Complex Systems Seminar
Can complex systems help design better chess-playing programs?

Olivier Mehani

February, 1 2006

Some figures about chess

- less than 20 rules
- 2×16 pieces
- 64 squares

Some figures about chess

- less than 20 rules
- 2×16 pieces
- 64 squares
- a branching factor[1] of 35
- 10^{120} legal board positions

[1] average number of possible moves *each* turn

Current implementations are deterministic

Efficient chess programs are based on this algorithm.

1. build then explore a tree of possible moves
2. rank each move
3. choose the best one

Current implementations are deterministic

Efficient chess programs are based on this algorithm.

1. build then explore a tree of possible moves
2. rank each move
3. choose the best one

Drawback: combinatorial explosion (tree size, number of positions,. . .)

Can complex systems bring better solutions ?

Neural networks or Genetic algorithms may be helpful at

- improving the deterministic algorithm (new exploration heuristic, new ranking method...)
- learning to be a better player
- replacing (maybe) the whole algorithm

Can complex systems bring better solutions ?

Neural networks or Genetic algorithms may be helpful at

- improving the deterministic algorithm (new exploration heuristic, new ranking method. . .)
- learning to be a better player
- replacing (maybe) the whole algorithm

What about these solutions ?

Outline

The minimax algorithm

The base of chess playing programs

The minimax algorithm

An algorithm good at games

More than one application in games

- full information games (tic-tac-toe)
- board-based games (chess, checkers,. . .)

Efficient enough to be chosen for all decent computer chess players

The minimax algorithm

The internals of the perfect minimax

- tree based algorithm
- assumes *perfect play*
- explores the *whole* tree

The minimax always finds the best solution...

The minimax algorithm

The internals of the perfect minimax

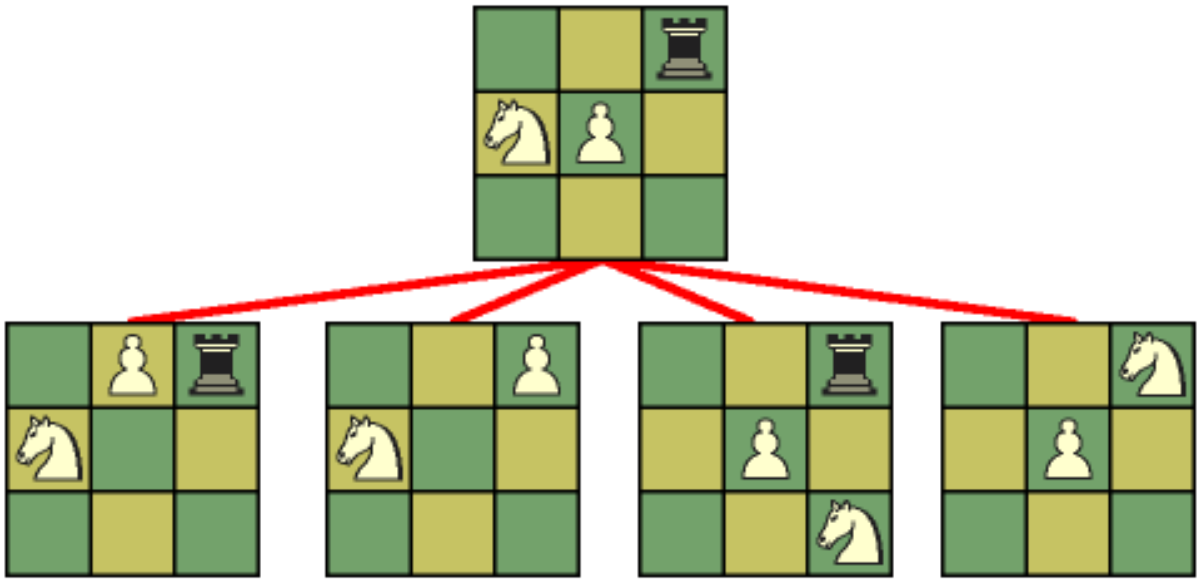
- tree based algorithm
- assumes *perfect play*
- explores the *whole* tree

The minimax always finds the best solution...
...assuming infinite time and memory.

The minimax algorithm

Tree building

From a given position, minimax generates all the legal moves



then works recursively, *depth first* until it finds every leaf.

The minimax algorithm

Evaluation of the leaves

A score is assigned to each leaf, depending on the outcome

win	highest score
draw	average score
loss	lowest score

The minimax algorithm

"Back propagation" of the evaluation

- The potential games are played backward.
- At every level, the move with the maximum score is chosen.
- Maximum score for the opponent means minimum score for the playing side.

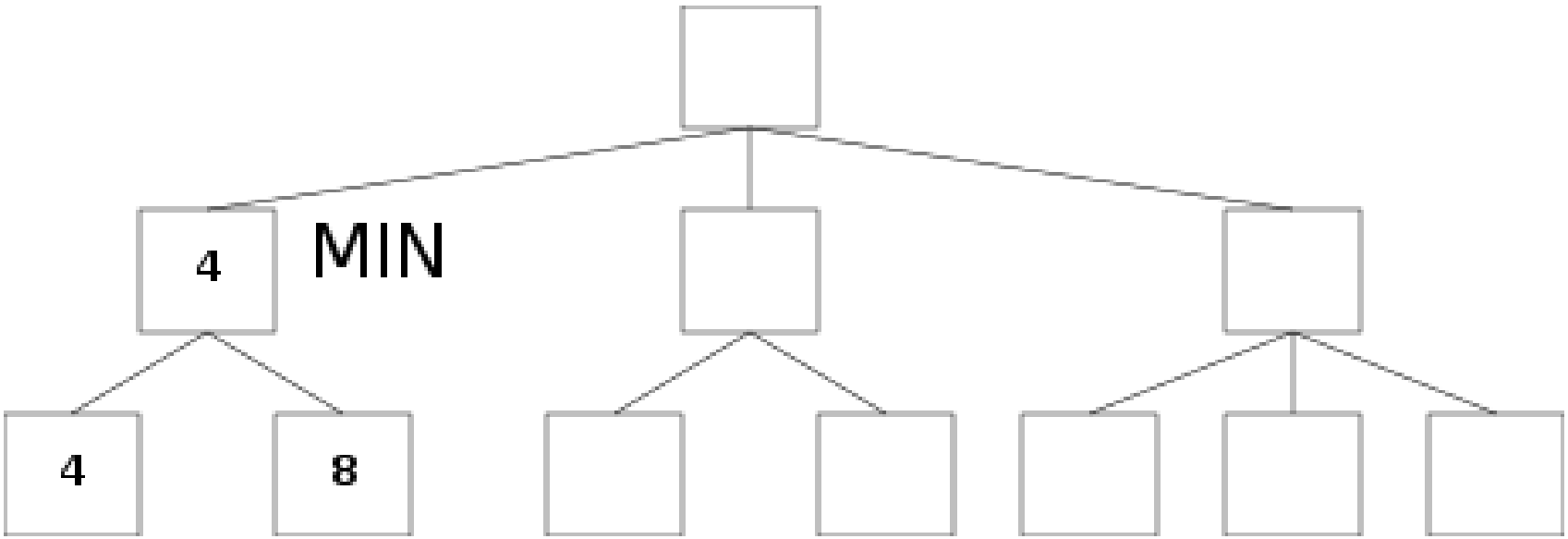
The minimax algorithm

"Back propagation" of the evaluation

- The potential games are played backward.
- At every level, the move with the maximum score is chosen.
- Maximum score for the opponent means minimum score for the playing side. $\Rightarrow$ min-max

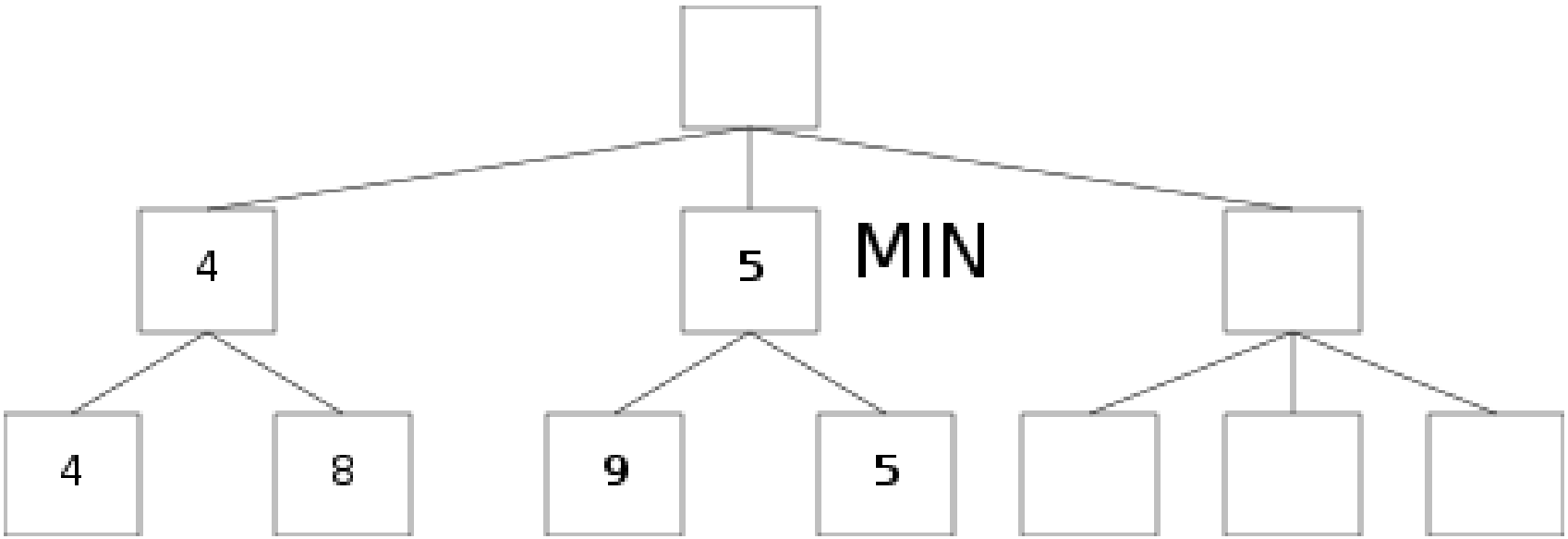
The minimax algorithm

An example of the scoring method



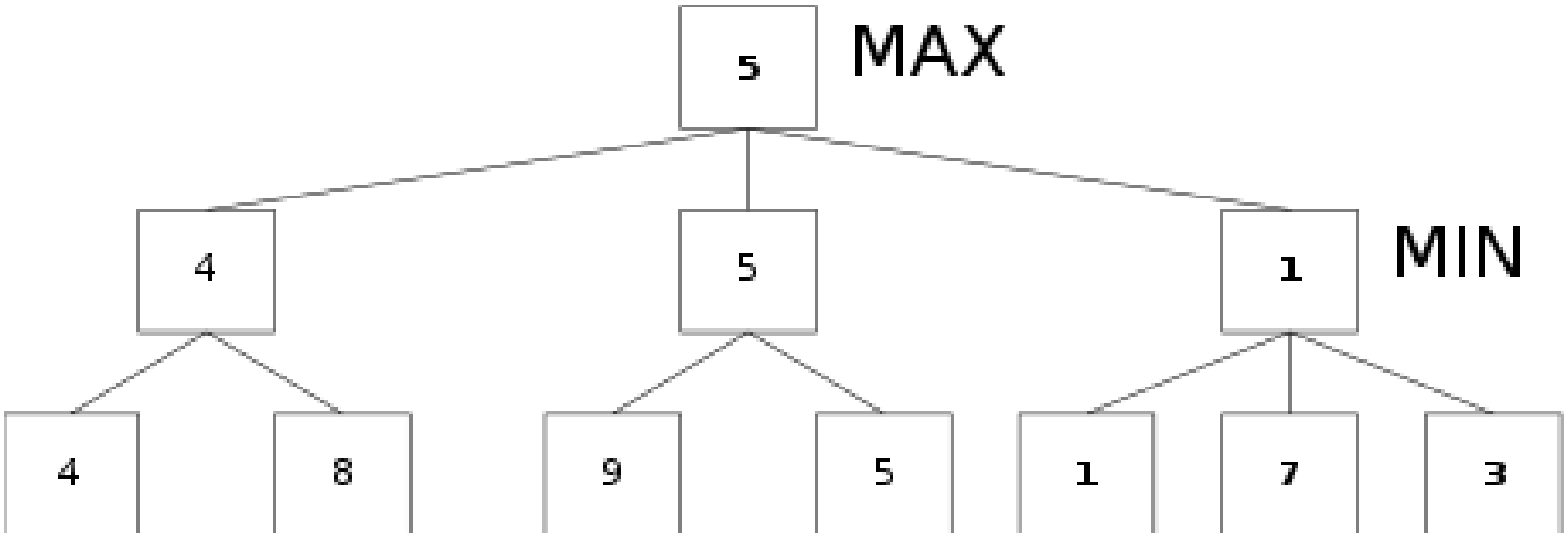
The minimax algorithm

An example of the scoring method



The minimax algorithm

An example of the scoring method



Classical approaches

Deterministic implementations of the minimax algorithm

The minimax algorithm

Classical approaches

- Heuristic scoring
- Alpha/Beta cut-off
- Plies database

Complex systems based approaches

Heuristic scoring

How to avoid exploring the tree down to the leaves ?

- Problem: too resource-consuming to explore the entire tree

Heuristic scoring

How to avoid exploring the tree down to the leaves ?

- Problem: too resource-consuming to explore the entire tree
- Solution: "guess" the value of a board configuration

Heuristic scoring

How to avoid exploring the tree down to the leaves ?

- Problem: too resource-consuming to explore the entire tree
- Solution: "guess" the value of a board configuration
- Several properties of the position are studied and weighted in order to score the board

Heuristic scoring

Important properties of the boards

- material balance
- number of possibles moves
- board control
- development of the pieces
- pawns formation
- king safety
- ...

Heuristic scoring
Drawbacks

- computationally difficult to extract the properties
- what weights should be attributed ?
- *horizon effect*

Alpha/Beta cut-off

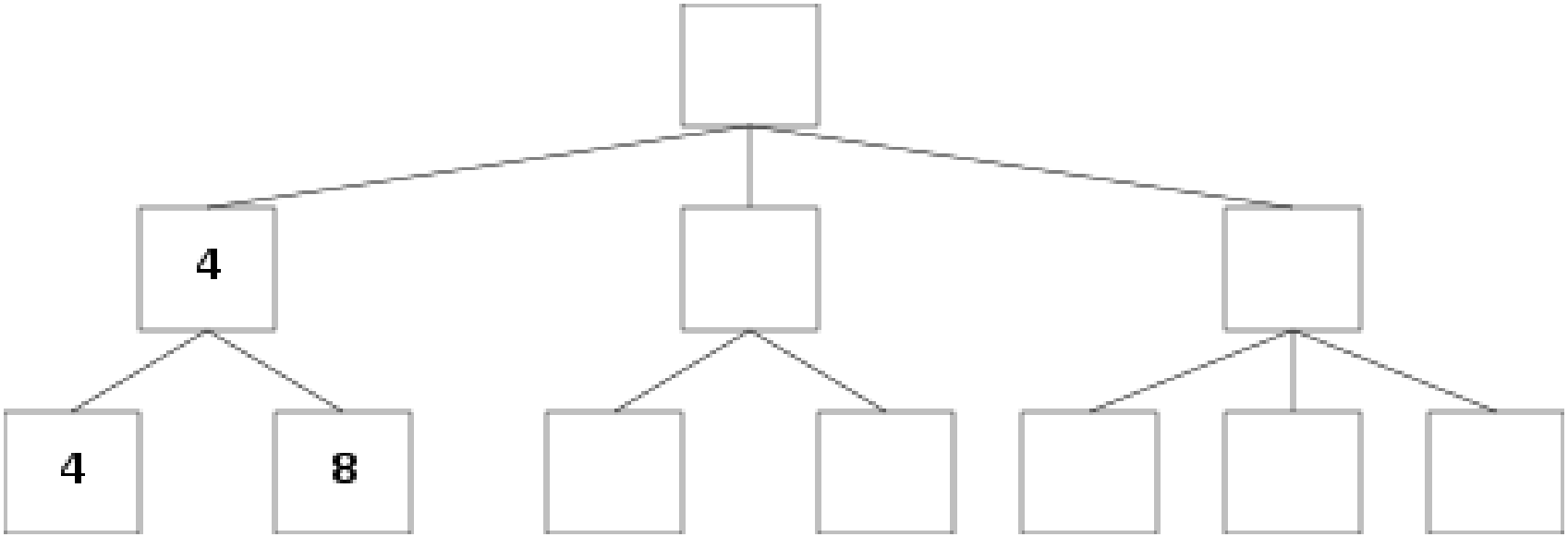
Speeding the scoring by cutting uninteresting branches

Subtrees are cut-off when

- one of the subnode has a lower score than parent's siblings (MIN level)
- one of the subnode has a higher score than the parent's siblings (MAX level)

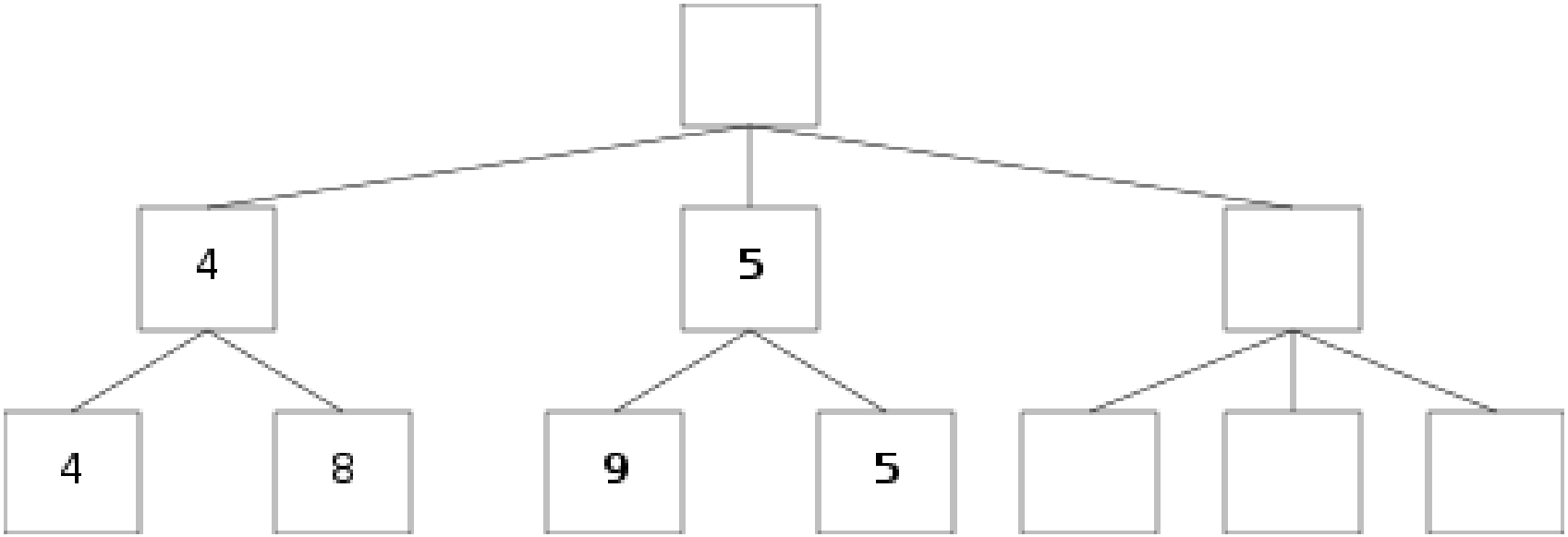
Alpha/Beta cut-off

An example cut-off



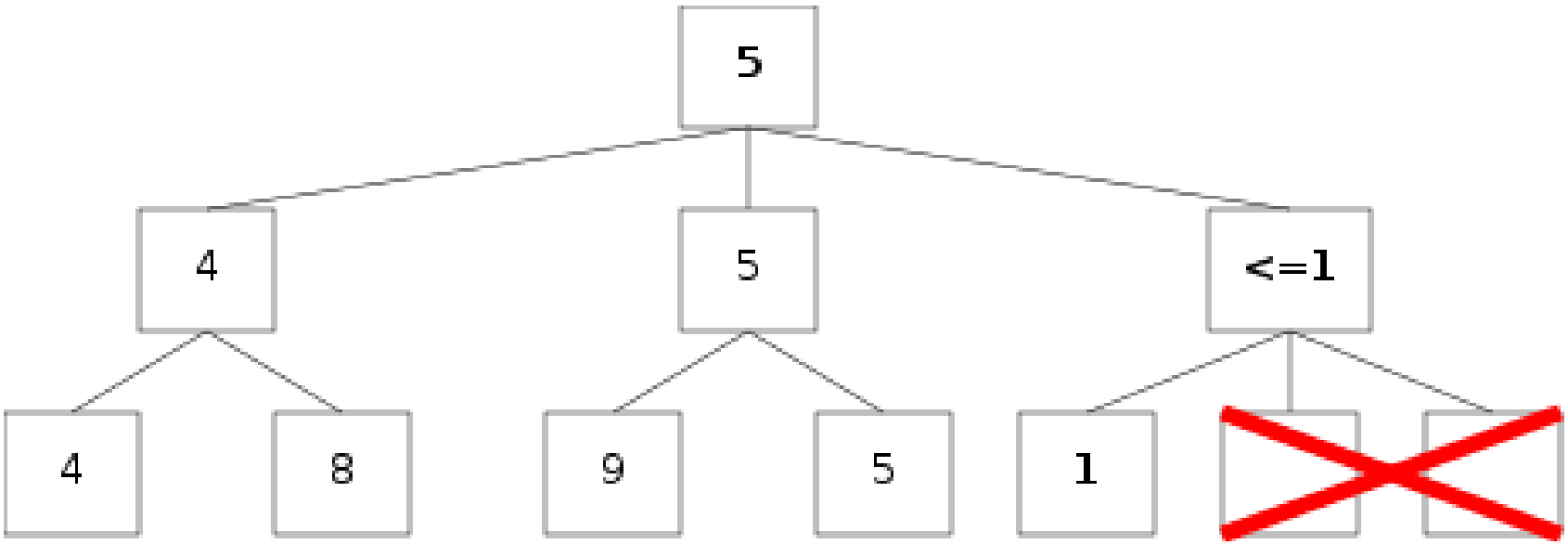
Alpha/Beta cut-off

An example cut-off



Alpha/Beta cut-off

An example cut-off

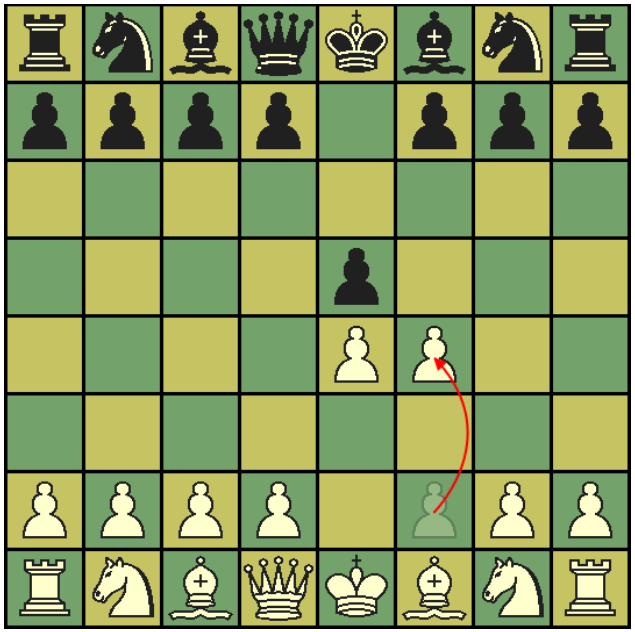


Alpha/Beta cut-off
Important remark

The order of the nodes during the exploration has an important impact on the number of subtrees actually visited.

Plies database

Adding memorized strategy to the game



The King's Gambit, a well known opening

Use of databases of opening and endgames

- use when the minimax would perform too poorly (too many or not enough pieces)
- may be used in cooperation to other methods to evaluate boards positions

Complex systems bases approaches

Can they do better?

The minimax algorithm

Classical approaches

Complex systems based approaches

- Genetically evolving the evaluation function
- Neural-network-based position evaluation
- Genetically programming a maximum search depth function
- Next-move-deciding neural network

Genetically evolving the evaluation function

What is actually evolved?

- The heuristic evaluation function is
  - a set of algorithms used to determine some properties of the position
  - a set of weights representing the importance of this property

Genetically evolving the evaluation function

What is actually evolved?

- The heuristic evaluation function is
  - a set of algorithms used to determine some properties of the position
  - a set of weights representing the importance of this property
- Here, the individuals are these weights (two sets depending on whether playing white of black)

Genetically evolving the evaluation function

Evaluation and fitness

- the individuals compete with each other
- the outcome of the games is used in the fitness function

Genetically evolving the evaluation function

Evaluation and fitness

- the individuals compete with each other
- the outcome of the games is used in the fitness function

$$\frac{\sum_{o=w,l,gd,bd} W_{eo} N_{eo} + W_{uo} N_{uo}}{N}$$

- $W \longrightarrow$ weights *of the GA*, $N \longrightarrow$ numbers of outcomes
- $w \longrightarrow$ win, $l \longrightarrow$ loss, $gd \longrightarrow$ good draw, $bd \longrightarrow$ bad draw
- $e \longrightarrow$ expected, $u \longrightarrow$ unexpected

Genetically evolving the evaluation function

Specificities of the GA

encoding 4 bit per variable in the chromosomes

selection depending on its fitness, an individual

- reproduces with another
- survives
- dies

overfitting avoidance 21 boards

optimization for speed individuals do not play against every other, random selection of a subest of the population

Genetically evolving the evaluation function

An interesting optimization

Boostrapping using a hard-coded black player

1. train whites against the black player
   $\Rightarrow$ perfect white opponent
2. train blacks against the white player
   $\Rightarrow$ perfect black opponent
3. go to 1

In three iterations

- always-drawing black player
- fairly good white player

Genetically evolving the evaluation function

Good performances but other issues

Big scalability problem

- only a reduced set of endgames has been used
- only three pieces on board

However evolving good players, it might be impossible to port to the whole game of chess.

Neural-network-based position evaluation

What is NeuroChess?

Board-evaluation functions using a neural network

- 175 input nodes
- evaluation using temporal differencing
- explanation-based learning
- training both from Grand Masters' games and self play

Neural-network-based position evaluation

Evaluation by temporal differencing

Each board s_t is evaluated according to the evaluation of the next board s_{t+2}

$$V^{\text{target}}(s_t) = \gamma V(s_{t+2})$$

- the final board is evaluated according to the outcome of the game
- $0 < \gamma < 1$ is a discount factor to favor boards leading to quick wins

Neural-network-based position evaluation

Evaluation by temporal differencing

Each board s_t is evaluated according to the evaluation of the next board s_{t+2}

$$V^{\text{target}}(s_t) = \gamma V(s_{t+2})$$

- the final board is evaluated according to the outcome of the game
- $0 < \gamma < 1$ is a discount factor to favor boards leading to quick wins

The neural network is expected to learn $V(s)$

Neural-network-based position evaluation

Explanation-based learning

- used to reduced the training time
- combining observations of the board with preliminary knowledge about the game (*i.e.* the rules)

Neural-network-based position evaluation

Explanation-based learning

- used to reduced the training time
- combining observations of the board with preliminary knowledge about the game (*i.e.* the rules)
  - another rule-neural network M
  - mapping from $V(s_t)$ to $V(s_{t+2})$
  - trained by backpropagation

Neural-network-based position evaluation

Explanation-based learning

- used to reduced the training time
- combining observations of the board with preliminary knowledge about the game (*i.e.* the rules)
  - another rule-neural network M
  - mapping from $V(s_t)$ to $V(s_{t+2})$
  - trained by backpropagation
- interesting fact: the main NN knows both $V(s_t)$ *and* $\frac{\partial V(s_t)}{\partial s_t}$
  $\Rightarrow$ may render usage of the minimax useless

Neural-network-based position evaluation

Compared performances

NeuroChess has been tested against GNU Chess[2]

After 2000 learning games actually learned to play better!

[2] standard Alpha/Beta based chess program

Neural-network-based position evaluation

Compared performances

NeuroChess has been tested against GNU Chess[2]

After 2000 learning games actually learned to play better but still lost in 70% of the cases...

An important point: 30% wins is more than standard backprop training for the same number of trainings

[2] standard Alpha/Beta based chess program

Genetically programming a maximum search depth function

What interest?

- minimax is a depth-first tree search
- computationnaly impossible to explore down to the leaves
- a stopping criteria (depth) is needed
  - usually hard coded
  - may be obtained more precisely using other methods

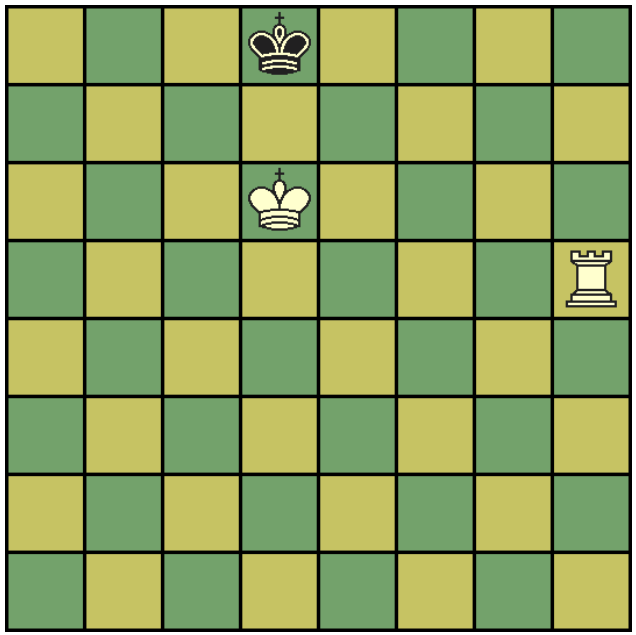
Genetically programming a maximum search depth function

Subset of the problem

- only K-R-K endgames
- two different functions

  Grand KRK determining the number of moves to checkmate

  Petite KRK telling whether a board lead to a checkmate in said moves



A King-Rook-King endgame

Genetically programming a maximum search depth function

Terminals and functions

Only the positions of the pieces are terminals:
wk_r, wk_f, wr_r, wr_f, bk_r, bk_f
Almost no chess knowledge in the base functions

$edge(i)$	1 if on an edge, 2 otherwise
$distance(i,j)$	positive difference of its arguments
$ifthen(i,j,k)$	j if $i = 1$, k otherwise
$compare(i,j)$	1 if $i < j$, 2 otherwise

Genetically programming a maximum search depth function

Fitness functions

Grand KRK

$$F = 1 - \frac{\sum_0^{16} \frac{C_i - I_i}{N_i}}{17},$$

$C_i \longrightarrow$ number of correct answers, $I_i \longrightarrow$ incorrect one, $N_i \longrightarrow$ total

Petite KRK

$$F = 1 + N_p - (C_p - I_p) - \frac{C_n}{N_n},$$

$p \longrightarrow$ positive, $n \longrightarrow$ negative

Genetically programming a maximum search depth function

The genetic algorithm

A regular genetic algorithm

- 75% crossover probability
- 15% mutation probability
- 5000 and 1000 individuals for the grand and the petite KRK (resp.)

Genetically programming a maximum search depth function

The results

Grand KRK 41% correct answers after 31 generations

Petite KRK 97% correst answers after 145 generations

Note: 16 petite KRK functions are needed to achieve the same work as the grand KRK one

Next-move-deciding neural network

A complete chess player

The *distributed chess project* tries to *genetically evolve* a complete *neural network* chess player able to

- recognize patterns of the board
- decide what to play next

Search and evaluation functions no longer needed

Next-move-deciding neural network

An interesting implementation

The GA is run on the *whole* internet, distributing the computation task to people's idle computers

- large computing power speeding the genetic process
- natural creation of subpopulations

Next-move-deciding neural network

Results

The networks were not trained for the complete game of chess

$\Rightarrow$ interesting subsets of the game

The results show that the networks *actually learned* to play chess

Conclusions

Common results

- scalability problem
- players using only these methods worse than usual ones
- but significant results

Conclusions

Common results

- scalability problem
- players using only these methods worse than usual ones
- but significant results

Using complex systems in chess is not totally worthless, but they should be used to improve, not compete with, actual implementations.

References


Chris Wyman.
Using genetic algorithms to learn weights for simple king-pawn chess endgames, 1999.
http://www.cs.utah.edu/~wyman/classes/ML_proj/paper.html.


Sebastian Thrun.
Learning to play the game of chess.
In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 1069–1076. The MIT Press, Cambridge, MA, 1995.
http://citeseer.ist.psu.edu/article/thrun95learning.html.


David Gleich.
Machine learning in computer chess: Genetic programming and KRK, 2003.
http://citeseer.ist.psu.edu/gleich03machine.html.


Ralf Seliger.
The distributed chess project.
http://neural-chess.netfirms.com/HTML/project.html.