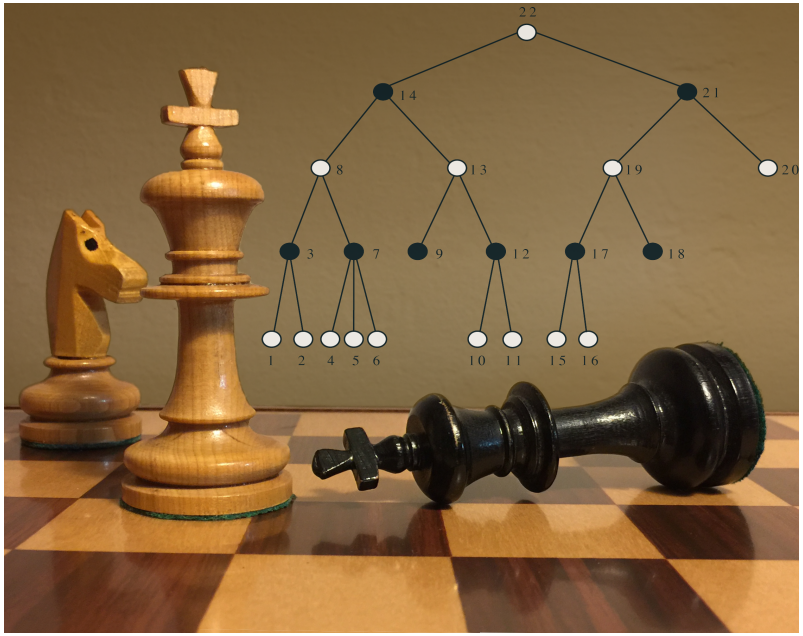
Superior Computer Chess with Model Predictive Control, Reinforcement Learning, and Rollout

Atharva Gundawar, Yuchao Li, and Dimitri P. Bertsekas

Based on the papers
“Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming", by D.P. Bertsekas, arXiv:2406.00592, Jun. 2024
“Superior Computer Chess with Model Predictive Control, Reinforcement Learning, and Rollout", by A. Gundawar, Y. Li, and D.P. Bertsekas, arXiv:2409.06477, Oct. 2024

Outline

Computer Chess - AlphaZero (2017)

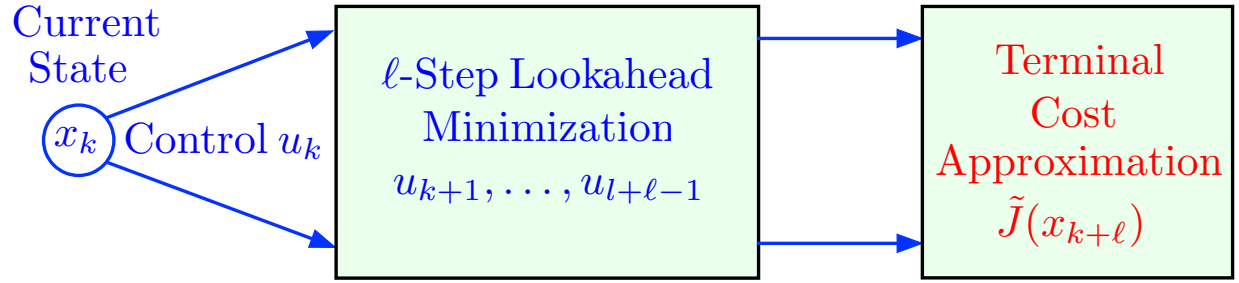


AlphaZero (and most chess programs) involve two algorithms:

- Off-line training of a position evaluator (among others), using deep NNs
- On-line play by multistep lookahead, and position evaluation at the end

- Most attention has been focused on the AlphaZero off-line training, which involves important innovations in NN technology, approximate policy iteration, etc
- The on-line algorithm part is more or less traditional. It is critically important for good performance
- On-line play in computer chess is strongly connected with MPC

Model Predictive Control (MPC): ℓ-Step Lookahead Optimization with Cost Approximation $\tilde{J}$ at the End



At x_k → Solve an ℓ-step lookahead problem with terminal cost $\tilde{J}$

1st ℓ Steps — Future

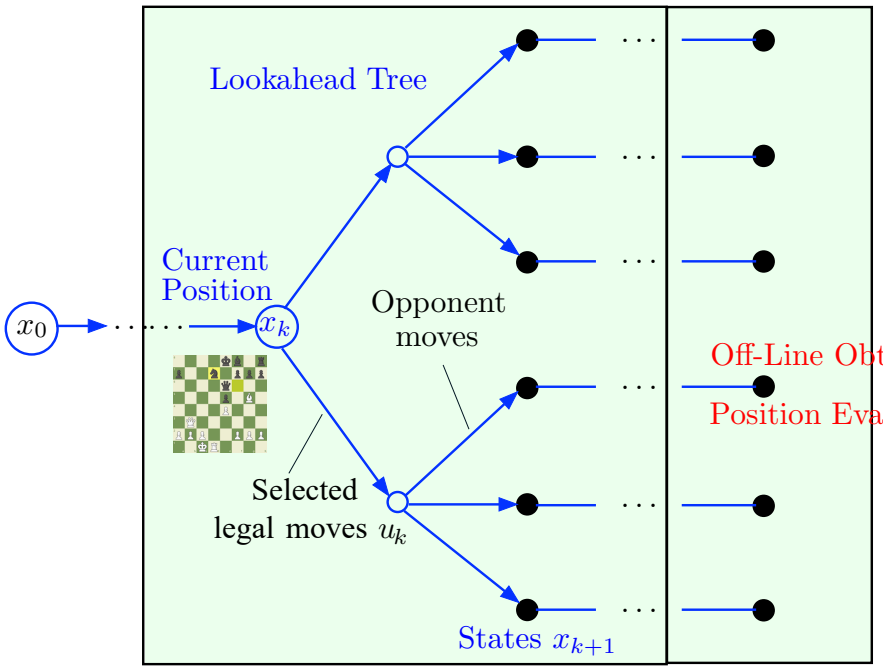
$$\min_{u_k, u_{k+1}, \ldots, u_{k+\ell-1}} \left\{ \sum_{m=0}^{\ell-1} g(x_{k+m}, u_{k+m}) + \tilde{J}(x_{k+\ell}) \right\}$$

Apply the first control $\tilde{u}_k$, discard the remaining controls

Discrete-time deterministic optimal control problem

- Dynamic system equation at time k: $x_{k+1} = F(x_k, u_k)$
- State and control at time k: x_k and u_k
- Cost at stage k: $g(x_k, u_k)$
- The minimization problem is often solved exactly: Shannon's type A scheme

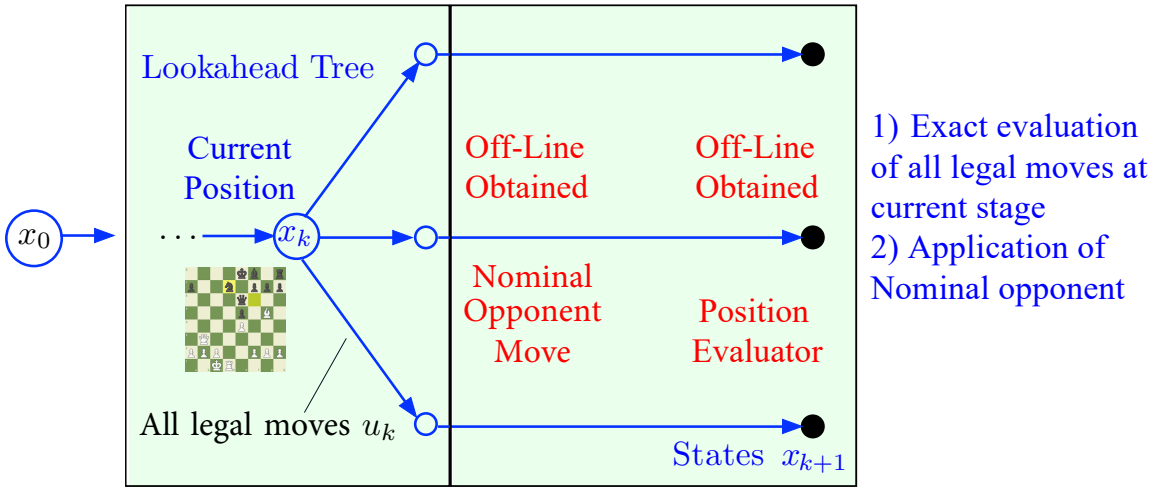
On-Line Play Architecture in Computer Chess



It is “isomorphic" to the MPC architecture, except:

- In chess the state and control spaces are discrete, while in MPC they are usually continuous (or mixed discrete-continuous). This difference is inconsequential
- In chess the lookahead tree is usually “pruned", Shannon’s type B scheme
- Exact computation in one-step lookahead is essential: There is theoretical significance, more on this later
- Another difference is that chess is a two-player game. More on this later, but think of chess against a fixed opponent (this makes chess a one-player game)

Principal Ideas of Our Scheme



We introduce a one-player MPC architecture (the true opponent is approximated by a “nominal" opponent)

We use two available chess engines as components (a meta algorithm)

- The position evaluator engine: Evaluates any given position
- The nominal opponent engine: Predicts the move of the true opponent of MPC-MC (exactly or approximately)
- Each move involves a Newton step starting at the position evaluation function

Modeling for Chess

- x_k: the chess position at time/move k
- u_k: a legal move at time k in response to position x_k
- $L(x_k)$: all legal moves at position x_k
- w_k: a legal move of an opponent at time k in response to position x_k followed by move u_k
- The resulting position at time $k+1$ following (x_k, u_k, w_k) is given by

$$x_{k+1} = f(x_k, u_k, w_k),$$

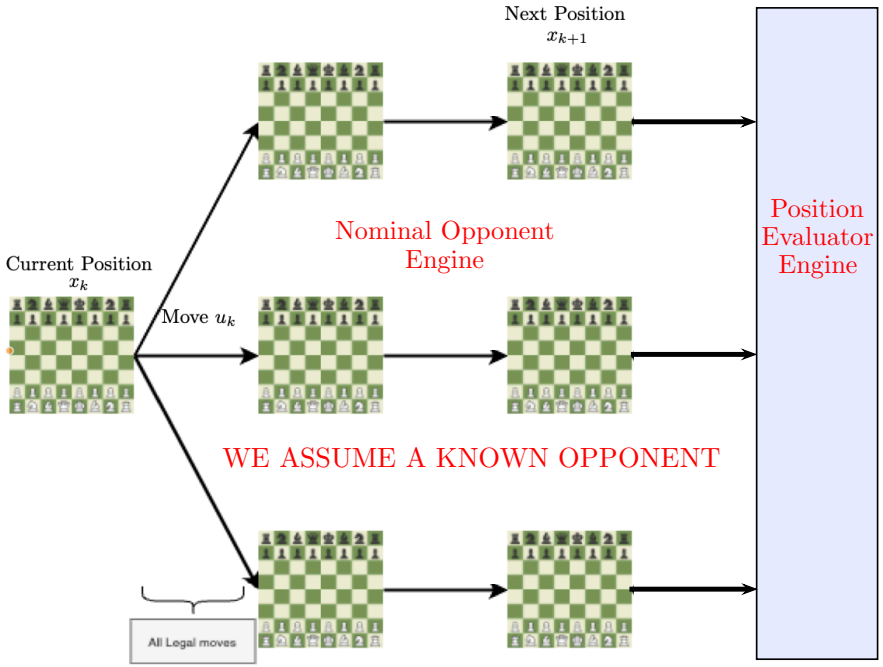
  where f is a known function
- Negative costs for winning positions, positive costs for losing positions, and zero costs at other positions.
- Our goal: minimizing cost, or equivalently, winning

Chess Engines: Move Generation and Position Evaluation

Screen shot from `https://www.chess.com/`

- Given a pair (x, u), a chess engine will produce a numerical evaluation $Q(x, u)$ of the position resulting from chose the move u at position x
- Based on the function $Q(x, u)$, a chess engine defines a policy μ by setting $\mu(x) \in \arg\min_{u \in L(x)} Q(x, u)$
- It also provides a evaluation function $E(x) = \min_{u \in L(x)} Q(x, u)$

The MPC-MC Architecture with One-Step Lookahead



We use two available chess engines as components. At a given position x_k:

- We generate all legal moves u_k at x_k
- For each pair (x_k, u_k), one engine is used to generate best responding move $\nu(x_k, u_k)$
- For each move u_k, we use the position evaluator engine to evaluate the position $x_{k+1} = f(x_k, u_k, \tilde{w}_k)$ corresponding to the nominal opponent $\tilde{w}_k = \nu(x_k, u_k)$
- We select the move u_k that corresponds to the position x_{k+1} with best evaluation

Deterministic MPC-MC: Known Opponent

- Nominal opponent $\nu(x_k, u_k)$ is the true opponent: dynamics are

$$x_{k+1} = F(x_k, u_k),$$

  where $F(x_k, u_k) = f(x_k, u_k, \nu(x_k, u_k))$
- We obtain an one-player optimal control/dynamic programming problem
- The optimal cost function $J^*(x)$ is the solution to the Bellman's equation

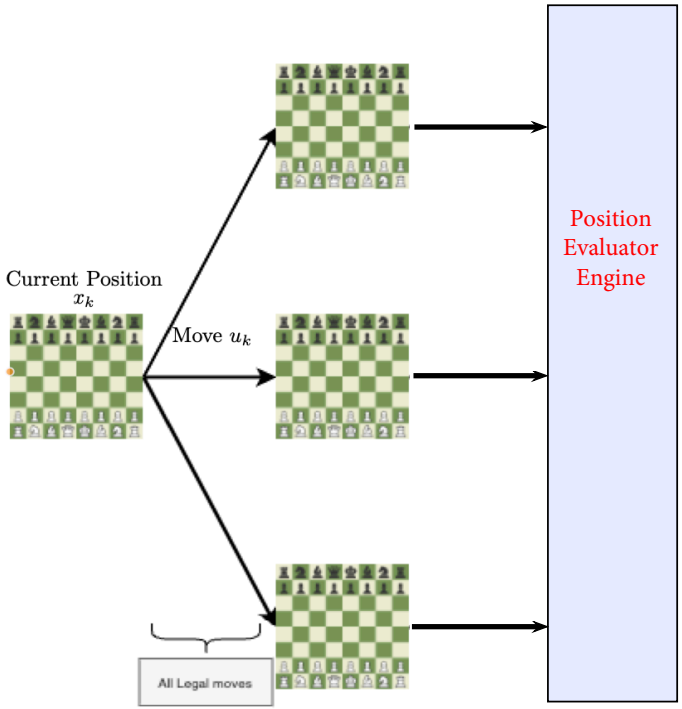
$$J^*(x) = \min_{u \in L(x)} J^*\big(F(x, u)\big),$$

  where $J^*(x) \neq 0$ for x where one player wins regardless of the choice of the opponent, and $J^*(x) = 0$ for theoretical draw
- Denote by $\tilde{\mu}$ the policy computed by MPC-MC, and $J_{\tilde{\mu}}$ its corresponding performance, then we have

$$\frac{J_{\tilde{\mu}}(x) - J^*(x)}{\tilde{J}(x) - J^*(x)} \to 0 \quad \text{as } \tilde{J} \to J^*$$

  where we use evaluation function E as $\tilde{J}$

MPC-MC: Variants



- Stochastic MPC-MC: use nominal opponent $\nu(x_k, u_k)$ to approximate an unknown opponent. The nominal opponent should be strong
- Fortified MPC-MC: compare u_k selected by MPC-MC with the move $\mu(x_k)$ selected by the engine, and choose u_k only if it is better than $\mu(x_k)$ according to the evaluation engine
- Half-step lookahead: evaluate directly the position-move pairs (x_k, u_k), easier to compute, and well-suited when nominal opponent is weak
- Multistep lookahead and so on ...

Computational Results Using the Stockfish (SK) Engine

We tested two variants of the algorithm

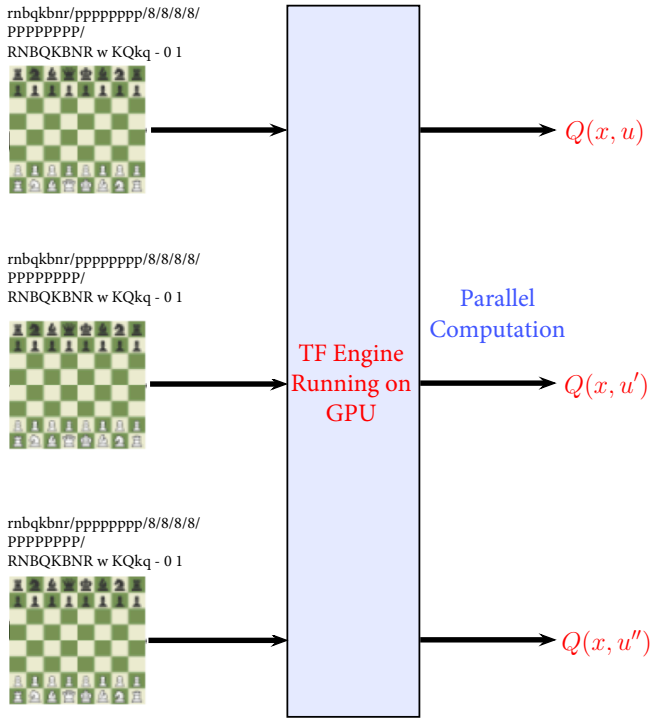
- Standard version
- Fortified version (plays a little better against very strong opponents, a little worse against weaker opponents)

Table: MPC-MC vs SK

SK Strength	Exact. Known Opponent		Approx. Known Opponent	
	Standard	Fortified	Standard	Fortified
0.5 secs	7.5-2.5	8-2	8-2	7-3
2 secs	5-5	5.5-4.5	5.5-4.5	6.5-3.5
5 secs	5-5	5.5-4.5	10-10	10.5-9.5

- We use MPC-MC (one-step lookahead), with SK as both the position evaluator and the nominal opponent, to play against SK
- Similar results obtained with other engines
- We can obtain better results with multistep lookahead
- Parallel computation is essential to reduce the move generation time

Computational Results Using a Transformer (TF) Engine



- We use the chess engines (TF for short) developed by Ruoss et. al, which use trained transformer network of varying sizes: 136 M and 270 M
- Given a chess position x, the transformers provides $Q(x, u)$ for all legal moves u
- With Graphical Computing Units (GPU), TF engines supports parallel move selection/position evaluation
- We tested half-step lookahead against TF engine:
  - 136 M: 7-3
  - 270 M: 7.5-2.5

Concluding Remarks

- We proposed a meta chess algorithm MPC-MC that improves the performance of existing chess engines
- Key ideas of the framework:
  - Evaluating all legal moves at the given position
  - Using nominal opponent to predict the moves of the true opponent
- The framework invokes a Newton step starting from the position evaluation engine, which leads to the performance improvement
- It has the potentials for other two-person zero-sum games without stochastic uncertainties, such as Xiangqi, Checkers, Go, and so on

Thank you!