Self-Adaptation of Playing Strategies in General Game Playing

Maciej Świechowski and Jacek Mańdziuk, *Senior Member, IEEE*

***Abstract*—The term General Game Playing (GGP) refers to a subfield of Artificial Intelligence which aims at developing agents able to effectively play many games from a particular class (finite, deterministic). It is also the name of the annual competition proposed by Stanford Logic Group at Stanford University, which provides a framework for testing and evaluating GGP agents.**

In this paper we present our GGP player which managed to win 4 out of 7 games in the 2012 preliminary round and advanced to the final phase. Our system (named MINI-Player) relies on a pool of playing strategies and autonomously picks the ones which seem to be best suited to a given game. The chosen strategies are combined with one another and incorporated into the Upper Confidence Bounds applied to Trees (UCT) algorithm. The effectiveness of our player is evaluated on a set of games from the 2012 GGP Competition as well as a few other, single-player games. The paper discusses the efficacy of proposed playing strategies and evaluates the mechanism of their switching. The proposed idea of dynamically assigning search strategies during play is both novel and promising.

***Index Terms*—Game Tree Search, General Game Playing, Monte Carlo Methods, Statistical Learning.**

I. Introduction

DESIGNING an artificial agent capable of exhibiting intelligent behavior in a variety of environments is one of the major goals of Artificial Intelligence (AI). General Game Playing (GGP) [1] is a step towards the accomplishment of this long-term goal. In short, GGP domain encompasses finite, deterministic, synchronous, multi-player, perfect-information games, which are defined in the so-called Game Description Language (GDL) [2] - a subset of Prolog. Since it is generally assumed in the paper that the reader is familiar with the GGP challenge, we will only briefly recall its basic principles. Readers who are less experienced with GDL-based game descriptions may consult one of the Internet repositories of sample games [3], [4].

Although official GGP tournaments started in 2005, it is worth recalling, that the history of game-independent playing agents dates back to more than 50 years ago with the introduction of Jacques Pitrat's work [5]. Modern research includes SAL [6], Hoyle [7] and METAGAMER [8], the last one having been a direct conceptual predecessor of (modern) GGP agents. The idea of METAGAMER brought together researchers interested in developing intelligent agents capable of playing a predefined class of chess-like games without human intervention. This multi-game environment consisted of the definition of a class of allowed games, a communication protocol, a game generator and resource limitations. In some sense, the goal denoted in [8] was to shift computer game-playing from an engineering back to a research discipline. GGP is currently the main application to multi-game playing.

On a general note, a program able to successfully participate in the GGP contest is a complex piece of software. In addition to the underlying AI methods and concepts, there are several "technical" issues having a great impact on the overall performance. All of them are worth research attention. In this paper, we present a player built on top of the so-called Upper Confidence Bounds Applied to Trees (UCT) method [9]. In GGP framework, UCT is currently a state-of-the-art approach to searching the game tree. It is aimed at providing balance between exploration and exploitation. The player uses strategies (implementing the concept of an informed search) instead of a blind Monte-Carlo Tree Search (MCTS) method [10]. The novelty of the proposed method is twofold. Most of the MCTS state-of-the-art agents use random playouts (see section IV-A for details) which are complemented with light-weighted playing policies. They are based on some statistical properties rather than elements related to games. Such elements, in turn, are common parts of heuristic evaluation functions in classical methods [11], [12], [13], [14]. Our idea was to consider selected methods of game-state estimation and adapt them to GGP in such a way that they can be used as strategies to guide the subsequently quasi-random Monte Carlo simulations. Secondly, these strategies are dynamically evaluated in terms of being adequate for a given game. Consequently, the tree search is performed in such a way that the highest-evaluated strategies start to dominate and are chosen for most of the simulations. At the same time, worse strategies have a marginalized impact on the search process. Such an adaptation of strategies enables the avoidance of performing inadequate or wasted simulations.

We have tested several strategies and eventually chosen six of them including a purely random search. History Heuristic [15] and Mobility [14] are nowadays a standard in game AI, so we only did a light tuning to adapt them to our approach. The Approximate Goal Evaluation is a concept introduced in GGP in [18], however our realization of this idea is different (more details are presented in section VIII-B). Statistical Symbols Counting was proposed in our earlier work [16]. The Exploration Strategy is designed from scratch for the purpose of our GGP program.

The other novel part, albeit of somewhat lesser importance than the strategy mixing mechanism, is a modified formula

MS is a PhD Student at Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland. e-mail:m.swiechowski@ibspan.waw.pl

JM is a professor at the Faculty of Mathematics and Information Science, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland. e-mail:mandziuk@mini.pw.edu.pl

applied to choosing a move to make. The aim of this formula is to perform a shallow min-max-type search around the root of the tree. Nevertheless, the experiments performed on games used in the GGP 2012 Competition proved that adaptive strategies have a major impact on the overall strength of our player.

The underlying idea of our GGP agent is to enhance simulation-based playing by speeding it up, lowering the amount of randomness and introducing mechanisms for dynamic discovery of strong lines of play. These improvements offer significant advantage over vanilla UCT approach. The improved player not only achieved a 28% better score against the test opponent but, most importantly, improved the record of wins from $1/9$ to $5/9$.

In the following sections the major components of MINI-Player are introduced and discussed. These are: game rules interpreter (section III), background concepts (section IV), a set of strategies used to guide simulations (section V), and a mechanism for real-time switching and evaluation of individual strategies (section VI). Experimental results are presented in section VII. In section VIII, the main differences to previous related works are outlined. Finally, section IX concludes the paper and discusses future research directions.

II. Summary of Earlier Work

The General Game Playing Competition has been played eight times since 2005. However, there has only been five unique champions: ClunePlayer [17] (2005), FluxPlayer [18] (2006), Cadia-Player [20] (2007, 2008, 2012), Ary [21] (2009, 2010) and TurboTurtle (2011). The first winner, Cluneplayer, employed a state evaluation function along with a min-max tree search method in a way similar to the common approach for two-player zero-sum games. The function operated on a weighted linear combination of predefined features: payoff, control and mobility. Each feature was tested against its stability and correlation with the game score. One of the problems encountered in this approach was a complete lack of stable features detected for some types of games. The following year's competition winner was FluxPlayer, whose underlying concept was based on the observation that games often possess common elements like boards, order relations, pieces and their quantities. Hence, Fluxplayer's state evaluation procedure took into account the existence of certain predefined structures. Furthermore, FluxPlayer applied fuzzy logic in order to detect a degree of truthfulness of terminal state conditions. The system used a variant of an iterative deepening depth-first search method to explore the game space. A similar idea was presented in [22], but instead of semantic structures the authors chose to identify the syntactic ones. In particular, they demonstrated the way of detecting successor relation, boards, counters, markers, pieces and their quantities directly from GDL description. The three most recent winners used an MCTS method which had also been successful in Go playing programs [23], [24], [9]. Instead of using any domain knowledge, these agents play random games until a terminal state is reached and fetch the game results. A game tree is gradually built with the average historical payoff for each action-state pair stored in the respective nodes. The search plays particularly well when combined with the UCT algorithm. Hence many enhancements and extensions to the method were proposed such as RAVE, MAST, TO-MAST, PAST [25], Transposition Tables [26], History Heuristic [15], and other [27]. Most of them, however, do not work for every type of game. In some cases, they offer no improvement or even cause a slight performance decrease, because of computational overhead. All of the proposed improvements are based on some statistical optimizations without using explicitly any game features. Game playing is approached through a K-armed bandit stochastic simulation, an approach based on the detection of some game features is presented in [28]. The authors analyze differences between the near end-of-game states in the form of GDL fluents. The fluents then become offensive or defensive features depending on whether they lead to a win state or prevent the player from reaching a loss state. This is an attempt at dynamic extraction of domain knowledge, however computationally quite expensive and with limited generalization capabilities, since features correspond to particular (fully grounded) GDL fluents.

III. Rules Interpreter

The interpretation of game rules is indispensable for computing legal moves, state transitions, terminal states, and goal conditions, which are all defined in GDL. The mechanism for rules interpretation is included in any GGP program and its effectiveness is vital for the agent's playing performance. Contrary to single-game programs which often use highly optimized game representation, GGP has a compact in declaration but computationally-heavy game definition format. This is an unavoidable burden to attain generality.

Most GGP programs use Prolog to deal with game rules interpretation since GDL is semantically a subset of Prolog (with some differences in notation). The rules are provided by the Gamemaster. It is a host with which players exchange messages during a game. The Gamemaster, apart from being a communication hub, serves as a kind of referee and checks player responses for legality. When a player responds with an illegal action in the current state, a random move is chosen instead. The Gamemaster also informs players about all performed actions, so they may update their internal game state representation.

In GGP tournament scenario [1] there are two clocks which govern the process of playing a game:

START CLOCK defines a period of time which starts when the Gamemaster sends the first message containing game rules to participants and lasts until the actual start of the game. This time period is devoted to the initial setup of a player and their preparation for playing the game.

PLAY CLOCK is a time period available for making a move. When a player fails to answer in time, the move is considered illegal.

The GDL-based game description needs to be interpreted in a suitable way. One of the options is a direct translation of GDL into Prolog. Playing programs typically perform such a translation upon receiving the rules. YAP (Yet Another

Prolog) [29] used by CadiaPlayer and EclipseProlog distribution [30] used by Centurio [31] are two of the favorites among GGP programs developers.

Other approaches are FluxPlayer using the so-called fluent calculus implemented in Prolog and Toss [32] with its own reasoning language of the same name. Details of our custom GDL interpreter will not be covered in this paper. Despite its rather technical nature it is a complicated and challenging issue presented in a separate publication [33]. Please note, that an in-house interpreter allowed us to implement all the strategies in as computationally-efficient way as possible.

IV. Background

This section covers some preliminary concepts needed to understand the rest of the paper.

A. Monte Carlo Tree Search

The MCTS method became highly popular in the game community after becoming the first successful approach to Go [23]. In the GGP competition it has been used since 2007. The basic idea of the MCTS simulation is to play a game acting randomly in order to reach the terminal state. In this state a goal value (a game result) for a particular player is computed and back-propagated to all states belonging to the respective path of play. This way a value of each state is estimated by the average result of all simulations which visited this state. Simulations are used to build a game tree whose nodes represent game states and edges represent players' actions.

Typically MCTS consists of four phases: *selection*, *expansion*, *simulation* and *back propagation (of results)*.

Selection: Starting from the root, in each node, choose the successor (child) node with the highest average score, until a leaf node is reached. Certainly, in this phase there is room for introducing additional heuristics to enhance selection.

Expansion: If a state associated with the leaf is not terminal, allocate $N > 0$ new child nodes.

Simulation: Perform simulation starting from the state stored in the leaf node until the end of a game.

Back propagation: Update all nodes on the path of simulation up to the root according to obtained simulation results.

B. UCT

The UCT stands for Upper Confidence Bounds Applied For Trees [9]. It is the most successful and widely used algorithm aimed at enhancing the selection process in MCTS. It provides a balance between exploration and exploitation. In this approach an action a^* is selected according to the following formula:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a)}} \right\} \qquad (1)$$

where *a* - is an action; *s* - is the current state; *A(s)* - is a set of actions available in state *s*; *Q(s,a)* - is an assessment of performing action *a* in state *s*; *N(s)* - is a number of previous visits of state *s*; *N(s,a)* - is a number of times an action *a* has been sampled in state *s*; *C* - is a coefficient defining a degree to which the second component (exploration) is considered.

C. History Heuristic

The history heuristic has been widely applied as a tree-search enhancement since 1989 [15].

The general idea is to transfer information about past actions taken in other states into the current state when the same action is available. In GGP, the history heuristic is typically used to affect the probability of choosing an unexplored action during simulations. For each action, the average score of simulations in which the action was played (regardless of the particular state in which it was performed) is stored. The better historical score an action has, the more likely it will be chosen again when available. The history heuristic is used by several top GGP players to guide the simulations. To the best of our knowledge, it was first applied to GGP by FluxPlayer [18].

Our implementation of this concept is based on the one proposed by [27]. This is a very recent research contribution in which the authors prove the superiority of this simple variant over more sophisticated implementations.

D. Mobility

Mobility in games stands for the number of legal actions available for the agent (usually in comparison to other players). Generally speaking, a drastic change in mobility often corresponds to performing strong offensive or strong defensive actions (for many board games capturing pieces is a good example). Having a greater number of actions available to the player is usually considered beneficial. In GGP, mobility was implemented as part of the evaluation function in the first winning program [17].

V. Strategy-Guided Simulations

The first two phases of MCTS, as well as the last one, are implemented in MINI-Player in a generally standard way (with slight modifications only). The main difference compared to other GGP approaches is attributed to the simulation mechanism. Hence, in the remainder of this section, we focus on the simulation phase, namely we introduce several search strategies developed for the purpose of the GGP search improvement. In the following section a procedure for picking and switching these strategies is presented and assessed. Certainly, using completely random simulations to reason about the game has both its advantages and drawbacks. On the one hand, the approach is very general and can be applied to any type of game. No specific assumptions are required, the idea is clear and easy to implement. In addition it scales well in a multi-core environment. On the other hand, however, for many games, no visible improvement can be observed until thousands or millions of simulations are made. The effectiveness of the method depends on the game's complexity, its branching factor, the average length, as well as its "convergent nature". By "convergent nature", we mean that every action taken by a player leads the game towards its terminal state (which is the case of connect-four, for instance). In contrast, a game in which players can move freely without decreasing their mobility or can repeat their positions are examples of non-convergent games.

Moreover, as stated in [20], simulation-based approaches sometimes lead to over-optimistic playing which relies on possible opponents' mistakes. Another issue is the non-deterministic nature of the MCTS approach. Generally speaking, some amount of randomness is advantageous in order not to omit potentially good moves. Contrary to a static heuristic function, which, when inadequate for a certain game, would constantly favor weak moves, the MCTS approach, given enough time, would sample all available actions. Too much randomness, however, hampers the use of domain knowledge gained from rules analysis or past simulations.

The valid question then is **whether it would not be beneficial to trade some simulation time for game learning process**. The state-of-the-art players incorporate some (though limited) modifications to MCTS to guide simulations, mainly the history heuristic mentioned in section IV-C.

Our agent uses several enhancements (strategies) to drive simulations. Each strategy takes a game state as an input and yields an ordered set of possible actions based on a specific ordering mechanism. Only one strategy per simulation is active. However, during a single match many simulations are performed and therefore more than one strategy (potentially all of them) may be used. Below, we present an overview of these strategies. Their fundamental idea is to use simple, rather than complex, rules, which is motivated by a low computational cost, as well as, simplicity and generality of the rules.

A. Random (R)

Strategy R is the least complex one. The player simply chooses actions at random as in the original MC simulations. This strategy acts as a natural fallback strategy when other ones fail. Even though random search is slow, it guarantees uniform convergence to the best move in conjunction with the UCT algorithm. Moreover, random selection does not require costly computations so it is better suited for very simple games which can be searched thoroughly. This strategy attains the highest number of simulations per second ratio among all strategies used in our system. Since the selection of games is not known in advance, introducing a certain amount of indeterminism hinders premature convergence (similar to the case of genetic algorithms or many other approximate optimization methods).

Based on the test results we can say that the best scenario is when a player merges one dominating strategy, well-suited to a given game with a number of random searches.

B. Approximate Goal Evaluation (AGE)

In GGP a set of goal rules determines the possible outcomes of the game for each of the players. If we filter out the rule with the highest value for a role assigned to MINI-Player, we can treat this rule as our ultimate, driving goal in a game. Usually, there is only a single rule that defines the highest score, but it may have a complex form, composed of multiple rule definitions (implications) connected by OR operators. In GDL the world is described by facts which are true in the current state. Anything that cannot be proved to be true is false. Hence, given that a GDL rule is an implication and that facts not implied by any rule are presumed false, it is sufficient to put together the results from the rule's implications in order to obtain a (compound) equivalence.

For instance, given:

$$\begin{aligned} R1 &\Rightarrow A \wedge B \wedge C \\ R1 &\Rightarrow D \\ R1 &\Rightarrow E \wedge F \end{aligned}$$

we can assume:

$$R1 \Leftrightarrow (A \wedge B \wedge C) \vee D \vee (E \wedge F)$$

The aim of AGE strategy is to focus on the rule with the highest goal value and consider moves which maximize the degree of its satisfiability. Therefore, in some sense, the AGE-using agent acts like a greedy player. Unfortunately, there is no built-in mechanism for computing a degree of rule's satisfaction in GDL/Prolog. This must be computed manually and there are several ways to do it. We decided to use fuzzy logic algebra with a logical resolution tree. The resolution tree is composed of two types of nodes which occur alternately:
AND nodes, each of which represents a set of conditions for a particular rule's instance in the form of a single implication with AND being a default logical connective between conditions.
OR nodes, each of which represents all implications existing in a rule's description (e.g. all implications of R1 in the example above). OR is a default logical operator here, since the final result is a sum of results of individual implications. For the sake of generality, an OR node is created even if the rule is defined based on one implication only. OR nodes are also created for state facts and constant facts. Let us briefly describe the reasoning behind this decision. The purpose of an OR node is to gather the complete set of (true) results for a particular rule. State and constant facts can be regarded as rules which are already calculated, with the results immediately available. The above interpretation allows the usage in our implementation model of the same object - an OR node - for both dynamic rules and facts. State facts are the only facts affected by the keywords *init* and *next*.

The first one specifies the initial set of state facts whereas the latter defines state transitions during state updates (after all moves have been performed). State facts are meant to be the basic elements that form a dynamic game state. Constant facts are directly defined in the rules of the game and (as the name suggests) do not change.

There are two values which are propagated bottom-up in the AND-OR tree in the recurrent process: a degree of approximate goal satisfaction *(AgVal)* and a tie-breaker *(TbVal)*.
In leaf OR nodes:

- *AgVal = 1* if the rule is satisfied, *0* otherwise. A rule is satisfied if there exists at least one grounding for the current query.
- *TbVal = AgVal*

In AND nodes:

- $$AgVal = \frac{1}{N}\sum_{i=1}^{N} AgVal_i$$

- $$TbVal = \frac{1}{N}\sum_{i=1}^{N} TbVal_i$$

  where i denotes the i-th child (OR) node.

In non-leaf OR nodes:

- $$AgVal = \max_{i=1,\ldots,N} AgVal_i$$
- $$TbVal = \frac{1}{N}\sum_{i=1}^{N} TbVal_i$$

  where i denotes the i-th child (AND) node.

The algorithm for computing both propagated values is similar to the regular logic proving scheme used by the interpreter. However, instead of just deriving true facts, we also apply the above formulas for calculating *AgVal* and *TbVal*. There are two differences though: in a regular case, while iterating the conditions in AND node, the first unsatisfied condition immediately proves the current rule false. There is no need to check the conditions further because we already know that the rule is not satisfied and will not produce any facts. In AGE case, we continue to check the conditions even if a failure is encountered, because we are interested in how many conditions actually hold true.

Recurrent rules, however, are treated differently. A failure of a condition inside a recurrent call acts as a termination condition for the recurrence. In such a case, the proving procedure is terminated, likewise in a regular non-AGE scenario, which prevents the system from entering an infinite loop.

A simulation that uses AGE strategy orders moves according to their next state's *AgVal* and then, in the case of even results, according to their *TbVal* values. This strategy, if active in a given simulation, is applied with probability $P = 0.75$ in each node. Otherwise, a random choice is made. Justification of this choice is presented in section V-G

C. Statistical Symbols Counting (SSC)

This strategy implements a simplified version of our former approach previously used as a stand-alone GGP agent's engine [16]. In the following description we assume that the reader possesses a basic knowledge of GDL. We recommend the GDL specification [2] in case of any questions in this matter. The SSC policy of play requires some learning period before it can be effectively applied. The learning phase relies on performing random simulations with some additional computations made in each encountered state.

First of all, the number of realizations of each state fact is counted. A state fact, like all GDL/Prolog terms, starts with a name followed by arguments, so state facts with a common name are grouped and counted together. For example, in the initial state of checkers (see [3] for a GDL checkers description), we observe:
$Count(\text{“}cell\text{”}) = 64$; there are 8×8 cells.
$Count(\text{“}control\text{”}) = 1$; there is one player having a turn.
$Count(\text{“}step\text{”}) = 1$; the step counter has only one value in each state.

TABLE I
STATE FACTS VERTICAL ARRANGEMENT. THE 0-TH COLUMN SYMBOLS ARE EMPHASIZED.

cell	**1**	1	b
cell	**1**	2	b
cell	**1**	3	b
cell	**2**	1	x
cell	**2**	2	b
cell	**2**	3	o
cell	**3**	1	b
cell	**3**	2	b
cell	**3**	3	x

$Count(\text{“}piece\_count\text{”}) = 2$; there are two scores, one for each of the players.
The other type of elements which are identified and counted are symbols, which are present on state facts argument lists with the same index (in the same position). In other words, if these facts were placed in vertical order, one below the other, these symbols would be placed in the same column (see Table I as an example).

We filter out symbols whose quantities do not change or change in a fixed manner. In order to detect them we compare the respective quantities in states occurring in different simulations paused at the same time steps (see Figure 1).

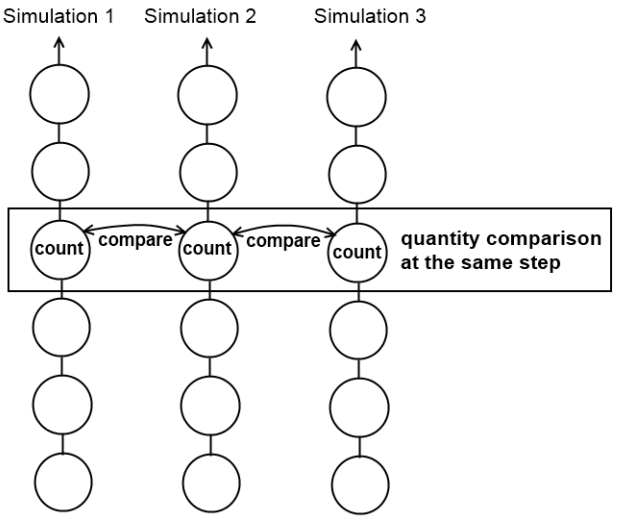

Fig. 1. Selection of dynamic features.

We label simulations as won or lost depending on whether or not the result is above the average defined in the GDL game description. For each symbol its average (*AVG*) quantity is computed for won (AVG_W) and lost (AVG_L) games, respectively, as well as, its maximum value (*MaxValue*). The weight assigned to a given symbol is calculated according to the following formula:

$$weight = \frac{AVG_W - AVG_L}{MaxValue} \quad (2)$$

These weights, linearly combined with the corresponding

numbers of symbol occurrences, form the evaluation function. Schematically the learning phase can be described as follows:

- identify elements, i.e. the numbers of state facts (contained within each fact), the numbers of symbols in columns (for each triple (fact, symbol, column))
- count the quantities of presence of these elements in each state during simulations
- distinguish the symbols whose quantities are affected by actions taken by the players (i.e. their changes are not predefined in a game and vary between matches)
- compute correlations between these quantities and a game score
- define the heuristic evaluation function as a weighted linear combination of these symbols quantities. Weights reflect the importance of the respective symbols and their positive/negative contribution to the player's position assessment

All the above-mentioned steps required to construct the evaluation function are performed by other strategies by means of simulations during the 95% of a START clock time, at which point the strategy is ready for the agent's use.

D. Mobility (M)

In the case of single-player games the strategy maximizes the number of actions available to our agent. In the case of N-player games ($N > 1$) it aims at the maximization of the following formula:

$$\sum_{i=1}^{N}(M_0 - M_i)$$

where M_i denotes the number of legal moves of the i-th player and $i = 0$ represents the role assigned to MINI-Player.

Mobility-based strategy is computationally intensive since prior to move evaluation, the next state and all legal actions in that state must be computed. This is why it is only applied to the first 6 actions in a given simulation. If the simulation lasts longer, it becomes random from the 7th step on. The limit for the strategy's maximum length (6) was defined based on preliminary tests made during the preparations for the 2012 GGP Competition.

E. Exploration (E)

The idea underpinning this strategy is to explore as many unknown states as possible. Let us define a difference between states B and A as the number of facts (GDL/Prolog terms) which are present in B but not in A:

$$diff(B, A) = B - A \tag{3}$$

The action-selection rule is defined as follows:

- $prev^i(S)$ denotes the i-th previous state to S traversed in the current simulation
- For each legal action a check its resulting state $S = next(a)$ and compute N differences between this state and N previous states:
  $d_i = diff(S, prev^i(S)), i = 1...N$
- Choose i which minimizes d_i to find the most similar state to S among the N last states
- Choose action a which satisfies: $a = arg\max_i(\min_i d_i)$

The action selection formula could be interpreted as follows: for each possible action a and its resulting state find the most similar state among the *N* last ones (the middle part of the algorithm) and assign this state to that action. Choose an action that maximizes the difference between the assigned state and the current state (the last part of the algorithm).

Basically, we look for the highest difference between a potential new state S and the set of last N states ($PREV$). A difference between state S and set $PREV$ is defined as the difference between S and the most similar to S member of $PREV$.

Using the set of chosen games, we experimentally measured the exploration performance of the algorithm defined as the number of distinct state facts visited per second, or equivalently, the number of distinct states visited so far. In both cases, the proposed method explores statistically more states than both the simple maximization of the difference between S and the last state or difference between S and the union of the last N states (i.e. a union of all facts which hold true in these states).

We have tested values of *N* from 1 to 8 and found $N = 3$ to be the best trade-off between depth and performance. $N = 4$ is also a viable, but visibly slower, choice. Values less than 3 result in playing in a similar way to random play.

Note that instead of using (3) we could have defined the state difference in a symmetrical way, i.e.:

$$diff(B, A) = (B - A) \cup (A - B) \tag{4}$$

We prefer the asymmetric definition mainly because it favors the existence of new facts in child states. We had tested this strategy in all single-player games available to us, as well as, in checkers, and found that the approach based on (3) yielded better results. Moreover, the wins occurred faster than in the case of using definition (4) .

F. History Heuristic (HH)

During a simulation MINI-Player maintains a global dictionary of actions taken (regardless of the state), e.g. *(mark 3 3 x)*, and their average scores. Moves performed by each role are tracked separately and once a terminal state is reached their average scores are updated by the current simulation result. An estimated historical strength of a move is then defined as the sum of obtained goal values in all simulations which included that move, divided by the number of such simulations - duplicates occurring during a single match are ignored.

In the case the HH strategy is chosen to be active during a given simulation, the player checks if there are any matches between currently available moves and those historically stored. If at least one such a match is found, the best scored move is selected. Otherwise, the action is performed at random. A similar approach is used by Cadia-Player in e-Greedy algorithm [27]. Contrary to our approach, however, the performance of HH is not monitored on-line and the heuristic is applied stand-alone, i.e. is not mixed with other play-out policies.

G. Tuning Parameters

Each non-random strategy includes, an individually tuned, parameter PR denoting the probability of choosing a move according to this strategy in a given node (state). For example, if a simulation guided by AGE lasts 100 steps, there are 100 decisions whether to take an action according to the AGE formula (with probability PR) or choose a random move (with probability $1 - PR$). Parameter PR should be high enough to allow the exploitation of a particular strategy, but at the same time low enough to avoid repeating the same lines of play. To derive a suitable value of PR, first we roughly identified games in which particular strategies give best results using default parameters, then we performed experiments with various PR values for each pair consisting of a strategy and the game for which the strategy is well-suited further to our observations. A player using that strategy mixed with a random one was tested against a basic UCT-player with a random search in 300 matches. The results are presented in Table II. For each strategy the respective value of PR lies in the row containing the bolded value (e.g. 0.70 for HH or 0.80 for E). The 95% confidence intervals are in a relatively close range within a game, so only the average value per game is included. It is easy to notice that results do not differ much in the proximity of the optimum. In addition, for four strategies the optimum for PR was found within a range of 0.7 and 0.85.

TABLE II
THE RESULTS OF TESTS AIMED AT TUNING THE PARAMETER PR (PROBABILITY OF USING A GIVEN STRATEGY IN THE CURRENTLY SIMULATED/VISITED NODE). SEE SECTION V-G FOR A DETAILED DESCRIPTION.

P	**AGE** Checkers	**SSC** Chess	**M** Othello	**E** Farming Quandries	**HH** Connect-4
0.20	59.17	50.00	54.33	54.83	54.67
0.40	66.00	51.00	55.50	56.50	55.83
0.50	65.67	51.33	68.17	58.17	59.33
0.60	70.67	51.00	73.83	59.50	59.00
0.65	79.17	50.83	73.67	60.67	57.33
0.70	76.33	51.33	81.33	61.17	**66.67**
0.75	**79.33**	49.50	83.67	61.67	62.92
0.80	75.83	53.17	81.17	**62.17**	59.17
0.85	72.67	54.17	**85.17**	58.33	54.83
0.90	69.33	54.64	79.17	42.50	50.67
1.00	55.67	**55.00**	68.17	34.83	44.17
95% Conf.	±4.46	±9.07	±3.90	±5.10	±5.13

H. Modeling the Opponents

In simulations performed by MINI-Player, opponents are modeled using the MCTS method enhanced with HH. This choice is, to some extent, justified by the popularity of the history heuristic - many players actually implement some version of this concept. Other strategies are not used for opponent modeling for multiple reasons, mainly the following ones:

- To maintain high simulation speed
- There are no premises that other approaches would model the opponents better than HH (most probably opponents do not use our strategies)
- To keep the evaluation of our strategies fair and comparable, i.e. our player is always simulated against players of the same type (random ones equipped with the history heuristic)

The effective inclusion of other than HH playing strategies in the task of modeling the opponents is one of our future research goals.

VI. STRATEGY SWITCHING MECHANISM

Using all strategies uniformly during a match would not be a good idea. First of all, if we are playing a game for which there is an optimal strategy in our repository (this is only a hypothetical assumption), then the best choice would be to use this optimal strategy in all simulations, not only in some percentage of them. Secondly, if some of the strategies are weak for a particular game, we would waste time on ineffective simulations.

To overcome this problem we use a dynamic strategy evaluation mechanism. The higher the evaluation of a strategy, the more simulations it is used in. Moreover, at some point, the worst performing strategies are discarded from the selection pool. After a number of experiments, we ended up with three methods of strategy evaluation. One of them (described in section VI-C) works well only under certain conditions and, due to not being general enough, was finally abandoned in our tests. The remaining two solutions are quite similar, though there are some practical differences between them. So far none of them appeared to be superior to the other. We present them in the following three subsections.

A. Upper Confidence Bounds (UCB)

Upper Confidence Bounds [10] is a well-known algorithm for maintaining a trade-off between exploration and exploitation. The method is also called a K-armed bandit algorithm because of its origin related to multi-armed bandit machines (in casinos). In the problem, there are k arms, each leading to a random reward. Probabilistic distributions of the arms are pairwise independent and unknown, but fixed for a given problem instance. An agent, at each step, can choose one arm to play with. The goal is to maximize the total reward. The UCB comes with a formula on how to balance the exploitation of the most lucrative arms recognized so far and exploration of other possibilities. A simple solution might be, for example, to play the currently best arm with the probability 0.5 and a randomly selected other one in the remaining cases. Such an approach, however, is not optimal. UCB offers a statistically justified solution which is optimal in the sense of the maximization of the expected value.

In the GGP framework the UCB method selects strategy s* according to the following formula:

$$s^*(n+1) = arg\max_{s \in S} \left\{ Q(s,n) + b\sqrt{\frac{ln\,(n)}{T(s,n)}} \right\} \qquad (5)$$

where *s** is a chosen strategy, *n* is the number of simulations performed so far, *Q(s,n)* - is the average payoff of strategy *s* in *n* simulations (this is the total result of *s*-driven simulations

divided by *T(s,n)*, *T(s,n)* is the number of simulations that used strategy s in n performed simulations, b is a weight assigned to the exploration part. After some tuning its value was set to 5. The algorithm is, to some extent, similar to the UCT method used for a node selection in the MCTS algorithm (cf. eq. 1), however, the assessments of nodes and strategies are performed independently.

B. Static Allocation

We also tried to order the strategies by their average results and assign to them predefined numbers of simulations based on their ranks. The distribution of the numbers of assigned simulations was optimized manually based on observations made in games we know how to play. In these games we have observed MINI-Player's performance and the quality of its move-selection mechanism in the case of various distributions of the strategies used in simulations. Based on the above-mentioned observations we decided on the following scheme: we allocated 22 simulations for the best strategy (ranked first), 11 for the second one, and then 7, 4, 2 and 1, respectively for strategies ranked from third to sixth.

If, for instance, the mean pay-off values of the strategies were the following: $S1 = 0.61$, $S2 = 0.59$, $S3 = 0.27$, $S4 = 0.49$, $S5 = 0.70$, $S6 = 0.48$, then the strategies were ordered $S5 > S1 > S2 > S4 > S6 > S3$ and allocated $[22, 11, 7, 4, 2, 1]$ simulation runs, respectively.

The order of execution was "close to uniform", with respect to the ranks of strategies, i.e., for example,

$$S5, S1, S5, S2, S5, S1, S5, S4, S5, S1, S5, S2, \ldots, S5.$$

In the above order $S5$, having 22 simulations to perform, appears twice more frequently than $S1$ with 11 simulations and approximately three times more frequently than $S2$ which is assigned to 7 simulations, etc. After a round of 47 simulations the evaluation process was repeated.

It is worth underlining that in this allocation scheme 70% of all simulations have one of the two best strategies assigned.

C. Square average result proportional

In this method strategies are assigned quantities of simulations to perform proportionally to their square average results.

First, we compute the average result of strategy i, $AVG_i, i = 1, \ldots, 6$ as a sum of results accomplished by this strategy, divided by the number of simulations that used this strategy, so far. In order to avoid the problem of dividing by zero, we forced the assignment of each strategy to at least one simulation in the first 10 simulations, and only then calculated the values of AVG_i for the first time.

Next, we normalize the averages:

$$AVG_i := \frac{AVG_i}{\min_i AVG_i}$$

Then we add the overflow OF_i, i.e. a fractional part from the previous evaluation (see below). In the first evaluation all OF_i are equal to zero. After OF_i is added, the average value is squared:

$$R_i = (AVG_i + OF_i)^2$$

Next, the allocation of simulations and new overflows are computed:

$$S_i = \lfloor R_i \rfloor$$
$$OF_i = R_i - \lfloor R_i \rfloor$$

where $S_i, i = 1, \ldots, 6$ is the number of times strategy i will be pursued in the current sequence. After a round of $\sum_i S_i$ simulations the strategy assignment process is repeated. The AVG_i values typically fall into the range of (1,2). Value of 2 means that there exists a strategy with the average outcome two times greater than the worst-evaluated strategy. This proportion depends on how a game is defined, but in a classical two-player zero -sum games the variance is usually lower (see Table IX which shows average outcomes of strategies). Let's consider a very probable example: $AVG = (1.1, 1.2, 1.4)$ and therefore $R = (1.21, 1.44, 1.96)$. Since we assign a number of simulations equal to the floor of R, we would not be able to distinguish between a better outcome and a worse one because all of them would become 1. The OF is used to add the fractional part, which is lost in the floor operation, to the next iteration of the allocation process. This enables maintaining appropriate proportions.

This method of allocation works well when there are only two strategies. When there are more of them, the distribution is too narrow, i.e. the quantities of simulations assigned to strategies are too close to each other. At first we decided to test just the linearly proportional allocations, without the square component applied to the average scores, but in that case the numbers obtained were even closer. That is why we eventually excluded this method from our tests and focused on those described in sections VI-A and VI-B.

D. Comparison of the switching mechanisms

Figure 2 presents the quantities of simulations which are assigned to the best evaluated strategy by the three methods discussed. The same data, in the case of the worst evaluated strategy, is presented in Figure 3.

Methods A (section VI-A) and B (section VI-B) yield similar performance during the first 20000 runs, when the number of simulations allocated to the best strategy equals approximately half of the total number of simulations performed. At the same time, the number of simulations allocated to the worst performing strategy is about 10 times smaller (c.a. 5% of the total number of simulations). The above comparison (50% vs. 5%) indicates that strategy allocation mechanisms in methods A and B are plausible and consistent. After 22500 simulations method A starts to favor the best performing strategy more than method B.

Method C (section VI-C), in the same time period of 20000 simulations, allocated approximately 25% of the simulations to the best performing strategy and around 15% to the worst one. Clearly, method C does not ensure adequate distinction between strategies.

Method A is the most complex in terms of required computations and, at the same time, has strong mathematical background. The algorithm must be recalculated before each simulation. For comparison, the two other methods need to

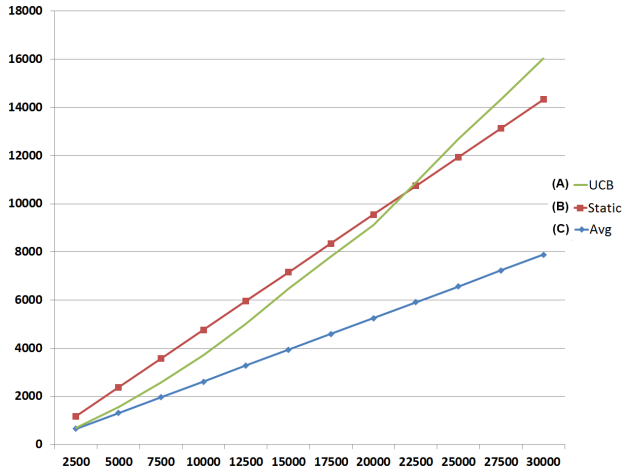

Fig. 2. The numbers of simulations allocated to the best evaluated strategy in a sample checkers game. The x-axis presents the total number of simulations.

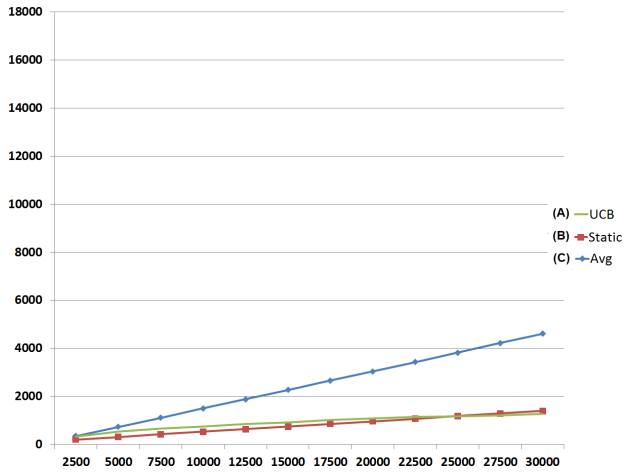

Fig. 3. The numbers of simulations allocated to the worst evaluated strategy in the same game as the one presented in Figure 2. The x-axis presents the total number of simulations.

be recalculated much less frequently, only when there are no more simulations assigned.

Method B is the simplest and the fastest one, which is a huge advantage. Since the allocation grain is static (pre-defined), the method cannot be optimal in all cases. This is, however, a safe and robust approach, with no risk of reaching a situation in which one strategy would receive all (or zero) simulations, for example.

Whether the first or the second method is better depends on the game. During the 2012 GGP Competition, method A was implemented by our agent. Most probably the optimal solution would consist of applying the UCB method with some, statically defined, lower and upper bounds, i.e. a kind of "the best of both-worlds" solution. An investigation into this issue is one of our current research goals.

VII. Empirical Results

A. Experimental setup

In addition to taking part in the 2012 GGP Competition, MINI-Player has been tested in a series of matches against two reference opponents. The first one was CadiaPlayer [20], a three times winner of the GGP Competition (2007, 2008 and 2012). The version we played against is from June 2011, available open source at the project's website [34]. We used the original authors' implementation. Although the program has most probably been optimized since then, the authors do not claim to have introduced any major changes to their agent. CadiaPlayer is undoubtedly regarded as a state-of-the-art GGP player and is the most renowned prize-winning tournament participant.

The second opponent was a clone of our system which did not use any simulation strategies other than a random one, enhanced by the history heuristic, and employed the classical move-selection method based on the nodes' average scores. We call this player MINI-Player-C ("C" stands for "classic").

In order to ensure sufficient complexity and diversity of games, we used the same set of games which our agent played during the 2012 GGP Competition plus a few single-player ones (games of this kind were not present during the competition).

Each game was played 270 times (rounds) with the roles swapped after each game. Role switching is important since in some games there exist favorable starting positions. Inspired by [20] we used a mechanism of aggregating the results similar to CadiaPlayer's. The game is considered won by a player if its score is greater than that of the opponent. The actual difference does not matter. An equal outcome results in a draw. The final formula for a player's aggregated score is the following (normalized to [0,100] interval):

$$Score = (|WINS| + \frac{|DRAWS|}{2}) * \frac{10}{27} \quad (6)$$

B. Move decision

A final decision which action to play during a match is taken on higher level than inner-simulation choices. Each strategy is too straightforward and not universal enough to be used as a unique, stand-alone tool. The root node in a game tree represents the current state. The selection of a move is technically equivalent to the selection of a direct child node of the root. Even though the whole UCT formula could, in principle, be applied to this task, it is better to use the exploitation part of equation 1 only, i.e. the average action value Q. During a match an agent should answer with the best possible move. Hence, most, if not all, UCT-based players choose an action either according to the Q value or select the most simulated node. The exploration part of equation 1 is usually reserved for tree building simulations.

Our action selection scheme is a bit more complex, based on - what we call - an Effective Quality (EQ). In a given decision context, let us define by *Our* - the number of moves available to our role and by *Total* - the total number of joint (tuple) moves available.

In order to calculate EQ value we distinguish four types of nodes among the next move candidates (see Table III).

For each child *node* of the root node (i.e. each candidate move) we calculate its EQ with respect to Table III. A *node* with the highest EQ value is chosen. If more than one *node*

TABLE III
EQ VALUE BASED ON THE ROOT CHILDREN *node* PROPERTIES.

Condition in a *node*	**EQ formula**
node.Terminal = True	$EQ = node.Q$
Our = 1	$EQ = \min_{i=1...N}(node.Child[i].Q)$
1 < Our < Total	$EQ = \frac{\min_{i=1...N}(node.Child[i].Q)+node.Q}{2}$
Our = Total	$EQ = \max_{i=1...N}(node.Child[i].Q)$

is tied with this value we extract our role action and choose one with the highest HH value. Since all strategies record their history, HH provides an additional mechanism for the final move recommendation. If some nodes are still tied, one of them is selected at random. When applying the above procedure, contrary to other Monte Carlo and UCT players, our agent analyzes not only the root children nodes but also their child nodes (root grandchildren), so, effectively, the tree is searched one level deeper.

An example of the EQ-based selection procedure is presented in Figure 4. In the root node there are four possible actions leading to four child nodes. In the leftmost one, the condition $1 < Our < Total$ holds, which leads to calculation of EQ according to the third case presented in Table III. The second node on the left is a terminal one with the game outcome (as well as EQ value) equal to zero. In the next one the condition $Our = 1$ is true, so the formula of the second case from Table III is applied. Finally, in the fourth child node the condition $Our = Total$ holds, and therefore, the last case of Table III is considered. The rightmost node is selected as the next agent's move with $EQ = 0.7$.

The main motivation behind the above-described method compared to typical UCT-based players is the common weakness detected in their play - they tend to play too optimistically, often relying on opponents' faults. Introduction of min function into the formula partly alleviates this problem. Furthermore, if a game features double moves, or two consecutive moves are strongly correlated, the modified approach has higher chances to select a locally optimal pair.

C. The results

As previously mentioned, among our two players, MINI-Player uses six previously introduced strategies with a switching mechanism A and the above-described EQ-based move selection, while MINI-Player-C uses plain UCT enhanced by HH and a standard (best Q) move decision scheme.

In the first experiment MINI-Player was pitted against CadiaPlayer. The results are presented in Table IV. Generally, our agent confirmed its potential in some games (roughly those which it had won during the contest), but at the same time lost in a few others (again, mainly those it had lost in during the competition). The positive exception from the above rule is Farming Quandries (won against Cadia, lost in the tournament). All in all, MINI-Player was the winner in five games and lost the remaining four.

In the second experiment we compared MINI-Player and CadiaPlayer with the basic MCTS agent (MINI-Player-C). The results are presented in Table V. MINI-Player outperformed MINI-Player-C in seven games and was inferior in the remaining two. Likewise CadiaPlayer won seven games, tied one and lost one, however, the sets of games in favor of MINI-Player and CadiaPlayer, respectively were not exactly the same.

TABLE IV
EVALUATION OF MINI-PLAYER'S STRENGTH VERSUS CADIAPLAYER. OUR PLAYER SUFFERED FROM A TECHNICAL ERROR IN THE FIRST GAME DURING THE COMPETITION WHICH WAS IMMEDIATELY CORRECTED. VALUES IN COLUMN *Clock* REPRESENT START CLOCK TIME (THE LEFT VALUE) AND PLAY (MOVE) CLOCK TIME (THE RIGHT VALUE). THE RESULTS ABOVE 50 ARE IN FAVOR OF MINI-PLAYER AND THOSE BELOW 50 FAVOR CADIAPLAYER. THERE ARE 95% CONFIDENCE INTERVALS PUT IN SQUARE BRACKETS (THE LOWER THE VALUE THE HIGHER THE CONFIDENCE)

Game	**Clock [s]**		**MINI vs. Cadia**	**GGP 2012 Result**
Connect4	40	15	41.67 [5.35]	Loss (error)
Cephalopod Micro	60	20	40.00 [5.84]	Loss
Free for all 2P	45	15	63.33 [5.14]	Win
Pentago	45	15	29.33 [5.43]	Loss
9 Board Tic-Tac-Toe	45	15	70.67 [5.16]	Win
Connect4 Suicide	45	15	53.33 [5.26]	Win
Checkers	60	20	54.33 [5.88]	Win
Farming Quandries	90	30	68.33 [4.21]	Loss
Piligrimage	90	30	42.67 [4.32]	Loss
Average			**51.40** [5.18]	

TABLE V
RESULTS OF MINI-PLAYER VS. MINI-PLAYER-C AND CADIAPLAYER VS. MINI-PLAYER-C. SEE DESCRIPTION OF TABLE IV FOR CLOCK VALUES AND INTERPRETATION OF RESULTS.

Game	**MINI vs. MINI-C**	**Cadia vs. MINI-C**
Connect4	61.33 [5.69]	59.67 [5.54]
Cephalopod Micro	59.33 [5.86]	73.33 [5.27]
Free for all 2P	75.33 [4.86]	65.67 [5.11]
Pentago	40.00 [5.84]	55.33 [5.93]
9 Board Tic-Tac-Toe	66.30 [5.42]	50.00 [5.76]
Connect4 Suicide	51.33 [5.72]	43.33 [5.50]
Checkers	79.33 [4.83]	69.33 [5.32]
Farming Quandries	66.67 [5.36]	59.67 [4.90]
Piligrimage	39.33 [5.40]	62.83 [5.39]
Average	**59.88** [5.44]	**59.91** [5.42]

The next experiment was devoted to the separation of the impact of adaptive-strategies from the modified move selection formula (EQ). In a small tournament two versions of MINI-Player using the respective mechanisms were tested against CadiaPlayer. The results are presented in Table VI.

Furthermore, in order to make the results easier to interpret, we computed the relative factors, i.e. the strength of both modifications separately and in a combined version divided by the baseline MINI-Player-C's score (see Table VII). A factor of 1.00 means that the player is performing at the level of a basic MCTS player with no enhancements. The usage of the EQ formula increases the overall strength by 7% whereas applying the adaptive selection of strategies by 24%. A combination of both enhancements results in a 28%

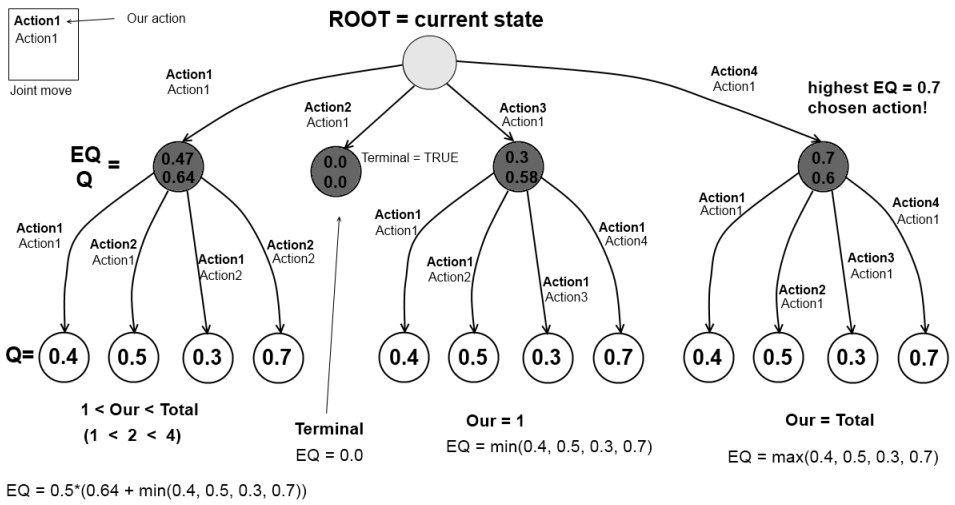

Fig. 4. An example showing the move selection procedure based on Effective Quality value. For the sake of completeness of the figure all four cases considered in Table III are presented, even though, in practice, such a situation is relatively unlikely to happen. The upper and lower values in the first-level nodes represent the EQ and Q values, respectively.

higher score, on average. In addition to the competition-based games, the test set included five single-player games, the results for which are shown in Table VIII. Contrary to multi-player games, which are generally of a competitive nature, in all single-player games the players were to find a solution for the game rather than compete with one another. For each game, the tests were repeated 270 times and the results were averaged. The comparison was even (2 wins of Cadia, 2 wins of our player, and a draw in one game), although our player achieved a better average score across all games.

In order to have an insight into the internal decision-making process related to the strategy selection we compared the scores assigned to each strategy at the end of the START clock phase, i.e. just before making the first move in a game for all nine tested two-player games. The results are presented in Table IX.

The average values are relatively low because in two games: Farming Quandries and Pilgrimage it was very difficult to reach higher payoffs. Many games ended with $0-0$ or $10-0$ scores (where 100 is the maximum). Interestingly, the results obtained using Exploration strategy in Farming Quandries are exceptionally high in comparison to the others.

In general, each strategy, except for Mobility and Random, was superior in at least one of the games. On the other hand, strategy R, though never being the strongest one attained second place in four games. The most successful were strategies HH and E, each appeared to be the best choice in three games. A close follower was AGE with two wins and three second

TABLE VI
RESULTS OF MINI-PLAYER USING ONLY ONE OF TWO FEATURES: MINI-EQ USES A MODIFIED MOVE DECISION; MINI-STR USES A MIX OF STRATEGIES WITH STANDARD MOVE DECISION VS. CADIAPLAYER. SEE DESCRIPTION OF TABLE IV FOR CLOCK VALUES AND INTERPRETATION OF RESULTS.

Game	MINI-EQ vs. Cadia	MINI-STR vs. Cadia
Connect4	54.84 [5.85]	40.00 [5.42]
Cephalopod Micro	36.67 [5.75]	40.00 [5.84]
Free for all 2P	39.81 [5.31]	59.56 [5.39]
Pentago	45.93 [5.94]	31.33 [5.53]
9 Board Tic-Tac-Toe	49.33 [5.72]	66.67 [5.41]
Connect4 Suicide	60.67 [5.40]	51.33 [5.33]
Checkers	24.00 [5.05]	53.67 [5.93]
Farming Quandries	39.33 [4.60]	64.67 [4.54]
Piligrimage	35.67 [4.09]	40.33 [4.55]
Average	**42.91** [5.30]	**49.73** [5.33]

TABLE VII
RESULTS ACHIEVED BY FOUR VERSIONS OF MINI-PLAYER AGAINST CADIAPLAYER RELATIVE TO THE BASE MINI-PLAYER-C'S PERFORMANCE.

Game	MINI-C	MINI-EQ	MINI-STR	MINI
Connect4	1.00	1.36	0.99	1.03
Cephalopod Micro	1.00	1.38	1.50	1.50
Free for all 2P	1.00	1.16	1.73	1.84
Pentago	1.00	1.03	0.70	0.66
9 Board Tic-Tac-Toe	1.00	0.99	1.33	1.41
Connect4 Suicide	1.00	1.07	0.91	0.92
Checkers	1.00	0.78	1.75	1.77
Farming Quandries	1.00	0.98	1.60	1.69
Piligrimage	1.00	0.96	1.09	1.15
Average	**1.00**	**1.07**	**1.24**	**1.28**

TABLE VIII
RESULTS OF MINI-PLAYER AND CADIAPLAYER IN SINGLE-PLAYER GAMES.

Game	Clock [s]		MINI-Player	CadiaPlayer
Hanoi	30	20	80.70 [0.55]	99.00 [0.52]
Knight Tour	30	20	100.00 [0.00]	100.00 [0.00]
8-Puzzle	30	20	64.80 [4.82]	3.60 [2.10]
Lights Out	30	20	18.50 [4.63]	0.00 [0.00]
Rubik's Cube	30	20	0.00 [0.00]	13.25 [1.17]
Average			**52.80** [2.00]	**43.17** [0.76]

TABLE IX
THE ASSESSMENT OF STRATEGIES JUST BEFORE THE FIRST MOVE WAS PLAYED IN A GAME. THE HIGHEST SCORES FOR EACH GAME ARE BOLDED.

Game	R	AGE	M	HH	E	SSC
Connect4	54.6	78.1	55.00	**80.6**	58.3	54.0
Cephalopod Micro	49.1	34.9	44.3	**57.9**	34.8	42.6
Free for all 2P	52.4	68.0	58.5	76.2	**94.5**	84.5
Pentago	48.9	**75.7**	43.1	47.0	45.4	45.7
9 Board Tic-Tac-Toe	51.3	**71.8**	49.7	32.0	48.5	46.6
Connect4 Suicide	54.1	51.4	49.0	**59.2**	45.0	49.6
Checkers	48.8	77.7	64.1	54.7	75.0	**78.5**
Farming Quandries	1.0	2.5	0.9	1.0	**37.2**	1.5
Piligrimage	0.1	0.2	0.3	0.4	**0.9**	0.2
Average	**40.0**	**51.1**	**40.5**	**45.4**	**48.8**	**44.8**

places. M was clearly the weakest choice - out of nine games it both failed to win and be a runner up.

The final experiment was aimed at testing synergistic behavior of strategies. In order to address this issue we developed an agent named MINI-1-S which was committed to the highest estimated strategy for a particular game. The question was whether MINI-1-S would be better than a regular version of MINI-Player. In the START clock, MINI-1-S uses all strategies in uniform proportions, and afterwards the strategy with the highest score is selected. The motivation is to test if there is emerging knowledge in alternating between strategies that is not captured by any one of the strategies on their own. The results of this experiment are presented in Table X.

TABLE X
RESULTS OF MINI-1-S PLAYER, WHICH USES ONLY THE BEST STRATEGY VERSUS MINI-PLAYER.

Game	MINI-1-S vs. MINI-Player
Connect4	52.41 [5.90]
Cephalopod Micro	45.92 [5.94]
Free for all 2P	48.33 [5.56]
Pentago	50.00 [5.96]
9 Board Tic-Tac-Toe	52.96 [5.75]
Connect4 Suicide	40.30 [5.60]
Checkers	57.78 [5.78]
Farming Quandries	58.89 [5.33]
Piligrimage	35.63 [5.33]
Average	**49.14** [5.68]

In four games, a mixture of strategies yielded better results than using only the major strategy. In four other games, the player provided with the (presumably) best strategy won against the MINI-Player. The ninth game ended with a draw. Several results are within a very close range, however, the strategy switching mechanism seems to be a safer option in general. The "best strategy" can change with respect to the game phase or game situation. A player using only a single strategy would not be able to react to such a change. Furthermore, the more strategies are considered, the shorter time for testing each of them in the START is allotted. Hence, the confidence in a final selection decreases.

VIII. RELATED WORK

In this section we present a detailed discussion on the differences between our method and the concepts previously introduced in the related literature.

A. Statistical Symbol Counting

In the SSC strategy introduced in section V-C we construct a simple evaluation function based on two types of elements: a number of state facts for each fact type and symbols counted in their respective columns. The way the two elements are defined, weighted and used in the evaluation function in the current approach is taken directly from our earlier work [16]. One significant difference is that, in the previous approach we included a third type of function component, called DynamicSymbols, of a far more complex definition. Roughly speaking, the idea was to detect symbols which to the highest degree change other symbols they appear with in fact realizations. Such symbols are naturally suspected of being pieces moving over the board or other dynamic game features. However, the major difference between SSC Strategy and our earlier work is that previously we used the evaluation function together with the iterative-deepening DFS as a stand-alone playing method. In the present approach we not only use a simplified version of this heuristic, but also the function itself is embedded in one of the six strategies. It is therefore used (if active) to guide the MC simulations and not to approximate the nodes' scores in the game tree. Scores in the tree are collected from the results of previously completed simulations.

B. Approximate Goal Evaluation

The idea of computing a degree of truthfulness of a goal formula was introduced in [18] and further analyzed in [19]. It was also implemented in FluxPlayer - the GGP Competition winner in 2006. Even though both FluxPlayer and MINI-Player take advantage of the estimated degree of a goal fulfillment there are two main differences between these two approaches. First of all, in FluxPlayer, the degree of a goal satisfaction is a part of the knowledge-based evaluation function used together with non-uniform iterative-deepening tree search. In our approach, the goal approximation is a part of the lightweight play-out strategy and (if active) is used to guide the simulation. One of the consequences is that in [18] the aim is to cut-off the DFS at some point and return a valid approximated score of a state. Likewise in the previous section on SSC strategy, we use AGE only to choose an action to be taken during a simulation and nodes' scores are exclusively based on the results of simulations. Hence, our approach to goal assessment is simpler than in [18], [19]. AGE is computed much more often and, to some extent, is prone to mistakes, since quasi-random simulations are repeated many times.

The second key difference between the two works is how the goal evaluation is actually computed. The authors of FluxPlayer use fuzzy logic to assign 0 and 1 values to facts which hold in the current state Z:

$$eval = \begin{cases} p, \text{ if } a \text{ holds in } Z \\ 1-p, \text{ otherwise} \end{cases}$$

Complex expressions, i.e. rules are computed using formally defined fuzzy logic t-norms and complementary s-norms:

$$eval(f \wedge g, Z) = T(eval(f, Z), eval(g, Z))$$
$$eval(f \vee g, Z) = S(eval(f, Z), eval(g, Z))$$

Yager norms family was chosen however the authors conclude that there is a degree of freedom in the choosing of a norm. The goal and terminal rules are unrolled, so they contain either facts or basic operations on facts: conjunction, disjunction and negation.

There is a serious problem with the above approach detailed in both [18] and [19], if using a natural candidate for p parameter (in $eval$ definitions), the state evaluation returns 0 if at least one subgoal is not fulfilled. To overcome this problem, the authors decided to choose an arbitrary $0.5 <= p < 1.0$. As long as only one goal rule is taken into consideration, the formula can correctly order states with respect to the degree of goal fulfillment. However, the computed value does not represent the absolute percentage extent to which the goal holds. This makes it difficult to compare goal rules of different definition, e.g. having a different number of conditions. We overcame both of the above-mentioned problems by using the approach presented in section V-B. Technically, we count the number of conditions which hold true in relation to the total number of conditions in AND tree nodes and apply the max norm in OR nodes.

C. Mobility

One of the strategies relies on a concept of mobility which is commonly used in a variety of standard game AI algorithms. In fact, no researcher claims to be the inventor of mobility, because the measure of available moves seems to be very intuitive and native in game domain. In GGP, the first competitive program which used mobility was ClunePlayer [17]. It was defined for two-player games in the following way:

$$\frac{M_{red} - M_{black}}{normalizationFactor}$$

where M_{red} denotes the number of our legal moves and M_{black} the number of opponent's legal actions. The normalization was performed by dividing by the maximum value of mobility over the sampled game states. We can expect that the formula had a serious impact on execution speed as once a greater value had been found, all previous computations had to be repeated. The author introduced a so-called $MobilityStability$ to test whether this heuristic is useful. If the variance of mobility i.e. the changes between consecutive states, was high, the heuristic had proportionally less impact on the overall evaluation function.

The difference compared to our approach is mainly twofold. Unlike in [17], we use the mobility to guide the simulation or more precisely to select an action which leads to a state with the highest mobility (consequently, we do not need to normalize this value and therefore avoid a lot of computation). Furthermore, we do not use a stability of mobility, since it is up to the strategy evaluation procedure to detect if this method is useful for a particular game.

IX. Conclusions

The paper describes a General Game Playing agent named MINI-Player, which combines various playing strategies in order to optimize the Monte Carlo Tree-Search process. The system evaluates the strategies and adapts their usage in real-time: the better the strategy is suited for a game (i.e. the higher it is evaluated), the more simulations use it as their playing policy.

MINI-Player took part in the official 2012 General Game Playing Competition. Despite some inefficiencies in play it managed to reach the final round, winning some games against former champions. Furthermore, the results of experiments evidently show an advantage of the MINI-Player's playing skills over the plain UCT-based player (MINI-Player-C). Since both players use the same code framework the comparison is straightforward and there are no external factors which might have come into play during the experiment.

Moreover, in a series of 2430 multi-player matches (9 games, each played 270 times) and 1350 single-player matches (5 games, 270 repetitions), MINI-Player appeared to be comparable to last year's version of CadiaPlayer, the current world champion and the most successful player ever. Performance in single-player games of both agents is also at a similar level. The results against CadiaPlayer proved that the proposed approach has potential and is worth further investigation and development. Certainly, the exact match-up is hard to judge since the pool of games available in GGP is practically unlimited and the decisive factors for gaining advantage in particular games are generally unknown.

Although our assessment of MINI-Player's play is generally very enthusiastic and favorable, the agent is definitely far from being a perfect player and suffers from several weaknesses. First of all, the efficiency of strategies measured during MC simulations does not always translate into a better play. This can be concluded from MINI-Player vs. MINI-Player-C direct tests, where in three games MINI-Player-C managed to score higher results.

We also observed that sometimes, for instance in Checkers, it would be better not to use statistical averaging of node scores, which are stored in the game-tree, but instead choose a move suggested by a certain strategy (e.g. AGE). We tried to include some confidence heuristic to deal with this problem, but have not yet come up with any robust solution.

Another issue is the tendency of MINI-Player to play too optimistically, in some sense relying on an opponents' faults. As observed in [20], this is a general problem with MC-based players. We managed to slightly alleviate this inefficiency by modifying an action selection formula.

General Game Playing is a rapidly growing subfield of game playing within AI/CI. Current GGP players are quite weak when faced against humans who know the rules of a game and possess a basic understanding of its mechanics. There are many possibilities into how such artificial players can be improved. Currently we are working on several enhancements to MINI-Player, notably adding new strategies, implementing more effective opponent modeling methods and including static rules' analysis in the strategy selection process. We also

plan to experiment more with single-player games. In such a class of games there is usually one path (or a small number of them) which leads to a victory and the goal is to find that path. The Exploration strategy seems to be best suited for such a task. However, for the moment we have decided to leave all the strategies to allow a higher variety in play. In general, we plan to incorporate the number of players in the strategy selection mechanism.

ACKNOWLEDGMENT

M. Świechowski was supported by the Foundation for Polish Science under International Projects in Intelligent Computing (MPD) and The European Union within the Innovative Economy Operational Programme and European Regional Development Fund.

This research was financed by the National Science Centre in Poland, based on the decision DEC-2012/07/B/ST6/01527.

REFERENCES

[1] M. Genesereth and N. Love, *General Game Playing: Overview of the AAAI competition*, AI Magazine, vol. 26, pp. 62-72, 2005.

[2] N. Love, T. Hinrichs, D. Haley, E. Schkufza, M. Genesereth, *General GamePlaying: Game Description Language Specification*, Technical Report LG-2006-01, 2006. Available at: `http://games.stanford.edu/`

[3] Dresden GGP Server site. Available at: `http://euklid.inf.tu-dresden.de:8180/ggpserver/public/show_games.jsp`

[4] GGP.org games repository. Available at: `http://www.ggp.org/view/tiltyard/games`

[5] J. Pitrat, *Realization of a general game-playing program*, IFIP Congress, pp. 1570-1574, 1968.

[6] M. Gherrity, *A Game Learning Machine*, Ph.D. Thesis, University of California, San Diego, 1993.

[7] S.L. Epstein, *Toward an ideal trainer*, Machine Learning, vol. 15, no. 3, pp. 251-277, 1994.

[8] B. Pell, *Metagame: A New Challenge for Games and Learning*, Heuristic Programming in Artificial Intelligence 3 - The Third Computer Olympiad. Ellis Horwood, pp. 237-251, 1992.

[9] Y. Wang and S. Gelly, *Modifications of UCT and sequence-like simulations for Monte-Carlo Go*, in Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 175-182, 2007.

[10] L. Kocsis and C. Szepesvari, *Bandit based Monte-Carlo planning*, In Proceedings of the European Conference on Machine Learning (ECML), LCNS, Springer-Verlag, pp. 282-293, 2007.

[11] C. Shannon, *Programming a Computer for Playing Chess*, Philosophical Magazine, vol. 41, no. 314, 1950.

[12] T.A. Marsland *Evaluation-Function Factors*, Journal of the International Computer Chess Association, vol. 8, no. 2, pp. 47-57, 1985.

[13] H. Nguyen, K. Ikeda and B. Le, *Extracting Important Patterns for Building State-Action Evaluation Function in Othello*, Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp.278,283, 2012.

[14] U. Lorenz and T.Tscheuschner, *Player modeling, search algorithms and strategies in multi-player games*, In Proceedings of the 11th international conference on Advances in Computer Games, pp. 210-224, 2005.

[15] J. Schaeffer, *The history heuristic and alphabeta search enhancements in practice*, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-11, vol. 11, pp. 1203-1212, 1989.

[16] J. Mańdziuk and M. Świechowski, *Generic Heuristic Approach to General Game Playing*, In SOFSEM 2012 Proceedings, Lecture Notes in Computer Science 7147, pp. 649-660, 2012.

[17] J. Clune, *Heuristic evaluation functions for general game playing*, In Proceedings of the 22nd AAAI Conference on Artificial Intelligence, The AAAI Press, pp. 994-999, 2007.

[18] S. Schiffel and M. Thielscher, *Fluxplayer: A successful general game player*, In Proceedings of the 22nd AAAI Conference on Artificial Intelligence, The AAAI Press, pp. 1191-1196, 2007.

[19] S. Shiffel, *Knowledge-Based General Game Playing*, PhD Thesis, Technische Universitt Dresden, 2011.

[20] H. Finnsson and Y. Björnsson, *CadiaPlayer: A Simulation-Based General Game Player*, IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 1, pp. 4-15, 2009.

[21] J. Mhat and T. Cazenave, *A Parallel General Game Player*, Künstliche Intelligenz, vol. 25, no. 1, pp. 43-47, 2010.

[22] G. Kuhlmann, K. Dresner and P. Stone, *Automatic Heuristic Construction in a Complete General Game Player*, In Proceedings of the Twenty-First National Conference on Artificial Intelligence, pp. 1457-1462, 2006.

[23] B. Brügmann, *Monte Carlo Go*, Masters Thesis, Max-Planck-Institute of Physics, 1993.

[24] C-S. Lee, M-H. Wang, G. Chaslot, J-B. Hoock, A. Rimmel, O. Teytaud, S-R. Tsai, S-C. Hsu, T-P. Hong, *The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments*, IEEE Transactions on Computational Intelligence and AI in games, vol. 1, no. 1, pp. 73-89, 2009.

[25] H. Finnsson and Y. Björnsson, *Simulation Control in General Game Playing Agents*, In Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09), 2009.

[26] A.L. Zobrist, *A New Hashing Method with Applications for Game Playing*, Tech. Rep. 88, Computer Sciences Dept., Univ. of Wisconsin, Madison, April, 1970. Also in Int. Computer Chess Assoc. Journal 13(2), pp. 169-173, 1990.

[27] M.J.W. Tak, M.H.M. Winands, Y. Björnsson, *N-Grams and the Last-Good-Reply Policy Applied in General Game Playing*, IEEE Transactions on Computational Intelligence and AI in games, vol. 4, no. 2, pp. 73-83, 2012.

[28] M. Kirci, J. Schaeffer and N.Sturtevant, *Feature Learning Using State Differences*, Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09), 2009.

[29] Yet Another Prolog (YAP) Web site. Available at: `http://www.dcc.fc.up.pt/~vsc/Yap/`

[30] ECLiPSe Constraint Programming System Web site. Available at: `http://eclipseclp.org/`

[31] M. Mller, M.T. Schneider, M. Wegner, T. Shaub, *Centurio, a General Game Player: Parallel, Java- and ASP-based*, Künstliche Intelligenz, vol. 25, no. 1, pp. 17-24, 2011.

[32] Ł. Kaiser and Ł. Stafiniak, *Translating the Game Description Langauge to Toss*, In Proceedings of the 2nd International General Game Playing Workshop, GIGA'11, pp. 9198, 2011.

[33] M. Świechowski and J. Mańdziuk, *Fast Logical Reasoning in General Game Playing*, (submitted)

[34] Cadia-Player project website. Available at: `http://cadia.ru.is/wiki/public:cadiaplayer:main/`

Maciej Świechowski received the B.Sc. and the M.Sc. in computer science from the Warsaw University of Technology, Warsaw, Poland, in 2009. He is currently pursuing the Ph.D. in the Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland under the International Projects in Intelligent Computing Programme.

His research interests include artificial intelligence in games, general game playing, graph theory, computational intelligence, and computer graphics. He has participated in the 2012 International General Game Playing Competition. He has also acquired practical programming experience in the mobile video games industry.

Jacek Mańdziuk (M'05, SM'10) is a Professor of Computer Science at the Warsaw University of Technology, Warsaw, Poland. He received the M.Sc. (Honors) and Ph.D. degrees in Applied Mathematics from the Warsaw University of Technology, Poland, in 1989 and 1993, respectively and the D.Sc. degree in Computer Science from the Polish Academy of Sciences, in 2000. In 2011 he was awarded the title of Full Professor (Professor Titular) by the President of the Republic of Poland. His current research interests include application of Computational Intelligence methods to game playing, bioinformatics, financial modeling and development of meta-heuristic general-purpose human-like learning methods.

He is the author of two books (including Knowledge-free and Learning-based Methods in Intelligent Game Playing, Springer 2010) and co-author of one textbook and over 90 refereed papers. Recently he has been twice a Chair of the IEEE SSCI Symposium on Computational Intelligence for Human-like Intelligence (Singapore 2013, Orlando 2014), a Program Co-Chair of the International Workshop on Adaptive Systems in Soft Computing and Life Sciences, and a panelist in Computational Intelligence and Games panel at the IEEE WCCI 2008.

Prof. Mańdziuk is an Associate Editor of the IEEE Transactions on Computational Intelligence and AI in Games, an Editorial Board Member of the International Journal On Advances in Intelligent Systems, a member of the Games Technical Committee of the IEEE CIS and a member of the Intelligent Systems Applications Committee of the IEEE CIS. He has been a founding chair of the IEEE CIG Task Force on Neural Networks for Games (since 2008) and a founding chair of the IEEE CIS Emergent Technology Technical Committee Task Force on Towards Human-like Intelligence (since 2011). He is a recipient of the Fulbright Senior Advanced Research Award.