Specialized vs. Multi-Game Approaches to AI in Games

Maciej Świechowski[1] and Jacek Mańdziuk[2]

[1] Phd Studies at Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
m.swiechowski@ibspan.waw.pl
[2] Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland
j.mandziuk@mini.pw.edu.pl

Abstract. In this work, we identify the main problems in which methodology of creating multi-game playing programs differs from single-game playing programs. The multi-game framework chosen in this comparison is General Game Playing, which was proposed at Stanford University in 2005, since it defines current state-of-the-art trends in the area. Based on the results from the International General Game Playing Competitions and additional results of our agent named MINI-Player we conclude on what defines a successful player. The most successful players have been using a minimal knowledge and a mechanism called Monte Carlo Tree-Search, which is simulation-based and self-improving over time.

Keywords: Games, Artificial Intelligence, General Game Playing, Heuristic Search

1 Introduction

The focus of this study is to review some of the approaches to AI in games and elaborate on a shift from these approaches to multi-game playing. We emphasize the difference between those two approaches while focusing on the latter. Games have been a legitimate challenge of AI since the early days of the field. There have been some common traits formulated by various researches [25], [21], [19], such as deduction, reasoning, problem solving, intelligent search, knowledge representation, planning, learning, creativity, perception, motion (manipulation) and natural language processing. Most of them, especially the first seven, are part of intelligent game playing. Researchers started to work on AI in the so-called mind-games, often driven by the ultimate goal of getting closer towards general intelligence. The programs began to be more and more specialized. Despite displaying often a remarkable set of skills, their intelligence and usefulness of the applied methods in the real-world problems are questionable. A top level chess-playing computer program [10] can defeat any human in chess but cannot help to form a medical diagnosis or even play a simple Tic-Tac-Toe game. A lack for multi-game playing systems has emerged. There have been many attempts

to bring back to life the early-day concepts of AI, such as SAL [9], Morph [16], Hoyle [5] and METAGAMER [26]. The most recent and probably the most well-designed is General Game Playing [8] which we introduce in the next section. In Section III we enumerate key differences to the methodology in both of the aforementioned disciplines. Next, we review a variety of heuristics applied to General Game Playing some of which are contributed by ours. Then, in section V, we provide some experimental results and conclude in section VI.

2 General Game Playing

General Game Playing (GGP) [8] is a realization of the multi-game concept and the current grand challenge of AI in games. The term was coined at Stanford University in 2005, together with specification of the international General Game Competition. The competition is held annually at the AAAI Conference (one exception is IJCAI in 2011) and works as an official world championships. The winners define the state-of-the-art solutions in the field. The GGP is about creating agents capable of playing games which can be defined in the Game Description Language (GDL) [17]. This declarative-logic language is a subset of Prolog. It is used to define rules of games as well as for communication between the agents and the Game Manager. The Game Manager is a communication hub between the players and a referee checking ensuring if the game is played legally. The communication, realized via HTTP, is also part of the GGP specification. General Game Playing revisits the early AI concepts. With programs tailored for playing specific games, most of the interesting analysis is done by the authors (or consulted experts). The goal of the GGP is to transfer this analysis to the programs, which start from scratch, without any knowledge. The programs are also autonomous i.e. no human intervention is possible.

3 General vs. Specialized Game Playing

In the previous section we emphasized the characteristic concepts of General Game Playing. There are various practical differences in research tools, key problems, implementation and the overall research mind-set, when comparing GGP and specialized game playing. We identify some of them in this section.

A) Rules Representation

In General Game Playing a rules interpretation system also known as an inference engine is required to be able to determine the game states and perform any kind of search. The GDL, which rules are represented in, is a declarative first-order logic language which contains no high level game-related logic. It means, that every concept present in games such as a piece, board, coordinate or adjacency of coordinates has to be defined from scratch within the language and there is no meta-information on what the concept is. Even mathematical formulas such as addition must be defined by logical rules for every arguments

and results. Such an approach is very general but very costly at the same time in terms of computational efficiency. As a consequence, the first choice to make in GGP is an inference method. It affects the player in a way how fast it will be able to search the game tree and also what the fidelity of the simulation will be. The more control over the inference process the more game-playing algorithms can be put inside which use the internal/intermediate state of the process. The speed of a rules interpreter also affects the feasible game-tree search which can be applied. It also quite convincingly hampers the methods of computational intelligence to be used in GGP as long as the start and play clocks apply. There are a few approaches to the problem of rules interpretation. Some players convert rules directly to Prolog. FluxPlayer uses fluent calculus implemented in ECLiPSE Prolog [1]. Many agents use custom-made interpreters for the GDL. Another problem, apart from efficiency, arises with the rules representation. In General Game Playing, we cannot design an efficient representation of a state because we do not know dimensions of the state (state-space, possible elements and arities of the elements) in advance. We do not even know what defines the state exactly although a sensible approach is to approximate the state by taking all facts used by *init* and *next* relations. An optimized representation such as 0x88 and Zobrist's Hashing in chess, can be used for Transposition Tables [12] and serialization required for the communication in a distributed parallel system.

B) Access to knowledge

In General Game Playing, there is no such thing as expert knowledge. Not only any human-intervention is illegal but also the games can be unknown beforehand or obfuscated so there would not exist an expert in most games. Moreover, the start clocks are usually set to from a couple of seconds to a couple of minutes, so there is no time for too complex methods. In specialized game playing lots of analysis can be done either by experts or by a long off-line tuning. In General Game Playing the agents either has to use robust simulation-based methods or discover the knowledge relatively quickly. It can be assumed that the discovered knowledge will not be very deep, but it may change in future when the GGP area becomes more advanced. The lack of expert knowledge is connected with the lack of databases of opening and endings. When a game is known, players quickly discover the reasonable ways how to start. Subsequent responses to the start moves are studied, players seek new moves and a database of opening grows. The endings are usually calculated using brute-force methods when the complexities of states become low enough to allow for this. Those two features are staples in single-game programs. Lastly, games in General Game Playing contain no interpretation of what particular predicates mean i.e. there is no connection to the game-specific objects. Even if there was such a connection defined in the rules (some kind of meta-information) both the designers of games and programs would have to predict names of all the possible game objects in advance. If the games' authors (or automatic generators) were not able to define custom types for the game objects, the GGP would not be general anymore. To sum it up, there are no predefined game-specific elements which agents could use

in construction of evaluation functions. Every building block for the evaluation function has to be discovered at runtime and only with certain probability of being accurate.

C) Game tree search

The introduction of start and play clocks requires General Game Playing programs to use the so-called anytime game tree search methods i.e. which can be interrupted at any time and return the currently best action. Moreover, in GGP, a search method cannot use reaching a particular tree level as a stop condition because the required time cannot be predicted beforehand. The most popular search method in GGP is Monte Carlo Tree-Search (MCTS) [3] which will be presented in the next section. Lack of heuristics may limit possible usage of methods such as alpha-beta pruning or MTD [27]. The problem which does not exist in the specialized game-playing is that in multi-game playing the chosen tree search should be universally good at the cost of being suboptimal in some games.

D) Diversity of games

There are many properties of games which can be took advantage of when designing an AI algorithm. Such properties include:

- Is the game zero-sum or not - zero-sum games can use the plain min-max idea. Moreover, there is no need to store all scores.
- Is the game co-operative or competitive - co-operation, to implement well, requires a completely different method.
- How many players there are - puzzles are inherently different whereas the goal of two-player games is usually more focused on beating an opponent. The game-tree structure can be also optimized for the number of players.
- Is the game turn-based or simultaneous - simultaneous games are usually more difficult. Action selection formula for turn-based games can be optimized.
- What is the Branching-Factor - a huge branching factor can render some game-tree search methods useless whereas a tiny branching-factor can make solving a game a viable attempt.
- Is there a board in the game - there have been various board-related concepts well-studied and established in games.

4 Heuristics in GGP

In this section we go through various heuristics applied in General Game Playing. We include both the related work and our contributions. We believe that the existence of such a rich variety of approaches shows that there is no single best solution to apply. Moreover, although the strongest programs use the Monte Carlo Tree-Search (MCTS) combined with the UCT [14] algorithm, which has become a *de-facto* standard, it is not clear whether with improvement in the heuristics field a different type of search will be a dominant one.

4.1 ClunePlayer

ClunePlayer [4] is the name of the first (2005) GGP Competition champion. It used the min-max search with an evaluation function. The key idea was an abstract model representing the so-called simplified game: $P : \Omega \rightarrow [0, 100]$ - a function which approximates the payoff $C : \Omega \rightarrow [-1, 1]$ - control: a relative mobility $T : \Omega \rightarrow [0, 1]$ - probability that a state is terminal $S_p : [1, \infty]$ - stability of the payoff $S_m : [1, \infty]$ - stability of the mobility; the Ω represents the set of legal states. The payoff function uses cardinality and distance between features as the building blocks. A feature is a GDL expression, which denotes a condition on a fact. Because its a condition and not the ground fact, it can contain uninstantiated variables. Candidate features are all conditions using facts defined in the GDL rules as well as new ones which are created by replacing variables with symbols of the respective domains. The cardinality means how many of facts fulfill the condition. The distances are used if a board relation and a next relation, which defines ordering of the board's coordinates, are detected semantically in the description using predefined template. Only stable features are used in the payoff function, where stability is computed based on the so-called adjacent variance (between the consecutive states) and total variance. The mobility is computed as follows:

$$C = \frac{Moves_a(\omega) - Moves_b(\omega)}{max(Moves_a(\omega) - Moves_b(\omega))} \tag{1}$$

where $Moves_a$ denotes the number of our player's legal moves. The formula as well as the whole approach is dedicated to two-player games only. The termination function is computed purely statistically using least-squares regression. The final formula for the evaluation function used in the min-max search combines five elements of the abstract game:

$$V = T * P + (1 - T) * [(50 + 50 * C) * S_c + S_p * P] \tag{2}$$

4.2 FluxPlayer

FluxPlayer [30] has been the second (2006) Competition champion. The key features of FluxPlayer are: using fuzzy logic to determine the degree to which any given state is terminal; identification of a few syntactic structures; fluent calculus implemented in a custom-made tool for reasoning in the GDL. In GDL, the rules and facts can be either true or false, there is no such concept as partial satisfaction. Moreover, the goal rule is defined to give a meaningful result only a in terminal state. However, if a rule is FALSE, it is convenient to have a mechanism of determining how close the rule is to TRUE. Moreover, it is useful to test the goal rule in non-terminal states because it is often a structural representation of the game objective. FluxPlayer is equipped with such a mechanism to reason about the terminal and goal rules. The procedure is recurrently applied to the GDL rules. When it reaches simple expressions (condition on facts) it returns 1 or 0 depending if the condition is true or not. More complex expressions can be

AND operators (between conditions in a rule), OR operator (defined explicitly between conditions or implicitly between implications) and distinct conditions. Distinct and negative conditions are ignored, whereas for AND and OR conjunctions, t-norms and s-norms are used respectively. The authors settled down with norms of the Yager family. Structures are identified using a syntactic templates. For example an order relation such as (*next* 1 2) is a binary, anti-symmetrical, injective and functional relation. The possible structures are step counter, order relation, board's pieces, quantities, control (turn) and board (ordered domains + pieces). The identified structures are used in a heuristic evaluation function which improves the basic fuzzy-logic goal/terminal satisfaction procedure. Where possible, complex expressions are replaced by the structures if the respective rule uses them. In general, the evaluation function still evaluates the degree of fulfillment for a terminal and goal states but in a more informed fashion. The search method used on top of this function is a modified iterative-deepening depth-first search with transposition tables. If a game is detected to be zero-sum and two-player, the alpha-beta pruning technique is used.

4.3 Boards Continued

Board detection is present in various General Game Playing approaches. One of them [11] is inspired by Hoyle [5] where game evaluators of two kind (structure and definition) are proposed following the idea of Hoyle's advisors. All of the game structure evaluators such as distance-initial, distance-to-target, count-pieces and occupied-columns are related to a two-dimensional board which is discovered using pattern detections algorithm. The novelty of the approach are smart algorithms used to sort GDL facts to store them in a consistent manner. Variance analysis helps to discover pieces which move around the board. A similar approach to this and FluxPlayer's [30] was proposed by [15]. The detection of boards and candidate features for an evaluation function is essentially similar but the application of the function is different. The authors introduce heuristics as linear mappings from the lowest to the highest goal value parametrized by the numeric value assigned to features (each one scaled to the [0,1] interval). The approach uses almost 200 of the so-called slave nodes from which each one is equipped with one heuristic. Another example exploring a board detection is [23] but this time boards of any shapes are taken into account. A distance between two board positions depends on which piece's movement we are interested in. The distance is equal to the number of steps needed to move a particular piece from one position to another. However, one of the problems of this approach is requirement to convert the GDL rules into a Disjunctive Normal Form (DNF) which is not always feasible as well as not taking certain constructions into account such as distinct keywords or negative conditions.

4.4 Monte Carlo Tree-Search Variations

The method was first proposed [2] as an approach to Go and was initially considered not serious. However, a confidence-based algorithm originating from gam-

bling mathematics called Upper Confidence Bounds Applied for Trees (UCT) [14] made the MCTS the state-of-the-art approach both in Go and General Game Playing. The MCTS together with the UCT works as follows: Start from the root of a game-tree and traverse down until reaching a leaf node. During this traversal choose an action a^*, i.e. an edge in the tree, according to the following formula:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a))}} \right\} \tag{3}$$

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. The action which falls out of the tree and the subsequent state define a new node which is added. This phase is called expansions since the tree expands. Next, starting from that state a random simulation is performed until reaching a terminal state (a simulation phase). In general, players are simulated by making random actions but this is a room for potential algorithms altering that choice. We will present several of them. The last phase is called back-propagation in which the goals achieved by players in the simulation are used to update the statistics in the tree. If the MCTS visits a terminal state inside the selection phase then the simulation phase is skipped.

Monte Carlo Tree-Search is the method of choice of all the GGP Competition's winners since 2007: CadiaPlayer [6], Ary [22] and TurboTurtle. Our agent, called MINI-Player [32] also is based on this technique. Many enhancements to the plain algorithm have been proposed. The most widely used is History Heuristic [29]. History Heuristic was proposed initially to determine order of branches to be visited by alpha-beta algorithm in Chess. It was first used in General Game Playing either by CadiaPlayer [6] or FluxPlayer [30]. The key idea is that actions occur repeatedly in simulations and certain actions are universally strong i.e. independent of the state they are taken in. Examples of actions which are usually good are capturing of pieces or placing markers in the middle in the classic connect-n games. The first application of the history heuristic was to assign non-uniform probability of choosing an action during a Monte Carlo simulation. The process was modeled as a Gibbs Sampling and the historical average score defined a parameter Q of the Gibbs Distribution. Then CadiaPlayer simplify the way history heuristic was used. The authors called the method ϵ greedy which means that an action with the best historical score is chosen with probability equal to ϵ and otherwise a random move is chosen. MINI-Player uses a similar approach. An extension of History Heuristic is called N-grams [33] in which not just one historical action is taken into account but arbitrary longer sequences. A similar approach presented in [33] as well is Last-Good Reply Policy (LGRP). The idea is to store the best replies for particular moves, think of a counterattack. The method originates from Go [2]. MINI-Player uses a portfolio of heuristics called strategies [32] because they encapsulate the whole logic of how an action during a semi-random simulation is chosen. The strategies include

History Heuristic, Mobility, Approximate Goal Evaluation, Statistical Symbols Counting and Exploration. The first three are quite self-explanatory. The exploration heuristic is aimed at choosing states which are bring the highest possible difference to explore the state-space more efficiently. In order not to get stuck in a local minimum we maximize the difference between the next consecutive state and the most similar state to it among several last visited ones. The Statistical Symbol Counting is our approach for board-like heuristics where we count occurrences of symbols at certain positions in their respective arguments lists. We assign weights to that occurrences according to the correlation between their average count and the game outcome. An evaluation function is then constructed as a linear combination of the respective weights and occurrences. A detailed description is contained in [20].

4.5 Selected other concepts

A player called MAGICIAN [34] features a construction of a simulation-based evaluation function. The function is based on counting and weighing the GDL terms in a similar fashion to ClunePlayer's and FluxPlayer's. The process is split into five phases: initialization, generalization, specialization, selection and weighting. The generalization phase replaces symbols found in the GDL description by "?" which means any symbol. Then the specialization generates all possible combination of terms by instantiating symbols from their domains. This way, more terms can be detected as useful candidates for the evaluation function than in the previous approaches. Another novel part is the way how the evaluation function is combined with the UCT search. When a GGP game is simple enough the question arises whether it can be solved. One of the approach aimed at solving games is [13]. It attempts to instantiate all rules in a DNF form and then use a symbolic breadth-first search (BFS) algorithm to generate the whole game graph to solve the game. We found three works involving setting neural networks in the GGP scenario. In the first one [24] a constructed neural network works as a state evaluation function. A dependency graph of GDL rules is translated directly into a neural network by the $C-IL^2P$ [7] algorithm. The result is a neuro-fuzzy goal inference engine. Unfortunately, a fully-fledged player was not build on top of this method. Another approach is NeuroEvolution of Augmenting Topologies NEAT [28]. A player named nrng.hazel uses NEAT with a shallow min-max search. NEAT combines genetic algorithms with neural networks. Each genotype encodes a neural network whereas each neural network encodes a heuristic evaluation function. Two populations of neural networks co-evolve. The reproduction uses the so-called explicit fitness sharing in which individuals of one specie share the same fitness level. The last approach also employs co-evolution of populations represented by an ant colony system [31]. Each ant models a General Game Playing agent which performs cyclical simulations leaving pheromone. A simulation is driven by a local knowledge, common cultural knowledge and the pheromone. The local knowledge is a simple evaluation function being a linear combinations of GDL state terms. The pheromone is defined as a factor which alters the weights in the evaluation function. The more

often a particular GDL term appears during a simulation the higher weight it gets. The global knowledge are sequences of states which appear in simulations of ants with the highest fitness level. The fitness level is computed in a tournament between populations where ants of one population play against random ants from the other population. It is proportional to the number of wins.

5 Results

Let us start from the results of the International General Game Playing Competitions. Most of the published work is expected to contain some positive results but both the choice of games and opponents can be arguable. The GGP Competitions use a decent variety of games and, moreover, there is no reason not to participate with a good player, so it is relatively safe to say that the winners are all around the most efficient and universal players. In the Table 1 we present the top 2 players from all the previous competitions and whether they are based on MCTS/UCT. In Table 2 we show the results of our own MINI-Player

Table 1. Results of MINI-Player vs. MINI-PlainUCT denoting a player without additional heuristics. A 95% confidence intervals are included in the square brackets

Year	**Winner**	**Runner-up**	**Winner MCTS?**	**Runner-up MCTS?**
2005	ClunePlayer	Goblin	NO	NO
2006	FluxPlayer	ClunePlayer	NO	NO
2007	CadiaPlayer	ClunePlayer	YES	NO
2008	CadiaPlayer	ClunePlayer	YES	NO
2009	Ary	FluxPlayer	YES	NO
2010	Ary	Maligne	YES	YES
2011	TurboTurtle	CadiaPlayer	YES	YES
2012	CadiaPlayer	TurboTurtle	YES	YES
2013	TurboTurtle	CadiaPlayer	YES	YES

playing 270 matches against itself stripped down only to the basic MCTS/UCT player. Although the fully-fledged player using strategies to guide the Monte Carlo simulations fares better in 7 of 9 games, the advantage is not as big as it could be expected. In some games such as Pentago or Pilgrimage, the simplest approach is better while in Connect-4 Suicide the MINI-Player's winning margin is insignificant. Players based on neural networks have not been strong in General Game Playing so far. The same rule applies for other approaches in the spirit of the computational intelligence which are common in the non-GGP game systems [18]. Restrictive time limits and lack of knowledge transfer hinder any long-term learning.

Table 2. Results of MINI-Player vs. MINI-PlainUCT denoting a player without additional heuristics. A 95% confidence intervals are included in the square brackets

Game	MINI vs. MINI-PlainUCT
Connect4	61.33 [5.69]
Cephalopod Micro	59.33 [5.86]
Free for all 2P	75.33 [4.86]
Pentago	40.00 [5.84]
9 Board Tic-Tac-Toe	66.30 [5.42]
Connect4 Suicide	51.33 [5.72]
Checkers	79.33 [4.83]
Farming Quandries	66.67 [5.36]
Pilgrimage	39.33 [5.40]
Average	**59.88** [5.44]

6 Conclusions

We have shown a numerous approaches to General Game Playing. Most competitive players use the Monte Carlo Tree-Search as the baseline method with either none or only light-weight heuristics. Some other players use a min-max inspired depth search whereas some other use neural networks without any deep search. Since 2007, only simulation-based players have been the winners of the GGP Competition. Together with our results, we conclude that in General Game Playing, a too heavily marked heuristic bias is unfavorable on a large-enough variety of games. The MCTS approaches have advantage of *not being inaccurate* for any game. That means that while not being predefined for any class of games, the players will always converge (at some arbitrary speed) to the optimal play. For some games, the convergence rate may be very slow but it is still better than inaccurate heuristic which is a barrier preventing a player from being successful. Not suitable heuristics may even make the player going towards the wrong goals. We predict that a truly multi-game player should be either aheuristic, using statistical methods and crunching simulations efficiently or based on self-adapting heuristics. Self-adaptation means that heuristics can be based on any aspect of the played game (**variety**) and that only such heuristics are currently used which are suitable for the game (**accuracy**). Although the MCTS currently dominates, it is interesting to see which approach will the most be successful in the future competitions.

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 partially funded by the National Science Centre in Poland, based on the decision DEC-2012/07/B/ST6/01527.

References

1. Apt, K.R., Wallace, M.: Constraint Logic Programming using Eclipse. Cambridge University Press, New York, NY, USA (2007)
2. Brgmann, B.: Monte carlo go. Technical report, Max Planck Institute for Physics (1993)
3. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1) (2012) 1–43
4. Clune, J.: Heuristic Evaluation Functions for General Game Playing. In: AAAI, AAAI Press (2007) 1134–1139
5. Epstein, S.: Toward an ideal trainer. Machine Learning **15**(3) (1994) 251–277
6. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: AAAI, AAAI Press (2008)
7. Garcez, A.S.d., Gabbay, D.M., Broda, K.B.: Neural-Symbolic Learning System: Foundations and Applications. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2002)
8. Genesereth, M.R., Love, N., Pell, B.: General game playing: Overview of the aaai competition. AI Magazine **26**(2) (2005) 62–72
9. Gherrity, M.: A Game-learning Machine. PhD thesis, University of California at San Diego, La Jolla, CA, USA (1993) UMI Order No. GAX94-14755.
10. Hsu, F.H.: Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. Princeton University Press, Princeton, NJ, USA (2002)
11. Kaiser, D.M.: Automatic feature extraction for autonomous general game playing agents. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS '07, New York, NY, USA, ACM (2007) 93:1–93:7
12. Kishimoto, A., Schaeffer, J.: Transposition table driven work scheduling in distributed game-tree search. In: Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence. AI '02, London, UK, UK, Springer-Verlag (2002) 56–68
13. Kissmann, P., Edelkamp, S.: Gamer, a General Game Playing Agent. KI **25**(1) (2011) 49–52
14. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Proceedings of the 17th European conference on Machine Learning. ECML'06, Berlin, Heidelberg, Springer-Verlag (2006) 282–293
15. Kuhlmann, G., Dresner, K., Stone, P.: Automatic heuristic construction in a complete general game player. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence. (July 2006) 1457–62
16. Levinson, R.: Morph ii: A universal agent - progress report and proposal. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA (1994)
17. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General game playing: Game description language specication (mar 2008)
18. Mańdziuk, J.: Computational Intelligence in Mind Games. In Duch, W., Mańdziuk, J., eds.: Challenges for Computational Intelligence. Volume 63 of Studies in Computational Intelligence. Springer-Verlag, Berlin, Heidelberg (2007) 407–442
19. Mańdziuk, J.: Knowledge-Free and Learning-Based Methods in Intelligenet Game Playing. Volume 276 of Studies in Computational Intelligence. Springer-Verlag, Berlin, Heidelberg (2010)

20. Mańdziuk, J., Świechowski, M.: Generic heuristic approach to general game playing. In Bielikov, M.r., Friedrich, G., Gottlob, G., Katzenbeisser, S., Tur n, G.r., eds.: SOFSEM. Volume 7147 of Lecture Notes in Computer Science., Springer (2012) 649–660
21. Mccarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence, Edinburgh University Press (1969) 463–502
22. Méhat, J., Cazenave, T.: Ary, a general game playing program. In: Board Games Studies Colloquium, Paris (2010)
23. Michulke, D., Schiffel, S.: Distance features for general game playing. In: Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11). (2011) 7–14
24. Michulke, D., Thielscher, M.: Neural networks for state evaluation in general game playing. In: Proceedings of the European Conference on Machine Learning (EMCL). (2009) 95–110
25. Nilsson, N.J.: Artificial Intelligence: A New Synthesis. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
26. Pell, B.: Metagame: A new challenge for games and learning. In: Programming in Artificial Intellegence: The Third Computer Olympiad. Ellis Horwood, Ellis Horwood Limited (1992) 237–251
27. Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A.: Best-first fixed-depth minimax algorithms. Artificial Intelligence **87**(12) (1996) 255 – 293
28. Reisinger, J., Bahceci, E., Karpov, I., Miikkulainen, R.: Coevolving strategies for general game playing. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, Piscataway, NJ, IEEE (2007) 320–327
29. Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. IEEE Trans. Pattern Anal. Mach. Intell. **11**(11) (November 1989) 1203–1212
30. Schiffel, S., Thielscher, M.: Fluxplayer: A successful general game player. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07), AAAI Press (2007) 1191–1196
31. Sharma, S., Kobti, Z., Goodwin, S.: Coevolving intelligent game players in a cultural framework. In: Proceedings of the IEEE Congress on Evolutionary Computing, Special Session on Computational Intelligence in Games. (2009)
32. Świechowski, M., Mańdziuk, J.: Self-Adaptation of Playing Strategies in General Game Playing. IEEE Transactions on Computational Intelligence and AI in Games (2014) Accepted for publication 20/06/2013. Available in Early Access. DOI: 10.1109/TCIAIG.2013.2275163.
33. Tak, M.J.W., Winands, M.H.M., Bjornsson, Y.: N-grams and the last-good-reply policy applied in general game playing. IEEE Transactions on Computational Intelligence and AI in Games **4**(2) (2012) 73–83
34. Walędzik, K., Mańdziuk, J.: An Automatically-Generated Evaluation Function in General Game Playing. IEEE Transactions on Computational Intelligence and AI in Games (2014) Accepted for publication 08/10/2013. Available in Early Access. DOI: 10.1109/TCIAIG.2013.2286825.