Towards Human-Competitive Game Playing for Complex Board Games with Genetic Programming

Denis Robilliard and Cyril Fonlupt

LISIC, ULCO, Univ Lille–Nord de France, FRANCE

Abstract. Recent works have shown that Genetic Programming (GP) can be quite successful at evolving human-competitive strategies for games ranging from classic board games, such as chess, to action video games. However to our knowledge GP was never applied to modern complex board games, so-called eurogames, such as Settlers of Catan, i.e. board games that typically involve four characteristics: they are non zero-sum games, multiplayer, with hidden information and random elements. In this work we study how GP can evolve artificial players from low level attributes of a eurogame named "7 Wonders", that features all the characteristics of this category. We show that GP can evolve competitive artificial intelligence (AI) players against human-designed AI or against Monte Carlo Tree Search, a standard in automatic game playing.

1 Introduction

Games are a classic AI research subject, with well-known successful results on games like chess, checkers, or backgammon. However, complex board-games, also nicknamed eurogames, still constitute a challenge, which has been initiated by such works as [1] or [2] on the game "Settlers of Catan", or [3] on the game "Dominion". Most often these games combine several characteristics among being no zero-sum games, multiplayer, with incomplete information and random elements, together with little formalized expert knowledge on the subject. Monte-Carlo Tree Search (MCTS), which gained much notoriety from the game of Go [4], seems a method of interest in this context because it does not require background knowledge about the game. Genetic Programming (GP) could also qualify as a possible alternative, for the very same reason.

In their pioneering work [5], Hauptman and Sipper were successful at developing winning strategies for chess endgames with GP, and Sipper also achieved impressive results on Robocode and Backgammon [6]. For instance the top evolved strategy for Backgammon was able to get a win percentage of 62.4% in a tournament against Pubeval, one of the strongest linear neural network player. However to the best of our knowledge, GP has never been used to automatically evolve a competitive player for a complex eurogame.

In order to simplify the obtaining of an AI for eurogames, many published works (whatever the AI engine) use some restricted game configuration or only a limited subset of the game rules. E.g. in [2] no trade interactions between players are allowed, and in [3] only a subset of the possible cards are used.

In this paper we focus on the creation of a GP evolved AI player for the "7 Wonders" (7W) eurogame, presented in the next section. In order to test the potential of GP, one of our requirements is to tackle the full game rules, including the trading mechanism.

In the next section we introduce the 7W game and its rules. Then we present the program architecture that was used for evolving GP players. After dealing with specific issues that arose during implementation or testing, we present our experiments and their results.

2 Description of the "7 Wonders" Game

Board games are increasingly numerous, with more than 500 games presented each year at the international Essen game fair. Among these, the game "7 Wonders" (7W), issued in 2011, obtains a fair amount of success, with about 100,000 copies sold per year and several expansion sets. It is basically a card game, whose theme is similar to the many existing "civilization" computer games, where players develop a virtual country using production of resources, trade, military and cultural improvements.

The 7W game belongs in the family of partially observable, multiplayer, stochastic, and also competitive games (although in any N-player game with $N > 2$, several players may share cooperative sub-goals, such as hindering the progress of the current leading player). All these characteristics suggest that 7W is a difficult challenge for AI.

In a 7W game, from 3 to 7 players[1] are first given a random personal board among the 7 available, before playing the so-called 3 ages (phases) of the game. At the beginning of each game age, each player gets a hidden hand of 7 cards. Then there are 6 rounds, where every player simultaneously selects a card from his hand and either:

- puts it on the table in his personal space;
- or puts it under his personal board to unlock some specific power;
- or discards it for 3 units of the game money.

The last decision (or move) is always possible, while the first two possible moves depend on the player ability to gather enough resources from his board or from the production cards he already played in his personal space. He can also use game money to buy resources from cards played by his left and right neighbors. This trading decision cannot be opposed by the opponent player(s) and the price is determined by the cards already played.

After playing their card, there is a so-called drafting phase, where all players give their hand of remaining cards to their left (age 1 and 3) or to their right (age 2) neighbor. Thus the cards circulate from player to player, reducing the hidden information. When there are less than 6 players, some cards from his original hand will eventually come back to every player . On the 6th turn, when the players have only two cards remaining in their hand, they play one of the two and discard the other (except with some player board conditions).

The goal of the game is to score the maximum victory points (VP), which are awarded to the players at the end of the game, depending on the cards played on the

[1] While the rule allows 2 player games, these are played by simulating a 3rd "dumb" player.

table, under the boards and the respective amounts of game money. The cards are almost all different, but come in families distinguished by color : resources (brown and gray), military (red), civil (blue), trade (yellow), sciences (green) and guilds (purple). The green family is itself sub-divided between three symbols used for VP count.

This game presents several interesting traits for AI research, that also probably explain its success among gamers:

- it has a complex scoring scheme combining linear and non linear features: blue cards provide directly VPs to their owner, red cards offer points only to the owner of the majority of red cards symbols, yellow ones allow to save or earn game money, green ones give their owner the number of identical symbols to the square, with extra points for each pack of three different symbols.
- resource cards have delayed effect : they mainly allow a player to put VPs awarding cards on later turns; this is also the case of green cards that, apart from the scoring of symbols, allow some other cards to be played for free later on.
- there is hidden information when the players receive their hand of cards at the beginning of each age of the game.
- there is a great interactivity between players as they can buy resources from each others to achieve the playing of their own cards. Some cards also give benefits or VPs depending on which cards have been played by the neighbors. Moreover the drafting phase confronts players with the difficult choice of either playing a card that gives them an advantage, or another less rewarding card that would advantage a neighbor after the drafting phase.
- the game is strongly asymmetric relatively to the players, since all player boards are different and provide specific powers (such as resources, or military symbols). Thus some boards are oriented towards specific strategies, such as maximizing the number of military symbols, or collecting green cards symbols, for example.

The number of different cards (68 for 3 players, from those 5 are removed randomly) and the 7 specific boards, together with the delayed effect of many cards and the non linear scoring, make it difficult to handcraft an evaluation function. Notably, the number of VPs gained in the two first game ages (card deals) is a bad predictor of the final score, since scoring points at this stage of the game usually precludes playing resource cards that will be needed later on.

We can give an approximation of the state space size for 3 players, by considering that there are 68 possible different cards, from those each player will usually play 18 cards. We thus obtain $\binom{68}{18} \times \binom{50}{18} \times \binom{32}{18} = 1E38$ possible combinations, neglecting the different boards and the partition between on-table and behind-the-board cards that would increase that number.

3 GP Individual Architecture

Devising a new strategy for a complex board game is typically choosing the best move (or decision) for the player at every decision step of the game. As explained in section 2, a player strategy for 7W is to choose the hopefully “best” couple composed of a card

and a way how to play it (on the board, under the board or discard). Our goal is to evolve a program that is able to cope well enough on average with every game position.

For this purpose, the design of an evaluation function, able to rate how a given board is promising for the current player, is a classic mean of obtaining an artificial player. Once this evaluation function has been developed, one can implement a variant of the well known minimax search algorithm. However, as explained in the presentation of the game, crafting such a function from scratch is very challenging in the case of 7W, due to the many different and often delayed ways of scoring points.

It feels natural to try and see if GP could evolve such an evaluation function. We partly proceed along the lines explored for chess endgames in [5]:

- in a similar way, we want to minimize the depth of the search, using less brute-force power and so use an efficient evaluation function for the current position;
- to the contrary, we will not use complex terminals for GP, for two motives. First there is not much expert knowledge at disposal to suggest complex GP inputs for 7W. Second, we want to try and obtain a good Koza's "A to I" ratio : can GP work out competitive players from raw game attributes ? We will see that the answer is positive.

Terminal set The terminal set is the most sensitive part as it should ideally embrace the whole set of variables that are able to predict the game evolution. Our terminal set is divided into two parts:

- a subset of 17 constant values of type real, regularly spaced in the interval $[-2, 2]$, intended to provide raw components for GP to build expressions;
- a subset of game related terminals. They try to embrace what a casual player may use among the information provided by the current game state. For example, a player will have a look at the military power of his left/right opponent, he may evaluate the number of victory points awarded by his science cards, look at how many different resources he has, and so on. As said above, we want as far as possible to avoid complex terminals, the sole exception being `Xvp` that computes the total number of victory points already gained by the player (a computation which usually needs too much time for a human to do during play).

All game related terminal are listed in Tab. 1. Their computational complexity is low, so they will guarantee a quick GP player.

Many more terminals, reflecting various information that a player can use, could be added. For instance most of the terminals introduced in Tab. 1 can be derived with a left and right version to check the same information for opponents, such as is already done with the military strength.

Function set The function set is kept simple. Following the classic example of Shannon's evaluation function for a chessboard [7] that computes a linear weighted sum of the number of pieces and of some piece combinations, we decided to restrict the function set to the 3 basic operations $\{+, -, *\}$. Note that even if the set may seem limited, it allows non linear combinations of the terminals.

Table 1. Game related terminal set for "Seven Wonders"

Terminal	Value type	Note
Xmil	int	military strength of the current player
Xlmil	int	military strength of the left player
Xrmil	int	military strength of the right player
Xtrade	int	number of own commercial cards (yellow cards)
Xtradrsc	int	number of own yellow cards providing resources
Xcivil	int	number of own civil cards
Xrsc	int	number of own resource cards
Xdiffrsc	int	number of own different resource types
Xscience	int	number of own science cards
Xsciencep	int	victory points earned by own science cards
Xgold	int	number of own gold coins
Xchain	int	number of cards that could be chained (i.e. played for free)
Xage	int	current age (i.e. card deal number)
Xturn	int	current ply of the current age
Xvp	int	total victory points earned so far by the player

4 Computing Individual Fitness

The fitness evaluation is a tricky part when we have to evaluate a player. It is quite obvious that playing against an under-average player or a random player will not provide interesting feedback. Furthermore as the game has some stochastic features, a player may win or lose even if his average level is better than those of his opponents. This means that we must be very careful when evaluating the GP player.

In order to evaluate the GP individuals, we need to oppose them to other AIs. In the absence of any human designed evaluation function that could be used in a minimax framework for providing an opponent, we opted for the development of a rule-based AI, and a MCTS player, that are presented below. Another solution could have been to oppose a GP player to another (or some others) from the GP population: first experiments seemed less promising, so this solution was put aside for the moment.

4.1 A rule-based AI

Designing rules for 7W seems rather easy and feels close to the way a beginner player can proceed, considering the cards already played, those in hand, and deciding one's next move. Our rule-based AI (rule-AI) follows a set of rules of decreasing priority, stated below, for the two first deals of a game (when a card is said as being "always played", it means of course if its cost is affordable):

- a card providing two or more different resource types is always played;
- a card providing a single resource type that is lacking to the rule-AI is always played;

– a military card is always played if rule-AI is not currently the only leader in military strength, and the card allows rule-AI to become the (or one of the) leading military player(s);
– at random either the civil card with the greatest victory points (VP) award or one science card is always played;
– a random remaining card is played;
– as a default choice, a random card is discarded.

Contrasting with the first two deals that mainly involve investment decisions, the last deal of the game, in the so-called third age, appears as the time to exploit the previous choices: the set of rules is now replaced by choosing the decision with best immediate VP reward.

Clearly we do not hope to reach championship level with such a simplistic set of rules, nonetheless the resulting player is able to beat human beginners. A comparison with MCTS success rate is given below in Sect. 4.3.

4.2 MCTS player

The Monte-Carlo Tree Search algorithm (MCTS) has been recently proposed for decision-making problems [8, 9, 4]. Applications are numerous and varied, and encompass notably games [10–14]. In games when evaluation functions are hard to design, MCTS outperforms alpha-beta techniques and is becoming a standard approach. Probably the best known implementation of MCTS is Upper Confidence Tree (UCT), presented below.

The idea is to build an imbalanced partial subtree by performing many random simulations from the current state of the game, and simulation after simulation biasing these simulations toward those that give good results, thus exploring the most promising part of the game tree. The construction is done incrementally and consists in three different parts : *descent* in the partial subtree, *growth* by adding a new child under the current leaf node, *evaluation* of the current branch by a random simulation of the end of the game.

The *descent* is done by using a bandit formula, i.e. at node s, among all possible children C_s, we choose to descend on next node $s' \in C_s$ that gives the best reward according to the formula :

$$s' \leftarrow \arg\max_{j \in C_s} \left[\bar{x}_j + K_{UCT} \sqrt{\frac{ln(n_s)}{n_j}} \right]$$

with $\bar{x}_j$ the average reward for the node j (it is the ratio of the number of victories over the number of simulations), n_j the number of simulations for node j and n_s is the number of simulation for the node s, with $n_s = \sum_j n_j$. The constant K_{UCT} is an exploration parameter used to tune the trade-off between exploitation and exploration. At the end of the *descent* part, a node which is outside the subtree has been reached and is added to the sub-tree (unless this branch already reach the end of the game). In order to *evaluate* this new node, a so-called *playout* is done: random decisions are taken until

the end of the game, when the winner is known and the success ratio of the new node and all its ascendant are updated.

We refer the reader to the literature cited at the beginning of this subsection for more information about MCTS, and also to [15] for a more detailed presentation of our MCTS dedicated to 7W: we simply sketch some implementation details in the following. A single N-player game turn (corresponding to the N player simultaneous decisions), is represented in the MCTS subtree by N successive levels, thus for a typical 3-player game with 3 ages and 6 cards to play per age, we get $3 \times 6 = 18$ decisions per player and the depth of the tree is $18 \times 3 = 54$. Of course we keep the simultaneous decisions, that is the state of the game is updated only when reaching a subtree level whose depth is a multiple of N, thus successive players (either real or simulated) make their decision without knowing their opponent choices for the curent turn. The average node arity can be estimated empirically at an average of 14 children per node, and a good value for K_{UCT}, also obtained empirically is between 0.3 and 0.5.

To implement MCTS one just need to know how to generate the possible moves for the current state of the game. Its drawback is its running time: obtaining a good level of play typically implies to simulate the completion (playout) of several thousands games. To obtain a better trade-off between speed and quality, we increase the number of playouts at game age 2 and 3 since the completion of the game simulations is shorter: when we state N simulations, we mean N for the second game deal, $0.66 * N$ for the first deal, and $1.33 * N$ when playing the last deal (thus it is N simulations on average for the whole game).

The tuning of MCTS, notably the K_{UCT} constant, may be a bit tricky: we presented this in [15], with the comparison of several MCTS enhancements. In the next sections we use a refined value for $K_{UCT} = 0.5$, that appears slightly better, and we present new results.

4.3 Comparison of rule-AI and MCTS

To serve as a reference in Sect. 5, we compared the success rates of the two AIs previously described. When we opposed two rule-AIs to the MCTS player with 1500 and 3000 playouts and $K_{UCT} = 0.5$ we obtain the resulting success rates on 5000 games:

Table 2. Comparisons of success rates (SR) for two identical rule-AIs and one MCTS player with either 1500 or 3000 simulations per move.

# MCTS simulations	rule-AI-0	rule-AI-1	MCTS
1500	18.92% ± 1.09	21.10% ± 1.13	59.98% ± 1.36
3000	17.80% ± 1.06	17.76% ± 1.06	64.44% ± 1.33

As expected the two clones of rule-AI obtains very similar success rates. The MCTS is a better player than rule-AI, and the bigger the number of simulations allowed per move for MCTS, the better the success rate, as expected in theory.

4.4 Choosing Moves and Assigning Fitness for GP

In order for the GP individual to choose a move, we examine the resulting game state of all possible GP moves, together with a random move of the opponents (remember that all players move simultaneously). We usually do not know exactly which cards are in the opponent hands, but we maintain the set of the possible cards in order to sample their possible decisions, doing N *determinizations*. That means that we simulate a random choice of opponents moves on their potential cards N times for each decision of the GP player (see e.g. [16, 15] for an illustration of this technique). We select the GP move that is associated to the biggest evaluation function value summed (or equivalently averaged) on the set of determinizations. This is indeed equivalent to an expectimax search of depth one.

Ideally an expectimax search should sample every possible opponent moves. In our case, as there can be up to 1764 possible combinations of opponent moves due to incomplete information, this would not be practically feasible: remember this must be done for all moves, in all games, for all individuals and all generations. Thus we fix the number of determinizations to $N = 11$, for the sake of rapidity: this is a low value but it already yields a satisfying level of play.

When GP needs the fitness of an individual, we have to assess its quality as an evaluation function. We proceed by playing a set of P games where the GP individual is opposed to other AIs. Finally the fitness is the success ratio (percentage of the games won) obtained by GP on the P games. It proved necessary to train GP on several hundred games to obtain a suitable fitness. Early experiments on 100 independent games gave fragile GP players, that lost almost always with some configurations of player boards (there are 210 such configurations). We settled on 500 games, composed with 25 different random player board configurations and 20 deals per configuration, to assess one individual fitness. Again for the sake of speed, we chose ruleAI as the only opponent for training GP since it is much faster than MCTS, but MCTS could still be used for validation.

5 Experiments and results

We recall that we evolve GP players (strictly speaking evaluation functions) trained against ruleAI, described previously, and the fitness is the success ratio obtained on 500 games. Each GP move is chosen as the one bringing the greatest evaluation value by a depth one expectimax on 11 random determizations of opponent moves.

The default GP parameters are: population size of 25 individuals only, tournaments of size 2, "steady state" worst individual replacement, crossover probability 0.6, point mutation probability 0.15 and 100 generations. The population size is small compared to usual practice, but it was required to reduce the computing time: a GP run against rule-AI still take two days on a 32 cores Intel Xeon CPU E5-4603 v2 @ 2.20GHz machine with multi-threaded fitness computation. The parallelization is done on the 500 games needed to assess a significant fitness. We chose a crossover probability rather less than standard, with a higher than usual mutation rate, for the sake of keeping more diversity in such a small population. Time constraints prevented us to perform a system-

atic study of these parameters, however the results derived from this setting are already satisfying.

Once a GP player is evolved, its success ratio is validated on 5000 games. Half of the validation games are played on the same 25 board configurations used for training, the other half is played on random configurations, and in both cases the card deals are completely independent from the learning phase. Note that while evolution is time consuming, the resulting GP program plays almost instantly. Validation on 5000 games against rule-AI takes less than an hour, while against MCTS it still takes several days, due to the MCTS cost.

5.1 Training GP versus 2 rule-AIs

When trained against two clones of rule-AI, GP yielded a best individual with fitness 0.754 at generation 93, that is the evolved player wins three games over four, while the expected win rate is 0.33 if it were of the same strength as its opponents. Once simplified (with a standard symbolic calculus package) this best individual, listed in Fig. 1, is amenable to some analysis and interpretation.

```
(6*Xdiffrsc + 8*Xmil + 5*Xtrade + 9.5*Xtradrsc + 5*Xvp
- 4.875)*Xtradrsc + 4*Xdiffrsc + 5*Xmil + 5*Xsciencep +
3.0*Xtrade + 17*Xtradrsc + Xvp - 11.5
```

Fig. 1. Best individual trained against 2 rule-AIs, winning 70% of games (obtained at generation 90, average fitness 0.56)

While this individual is not a linear function, it remains a rather simple quadratic polynomial and still looks like a traditional weighted combination of attributes. We can see that some terminals are associated to strong weights, notably `Xtradsrc` the number of cards providing a choice of alternative resources, `Xmil` the military strength and `Xdiffrsc` the number of different available resources. Intuitively these are important inputs since having access to different resources help in developing one's game and it is rather difficult to win a game without any success in the military strategy. Indeed our rule-AI uses similar information as highest criteria for decision taking — but with much less success!

The validation experiment is presented in Tab. 3. The GP fitness being measured on the training set, it proved to be too optimistic, as expected in theory. The validation win rate is nonetheless superior to rule-AI and unexpectedly very close to the success rate of MCTS parametered with 1500 simulations per move. It feels rather remarkable that GP could evolved such a successful formula, that is:

- unique for the whole game,
- using only raw game attributes,
- able to choose the next move with a search of depth only one, which is almost instantaneous.

Table 3. Comparisons of success rates (SR) for two identical rule-AIs and the GP player of Fig. 1.

rule-AI-0	rule-AI-1	GP
22.20% ± 1.15	21.52% ± 1.14	56.28% ± 1.37

5.2 Validation against MCTS with 1500 and 3000 playouts

In Tab. 4 we validate our best GP individual against rule-AI and MCTS. GP is the best of the three, while MCTS wins the second rank. It seems curious since MCTS scored better than GP when opposed only to ruleAI, but this is an illustration of the difficulties raised when opposing three players: the strategies of two players may combined to the detriment of the last. One can notice than MCTS still wins at least one third of the games so it means that it is rule-AI which gives way to either GP or MCTS. By contrast, GP appears rather robust in this context against the increase in MCTS simulations.

Table 4. Comparisons of success rates (SR) for one rule-AI, one MCTS with 1500 playouts, and the GP player of Fig. 1.

# MCTS simulations	rule-AI	MCTS	GP
1500	24.14% ± 1.22	34.13% ± 1.35	41.72% ± 1.40
3000	20.40% ± 1.12	39.11% ± 1.35	40.49% ± 1.36

In Tab. 5, MCTS is opposed to 2 clones of GP. With 1500 playouts MCTS is no more able to win one third of the games, thus clearly meaning that GP has a superior play ability. But once the number of playouts is raised to 3000, this time MCTS is the winner. Again, it is an illustration of the dependencies between more than two players: there is no weak rule-AI player that GP can loot, and probably the two similar GP individual hinder themselves by trying to share the same strategy.

Table 5. Comparisons of success rates (SR) for one MCTS with 1500 playouts, against two GP player clones of Fig. 1.

# MCTS simulations	GP-1	GP-2	MCTS
1500	34.40% ± 1.32	35.34% ± 1.33	30.26% ± 1.27
3000	32.34% ± 1.30	31.82% ± 1.29	35.84% ± 1.33

These results show that GP can evolve successful players that can compete with MCTS, the current method of choice when evaluation functions are not easy to obtain. Notice that even if our GP individual plays remarkably against the other artificial opponents, it is not yet tough enough to deal with experienced human players. The absence

of information about opponent moves is a strong limitation that could be exploited by humans.

On the one hand MCTS keeps the advantage of being improved simply by increasing the number of simulations (although it may become too slow to be acceptable), while on the other hand the GP player is several order of magnitude faster but cannot be improved as easily.

6 Conclusion and future works

This study has shown that GP can evolve very competitive players for complex board games in the eurogame category, even from basic inputs. The main technical problem we encountered was the huge amount of computing time needed to obtain a significant fitness. This prevented us, at least for the moment, to obtain GP players trained against MCTS. We stress again that, on the opposite, once evolved, the resulting player is almost instantaneous, several orders or magnitude faster than MCTS.

Training only against the ruleAI, which is a weak player, nonetheless allowed GP to beat MCTS with 1500 playouts on average, and to compete with a 3000 playout MCTS in a mixed players context. This a significant result which was unexpected, especially as 3000 playouts already incurs a significant delay for MCTS.

Many tracks are opened by this study. Some are GP oriented such as co-evolving programs by assessing fitness against the other individuals of the population; or trying smarter terminals and expanding the function set to include e.g. "if" statements; or splitting the program in three subroutines, one for each deals of the game, in order to obtain an increased level of play by adjusting the evaluation function to the current phase of the game. Tuning the evolution parameters, and also the number of determinizations are natural extensions, and also testing our GP evaluation function in an expectimax of depth greater than one.

A more fundamental idea could be to try to bridge the gap between GP and MCTS, e.g. using GP as a surrogate estimation of a fraction of the playouts, in order to speed up MCTS, or using GP to build hyper-heuristics using MCTS sampling, GP evolved evaluation functions and even rule-AI as building bricks. We could also try to learn game patterns, following the ideas in [17]. At last, tackling other eurogames is also a future objective.

References

1. Michael Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. In Mehdi Q, Gough N, Natkin S, and Al-Dabass D, editors, *5th international conference on computer games: artificial intelligence, design and education*, pages 384–388, 2004.
2. István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-Carlo tree search in Settlers of Catan. In *Advances in Computer Games*, pages 21–32. Springer Berlin Heidelberg, 2010.
3. Ransom K. Winder. Methods for approximating value functions for the Dominion card game. *Evolutionary Intelligence*, 2013.
4. GMJB Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, JWHM Uiterwijk, and H Jaap Van Den Herik. Monte-Carlo strategies for computer Go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.

5. Ami Hauptman and Moshe Sipper. GP-endchess: Using genetic programming to evolve chess endgame players. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 120–131, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.
6. Moshe Sipper. Evolving game-playing strategies with genetic programming. *ERCIM News*, 64:28–29, January 2008. Invited article.
7. Claude E. Shannon. XXII. Programming a computer for playing chess. *Philosophical Magazine (Series 7)*, 41(314):256–275, 1950.
8. Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.
9. Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007.
10. Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.
11. Richard J Lorentz. Amazons discover Monte-Carlo. In *Computers and games*, pages 13–24. Springer, 2008.
12. Tristan Cazenave. Monte-Carlo Kakuro. In H. Jaap van den Herik and Pieter Spronck, editors, *ACG*, volume 6048 of *Lecture Notes in Computer Science*, pages 45–54. Springer, 2009.
13. Broderick Arneson, Ryan B Hayward, and Philip Henderson. Monte-Carlo tree search in Hex. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):251–258, 2010.
14. F. Teytaud and O. Teytaud. Creating an upper-confidence-tree program for Havannah. *Advances in Computer Games*, pages 65–74, 2010.
15. Denis Robilliard, Cyril Fonlupt, and Fabien Teytaud. Monte-carlo tree search for the game of 7 wonders. In T. Cazenave, M.H.M. Winands, and Y. Björnsson, editors, *Computer Games: Third Workshop on Computer Games, CGW 2014, Held in Conjunction with the 21st European Conference on Artificial Intelligence, ECAI 2014*, volume 504 of *Communications in Computer and Information Science*, pages 64–77. Springer International Publishing, 2014.
16. Daniel Whitehouse, Edward J Powley, and Peter I Cowling. Determinization and information set Monte-Carlo tree search for the card game Dou Di Zhu. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 87–94. IEEE, 2011.
17. Jean-Baptiste Hoock and Olivier Teytaud. Bandit-based genetic programming. In *Genetic Programming*, pages 268–277. Springer, 2010.