various *a priori* unknown games, even when restricted to a certain genre, an agent has to be capable of autonomous construction of the assessment function based on experience or syntactical game analysis, self-adaptation of search mechanism, incremental learning, opponent modeling, or sharing and generalizing knowledge across several learning tasks. Especially the first (evaluation function) and the last (knowledge transfer) abilities seem to be critical for overall strong performance in GGP.

From a human player's perspective, repeating the same game against various opponents or playing several different games against the same opponent is a great chance for an improvement focused respectively on mastering the playing of a particular game or playing against a particular opponent. In GGP, however, this idea is not easily implementable since players are not aware beforehand on the nature of the game (only the rules of the game are presented). What is more, the same game may be defined in many equivalent alternative ways, and the task of game identification is computationally demanding. Without any indication of the game's name (or any other "labeling"), a GGP agent must rely exclusively on its built-in or autonomously developed game identification mechanism.

The other challenge, which is of special interest in this paper, is the building of appropriate position assessment mechanisms. Based on previous accomplishment, one may say that massive MC search-based methods appeared to be superior over more focused, more creative and - in some sense - more "intelligent" methods described in section II and have displayed dominance in the preliminary period of GGP competitions. This situation resembles one observed in the case of single-game approaches, where, after the initial period of building universal, *learning* agents, mainstream development was concentrated on brute-force based methods supported by pre-defined knowledge (large volumes of auxiliary library data and hand-crafted evaluation functions carefully tuned by human grandmasters), with practically no use of any autonomous or, at least, externally guided learning mechanisms.

Even though increasing computational power allows performing more and more extensive simulations, as well as deeper and deeper game tree searching, the world computer champions (in both single- and multi- game playing competitions) are not really becoming more intelligent (rather they are clearly becoming more efficient).

In the context of the challenge related to autonomous development of the evaluation function, **we believe that one could try to combine the best of two worlds, i.e. using computational power to the greatest extent possible, but applying the technology not solely for the purpose of search/simulations, rather, conducting search/simulations to fulfil the long-term goal of building strong assessment**

function(s). This idea is particularly tempting for the GGP framework, where a lack of straightforward mechanisms for the identification and analysis of the game's structure and relevant features, as well as practically endless possibilities of games to be played, make some degrees of simulations almost inevitable.

A. Simulation-based construction of the evaluation function

Our recent approach to GGP, first proposed in [34], relies on limited use of MC simulations, the purpose of which is to construct evaluation function that may subsequently be used together with "classical" search methods, such as MTD(f) [35], [36]. The approach was partly inspired by the ideas proposed in Cluneplayer and Fluxplayer, as described in sections III-A and III-B, respectively. Nevertheless, there are significant differences at both the algorithmic and conceptual levels. In short, the concepts proposed in the above mentioned previous works are mixed in a new way and enhanced with novel ideas introduced.

The assessment function is developed using a knowledge-free concept, in the form of a linear combination of several numerical game-specific *features* that are inferred from the game rules through a five-step procedure composed of the following phases: *initialization*, *generalization*, *specialization*, *selection* and *weighting*. Features are represented in the form of expressions that are similar to those of GDL, e.g. $(cell\ ?x\ ?y\ b)$.

The first step consists in defining the initial pool of potential features. Since there is no indication about the potential worth of particular features, all suitable statements that exist in the GDL game definition are extracted and added into the pool.

In the generalization phase, we extend the initial pool of candidate features. In each of the statements we replace all constants with variables, generating all possible combinations of constants and variables. For example, a feature $(cell\ ?\ 1\ b)$, will be transformed into a set of the four following expressions (features): $(cell\ ?\ 1\ b)$, $(cell\ ?\ ?\ b)$, $(cell\ ?\ 1\ ?)$ and $(cell\ ?\ ?\ ?)$.

In the next phase we generate specialized features, which contain fewer variables than the original statements. For instance, having the feature $(cell\ 2\ 3\ ?)$, we generate the following three features $(cell\ 2\ 3\ o)$, $(cell\ 2\ 3\ x)$ and $(cell\ 2\ 3\ b)$. In order to do this in a proper manner and at the same time prevent an explosive growth of the feature set, we restrict each variable to its domain. These domains are defined in an approximate manner according to a simple routine proposed in [29].

The set of pre-selected features is subsequently analyzed via some simple simulations. First, we generate some number of short (several moves long) game sequences, starting with randomly selected game states. These sequences are composed of consecutive or "nearly consecutive" states (i.e., with a distance of 2 or 3 plies) resulting from simulating moves of all players. Additionally, for each of the sequences, we perform a limited number of MC simulations starting from the last state of the sequence till the end-of-game state thus arriving at some statistics on the game outcomes as accomplished by each player. Based on the statistics, we then calculate the *correlation* $C(f)$ with expected score for each player and the *stability* $S(f)$ of each identified feature f. Stability is defined by

$$S(f) = \frac{TV(f)}{TV(f) + 10SV(f)} \tag{2}$$

where $TV(f)$ denotes the total variance of feature f (across all random game states) and $SV(f)$ is the average variance of feature f within a sequence.

The higher the value of $S(f)$, the more promising feature f is as a component of the evaluation function. The reasoning behind such an assessment is two-fold. Firstly, if the value of f oscillates widely between consecutive positions, the prognostic value of the feature related to the final game outcome is clearly doubtful due to strong horizon effect. Secondly, if the feature value differs only slightly between separate plays that lead to distinctive outcomes, its relevance to the evaluation function is also questionable.

In the last phase, a final selection of the most promising features to be included in the evaluation function is conducted via appropriate weight assignments. We start with an ordering of the features according to their level of stability and correlation with the evaluated score (i.e., the feature f with the highest value of $\min(|C(f)|, S(f))$ is ranked as 1). Although these two measures are not directly comparable, they can be normalized with a common range of $([0, 1])$. Even though, in practice, often $S(f) > C(f)$, as concluded in our experimental studies, using a combined measure is crucial and none of the components can be removed without deterioration of the system's performance.

Having obtained the ordered features, the number of high ranking features is then defined arbitrarily based on preliminary simulations. Obviously it is not a universally optimal choice, thus, one of our current research goals is to devise a selection mechanism optimized for the particular game of interest (mainly guided by the game complexity).

The final form of the evaluation function is then built as a linear combination of the chosen features, with the weights defined by the product of the correlations to the evaluated score and stabilities, (i.e., $C(f) \cdot S(f)$). Note, that since this time we use a plain (not absolute) values of $C(f)$ and since $S(f)$ is always positive, the sign of the resulting weight is equal to that of the correlation.

B. Experimental results

We have tested two search algorithms that employ our evaluation function. Firstly, we devised and implemented a modified version of the UCT algorithm, which we labeled as Guided UCT (GUCT). The results of the experiments conducted using this algorithm can be found in [34].

In the second set of experiments, we decided to verify the efficacy of the evaluation function based on a min-max tree search algorithm. In our implementation, we employed a modified version of the MTD(f) algorithm. Some modification was required, because while GGP operates on games with arbitrary number of simultaneously-moving players, MTD(f)

has been designed solely for two-alternately-moving-player games. The simple solution to this problem adopted in our present study is to take a paranoid approach, i.e. assume that all other players work together against us and make their decisions after us. This effectively transforms the game into a two-player one.

We orchestrated tournaments in several games pitting our MTD(f) agent against the plain-UCT counterpart (requiring no evaluation function). Initially we started with a selection of 13 games available on the Dresden GGP Server [37], selecting those based on real-world ones (to simplify analysis and facilitate intuitive understanding of the results). After the initial phase of experimentation, we decided to limit the set to five games: a very sophisticated one (chess); a moderately sophisticated one for which our evaluation function is extremely successful (checkers); two games in which our agent displays mixed results (connect4 and connect5) and, finally, one game which proves to be extremely hard for our approach (Othello). All the selected games closely resemble their real-world respective counterparts, with the exception of connect5 which is, actually, a GGP version of small-board go-moku.

One of the important differences between min-max and UCT-family algorithms is their ease of parallelization. While UCT can be relatively easily parallelized with significant performance gain, MTD(f) parallelization techniques are much more sophisticated and, typically, offer less performance gain. At the time of the experiments, our agent was based on a single-threaded MTD(f) implementation, therefore we decided to test it against both single and multi-threaded versions of UCT. The former, gives both agents access to exactly the same processing power, the latter, takes into consideration the fact that UCT can be parallelized more easily than MTD(f).

For each (game, algorithm) pair, we would hold a tournament of 220 to 520 games (depending on the stability of the results), with varying move time limits from 1 to 60s. To account for non-symmetrical games, the agents would swap sides after each match. Each agent would gain 1 point for winning a game, lose 1 point for losing and get 0 points for a draw. Percentage results for MTD(f) agent are presented in Tables I and II. A value of 0% denotes players of equal level, any greater value thus indicates a superiority of MTD(f).

Overall, the results clearly exhibited the potential of our approach. In two cases (checkers and connect4), MTD(f)-based agent significantly outperforms both single-threaded and multi-threaded plain UCT agent. Both these games are moderately sophisticated, having average branching factors of several plies and, obviously, possessing characteristics that enabled our algorithm to generate efficient evaluation functions.

In the case of chess, the results are ambiguous. While our MTD(f) agent reliably wins with single-threaded UCT, its advantage is statistically insignificant to outperform the handicap of the multi-threaded algorithm. Previous results with Guided UCT [34] indicate, however, that this is not due to the deficiencies of the evaluation function but rather because of the high branching factor of chess, which makes min-max analysis extremely time-consuming and, therefore, ineffective.

TABLE I
MTD(F) VS. SINGLE-THREADED UCT TOURNAMENT RESULTS

	Checkers	Chess	Connect4	Connect5	Othello
1s	100%	5%	63%	100%	3%
2s	100%	0%	68%	94%	22%
3s	100%	0%	75%	81%	28%
5s	100%	0%	65%	94%	-6%
8s	100%	0%	68%	81%	25%
10s	100%	5%	90%	88%	-11%
15s	100%	20%	75%	56%	-6%
20s	100%	20%	63%	66%	-11%
30s	100%	30%	85%	25%	-6%
45s	100%	45%	80%	13%	-14%
60s	100%	35%	75%	-13%	6%
90s	100%	30%	65%	-56%	-19%
120s	100%	40%	85%	-25%	11%

TABLE II
MTD(F) VS. MULTI-THREADED UCT TOURNAMENT RESULTS

	Checkers	Chess	Connect4	Connect5	Othello
1s	100%	4%	45%	100%	-28%
2s	100%	-11%	13%	88%	6%
3s	100%	4%	38%	63%	-47%
5s	100%	7%	65%	88%	-39%
8s	100%	-7%	35%	56%	-61%
10s	100%	-7%	35%	63%	-50%
15s	100%	-25%	53%	9%	-78%
20s	100%	-11%	60%	-44%	-50%
30s	75%	-18%	58%	-69%	-33%
45s	95%	-11%	78%	-69%	-44%
60s	100%	-11%	60%	-75%	-47%

Similarly to previous results [34], the outcomes of connect5 and Othello tournaments are less optimistic. It seems that connect5 evaluation function is of relatively low quality, enabling the outplaying of multi-threaded UCT only in the case of extremely tight time regime. As soon as UCT is able to perform a decent number of simulations, it wins easily with our MTD(f) agent. When considering the poor scores in Othello, it is worth recalling that this game has been included in our test set simply due to its high difficulty to our system.

V. CONCLUSIONS

As presented above, multi-game playing remains one of more interesting and worthy challenges in the AI/CI area. General Game Playing framework provides the research community with a conceptual platform for creating multi-game playing agents capable of playing a wide variety of games (namely: all finite discrete multi-player games).

While many of the currently most successful solutions rely on high processing power and huge numbers of random simulations employing UCT-based game tree analysis methods, there are indicators that their playing level can be further improved via smart analysis of game rules and introduction

of various heuristical evaluators. One such approach has been presented in this paper.

While there definitely is a lot of room for improvement, we find our so-far results to be very promising and are already considering several possible paths of continuation of the presented research. Note that it is possible to further tune both search algorithms (e.g. combining the power of UCT simulations with min-max analysis), as well as the evaluation function construction process and its representation. The trivial components selection method employed in the current agent version can be replaced by much more intelligent alternatives, possibly using CI methods such as memetic computing [38]. Apart from that, various CI methods [39] can be employed to further tweak and improve the evaluator as the game progresses.

ACKNOWLEDGMENT

This research is partially supported by Multi-plAtform Game Innovation Centre (MAGIC) in Nanyang Technological University. MAGIC is funded by the Interactive Digital Media Programme Office (IDMPO) hosted by the Media Development Authority of Singapore. IDMPO was established in 2006 under the mandate of the National Research Foundation to deepen Singapore's research capabilities in interactive digital media (IDM), fuel innovation and shape the future of media.

REFERENCES

[1] J. Mańdziuk, *Knowledge-Free and Learning-Based Methods in Intelligenet Game Playing,* vol. 276 of Studies in Computational Intelligence, Springer-Verlag, Berlin, Heidelberg, 2010

[2] ——, "Towards cognitively-plausible game playing systems," *IEEE Computational Intelligence Magazine*, vol. 6, no. 2, pp. 38–51, May, 2011

[3] ——, "Computational Intelligence in Mind Games," in *Challenges for Computational Intelligence*, W. Duch and J. Mańdziuk, Eds., Volume 63 of Studies in Computational Intelligence, Springer-Verlag, Berlin, Heidelberg, pp. 407–442, 2007

[4] ——, "Some thoughts on using Computational Intelligence methods in classical mind board games," in *Procedings of the 2008 International Joint Conference on Neural Networks (IJCNN 2008)*, Hong Kong, China, pp. 4001–4007, 2008

[5] M. Gherrity, "A game-learning machine," Ph.D. dissertation, University of California, San Diego, CA, 1993.

[6] S. L. Epstein, "Identifying the right reasons: Learning to filter decision makers," in *Proceedings of the AAAI 1994 Fall Symposium on Relevance*, R. Greiner and D. Subramanian, Eds. New Orleans: AAAI Press, 1994, pp. 68–71.

[7] S. L. Epstein, J. Gelfand, and J. Lesniak, "Pattern-based learning and spatially-oriented concept formation in a multi-agent, decision-making expert," *Computational Intelligence*, vol. 12, no. 1, pp. 199–221, 1996.

[8] R. A. Levinson, "MORPH II: A universal agent: Progress report and proposal," Jack Baskin School of Engineering, Department of Computer Science, University of California, Santa Cruz, Tech. Rep. UCSC-CRL-94-22, 1994.

[9] B. Pell, "Metagame: a new challenge for games and learning," in *Heuristic Programming in Artificial Intelligence. The Third Computer Olympiad*, H. J. van den Herik and L. V. Allis, Eds., Ellis Horwood, 1992.

[10] ——, "Strategy generation and evaluation for meta-game playing," Ph.D. dissertation, Computer Laboratory, University of Cambridge, UK, 1993.

[11] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, Cambridge, MA, 1974.

[12] R. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[14] S. L. Epstein, "Learning to play expertly: A tutorial on Hoyle," in *Machines that learn to play games*, J. Fürnkrantz and M. Kubat, Eds. Huntington, NY: Nova Science, 2001, pp. 153–178.

[15] R. A. Levinson and R. Snyder, "Adaptive pattern-oriented chess," in *Proceedings of the 8th International Workshop on Machine Learning*, L. Birnbaum and G. Collins, Eds. Morgan Kaufmann, 1991, pp. 85–89.

[16] B. Pell, "Metagame in symmetric chess-like games," in *Heuristic Programming in Artificial Intelligence. The Third Computer Olympiad*, H. J. van den Herik and L. V. Allis, Eds., Ellis Horwood, 1992.

[17] ——, "A strategic metagame player for general chess-like games," *Computational Intelligence*, vol. 12, pp. 177–198, 1996.

[18] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*. New York: Springer-Verlag, 1997.

[19] A. L. Samuel, "Some studies in machine learning using the game of checkers II - recent progress," *IBM Journal of Research and Development*, vol. 11, no. 6, pp. 601–617, 1967.

[20] G. Tesauro, "Practical issues in Temporal Difference Learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.

[21] ——, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[22] F. Hsu, *Behind Deep Blue*. Princeton, NJ: Princeton University Press, 2002.

[23] M. Buro, "The Othello match of the year: Takeshi Murakami vs. Logistello," *ICCA Journal*, vol. 20, no. 3, pp. 189–193, 1997.

[24] B. Sheppard, "World-championship-caliber scrabble," *Artificial Intelligence*, vol. 134, pp. 241–275, 2002.

[25] M. Genesereth and N. Love, "General Game Playing: Overview of the AAAI Competition," http://games.stanford.edu/aaai.pdf, 2005.

[26] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, pp. 62–72, 2005.

[27] N. Love, M. Genesereth, and T. Hinrichs, "General game playing: Game description language specification," Stanford University, Stanford, CA, Tech. Rep. LG-2006-01, 2006, http://logic.stanford.edu/reports/LG-2006-01.pdf.

[28] J. Clune, "Heuristic evaluation functions for General Game Playing," in *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*. Vancouver, BC, Canada: AAAI Press, 2007, pp. 1134–1139.

[29] S. Schiffel and M. Thielscher, "Fluxplayer: A successful General Game Player," in *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*. Vancouver, BC, Canada: AAAI Press, 2007, pp. 1191–1196.

[30] H. Finnsson and Y. Björnsson, "Simulation-based approach to General Game Playing," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*. Chicago, IL: AAAI Press, 2008, pp. 259–264.

[31] J. Mhat and T. Cazenave, "A parallel General Game Player," *Künstliche Intelligenz*, vol. 25, no. 1, pp. 43–47, 2010.

[32] L. Kocsis and C. Szepesvari, "Bandit based Monte Carlo planning," in *Proceedings of the 15th European Conference on Machine Learning (ECML'06)*, 2006, pp. 282–293.

[33] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," *Neural Information Processing Systems 2006 Workshop on On-line trading of exploration and exploitation*, 2006.

[34] K. Walędzik and J. Mańdziuk, "Multigame playing by means of UCT enhanced with automatically generated evaluation functions," in *Proceedings of the 4th Artificial General Intelligence Conference*, ser. Lecture Notes in Artificial Intelligence. Mountain View, CA: Springer, 2011, pp. 327–332.

[35] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin, "Best-first fixed-depth game-tree search in practice," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, vol. 1, Montreal, Quebec, Canada, 1995, pp. 273–279.

[36] ——, "A minimax algorithm better than SSS*," *Artificial Intelligence*, vol. 87, no. 1–2, pp. 255–293, 1996.

[37] http://euklid.inf.tu-dresden.de:8180/ggpserver/public/show_games.jsp.

[38] X. S. Chen, Y. S. Ong, M. H. Lim and K. C. Tan, "A Multi-Facet Survey on Memetic Computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 591–607, 2011.

[39] Q. H. Nguyen, Y. S. Ong, M. H. Lim and K. C. Tan, "A Probabilistic Memetic Framework," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 604–623, 2009.