Development of a Program for Playing Progressive Chess

Vito Janko[1] and Matej Guid[2]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
[2] Faculty of Computer and Information Science, University of Ljubljana, Slovenia

Abstract. We present the design of a computer program for playing Progressive Chess. In this game, rather than just making one move per turn, players play progressively longer series of moves. Our program follows the generally recommended strategy for this game, which consists of three phases: looking for possibilities to checkmate the opponent, playing generally good moves when no checkmate can be found, and preventing checkmates from the opponent. In this paper, we focus on efficiently searching for checkmates, putting to test various heuristics for guiding the search. We also present the findings of self-play experiments between different versions of the program.

1 Introduction

Chess variants comprise a family of strategy board games that are related to, inspired by, or similar to the game of Chess. Progressive Chess is one of the most popular Chess variants [1]: probably hundreds of Progressive Chess tournaments have been held during the past fifty years [2], and several aspects of the game have been researched and documented [3–6]. In this game, rather than just making one move per turn, players play progressively longer series of moves. White starts with one move, Black plays two consecutive moves, White then plays three moves, and so on.

Rules for Chess apply, with the following exceptions (see more details in [2]):

– Players alternately make a sequence of moves of increasing number.
– A check can be given only on the last move of a turn.
– A player may not expose his own king to check at any time during his turn.
– The king in check must get out of check with the first move of the sequence.
– A player who has no legal move or who runs out of legal moves during his turn is stalemated and the game is drawn.
– En passant capture is admissible on the first move of a turn only.

There are two main varieties of Progressive Chess: Italian Progressive Chess and Scottish Progressive Chess. The former has been researched to a greater extent, and a large database of games (called 'PRBASE') has been assembled. In Italian Progressive Chess, a check may only be given on the last move of a *complete* series of moves. In particular, if the only way to escape a check is to give check

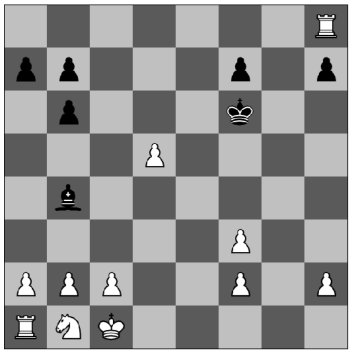


Fig. 1. Black to move checkmates in 8 (consecutive) moves.

on the first move of the series, then the game is lost by the player in check. In Scottish Progressive Chess, check may be given on any move of a series, but a check also ends the series. It has been shown that the difference very rarely affects the result of the game [7].

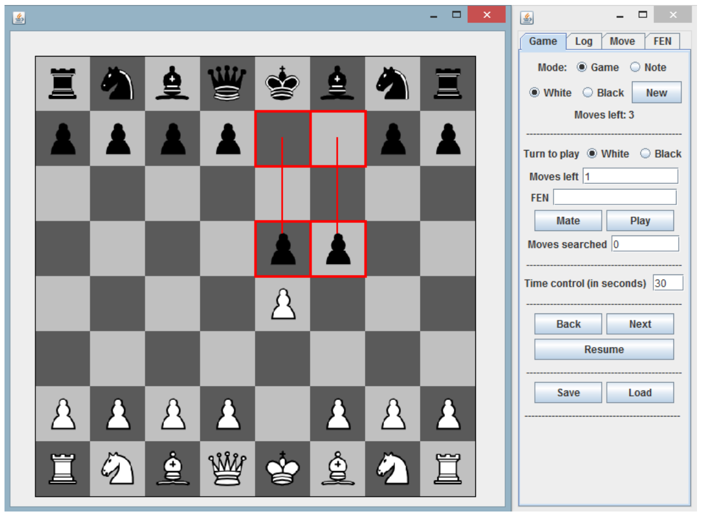
The strategy for both players can be summarized as follows. Firstly, look for a checkmate; if none can be found, ensure that the opponent cannot mate next turn. Secondly, aim to destroy the opponent's most dangerous pieces whilst maximizing the survival chances of your own [2]. Searching for checkmates efficiently – both for the player and for the opponent – is thus an essential, the single most important task in this game.

The diagram in Fig. 1 shows an example of a typical challenge in Progressive Chess: to find the sequence of moves that would result in checkmating the opponent. Black checkmates the opponent on the 8^{th} consecutive move (note that White King should not be in check before the last move in the sequence).

Our goal is to develop a strong computer program for playing Progressive Chess. We know of no past attempts to build Progressive Chess playing programs. In the 90's, a strong Progressive Chess player from Italy, Deumo Polacco, developed *Esaù*, a program for searching for checkmates in Progressive Chess. According to the program's distributor, AISE (Italian Association of Chess Variants), it was written in Borland Turbo-Basic, and it sometimes required several hours to find a checkmate. To the best of our knowledge, there are no documented reports about the author's approach, nor whether there were any attempts to extend *Esaù* to a complete Progressive Chess playing program.

From a game-theoretic perspective, Progressive Chess shares many properties with Chess. It is a finite, sequential, perfect information, deterministic, and zero-sum two-player game. The state-space complexity of a game (defined as the number of game states that can be reached through legal play) is comparable to that of Chess, which has been estimated to be around 10^{46} [8]. However, the per-turn branching factor is extremely large in Progressive Chess, due to the combinatorial possibilities produced by having several steps per turn.

In another chess variant, Arimaa, where "only" four steps per turn are allowed, the branching factor is estimated to be around 16,000 [9]. So far, human



Mode: [✓ Game] Note

[✓ White] Black

Moves left: 3

Turn to play [✓ White] Black

Moves left 1

FEN

Moves searched 0

Time control (in seconds) 30

Fig. 2. Our Progressive Chess playing program. Black's last turn moves are indicated.

players prevailed over computers in every annual "Arimaa Challenge" competition, and the high branching factor is considered as the main reason why Arimaa is difficult for computer engines [10]. We thus expect Progressive Chess to provide a challenging new domain in which to test new algorithms, ideas, and approaches.

The paper is organized as follows. In Section 2, we describe the design of our Progressive Chess playing program. In Section 3, we focus on the specific challenge of searching for checkmates. Experimental design and results of the experiments are presented in Sections 4 and 5. We then conclude the paper.[1]

2 Application Description

The graphical user interface of our Progressive Chess playing program is shown in Fig. 2. We implemented the Italian Progressive Chess rules (see Section 1 for details). The application provides the following functionalities:

– playing against the computer,
– searching for checkmates,
– watching the computer playing against itself,
– saving games,
– loading and watching saved games.

The user is also allowed to input an arbitrary (but legal) initial position, both for playing and for discovering sequences of moves that lead to a checkmate. The application and related material is available online [11].

[1] The solution to Fig. 1: Bb4-d6, b6-b5, b5-b4, b4-b3, b3xa2, a2xb1N, Nb1-c3, Bd6-f4.

2.1 Search framework

As indicated in the introduction, one of the greatest challenges for AI in this game is its combinatorial complexity. For example, on turn five (White to move has 5 consecutive moves at disposal) one can play on average around 10^7 different series of moves. Games usually end between turns 5-8, but may lengthen considerably as both players skill increases. Generating and evaluating all possible series for the side to move quickly becomes infeasible as the game progresses. Moreover, searching through all possible responses after each series is even less feasible, rendering conventional algorithms such as minimax or alpha-beta rather useless for successfully playing this game.

Generally speaking, our program is based on heuristic search. However, the search is mainly focused on sequences of moves for the side to move, and to a much lesser extent on considering possible responses by the opponent. In accordance with the aforementioned general strategy of the game, searching for the best series of moves consists of three phases:

Searching for checkmate In the first phase, the aim of the search is to discover whether there is a checkmate available. If one is found, the relevant series of moves is executed and the rest of the search is skipped. Checkmates occur rather often, thus finding them efficiently is crucial for successfully playing this game.

Searching for generally good moves Another search is performed, this time trying to maximally improve the position. Usually, the aim of this phase is to eliminate the opponent's most dangerous pieces, and to maximize the survival chances of own pieces. For example, giving check on the last move of a turn is considered a good tactic, as it effectively reduces the opponent's sequence of moves by one. The king should be given air (e.g., a king on the back rank is often at risk). Pawn promotions are also an important factor to consider. It is often possible to prevent inconvenient opponent's moves by placing the king so that they will give premature check etc. If the allocated time does not allow to search all available series of moves, only a subset of the most promising ones (according to the heuristics) is searched. The series are then ordered based on their heuristic evaluation.

Preventing checkmate The previous phase generates a number of sequences and their respective evaluation. It is infeasible to perform a search of all possible opponent replies for each sequence. However, it is advisable to verify whether we are getting mated in the following turn. The most promising sequence of moves is checked for opposing checkmate. In case it is not found, this sequence of moves is then executed. Otherwise the search proceeds with the next-best sequence, and the process then repeats until a safe move is found, or the time runs out. In the latter case, the best sequence according to the heuristic evaluation is chosen. In this phase, again a quick and reliable method for finding checkmates is required.

In Section 2.2, we describe the heuristics for finding generally good moves (see the description of the second phase above). In Section 3, we describe our approach to searching for checkmates, which is the main focus of this paper.

2.2 Position heuristics

As described in Section 2.1, heuristic evaluation of particular series of moves is used for finding generally good moves (when checkmate is not available), and taking into account the opponent's replies is often infeasible. Relatively complex heuristics are therefore required. Discovering good heuristics is challenging, as the game is relatively unexplored. Furthermore, a fair bit of what is known to be good in Chess does not apply for Progressive Chess. Defending pieces is an obvious example: while it may be a good idea in orthodox chess, it is often completely useless in Progressive Chess.

We hereby briefly describe the most important heuristics that our program uses for finding sensible sequences of moves (when checkmate is not available):

Material count The Shannon value of pieces (Queen = 9, Rook = 5 etc.) hardly apply in Progressive Chess. Bishops are better than Knights in the early stages. In the ending, however, Knights are much better than Bishops because of their ability to reach any square. Pawns are much more dangerous than in the original game, since their promotions often cannot be prevented. Finally, queens are extremely dangerous, because of their huge potential for delivering checkmates. Additional experiments are still required to determine a suitable relative value of the pieces.

King safety Kings tend to be safe in the open air, preferably not at the edge of the board. Given the nature of the game it is usually trivial to checkmate a king that is enclosed with its own pieces, so the usual pawn defences or castling are discouraged. Practice showed that king is safest on the second rank; away from opponent pieces, but still safe from back rank mates.

Pawn placements Pawns can promote in five moves from their starting position. Stopping them becomes essential as game progresses. It can be done by blockading them with pieces, placing pawns in such formation that opposing pawns cannot legally bypass them, or using the King to prevent promotions due to a premature check. Positions where there is no legal promotion from the opponent side are rated higher. It is also favorable to advance pawns, bringing them closer to the promotion square.

Development Development is a risky proposition, since pieces in the center are more easily captured, and they can often be brought into action from their initial positions rather quickly. Nevertheless, pieces with higher mobility are positively rewarded.

Opening book The opening book is upgrading based on results of previous games. The statistics are then used as a part of heuristic evaluation.

Search extensions For leaf nodes at the end of the turn it is possible to simulate some opponent replies. Searching only a limited amount of moves may not give an accurate representation of opponent's best reply, but it gives a general idea. For example, it prevents spending five moves for promoting a pawn that could be taken right on the next move by the opponent.

3 Searching for checkmate

Searching for checkmates efficiently is the main focus of this paper. In this section, we explore various attempts to achieve this goal. It can be considered as a single agent search problem, where the goal is to find a checkmate in a given position. Alternative problem setting would be to find *all* checkmates in the position, or to conclude that one does not exist, without exploring all possibilities.

The A* algorithm was used for this task. We considered various heuristics for guiding the search. In the experiments, we observed the performance of two different versions of the algorithm (see Section 3.1), and of five different heuristics (see Section 3.2). Experimental design is described in Section 4.

3.1 Algorithm

The task of finding checkmates in Italian Progressive Chess has a particular property – all solutions of a particular problem (position) lie at a fixed depth. Check and checkmate can only be delivered on the last move of the player's turn, so any existing checkmate must be at the depth equal to the turn number. A* uses the distance of the node as an additional term added to the heuristic evaluation, guiding the search towards shorter paths. In positions with a high turn number (where a longer sequence of moves is required) this may not be preferred, as traversing longer variations first is likely to be more promising (as they are the only ones with a solution). One possibility to resolve this problem is to remove the distance term completely, degrading the algorithm into best-first search. An alternative is to weight the distance term according to the known length of the solution. Weight $a/length$ was used for this purpose, where the constant a was set arbitrarily for each individual heuristic. In all versions we acknowledged the symmetry of different move orders and treated them accordingly. In the experiments, we used both versions of the algorithm: *best-first search* and *weighted A**.

3.2 Heuristics

For the purpose of guiding the search towards checkmate positions, we tried an array of different heuristics with different complexities, aiming to find the best trade-off between the speed of evaluation and the reliability of the guidance. This corresponds to the well known search-knowledge tradeoff in game-playing programs [12]. All the heuristics reward maximal value to the checkmate positions. It is particularly important to observe that in such positions, all the squares in the immediate proximity of the opponent's King must be covered, including the King's square itself. This observation served as the basis for the design of the heuristics. They are listed in Table 1.

Fig. 3 gives the values of each heuristic from Table 1 for the pieces in the diagram. In *Manhattan*, pawns are not taken into account, resulting in the value of 13 (2+5+6). The value of *Covering* is 3, as the squares a1, a2, b1 are not covered. The *Ghost* heuristic obtains the value of 7 (2+1+2+2): the Rook needs

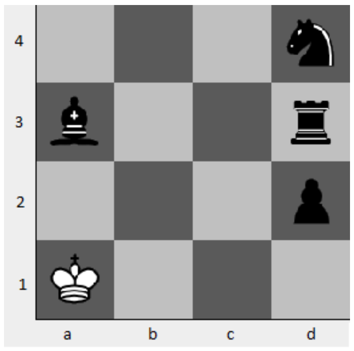


Fig. 3. The values of heuristics listed in Table 1 for this position are as follows. Manhattan: 13, Ghost: 7, Covering: 3, Squares: 8.

Table 1. Heuristics for guiding the search for checkmates.

Name	Description
Baseline	Depth-first search without using any heuristic values.
Manhattan	The sum of Manhattan distances between pieces and the opponent's King.
Ghost	The number of legal moves pieces required to reach the square around the king, if they were moving like "ghosts" (ignoring all the obstacles).
Covering	The number of squares around the king yet to be covered.
Squares	The sum of the number of moves that are required for each individual piece to reach every single square around the king.

two moves to reach the square immediate to the king, the Bishop needs one, the Knight needs two moves (to reach a1), the Pawn needs one move to promote and one move with the (new) Queen. The value of *Squares* heuristic is 8 (2+3+2+1): the square a1 can be reached in two moves with the Knight, a2 can be reached in three moves with the Knight or Rook, b1 can be reached in two moves with the Rook, b2 can be reached in one move with the Bishop. The player's King is never taken into account in the calculations of heuristic values.

Aside from covering squares around the opponent's King, there are two more useful heuristics that can be combined with the existing ones; we named them *Promotion* and *Pin* (see Table 2).

A majority of checkmates that occur later in the game include promoting one of the Pawns, getting an extra attacker for delivering checkmate. Rewarding promotions of the Pawns is therefore beneficial.

Another useful heuristic takes advantage of a "self-pin." Fig. 4 shows the controversial "Italian mate," which is enthusiastically championed by some but is felt by others to be undesirably artificial [7]. It occurs where the only way to escape a check is to give a check in return, making that move illegal. The position on the left diagram is from a game Boniface – Archer (played in the year 1993), where White played 7 c4, Kd2, Kc3, Kb4, Nf3, Rd1, Rxd7. The final position

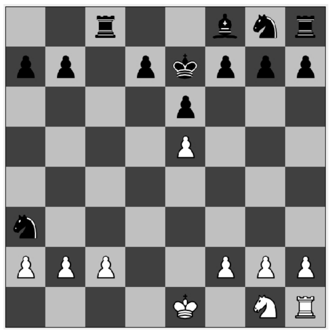
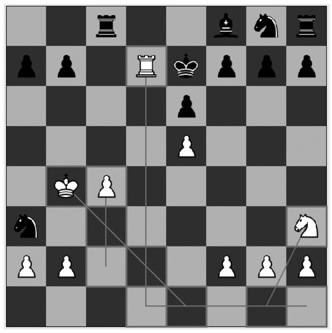



Fig. 4. Left: White to move checkmates in 7 moves. Right: the final position.

(diagram on the right) is checkmate according to Italian rules. Our program found an alternative solution (albeit with the same idea), putting the Knight on h3. The moves played were indicated by the program. The solution shows the idea of exploiting the self-pin, moving the King to an appropriate square.

4 Experimental design

The goal of the experiments was to verify empirically how promising is our approach for finding checkmates in an efficient manner. In particular, (1) which of the two search algorithms performs better (*best-first search* or *weighted A**), (2) which is the most promising heuristic to guide the search (*Manhattan*, *Ghost*, *Covering*, or *Squares*), and (3) what is the contribution of the two additional two heuristics (*Promotion* and *Pin*; see Table 2).

Another research question was who has the advantage in Progressive Chess: White or Black (note that this is not so clear as in orthodox chess). *The Classified Encyclopedia of Chess Variants* claims that masters have disagreed on this question, but practice would indicate that White has a definite edge [2].

4.1 Experiment

Two sets of experiments were conducted. Firstly, we observed how quickly do different versions of the program find checkmates on a chosen data set of positions with different solution lengths (see Section 4.2). Both average times and success rates within various time constraints were measured. The search was limited to 60 seconds per position (for each version of the program).

Table 2. Additional heuristics that can be combined with the existing ones.

Name	Description
Promotion	How far are Pawns to the square of promotion, also rewards extra queens.
Pin	How far is the King to the closest square where self-pin could be exploited.

Secondly, self-play experiments were performed between the programs with the same algorithm (weighted A* with the two additional heuristics) and various heuristics. The programs played each other in a round-robin fashion. The winning rates were observed for each version of the program, and both for Black and White pieces. In the second phase of the game (see Section 2.1), a small random factor influenced the search so the games could be as diverse as possible. Four different, increasingly longer time settings were used in order to verify whether different time constrains affect the performance.

4.2 The checkmates data set

We collected 900 checkmates from real simulated games between programs. In each turn in the range from 4 to 12, there were 100 different checkmates included. The shortest checkmates in Progressive Chess can be made on turn 3, however, they are few and rather trivial. Longer games are rare, and even then there are usually very few pieces left on the board, making the checkmate either trivial or impossible. The above distribution allowed us to observe how the length of the solution affects the search performance.

5 Results

5.1 Average times for finding checkmates

Fig. 5 gives the average times for finding checkmates with the best-first search algorithm. It roughly outlines the difficulty of the task: finding checkmates is easier when the solution is short (turns 4–6), more difficult when the solutions are of medium length (turns 7–10), and easier again in the later stage (turns 11–12), as the material on the board dwindles.

It is interesting to observe that the baseline heuristic (i.e., depth-first search) even outperforms some other heuristics at turns 4–6 and 11–12, i.e., when the

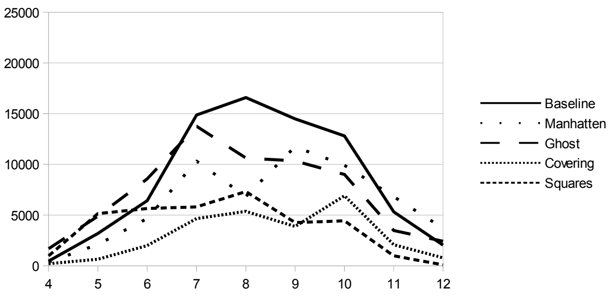


Fig. 5. Average times (in milliseconds) with the best-first search algorithm. The horizontal axis represents the length of the solution.

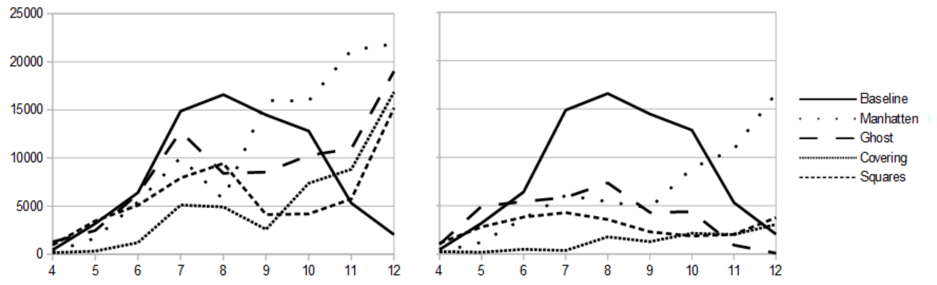


Fig. 6. On the left are average times (in milliseconds) with the A* algorithm, on the right are average times (in miliseconds) with the improved A* algorithm.

solution is less difficult (note that the problems at higher turns typically contain less material on the chessboard). The Covering heuristic performs best up most of the time (up to turn 9), and the Squares heuristic performs best at the later stages.

The average times with the weighted A* algorithm are given in Fig. 6. They are slightly shorter than the ones obtained by the best-first search algorithm up to turn 9. However, the average time increases greatly at the later stages. The main reason is that the heuristics tend to fail to find the solutions in later stages (note that each failed attempt is "penalized" with 60,000 milliseconds, i.e., the time limit for each problem).

However, the performance of the A* algorithm improves dramatically when it also uses the two additional heuristics: Promotion and Pin (see Fig. 6). In particular, the Promotion heuristic turns out to be very useful at the later stages in the game.

Overall, the A* algorithm with the two additional heuristics performed best, and Covering heuristic turned out to be the most promising one. In particular, since most of the games in Progressive Chess finish before turn 10.

5.2 Success rates

Fig. 7 demonstrates how many checkmates were found at any given point of time (in seconds). The Covering heuristic performed clearly best at every cutoff point. It found 80% (722 out of 900) checkmates in less then a second, and 99% (891) checkmates within the time limit of 60 seconds. The improved A* algorithm (using the two additional heuristics) was used in this experiment.

5.3 Self-play experiments

The results of the self-play experiments are given in Fig. 8, showing the number of wins for each heuristic. The Covering heuristic clearly outperformed all the other heuristics, and each heuristic performed better with black pieces.

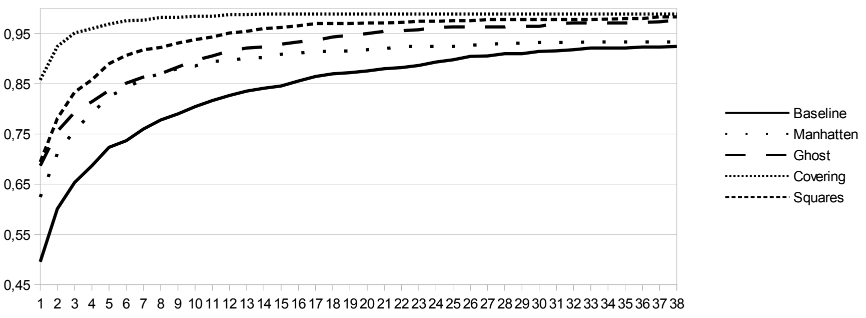


Fig. 7. Percentage of checkmates found in less then x seconds.

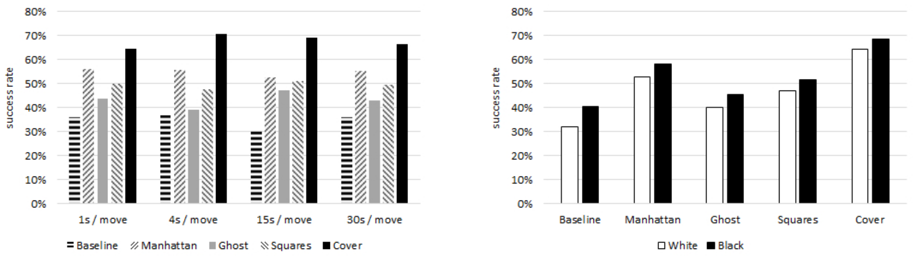


Fig. 8. The success rate for each program (left) and for each piece color (right).

There was the total of 31,260 games played, and each program played the same number of games. 20,340 games were played at the time control of 1s/move, 6,900 at 4s/move, 2540 at 15s/move, and 1,480 at 30s/move. The average length of the games was 8.3 turns ($\sigma = 2.8$). The success rate of white pieces against black pieces was 47.2% vs. 52.8%, which suggests that it is Black that has a slight advantage in Progressive Chess. Only 13.7% of the games ended in a draw.

6 Conclusions

The aim of our research is to build a strong computer program for playing and learning Progressive Chess. This chess variant was particularly popular among Italian players in the last two decades of the previous century [2]. By developing a strong computer program, we hope to revive the interest in this game both among human players, who may obtain a strong playing partner and an analysis tool, as well as among computer scientists. In particular, the extremely large branching factor due to the combinatorial explosion of possibilities produced by having several moves per turn makes Progressive Chess both an interesting game and a very challenging environment for testing new algorithms and ideas.

Our program follows the generally recommended strategy for this game, which consists of three phases: (1) looking for possibilities to checkmate the opponent, (2) playing generally good moves when no checkmate can be found, and (3) preventing checkmates from the opponent. In this paper, we focused on efficiently searching for checkmates, which is considered as the most important task in this game. We introduced various heuristics for guiding the search. The A* algorithm proved to be suitable for the task. In the experiments with (automatically obtained) checkmate-in-N-moves problems, the program found the solutions very quickly: 80% within the first second, and 99% within one minute of search on regular hardware.

A self-play experiment (more than 30,000 games played) between various versions of the program lead to the success rate of 47.2% vs. 52.8% in favor of Black. Notably, each version of the program performed better with black pieces.

Our program requires significant further work to achieve the level of the best human players. Particularly in the second phase of the game (which is not directly associated with searching for checkmates) we see a lot of room for improvements, possibly by introducing Monte-Carlo tree search techniques [13]. The question of who has the advantage in Progressive Chess is therefore still open and could be the subject of further investigation.

References

1. Pritchard, D.: Popular chess variants. BT Batsford Limited (2000)
2. Pritchard, D.B., Beasley, J.D.: The classified encyclopedia of chess variants. J. Beasley (2007)
3. Leoncini, M., Magari, R.: Manuale di Scacchi Eterodossi. Siena-Italy (1980)
4. Dipilato, G., Leoncini, M.: Fondamenti di Scacchi Progressivi. Macerata-Italy (1987)
5. Castelli, A.: Scacchi Progressivi. Matti Eccellenti (Progressive Chess. Excellent Checkmates). Macerata-Italy (1996)
6. Castelli, A.: Scacchi progressivi. Finali di partita (Progressive Chess. Endgames). Macerata-Italy (1997)
7. Beasley, J.: Progressive chess: How often does the "italian rule" make a difference? `http://www.jsbeasley.co.uk/vchess/italian_rule.pdf` (2011) [accessed 06-March-2015].
8. Chinchalkar, S.: An upper bound for the number of reachable positions. ICCA Journal **19**(3) (1996) 181–183
9. Wu, D.J.: Move Ranking and Evaluation in the Game of Arimaa. PhD thesis, Harvard University, Cambridge, USA (2011)
10. Kozelek, T.: Methods of MCTS and the game arimaa. Master's thesis, Charles University, Prague, Czech Republic (2010)
11. Janko, V., Guid, M. `http://www.ailab.si/progressive-chess/`
12. Junghanns, A., Schaeffer, J.: Search versus knowledge in game-playing programs revisited. In: 15th International Joint Conference on Artificial Intelligence, Proceedings. Volume 1., Morgan Kaufmann (1999) 692–697
13. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers. (2006) 72–83