Artificial Intelligence 4 Games

More MCTS Modifications

2020

Jakub Kowalski, University of Wrocław

N-Gram Selection Technique

(NST)

Last-Good-Reply Policy

(LGR)

Literature

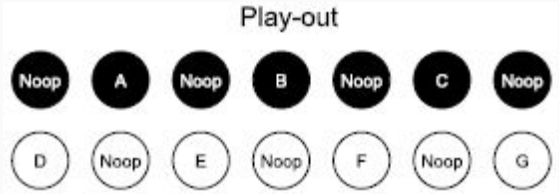
★ M. Tak, M. Winands, Y. Björnsson: N-grams and the Last-Good-Reply policy applied in General Game Playing. TCIAIG, vol. 4, no. 2, pp. 73–83, 2012

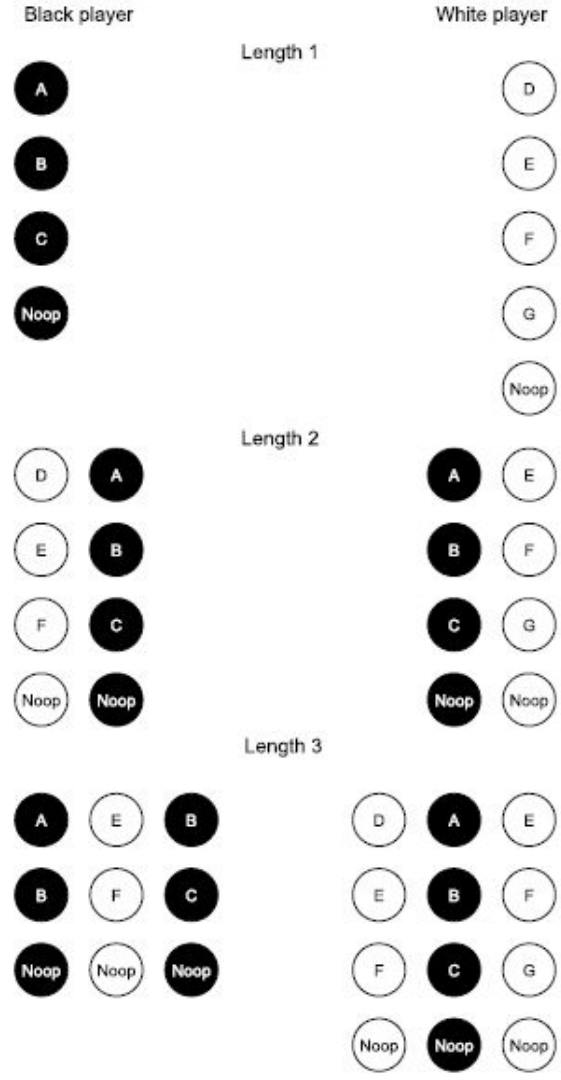
N-Gram Selection Technique

As we said before, NST is an extension of MAST that keeps track of move sequences instead of single moves.

This, in some way, gives context to the gathered statistics, but makes computations more expensive.

For each playout, we compute consecutive move sequences up to a given length N.





NST parameters

The global table with average move rewards is now more complex, as (for each player) we remember reward for each possible N-gram of moves.

Thus, we store average rewards for sequences, e.g. $Q_p(a)$, $Q_p(a, b)$, $Q_p(a, b, c)$, … Based on those values we calculate score, that can be used in ε-greedy or Gibbs sampling for move choice during the simulation phase.

To prevent the algorithm from using low-quality statistics, the authors propose to introduce an additional parameter ***k***.

The score of the move ***a*** is the average of ***Q*** values that starts with ***a***. But, we want to omit the sequences that were visited less than ***k*** times. Thus, if sequence of length 2 was not sampled enough times, the value of $Q_p(a)$ (standard MAST) will be returned.

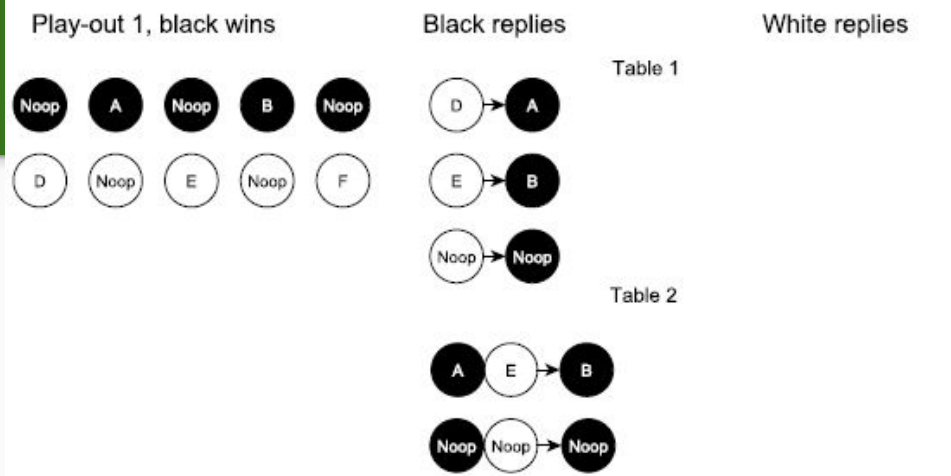
In Stanford's GGP domain the tests have been made for ***N***=3 and ***k***=7.

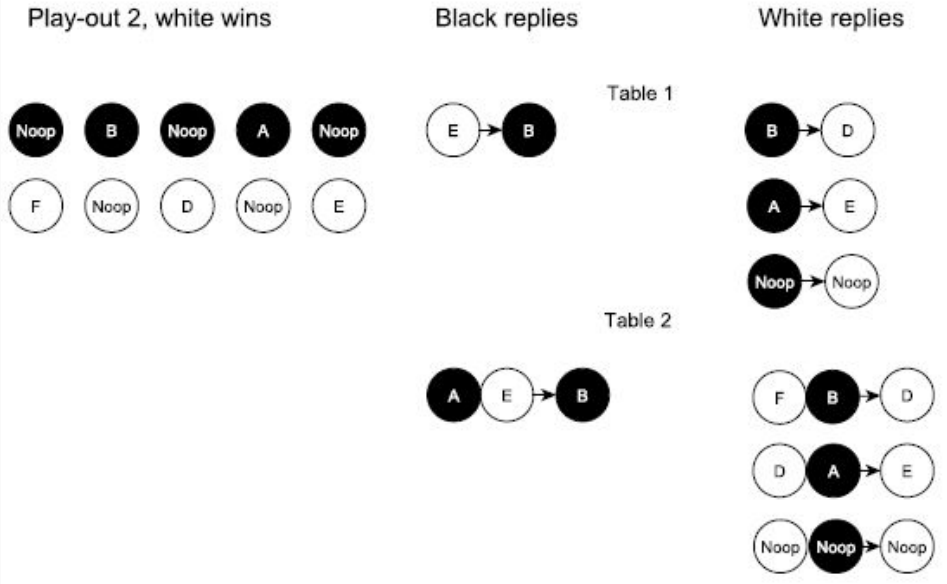
Last-Good-Reply Policy

This is a simulation strategy that plays a move that is a successful reply to the immediately preceding moves whenever available.

We store two tables with best replies for N-grams of size 1 and 2, and update them after each simulation.

If, after a simulation, the player's reward is assumed good enough (winning/draw) the best replies are updated. Otherwise, they are removed from the tables.





LGR usage

The best-reply tables can be used both during playouts and in the MCTS tree expansion when choosing a new node to add to the tree .

If the sequence of length 2 is legal and available in the table its value (best-reply move) is used.

If not, we try to use the reply for the last move of the previous player.

If also this move is illegal, or the data is not available, we use the default policy (e.g. MAST).

The LGR policy (and also NST) naturally works for more than two players, as it uses the player ordering as a base for N-grams.

Also, as we see from examples, LGR/NST can be used for simultaneous moves games, as long as we fix the ordering of players we look at.

Apart from GGP, LGR policy has been proven to work in e.g., Go and Havannah.

Sequential Halving applied to Trees

(SHOT)

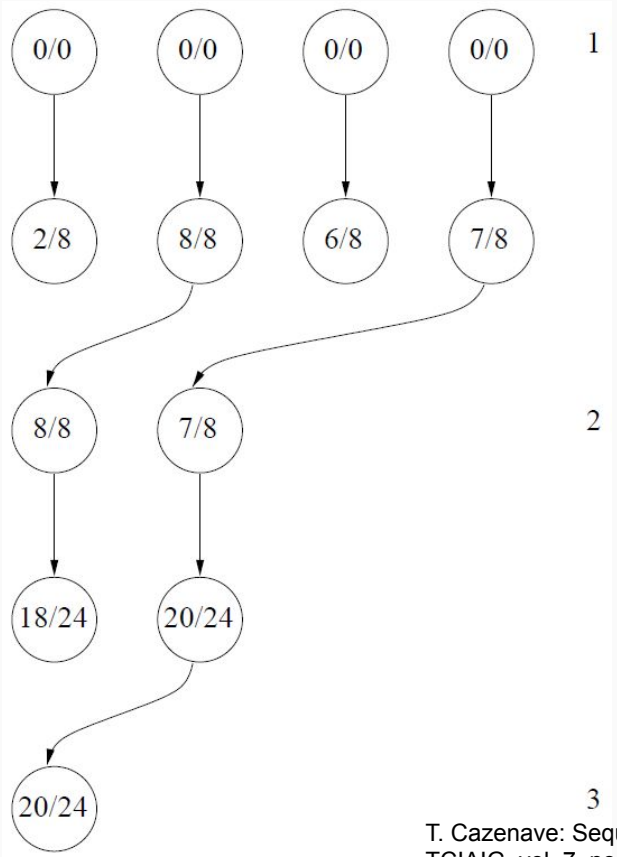
Literature

★ T. Cazenave: Sequential Halving applied to Trees. TCIAIG, vol. 7, no. 1, pp. 102–105, 2015

Sequential Halving

The algorithm is based on a sequential elimination of moves, and distribution of computational budget based on the expected move quality.

In rounds, we play a fixed amount of playouts for each of the remaining moves. After each round, we remove half of the moves with the worst scores. The algorithm returns the one move that remains.



Example:

- ➔ 4 moves,
- ➔ budget of 64 playouts.

T. Cazenave: Sequential Halving applied to Trees. TCIAIG, vol. 7, no. 1, pp. 102–105, 2015

Sequential Halving applied to Trees (SHOT)

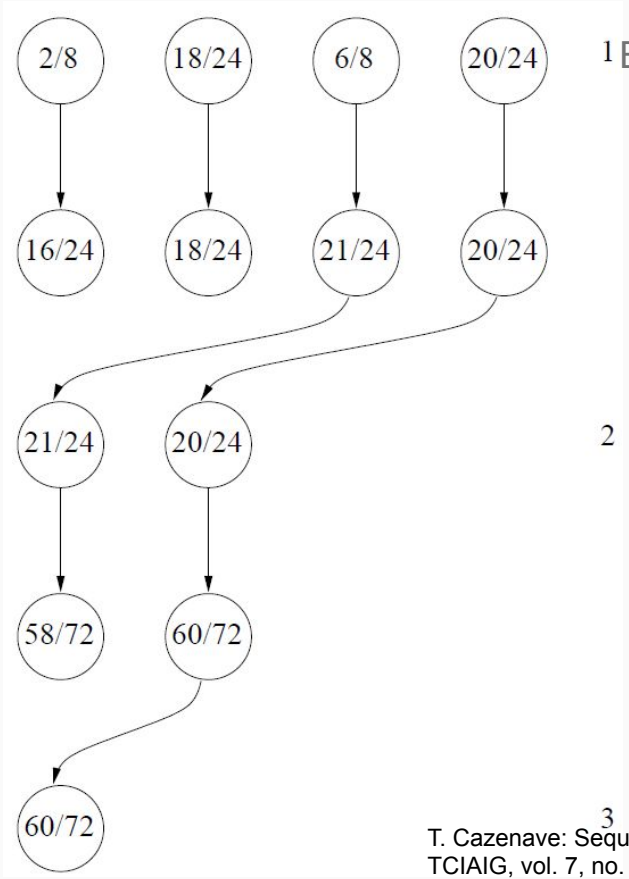
Here, we have to consider revisiting already tested node with an increased budget.

We apply Sequential Halving on the overall budget as it was a whole.

SHOT properties:

- ★ Can also work in time-based scenario
- ★ Spends less time in tree than UCT
- ★ At equal time setting usually beats UCT in combinatorial games
- ★ No parameter tuning

Thus, may be worth a try.



Example:

- ➔ 4 moves,
- ➔ node with 64 playouts
- ➔ new budget of 128 playouts.

T. Cazenave: Sequential Halving applied to Trees. TCIAIG, vol. 7, no. 1, pp. 102–105, 2015

SHOT Pseudocode

SequentialHalving ($budget$)
$S \leftarrow [possibleMoves]$
while $|S| > 1$ **do**
  for each move m in S **do**
    play (m)
    perform $\left\lfloor \frac{budget}{|S| \times \lceil log_2(|possibleMoves|) \rceil} \right\rfloor$ playouts
    undo (m)
  end for
  $S \leftarrow$ set of $\left\lceil \frac{|S|}{2} \right\rceil$ moves in S with the largest empirical average
end while
return the move in S

SHOT ($board$, $budget$, $budgetUsed$, $playouts$, $wins$)
$S \leftarrow$ possible moves
if board is terminal **then**
  update $playouts$, $wins$ and return
end if
if $budget = 1$ **then**
  $result \leftarrow playout(board)$
  update $playouts$, $budgetUsed$, $wins$ and return
end if
if $|S| = 1$ **then**
  $u \leftarrow 0, p \leftarrow 0, w \leftarrow 0$
  SHOT ($play(board, move)$, $budget$, u, p, w)
  update $playouts$, $budgetUsed$ and $wins$
  return $move$
end if
$t \leftarrow$ entry in the transposition table
if $t.budgetNode \leq |S|$ **then**
  for move m in S with zero playouts **do**
    $result \leftarrow playout(play(board, m))$
    update $playouts, budgetUsed, wins$ and t
    return if $budget$ playouts have been played
  end for
end if

sort moves in S according to their mean
$b \leftarrow 0$
while $|S| > 1$ **do**
  $b \leftarrow b + max(1, \left\lfloor \frac{t.budgetNode + budget}{|S| \times \lceil log_2(|possibleMoves|) \rceil} \right\rfloor)$
  for move m in S by decreasing mean **do**
    if $t.playouts[m] < b$ **then**
      $b1 \leftarrow b - t.playouts[m]$
      if at root and $|S| = 2$ and m is the first move in S **then**
        $b1 \leftarrow budget - budgetUsed - (b - t.playouts[secondMove])$
      end if
      $b1 \leftarrow min(b1, budget - budgetUsed)$
      $u \leftarrow 0, p \leftarrow 0, w \leftarrow 0$
      SHOT ($play(board, m)$, $b1$, u, p, w)
      update $playouts, budgetUsed, wins$ and t
    end if
    break if $budgetUsed \geq budget$
  end for
  $S \leftarrow$ sort $\left\lceil \frac{|S|}{2} \right\rceil$ best moves in S
  break if $budgetUsed \geq budget$
end while
update $t.budgetNode$
return first move of S

Progressive strategies

➔ bias

➔ unprunning / widening

Literature

★ G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, B. Bouzy: Progressive Strategies for Monte-Carlo Tree Search. New Math. Nat. Comput., vol. 4, no. 3, pp. 343–357, 2008.

★ R. Coulom: Computing Elo Ratings of Move Patterns in the Game of Go. Int. Comp. Games Assoc. J., vol. 30, no. 4, pp. 198–208, 2007.

Progressive bias

This method is one of the easiest way to include domain-specific heuristic knowledge in the MCTS selection/expansion phase.

Assuming that for each node ***v*** we can compute some heuristic evaluation function ***H*** that will score the game state, we have a following function (as previously ***N(v)*** is the number of visits in the node):

$$f(v) = \frac{H(v)}{N(v) + 1}$$

We modify the UCT selection, by simply adding ***f(v)*** to UCB1 evaluation of a node. Thus, for a small number of visits the heuristic-part will be dominant, but with more simulations going through the node it will gradually lose its impact.

If the **H(v)** is expensive to compute, it may be wise to postpone its computation until some fixed number of visits - which limits the number of nodes that need such evaluation.

On the other hand, the term **f(v)** can also be used for the expansion purposes, to prioritize promising nodes first.

Progressive unpruning/widening

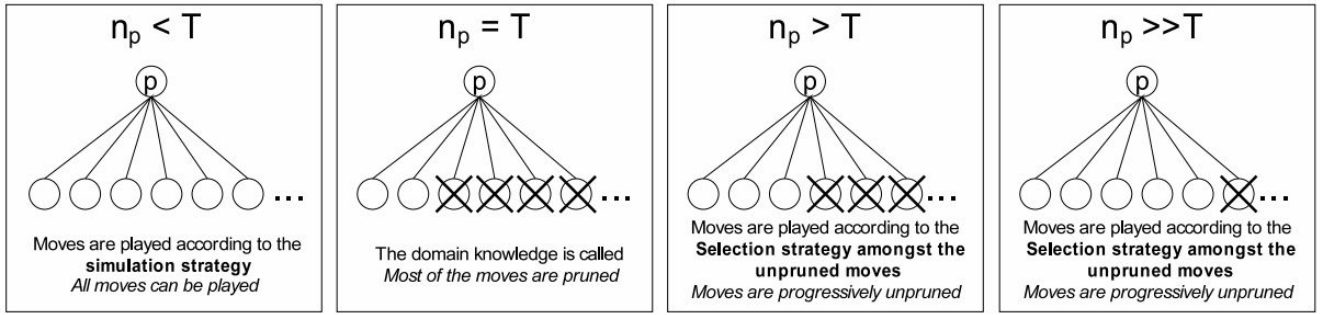
Here, the idea is to quickly narrow down the branching factor based on the initial values of the nodes, and then as more time becomes available, increase it progressively.

This method has proven to be working on Go, in combination with heuristic knowledge that somewhat ensures that the initial node evaluation is reliable.

When number of games n_p equals the threshold T, we prune (exclude from simulations) all children except k_{init} ones with the best so-far evaluation / heuristic value. The k-th node is unpruned when the number of simulations in the parent surpasses

$$A \times B^{k-k_{init}}$$

Values tested for Go were: k_{init}=5, A=50, B=1.3



Progressive unpruning/widening

It is however possible to use progressive unprunning in a knowledge-free way, with the initial evaluation based on a sufficient number of standard-MCTS simulations.

The main idea works the same, as after the threshold (which is of course game-dependent) we reduce the number of nodes under a consideration and gradually increase their number growing logarithmically on the number of simulations.

We can alternatively state the same idea the following way: ***k***-th move is added when t_{k-1} simulations have been performed, where

$$t_0 = 0 \quad t_{k+1} = t_k + A \times B^k$$

Another Go implementation based on this method used ***A***=40, ***B***=1.4

The exact game-specific parameters depends on the branching factor, computation efficiency and reliability of the initial node evaluations.

Video game-based modifications

- ➔ MixMax backups
- ➔ Macro-actions
- ➔ Partial expansion
- ➔ Reversal penalty
- ➔ Roulette wheel selection

Literature

- ★ F. Frydenberg, K. R. Andersen, S. Risi, J. Togelius: Investigating MCTS Modifications in General Video Game Playing. IEEE COG, pp. 107–113 2015.
- ★ E. J. Jacobsen, R. Greve, J. Togelius: Monte Mario: Platforming with MCTS. GECCO 14, pp. 293–300, 2014.
- ★ E. J. Powley, D. Whitehouse, P. I. Cowling: Monte Carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. IEEE CIG, pp. 1–8, 2013.

Video game-based modifications

Here, we want to describe a series of small MCTS modifications inspired by the application of MCTS in the domains of real-time video games, usually when controlling a character (with a large branching factor) traversing the game's world.

Examples consists of Mario, Bomberman (CodinGame adaptation is called Hypersonic), and various games from General Video Game AI domain.

Modifications

- ➔ MixMax backups
- ➔ Macro-actions
- ➔ Partial expansion
- ➔ Reversal penalty
- ➔ Roulette wheel selection

MixMax backups

It is often a case that the path ahead our character is dangerous, yet awaiting dangers are deterministic and predictable, vide monster routes in platformers or explosions in bomberman.

Nevertheless, applying standard MCTS cause the character to act cowardly, as the average rewards take into account all possible death-paths even if they are easy to avoid.

The goal of MixMax is to, at least partially, make up for this.

Let ***Q(s)*** be the average reward for state ***s***, as in the standard UCB1 formula. We want to substitute the occurrence of ***Q(s)*** by

$$q \cdot M(s) + (1-q) \cdot Q(s)$$

where ***M(s)*** is the maximum score obtained from ***s***. So we weight the contribution of good paths, reducing the defensiveness of the character.

It was reported that the MixMax approach may not work greatly on its own, but it is a good addition to other modifications. For games tested in the literature values of ***q*** around 0.1 worked.

Macro-actions

If the domain is large but the world is relatively sparse, the computational limitations may result in a character being very shortsighted.

We can then introduce the macro-actions to artificially increase the search depth at the cost of precision.

Macro-actions consist of modifying the expansion process such that each action is repeated a fixed number of times before a child node is created. Thus, each edge is labeled by a series of identical actions.

This lose of granularity may work as stated for games without instant death situations (eg. racing-style). In other cases (e.g. platformers) it may be safer to include a domain knowledge of being *near the danger* (enemy, pit) to switch back to micro-level planning (actions of size 1).

This is another enhancement that usually pays pay-off when combined with other modifications, and only for a specific types of games. Note that its role is similar as this of the Partial expansion (and partially Reversal penalty).

Partial expansion

Another approach to increase the search depth in case of a large branching factor games is to lift the restriction that we need to first expand all node's children.

Thus, at the cost of exploration, we allow to consider grandchildren before testing all children before.

In the tree policy phase, we compute the urgency of creating a new child. If this value exceeds the confidence of any of the children nodes then we perform expansion. Otherwise, we simply go to the best already visited child.

$$k + C\sqrt{\frac{2ln(N(s))}{1 + c_n}}$$

This urgency formula looks similar to the UCB one, with constant ***C*** and number of node visits ***N(s)***, but we also take into account the number of expanded children in the current node c_n. Parameter ***k*** is a default reward for a non-existing child, e.g. 0.5.

Reversal penalty

For the games with “physical positions” it is very common for simulated games to create many oscillations, e.g. going back and forth between a few adjacent tiles.

This usually leads to unnecessary and redundant series of moves and thus wasted computation resources.

There are many ways how one can modify search to prevent reversal moves, it is also very game-dependent.

One may simply forbid making a move that is cancels the previous one. Note that for some games such hard-removal may be dangerous.

We may also go deeper and forbid some hand-crafted sequences of action about which we know they are redundant.

Other approach is to remember some number (e.g. 5) of last-visited positions in every MCTS node, and if a child node represents one of this positions, a small penalty is applied (e.g. its UCT value is multiplied by 0.95).

Roulette wheel selection

This modification aims at the ordering of expanding new nodes.

Instead of doing this with uniform random, we want to introduce some domain knowledge to increase the probability of exploring the most promising actions first, and skew the search tree in a desirable direction.

The idea is not as strict as in move ordering that is usually used in min-max approach, but it serves a similar goal.

For each action, we assign weight, corresponding to a desired priority of this action.

When choosing which of untried children to expand, we use a roulette wheel (weight-proportionate) selection.

Roulette wheel selection is not much influential on its own, but it has been shown a very good performance in conjunction with Partial expansion.

Summary

Summary

- ★ There are MANY variants of MCTS
- ★ Some of them are really working
- ★ But they are often well-suited for a specific type of game / occurring problem

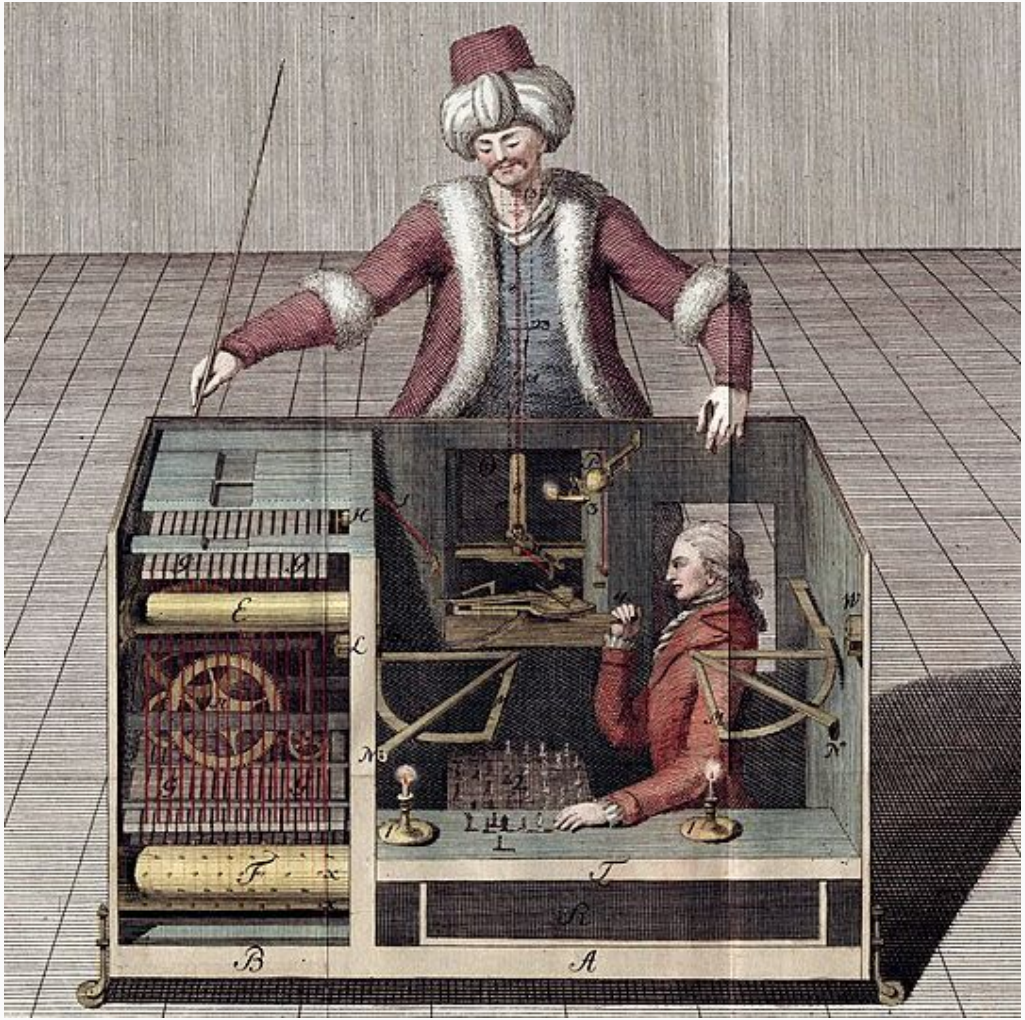
4 Variations
- 4.1 Flat UCB
- 4.2 Bandit Algorithm for Smooth Trees
- 4.3 Learning in MCTS: TDL; TDMC(λ); BAAL
- 4.4 Single-Player MCTS: FUSE
- 4.5 Multi-player MCTS: Coalition Reduction
- 4.6 Multi-agent MCTS: Ensemble UCT
- 4.7 Real-time MCTS
- 4.8 Nondeterministic MCTS: Determinization; HOP; Sparse UCT; ISUCT; Multiple MCTS; UCT+; MC$_{\alpha\beta}$; MCCFR; Modelling; Simultaneous Moves
- 4.9 Recursive Approaches: Reflexive MC; Nested MC; NRPA; Meta-MCTS; HGSTS
- 4.10 Sample-Based Planners: FSSS; TAG; RRTs; UNLEO; UCTSAT; ρUCT; MRW; MHSP

5 Tree Policy Enhancements
- 5.1 Bandit-Based: UCB1-Tuned; Bayesian UCT; EXP3; HOOT; Other
- 5.2 Selection: FPU; Decisive Moves; Move Groups; Transpositions; Progressive Bias; Opening Books; MCPG; Search Seeding; Parameter Tuning; History Heuristic; Progressive History
- 5.3 AMAF: Permutation; α-AMAF Some-First; Cutoff; RAVE; Killer RAVE; RAVE-max; PoolRAVE
- 5.4 Game-Theoretic: MCTS-Solver; MC-PNS; Score Bounded MCTS
- 5.5 Pruning: Absolute; Relative; Domain Knowledge
- 5.6 Expansion

6 Other Enhancements
- 6.1 Simulation: Rule-Based; Contextual; Fill the Board; Learning; MAST; PAST; FAST; History Heuristics; Evaluation; Balancing; Last Good Reply; Patterns
- 6.2 Backpropagation: Weighting; Score Bonus; Decay; Transposition Table Updates
- 6.3 Parallelisation: Leaf; Root; Tree; UCT-Treesplit; Threading and Synchronisation
- 6.4 Considerations: Consistency; Parameterisation; Comparing Enhancements

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, D., Colton, S. A survey of Monte Carlo tree search methods. TCIAIG, vol. 4, no. 1, pp. 1–43, 2012.

Thanks!



Bonus reference quiz