Politechnika Wrocławska
Wydział Elektroniki

Metaheuristic Chess Artificial Intelligence

Maciej Borkowski
195968@student.pwr.edu.pl

Paweł Pałus
197004@student.pwr.edu.pl

Mariusz Waszczyński
xxxxxx@student.pwr.edu.pl

Abstract

something 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

About the application

3.1 UCI

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 1).

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 choos-

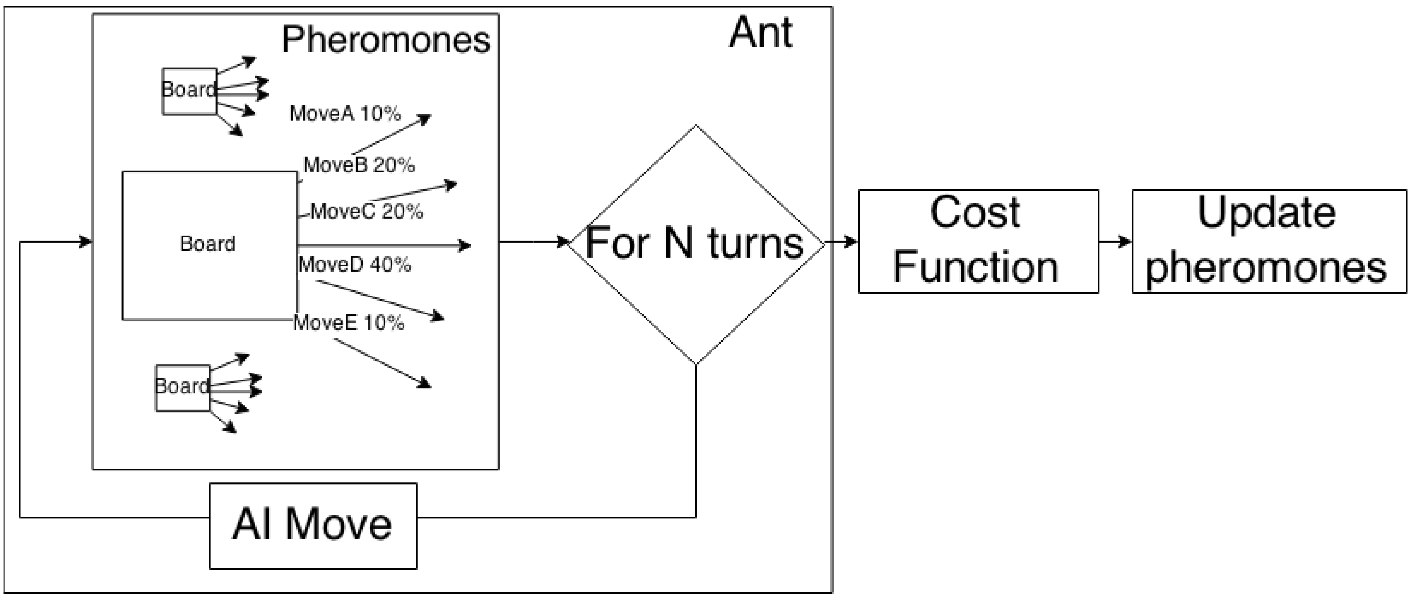


Figure 1: Diagram of the implemented ant colony search algorithm with visualisation of pheromones for one iteration with one ant

ing move M (with real value v):

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

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.

   $$v_{new} = v_{old} + \frac{i}{m} cost \quad (2)$$

   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.

   $$v_{new} = v_{old} * (1 - dissipation) \quad (3)$$

   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:

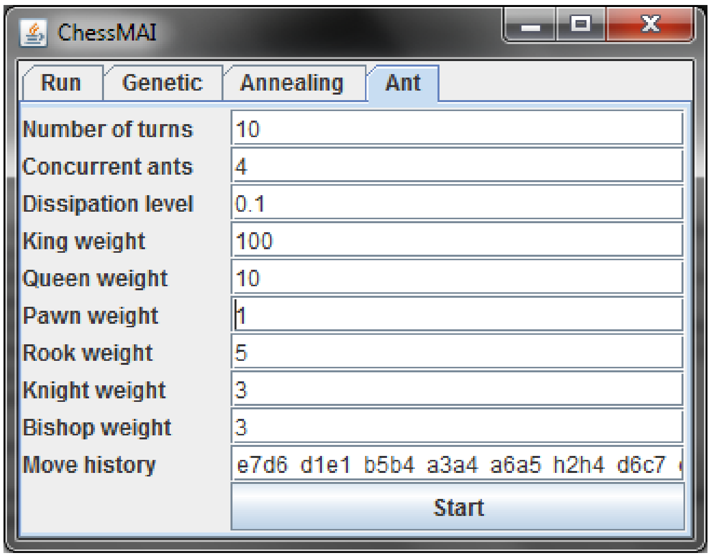
1. String representation of a board, left to right, bottom to up, where # means no chessman, upper case letters mean white chessmen and lower case letters mean black chessmen
2. A list of moves. Each move is described by five integer values. First two are the coordinates of chessman to move, third and 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 2). 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



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 2: Ant colony dialog boxes

time and if we want ants to have more broad "knowledge" of possible moves in the beggining of the game.

- 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

4.2.1 what?

4.2.2 gui/experiment

4.2.3 result

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

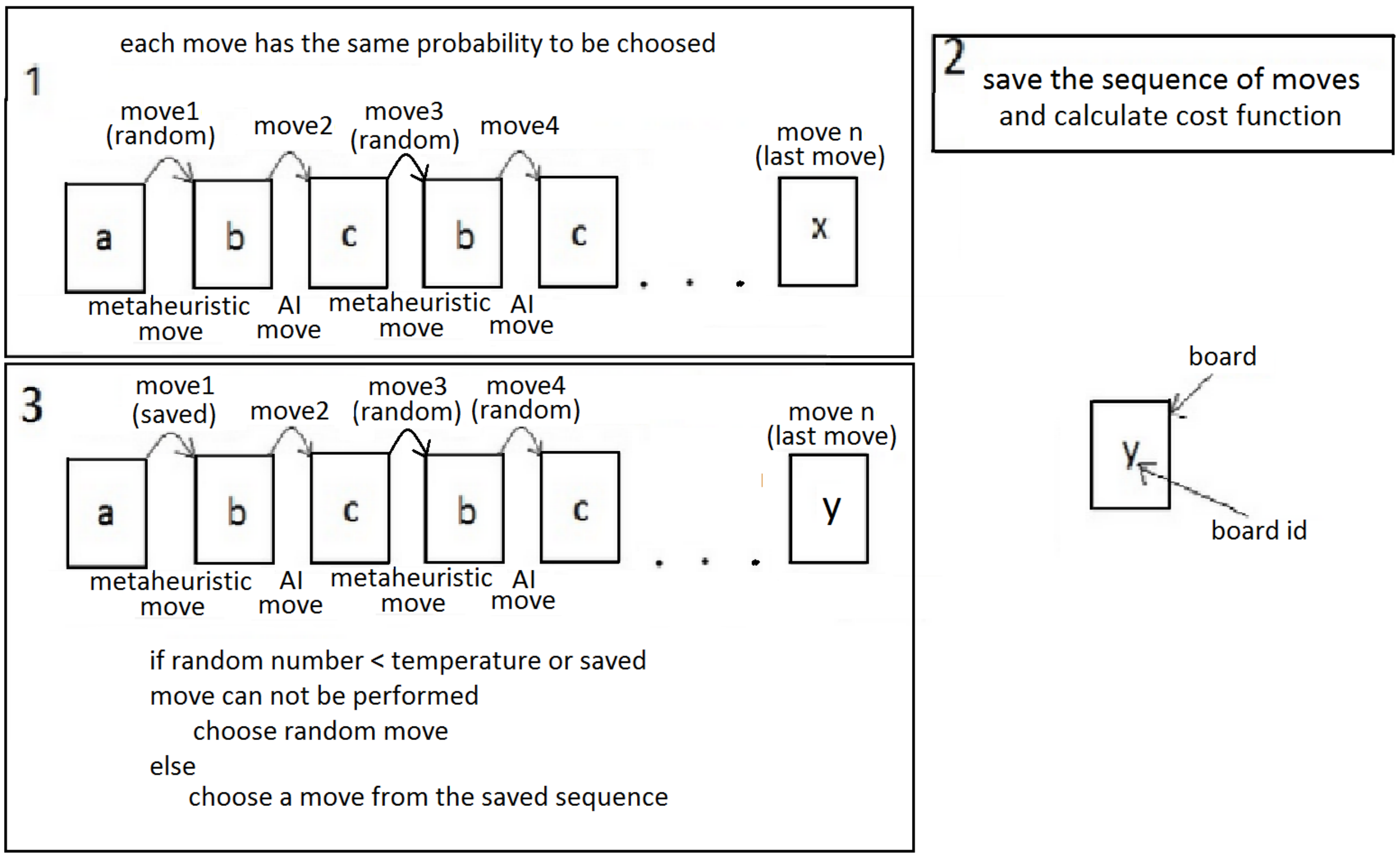


Figure 3: Diagram of the implemented simululated annealing algorithm

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 4.

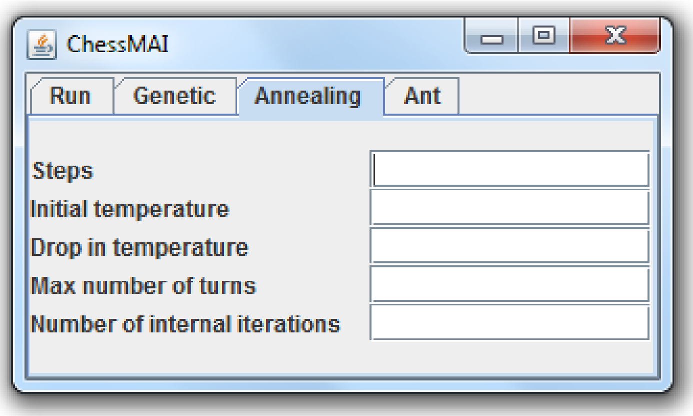


Figure 4: Simulated annealing dialog boxes

- 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

- 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 5 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

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. 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.

References

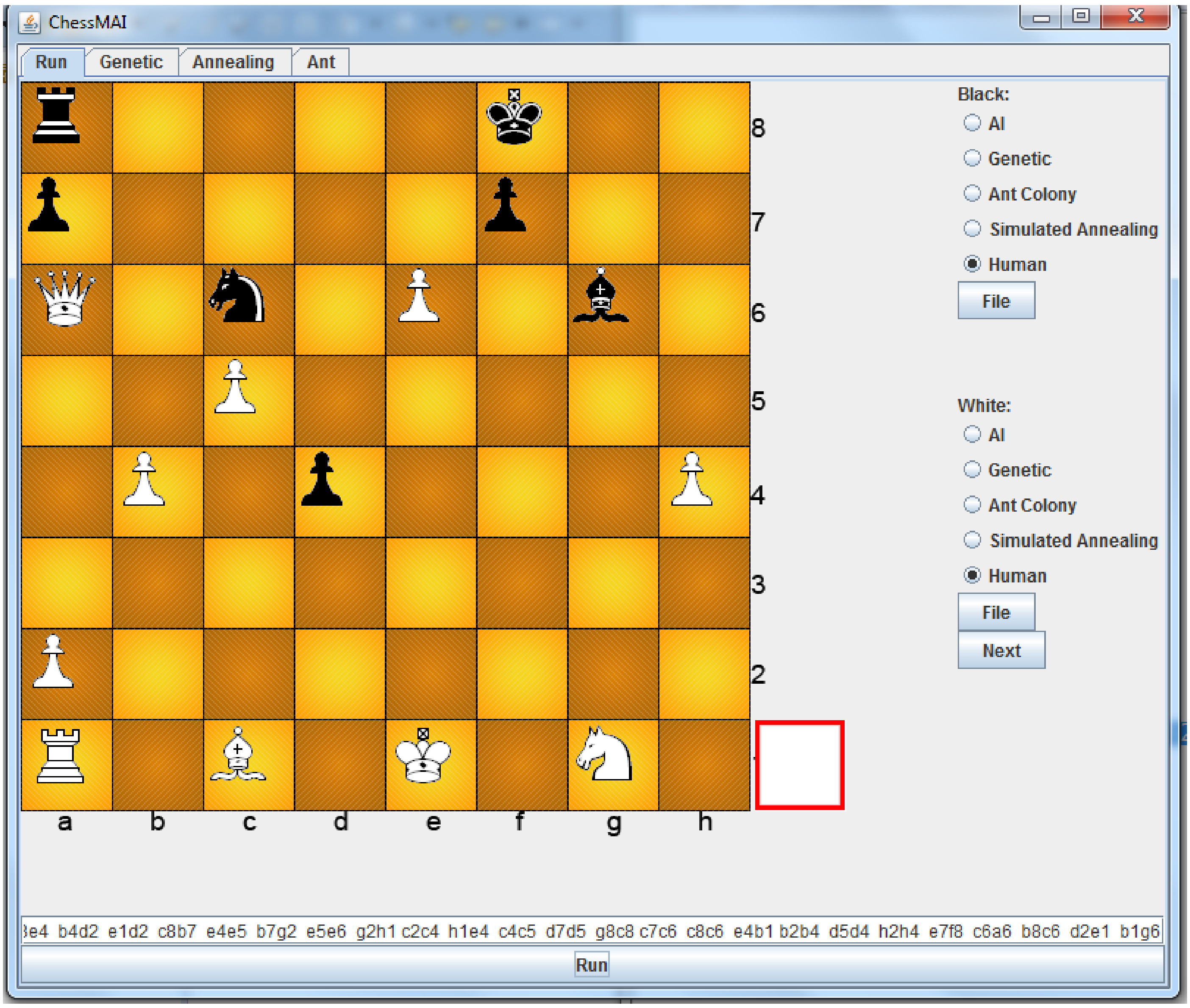
[1] Vecek, N. ; Crepinsek, M. ; Mernik, M. ; Hrncic, D., A comparison between different chess rating systems for ranking evolutionary algorithms

[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



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 5: initial board