Politechnika Wrocławska

Wydział Elektroniki

Metaheuristic Chess Artificial Intelligence

Maciej Borkowski
195968@student.pwr.edu.pl

Paweł Pałus
197004@student.pwr.edu.pl

Mariusz Waszczyński
192660@student.pwr.edu.pl

Abstract

something something something something something something something something something something something something something something something something something something something

Index Terms: *chess, metaheuristics, artificial intelligence, ant colony, genetic, simulated annealing*

1 Introduction

Hello

2 Optimization Problem

The problem describes a standard game of chess, with a square board of 64 fields. Two players have to consecutively move a piece the board onto another field according to complex, well-defined rules. Our task is to find the series of movements in a game of chess that gives the best chance of winning the game in the end. The starting position of pieces can be arbitrary.

2.1 Mathematical model

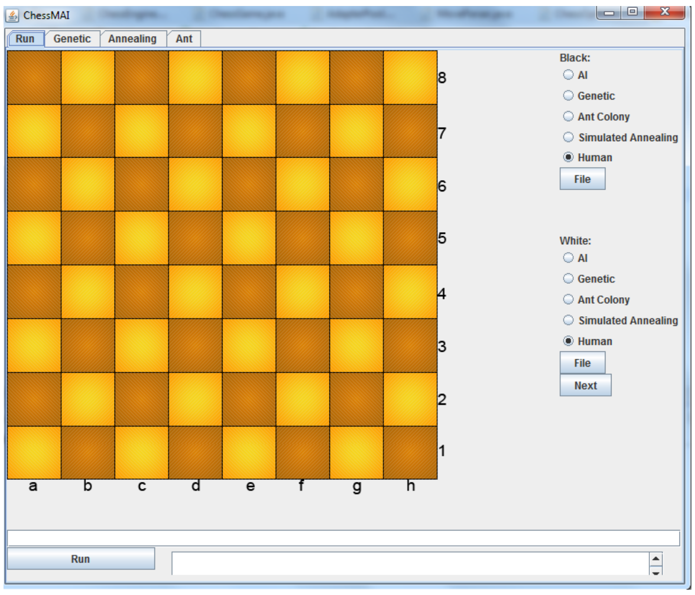
Given:

- $t \in N$ - initial game turn
- $n \in N$ - maximum number of game turns intended by for playing
- $i, j \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ - number of rows and columns
- $B_{8,8,t}$ - fields on the board taking values $v_{i,j,t} \in R$ describing the field on board(e.g. the chessmen standing on field) during turn t
- f($B_{8,8,t}, v_{i,j,t}$) - takes negatives values for bad situations on the field $v_{i,j,t}$ (e.g. an opponent's piece) positive for good situations (e.g. player's piece)

To find:
a series of movements starting from $B_{8,8,t}$ and ending in $B_{8,8,t+n}$ according to rules of chess
such that
$\sum_{i,j} f(B_{8,8,t+n}, v_{i,j,t+n})$ is maximised

3 Experimentation system

Experimentation system will by discussed with images.



Black: AI, Genetic, Ant Colony, Simulated Annealing, [✓ Human]

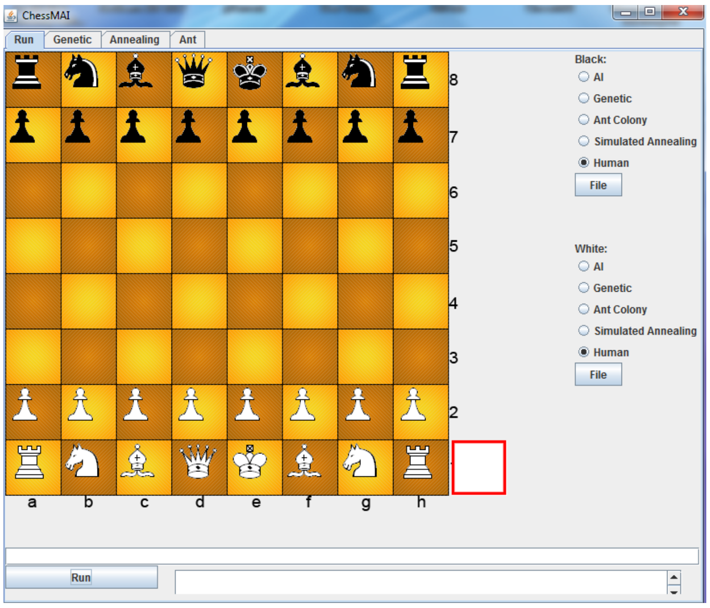
White: AI, Genetic, Ant Colony, Simulated Annealing, [✓ Human]

Figure 1: Initial Run panel

Figure 1 shows run panel that is displayed after starting the application. On the right side of the application are options on white and black player. Checking proper radio button one can choose what player will play a game as white player and also as black player. The choices are: AI(firenzina engine), human, or one of 3 metaheuristics. There are also file buttons that are used to input knowledge for ant search algorithm (proper file button for proper white or black player), and one next button that is used to make movements one after another from a textbox that is located in annealing tab. It will be further discussed in 4th section. the Next button is not displayed in another figures.

Figure 2 shows the run panel after pressing the run button. Initial locations of chessmen are set according to chess rules. In the button right corner of the board the square taking black or white depending on whose player is expected to make a move appeared.

the application allows possibility to start game from other



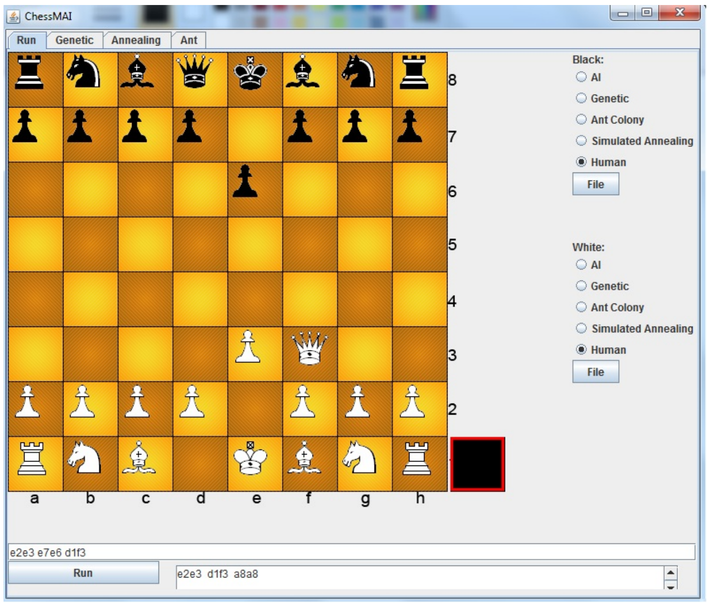
Black:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

White:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

Figure 2: Initial Run panel after pressing run button



Black:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

White:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

```
e2e3 e7e6 d1f3
```

```
e2e3 d1f3 a8a8
```

Figure 3: Initial board after certain sequence of movements

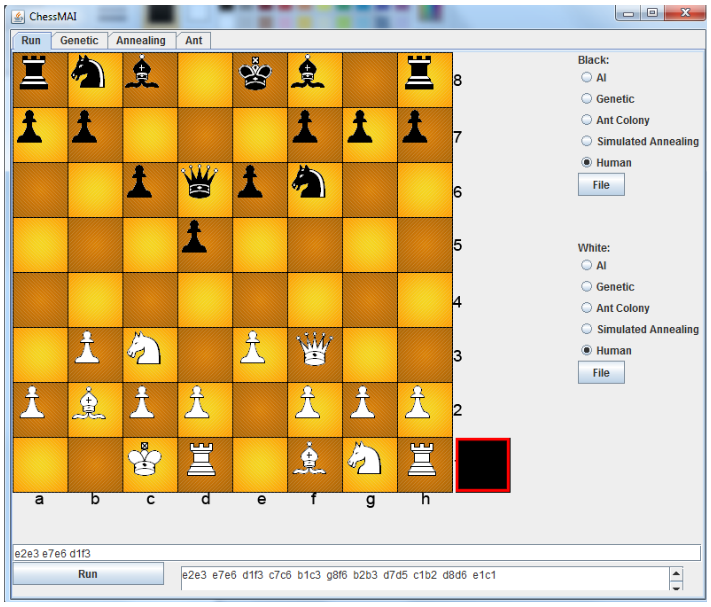
than standard initial locations of chessmen what is depicted in figure 3. For this purpose sequence of movements should be typed (or pasted) to the textbox located above the run button.

On the right side from the run button one can see the textbox responsible for showing the history of the movements as depicted in figure 4. This is full history of movements including movements that was made using the textbox above the run button.

At the top of the application are tabs for metaheuristics and the run panel.

3.1 UCI

UCI (Universal Chess Interface) - an open communication protocol that enables a chess program's engine to communicate with its user interface



Black:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

White:
- AI
- Genetic
- Ant Colony
- Simulated Annealing
- [✓ Human]

File

```
e2e3 e7e6 d1f3
```

```
e2e3 e7e6 d1f3 c7c6 b1c3 g8f6 b2b3 d7d5 c1b2 d8d6 e1c1
```

Figure 4: History of movements

3.1.1 Firenzina

3.2 GUI

About the GUI

4 Algorithms

4.1 Ant colony (Maciej Borkowski)

4.1.1 Idea

For ant colony search algorithm a mapping is created between a placement of chess pieces on a chessboard and a list of possible moves the current player is able to do, when provided such board. Each move on this list is additionally annotated with a real value, which describes the fitness of the move. Moves with higher fitness ought to yield us better results. Such mapping is called a pheromone and a set of them pheromones (Figure 5).

Ant is defined here as a chess player, that uses pheromones to choose a move when it is metaheuristic's time to make a choice of movement. Ant can be a part of a colony, in which case the colony provides the pheromones or it can be independent (used for Greedy Mode). Pheromones can be saved to and loaded from a file. When an ant finds itself on a board that has not yet been added to pheromones a new pheromone is created (possible moves for the board are computed and assigned equal real values).

Ant can work in one of two modes:

- Adventurous Mode
  Used for learning. In this case the ant works for the betterment of its colony. It chooses moves randomly, according to the values of pheromones. This strategy improves the pheromones, by visiting a wide range of possible boards, which results in frequent updates and addition of new pheromones. The probability of choosing move M (with real value v):

$$P(M) = \frac{v_m + |min(V)|}{\sum V + n|(min(V))|} \tag{1}$$

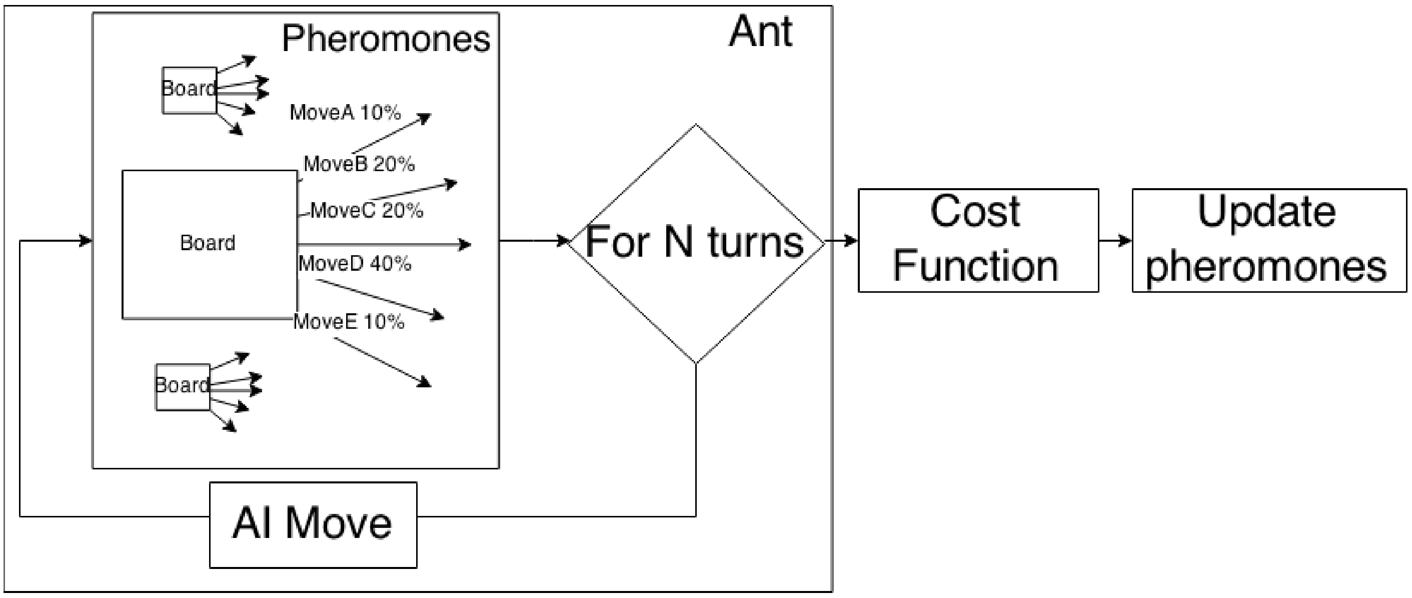


Figure 5: Diagram of the implemented ant colony search algorithm with visualisation of pheromones for one iteration with one ant

where V are all values in a given pheromone, v_m the value for move and n is the length of V.

- Greedy Mode
  Used for testing and real games. In this case the ant plays for the best end result in its game. Ant chooses a move from the pheromone with the highest value to choose the best move in each turn.

The process of learning consists of many iterations of ants in Adventurous Mode working as a colony. Each iteration amounts to a few phases:

1. Start new games and wait for them to end
   Each ant plays one game of chess and remembers all boards it has run across, all moves it has chosen to do and the the cost function of the series of movements (value of cost function for last board).
2. Update pheromones
   For each ant the pheromones connected to visited boards are updated by a fraction of the value of cost function of the whole series of movements.
   $$v_{new} = v_{old} + \frac{i}{m} cost \tag{2}$$
   where v_{new} is the new value, v_{old} is the old value, i is the index of the movement in this series of movements, m is the length of the series of movements and *cost* ist the value of cost function.
3. Dissipate pheromones
   Pheromone for each of the boards that has been visited at least once by any ant in this game is decreased by multiplying the value by a parameter.
   $$v_{new} = v_{old} * (1 - dissipation) \tag{3}$$
   where v_{new} is the new value, v_{old} is the old value and *dissipation* is a parameter.

Pheromones can be saved to a file. The file consists of a list of pheromones, each is described with two lines:

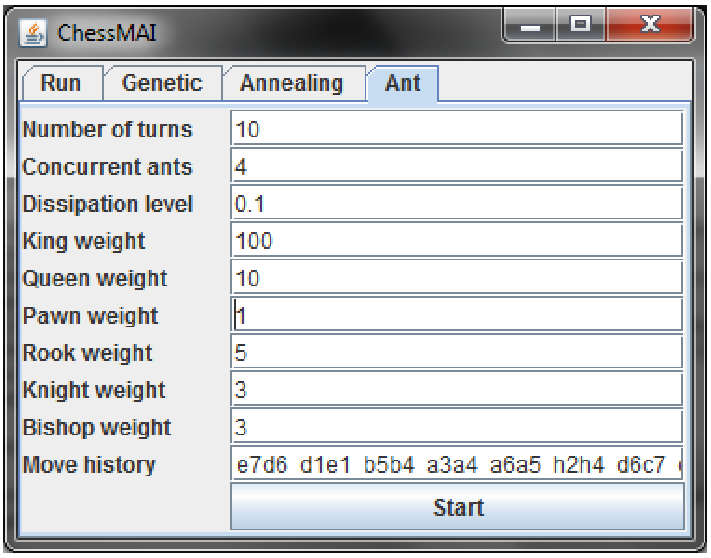
1. String representation of a board, left to right, bottom to up, where # means no chessman, upper case letters mean white chessmen and lower case letters mean black chessmen
2. A list of moves. Each move is described by five integer values. First two are the coordinates of chessman to move, third and fourth where to move the chessman to, the fifth is a special value used for promotion (when a pawn becomes another chess piece) and the sixth a real value of pheromone describing its effectiveness.

4.1.2 Experiments

Firstly, it has to be accentuated that chess is a complex game, the number of possible boards that have to be remembered in pheromones is huge and for each ant move another move has to be done by external Artificial Intelligence. Because of these reasons the learning phase of this metaheuristic takes a very long time. To get results appearing significantly different from completely random ones, hours have to be spent on learning. This makes it very difficult and time-consuming to experiment with parameters properly.

For the experiments the parametrization of many ant search specific variables has been put into dialog boxes in the application(Figure 6). This way user can change these values easily and create his own colonies. The user-available parameters are as follows:

- Number of turns
  The number of turns for each iteration. Each iteration consists of an all concurrent ants playing one game of chance up to a win, lose, draw or the artificial end of game, when it takes too long. This parameter should be kept low if we want the learning phase to take less time and if we want ants to have more broad "knowledge" of possible moves in the beggining of the game.



Number of turns: 10
Concurrent ants: 4
Dissipation level: 0.1
King weight: 100
Queen weight: 10
Pawn weight: 1
Rook weight: 5
Knight weight: 3
Bishop weight: 3
Move history: e7d6 d1e1 b5b4 a3a4 a6a5 h2h4 d6c7

Figure 6: Ant colony dialog boxes

- Concurrent ants
  The number of ants playing a game in each iteration. The more of them the longer each iteration takes.

- Dissipation level
  The dissipation parameter of ant search algorithm dissipation equation (Equation 3).

- Piece weight
  Each piece has its own weight used in cost function (Equation ??)

- Move history
  This parameter sets the starting position of the game from any point, provided a valid chess move history. Each ant in each iteration starts its game from this point and plays up until the end of the game (including the end of maximum number of turns defined as another parameter)

The more obvious rules had to be applied to get to the point of metaheuristic being better than a random algorithm and able to win one game out of hundreds when playing against the artificial intelligence. Most importantly the weight of king should far bigger than other figures, the number of turns small, and a move history provided that gives a possibility of winning in a few turns. Increasing the number of concurrent ants and, at the same time, the dissipation level makes the results possibly even better but by a very small margin at the cost of a longer learning phase (making it difficult to test).

4.1.3 Result

Meddling in all of these parameters proved to be insufficient in obtaining better results. After careful debugging of the application, the conclusion has been made, that the possible reason for improvement could be to increase the probability of choosing good moves instead of dwell on the bad moves. Even though the move choosing equation (Equation 1) gives more probability to good moves, it is not very significant to the sum of all the probabilities, because there are generally a lot of bad moves. One of the ideas was to sum up the logarithms of values in equation, which is often a way of dealing with those kind of issues. This however only made the values of pheromones more close to each other so the probability of choosing a good move didn't really change that much.

The really important change was observed when the Equation 1 was modified to one with a "tolerance" factor multiplied by each value except the best one. This way the probability of choosing the best move in training is big, enabling the colony to thoroughly establish how good of a situation on the board it results in. This can quite easily culminate in a fall into a local maximum of cost function, but it is not a bad situation to be in chess - at least we end up in a better situation than before. Additionally to really fall into a local maximum the moves have to contain a lot of weight gain, so it would be most probably a checkmate anyway, which is, for all intents and purposes, a global maximum. Indeed, if a winning sequence of moves has been established in this version the colony started winning very often.

4.2 Genetic algorithm (Mariusz Waszczyński)

4.2.1 Idea

Our second proposal of metaheuristic algortihms is genetic algortihm. It is pure related to a human nature evoluates from the ages. Looking for humans, it gives interesting results. The population consist with many individuals - games. The individual is a set of gens which here is called moves. From programmicaly point of view a mapping is created between a placement of chess pieces on a chessboard and a move the current player, when provided such board. Each individual is additionally annotated with a real value, which describes the fitness of the final move.

In application each metaheuristic algorithm is defined as a player. Even Artificial Inteligence is marked as player. Hence different players can play against each other. Genetic player is divided into two modes (as ant colony algorithm has):

- Adventurous Mode
  Used for learning. In this case the individual is created. The moves are chosen randomly.

- Greedy Mode
  Used for testing and real games. Based on global individual choose appropriate moves. If the individual has not current a placement of chess pieces on a chessboard, move is picked randomly.

Entire proccess is divided into two phases: generating a population and learning. Genereting population phase creates a new thread for playing individuals, waits for them to end and updates their fitness. When the first phase has been finished the learning process which consist of many interations is runned.

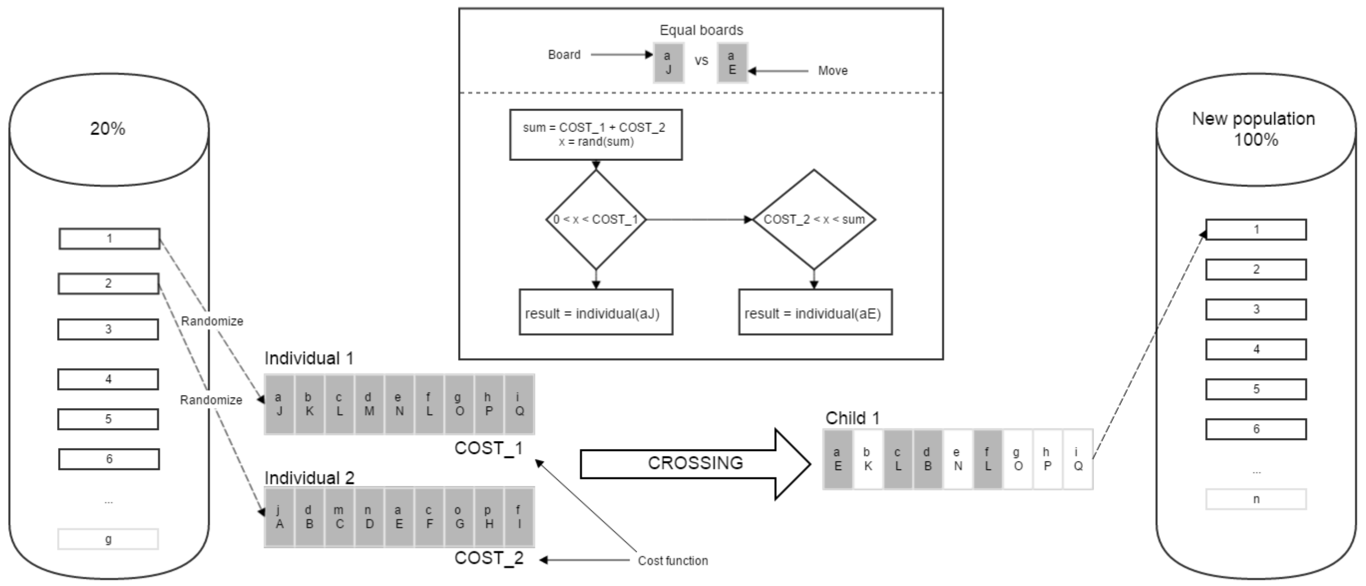


Figure 7: Genetic: crossing phase

The interation has follwing steps:

1. Selection
   The general assume is 80 percent of population have wrong gens and are going to remove (Figure 8).

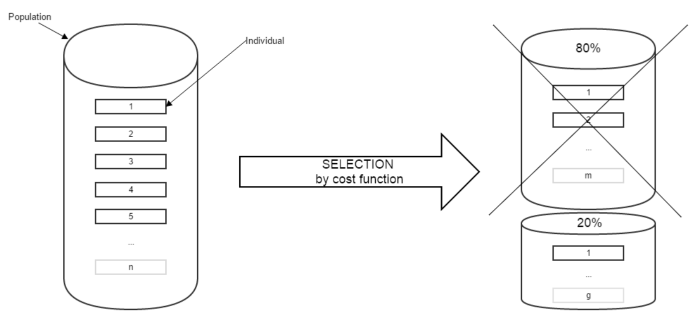


Figure 8: Genetic: selection phase

2. Choosing the global individual
   From the remaining population the individual which has the best cost function is compared with global individual. If it is higher, then global individual is overwritten (Figure 9).

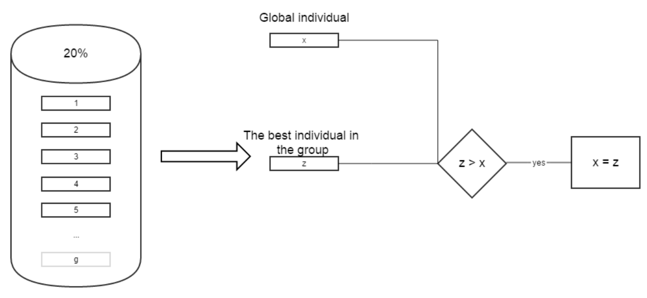


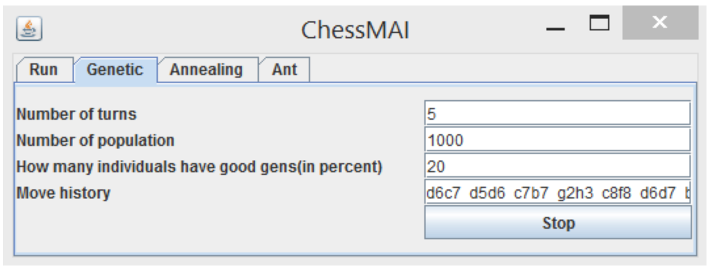
Figure 9: Genetic: comparing with the global individual

3. Crossing phase
   Two individuals are chosen randomly. Every board[1] from first individual is comparing with every board from second individual. If the boards are equal the proper is chosen and added to the descendant. To choose optimal one probability with cost function are used. Cost function from first and second individual is summed. The sum is a boundary for randomize the number. If the randomized value is from zero to the value of first individual cost function then first individual is chosen, otherwise second individual. Nevertheless If board has not the equivalent, is added to the descendant. Process is showed on Figure 7. This step is repeating up to fill population, in order to avoid losings.

[1]a placement of chess pieces on a chessboard

When user decide to stop learning, global individual is stored to file which can be used in normal game - greedy mode. Optimal time for 5 moves for learning is couple hours.

4.2.2 Experiment



Number of turns	5
Number of population	1000
How many individuals have good gens(in percent)	20
Move history	d6c7 d5d6 c7b7 g2h3 c8f8 d6d7 b

Figure 10: Genetic tab in application

In order to play with genetic algorithm, firstly the optimal global individual should be finded. The genetic tab is designed for learning purposes. It contains four text fields.

Number of turns
Determine how many moves the individual have to do. Low value should be used to confine time for learning.

Number of population
Determine size of population.

How many individuals have good gens
: Needed for selection phase. Defined 20 percent by default.

Move history
: Learning proccess can start from specified board.

Button "start" implies generating population and turning learning proccess on. Optimal time for learning depends on parameters provided by user. Significant parameters are "Number of turns" and "How many individuals have good gens". First one can minimalize time of learning to achive certain point of knowledge. Second one has direct impact of removing individuals and is associated with "Number of population" parameter.

4.2.3 Result

Unfortunetelly checkmate was not achived. For almost 2 tousand tries Artificial Inteligence was winning. Should be emphasize that important is initial board from which games is starting. If cost function is very high at the beginning then is the most probability that genetic algorithm will win. Of course significant impact has manner how cost function is calculated. What factors are taking into consideration, what vages has pieces.

4.3 Simulated Annealing (Paweł Pałus)

4.3.1 Idea

Simulated annealing is a method for finding a good (not necessarily optimal) solution to an optimization problem. There are many optimization algorithms, including ant search, genetic algorithms, tabu search, and more. Simulated annealing's advantage is that it avoids getting caught at local minimums solutions that are better than any others nearby, but are not the best.

in the first game the Algorithm chooses random possible moves, all moves performed by AI and the algorithm are saved. In the end of the game cost function is calculated and saved. Then the algorithm simulate another game (iteration) in which chooses a saved moves, or a random possible moves and it depends on temperature. If the algorithm choosed random move, opponent move is performed by AI, but if saved move was choosed then opponent move is also a saved move. Not always saved move can be performed therefore in case of one cannot make a saved move, a random move is performed. If there is checkmate or maximum number of turns is achived game ends and cost function is calculated if this cost function is greater than previous one then old saved moves and cost function are exchanged for present. There can occour a situation in which game ends before achieving maximum number of turns (for example if there is a checkmate) and its cost function is greater than previous one. Then in the next simulated game the algorithm will have less saved moves for example 4 and when this number is exceeded random moves are performed in metaheuristic's turns and AI moves in opponent turns. The cost function base only on weights of chess pieces. Weights for chess pieces are as follows:

pawn - 1
knight - 5
bishop - 5
rook - 7
queen - 10
king - 1000

Weights are positive for own pieces and negetive for opponent pieces. It is important to assign a weight to the king instead of giving hight (in absolute value) value for checkmate, because then it is possible to consider better and worse situations for games that has no max number of turns. without that the algorithm would have all the time the same sequence of saved moves expect very unlikly situation that metaheuristics wins, then the sequence would change only once.

4.3.2 GUI

GUI is depicted in figure 12).

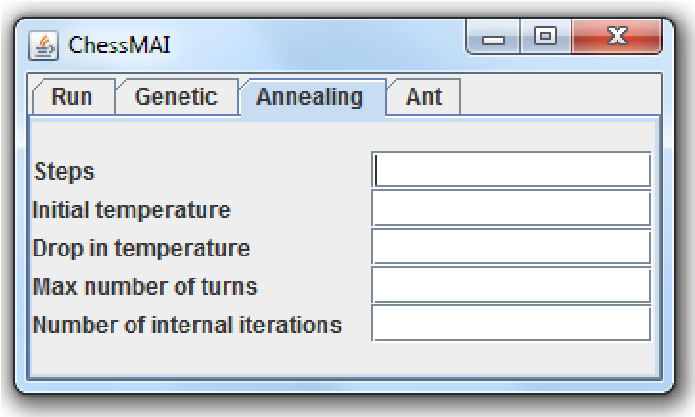


Figure 12: Simulated annealing dialog boxes

- Steps
  Here is a place to input sequence of moves, after that one can display all or a part of a game in a way that every another move is displayed after pressing a button
- Initial temperature
  A decimal number in a range from 0 to 1.0. The greater the temperature the greater the chance of choosing a random move. Temperature decreases after every iteration of algorithm.
- Drop in temperature
  A value by which the temperature drops after each iteration.
- Max number of turns
  An integer number that says how many turns at most can be performed in one game. A turn is defined as two moves, but to simplify number of turns increases after every black player move.
- number of internal iterations
  An integer number that says how many times the algorithm should repeat an action of searching better solution.

4.3.3 eqperiments

It is hard to tell what parameters should be to abtain the best results due to big randomness of the algorithm work. As good parameters have been choosed:

- Initial temperature = 0.8

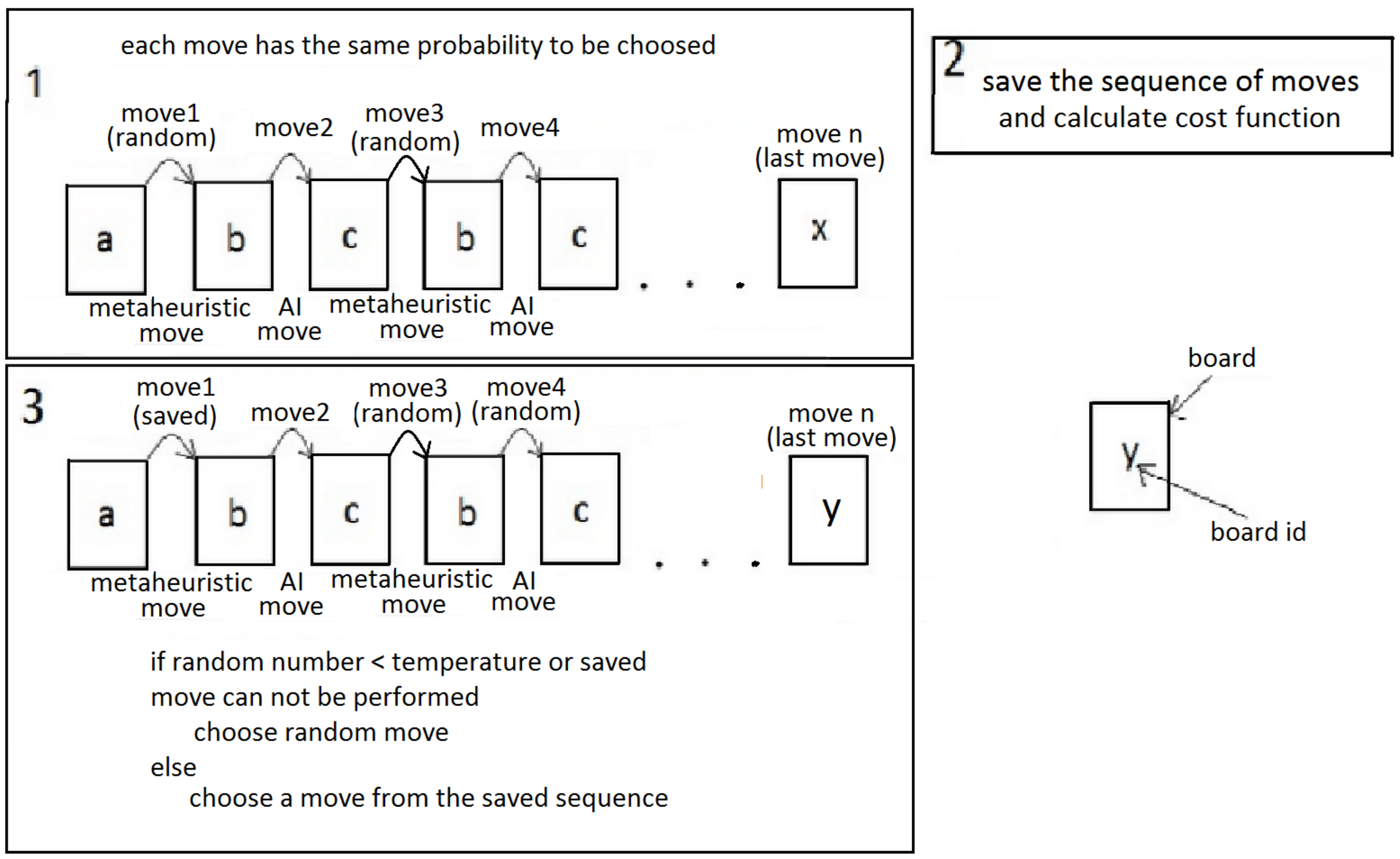


Figure 11: Diagram of the implemented simululated annealing algorithm

- Drop in temperature $= \frac{Initial\ temperature}{number\ of\ internal\ iterations} \cdot 0.9$

  in this case the temperature for the last iteration equals 10% of the initial temperature. that means 7.2%

Of course the greater number of the "number of internal iterations" parameter the greater the chance to achieve better solution. Experiments was made in a range from 10 to 20000 for "number of internal iterations" parameter parameter. Experiments was made most often for value of 10 for "Max number of turns" parameter. There were considered 2 kinds of initial board, standard initial situation for chess and situation depicted in figure 5.

For above board metaheuristic player is white and there is actually its turn.

4.3.4 result

Unfortunetelly checkmate was not achived. The best result for standard initial chess board was cost function: 21.0 for parameters:
- internal iteration number: 10000
- max turn number: 10
- initial temperature: 0.8
- drop in temperature: $\frac{0.8}{10000} \cdot 0.9 = 0.000072$
in time: 1485 seconds

and for initial situation depicted in figure 13 the best result was cost function equals 31.0 for parameters:
- internal iteration number: 20000
- max turn number: 10
- initial temperature: 0.8
- drop in temperature: $\frac{0.8}{20000} \cdot 0.9 = 0.000031$
in time: 3734 sekund

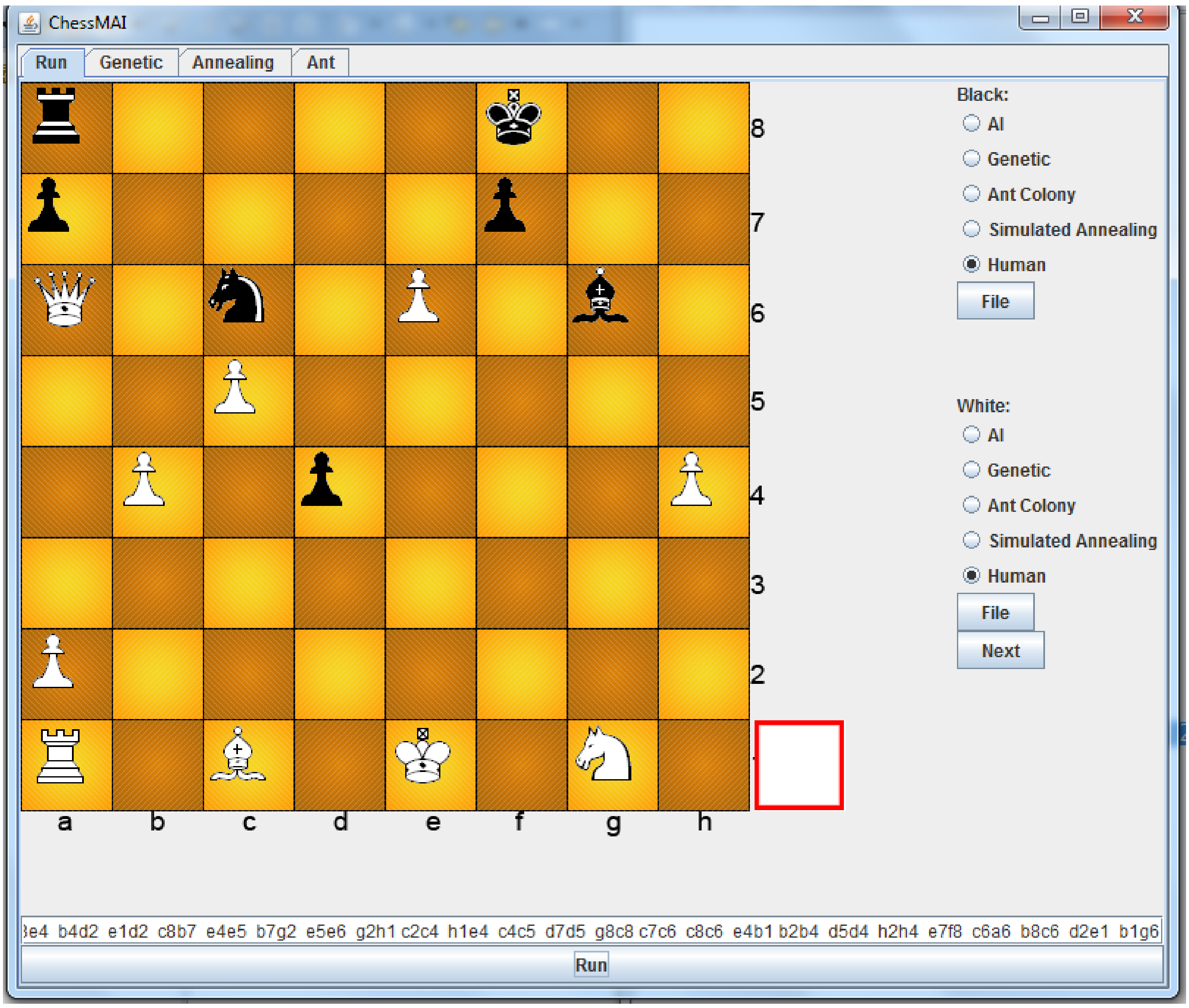
Different wariants of the algorithm could be considered, the algorithm could simultaneously play for example 5 different games instead of one and at the end of all games choose moves (that would exchange previous saved moves) from the game that achieved the greatest cost function (as far as this cost function would be greater than previous one), or the algorithm could save only its moves and every opponent move would be performed by AI.

5 Improvements

- put all the variables/modes etc in GUI - adding new engines and randomize their use between moves - importing files for further learning

6 Conclusion

It is rare to achive a checkmate by the algorithm due to several precised moves has to be performed to obtain one-time increasing of the cost function. The algorithm would work better if the cost function better expressed a situation on the board. Actually only a material situation is considered, there is a possibility to consider a positional situation, for example the weight of a pawn would increase after each move towards promotion field or the weight of a rook would increase with the number of possible moves that can be performed by the rook, it could be also considered if certain chess piece is protected or not.



Black: AI, Genetic, Ant Colony, Simulated Annealing, [✓ Human]

White: AI, Genetic, Ant Colony, Simulated Annealing, [✓ Human]

```
3e4  b4d2  e1d2  c8b7  e4e5  b7g2  e5e6  g2h1 c2c4  h1e4  c4c5  d7d5  g8c8 c7c6  c8c6  e4b1 b2b4  d5d4  h2h4  e7f8  c6a6  b8c6  d2e1  b1g6
```

Figure 13: initial board

References

[1] Vecek, N. ; Crepinsek, M. ; Mernik, M. ; Hrncic, D., A comparison between different chess rating systems for ranking evolutionary algorithms

[2] Dorigo, M. ; Maniezzo, V. ; Colorni, A., Ant system: optimization by a colony of cooperating agents

[3] David, O.E. ; van den Herik, H.J. ; Koppel, M. ; Netanyahu, N.S. , Genetic Algorithms for Evolving Computer Chess Programs

[4] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi., Optimization by Simulated Annealing

[5] R. Huber, S. Meyer-Kahlen, Universal Chess Interface, http://www.shredderchess.com/chess-info/features/uci-universal-chess-interface.html, 2015/06/01