Artificial Intelligence

Bandits, MCTS, & Games

Marc Toussaint
University of Stuttgart
Winter 2018/19

Motivation:

The first lecture was about tree search (a form of sequential decision making), the second about probabilities. If we combine this we get Monte-Carlo Tree Search (MCTS), which is the focus of this lecture.

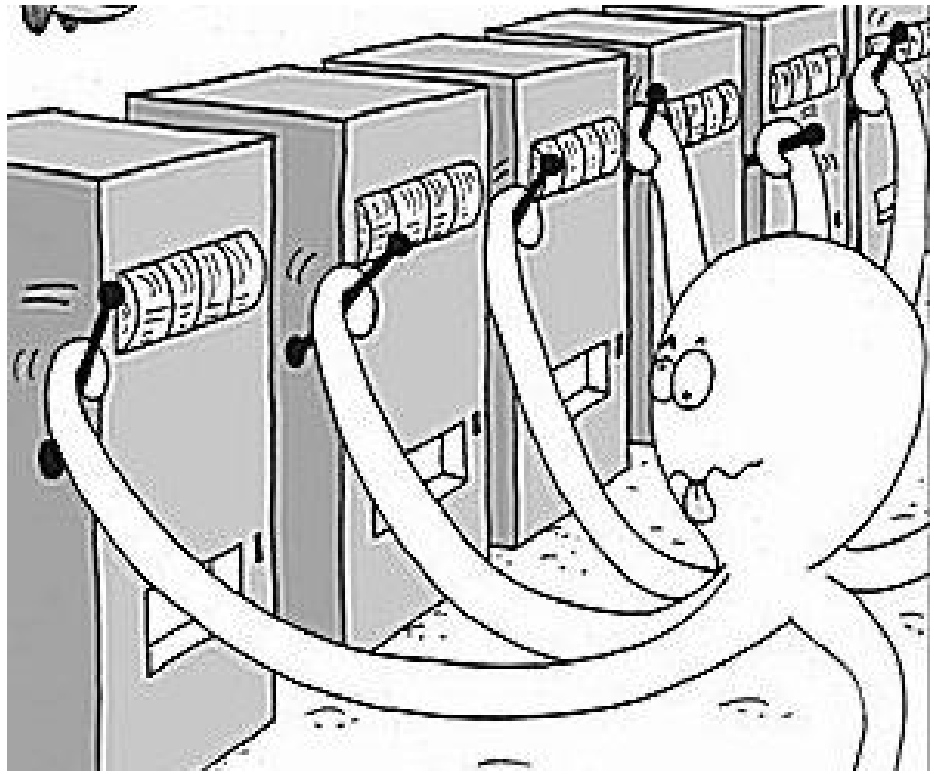
But before discussing MCTS we introduce an important conceptual problem: Multi-armed bandits. This problem setting is THE prototype for so-called exploration-exploitation problems. More precisely, for problems where sequential decisions influence both, the state of knowledge of the agent as well as the states/rewards the agent gets. Therefore there is some tradeoff between choosing decisions for the sake of learning (influencing the state of knowledge in a positive way) versus for the sake of rewards—while clearly, learning might also enable you to better collect rewards later. Bandits are a kind of minimalistic problem setting of this kind, and the methods and algorithms developed for Bandits translate to other exploration-exploitation kind of problems within Reinforcement Learning, Machine Learning, and optimization.

Interestingly, our first application of Bandit ideas and methods is tree search: Performing tree search is also a sequential decision problem, and initially the 'agent' (=tree search algorithm) has a lack of knowledge of where the optimum in the tree is. This sequential decision problem under uncertain knowledge is also an exploitation-exploration problem. Applying the Bandit methods we get state-of-the-art MCTS methods, which nowadays can solve problems like computer Go.

We first introduce bandits, the MCTS, and mention MCTS for POMDPs. We then introduce 2-player games and how to apply MCTS in this case.

Bandits

Multi-armed Bandits



- There are n machines
- Each machine i returns a reward $y \sim P(y; \theta_i)$
  The machine's parameter θ_i is unknown
- Your goal is to maximize the reward, say, collected over the first T trials

Bandits – applications

- Online advertisement

- Clinical trials, robotic scientist

- Efficient optimization

Bandits

- The bandit problem is an archetype for
  - Sequential decision making
  - Decisions that influence knowledge as well as rewards/states
  - Exploration/exploitation
- The same aspects are inherent also in global optimization, active learning & RL

- The Bandit problem formulation is the basis of UCB – which is the core of serveral planning and decision making methods
- Bandit problems are commercially very relevant

Upper Confidence Bounds (UCB)

Bandits: Formal Problem Definition

- Let $a_t \in \{1, .., n\}$ be the choice of machine at time t
  Let $y_t \in \mathbb{R}$ be the outcome

- A policy or strategy maps all the history to a new choice:

$$\pi : \ [(a_1, y_1), (a_2, y_2), ..., (a_{t\text{-}1}, y_{t\text{-}1})] \mapsto a_t$$

- Problem: Find a policy π that

$$\max \langle \sum_{t=1}^{T} y_t \rangle$$

  or

$$\max \langle y_T \rangle$$

  or other objectives like discounted infinite horizon $\max \langle \sum_{t=1}^{\infty} \gamma^t y_t \rangle$

Exploration, Exploitation

- "Two effects" of choosing a machine:
  - You collect more data about the machine $\rightarrow$ knowledge
  - You collect reward

- For example
  - Exploration: Choose the next action a_t to $\min\langle H(b_t)\rangle$
  - Exploitation: Choose the next action a_t to $\max\langle y_t\rangle$

Upper Confidence Bound (UCB1)

1: Initialization: Play each machine once
2: **repeat**
3: Play the machine i that maximizes $\hat{y}_i + \beta\sqrt{\frac{2\ln n}{n_i}}$
4: **until**

- $\hat{y}_i$ is the average reward of machine i so far
- n_i is how often machine i has been played so far
- $n = \sum_i n_i$ is the number of rounds so far
- β is often chosen as $\beta = 1$

- The bound is derived from the Hoeffding inequality

See *Finite-time analysis of the multiarmed bandit problem*, Auer, Cesa-Bianchi & Fischer, Machine learning, 2002.

UCB algorithms

- UCB algorithms determine a **confidence interval** such that

$$\hat{y}_i - \sigma_i < \langle y_i \rangle < \hat{y}_i + \sigma_i$$

  with high probability.
  UCB chooses the upper bound of this confidence interval

- *Optimism in the face of uncertainty*

- Strong bounds on the regret (sub-optimality) of UCB1 (e.g. Auer et al.)

UCB for Bernoulli**

- If we have a single Bernoulli bandits, we can count

$$a = 1 + \text{\#wins}\ , \quad b = 1 + \text{\#losses}$$

- Our posterior over the Bernoulli parameter μ is $\mathrm{Beta}(\mu \,|\, a, b)$
- The mean is $\langle \mu \rangle = \frac{a}{a+b}$
  The mode (most likely) is $\mu^* = \frac{a-1}{a+b-2}$ for $a, b > 1$
  The variance is $\mathrm{Var}\{\mu\} = \frac{ab}{(a+b+1)(a+b)^2}$
  One can numerically compute the *inverse cumulative Beta distribution*
  $\rightarrow$ get exact quantiles

- Alternative strategies:

$$\operatorname*{argmax}_i \text{90\%-quantile}(\mu_i)$$

$$\operatorname*{argmax}_i \langle \mu_i \rangle + \beta \sqrt{\mathrm{Var}\{\mu_i\}}$$

UCB for Gauss**

- If we have a single Gaussian bandits, we can compute
  the mean estimator $\hat{\mu} = \frac{1}{n}\sum_i y_i$
  the empirical variance $\hat{\sigma}^2 = \frac{1}{n-1}\sum_i (y_i - \hat{\mu})^2$
  and the estimated variance *of the mean estimator* $\text{Var}\{\hat{\mu}\} = \hat{\sigma}^2/n$

- $\hat{\mu}$ and $\text{Var}\{\hat{\mu}\}$ describe our posterior Gaussian belief over the true underlying μ
  Using the err-function we can get exact quantiles

- Alternative strategies:

$$\text{90\%-quantile}(\mu_i)$$

$$\hat{\mu}_i + \beta\sqrt{\text{Var}\{\mu_i\}} = \hat{\mu}_i + \beta\hat{\sigma}/\sqrt{n}$$

UCB - Discussion

- UCB over-estimates the reward-to-go (under-estimates cost-to-go), just like A^* – but does so in the probabilistic setting of bandits

- The fact that regret bounds exist is great!

- UCB became a core method for algorithms (including planners) to decide what to explore:

  In tree search, the decision of which branches/actions to explore is itself a decision problem. An "intelligent agent" (like UBC) can be used within the planner to make decisions about how to grow the tree.

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS)

- MCTS is very successful on Computer Go and other games
- MCTS is rather simple to implement
- MCTS is very general: applicable on any discrete domain

- Key paper:
  Kocsis & Szepesvári: *Bandit based Monte-Carlo Planning*, ECML 2006.
- Survey paper:
  Browne et al.: *A Survey of Monte Carlo Tree Search Methods*, 2012.
- Tutorial presentation:
  `http://web.engr.oregonstate.edu/~afern/icaps10-MCP-tutorial.ppt`

Monte Carlo methods

- General, the term *Monte Carlo simulation* refers to methods that generate many i.i.d. random samples $x_i \sim P(x)$ from a distribution $P(x)$. Using the samples one can estimate expectations of anything that depends on x, e.g. $f(x)$:

$$\langle f \rangle \;=\; \int_x P(x)\; f(x)\; dx \;\approx\; \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

  (In this view, Monte Carlo approximates an integral.)

- Example: What is the probability that a solitair would come out successful? (Original story by Stan Ulam.) Instead of trying to analytically compute this, generate many random solitairs and count.

- The method developed in the 40ies, where computers became faster. Fermi, Ulam and von Neumann initiated the idea. von Neumann called it “Monte Carlo” as a code name.

Flat Monte Carlo

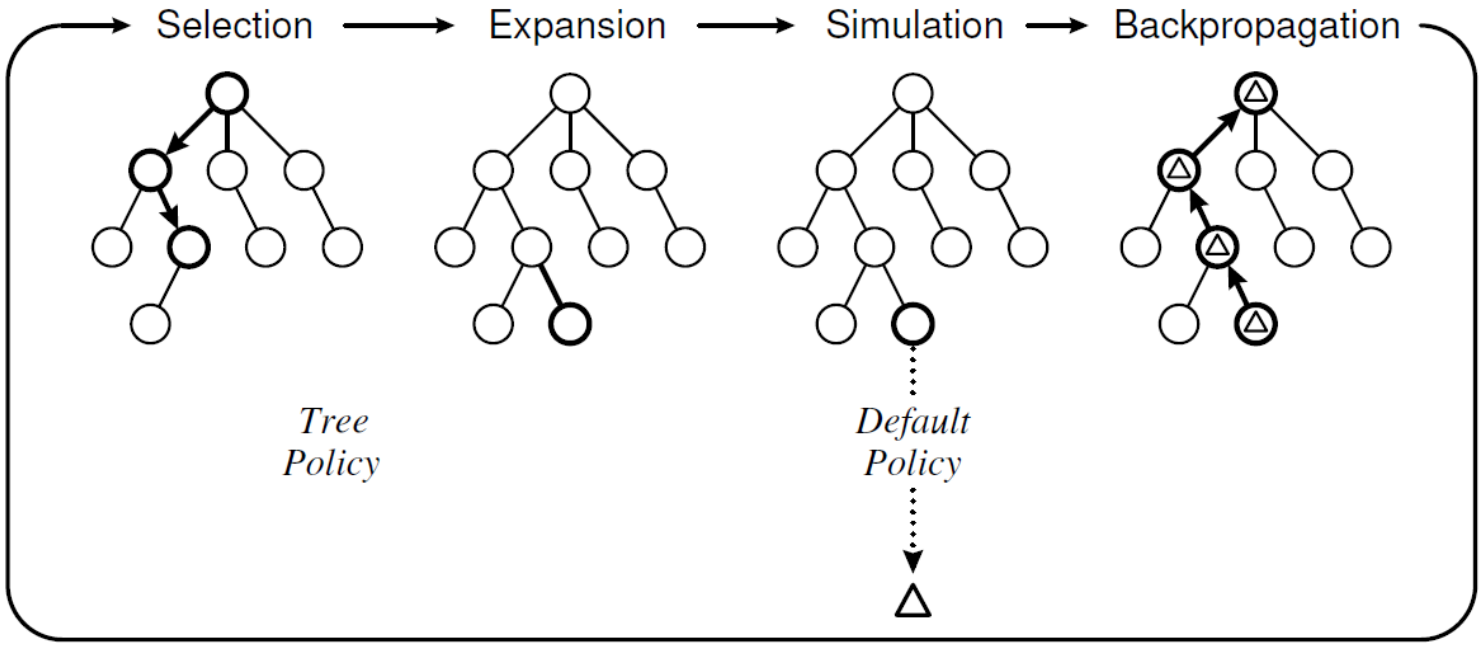
- The goal of MCTS is to estimate the utility (e.g., expected payoff Δ) depending on the action a chosen—the **Q-function**:

$$Q(s_0, a) = \mathsf{E}\{\Delta | s_0, a\}$$

  where expectation is taken with w.r.t. the whole future randomized actions (including a potential opponent)

- *Flat Monte Carlo* does so by rolling out many random simulations (using a RolloutPolicy) without growing a tree
  The key difference/advantage of MCTS over flat MC is that the tree growth focusses computational effort on promising actions

Generic MCTS scheme



from Browne et al.

1: start tree $V = \{v_0\}$
2: **while** within computational budget **do**
3: $v_l \leftarrow$ TREEPOLICY(V) chooses and creates a new leaf of V
4: append v_l to V
5: $\Delta \leftarrow$ ROLLOUTPOLICY(V) rolls out a full simulation, with return Δ
6: BACKUP(v_l, Δ) updates the values of all parents of v_l
7: **end while**
8: return best child of v_0

Generic MCTS scheme

- Like FlatMC, MCTS typically computes full roll outs to a terminal state. A heuristic (evaluation function) to estimate the utility of a state is not needed, but can be incorporated.
- The tree grows unbalanced
- The TREEPOLICY decides where the tree is expanded – and needs to trade off exploration vs. exploitation
- The ROLLOUTPOLICY is necessary to simulate a roll out. It typically is a random policy; at least a randomized policy.

Upper Confidence Tree (UCT)

- UCT uses UCB to realize the TreePolicy, i.e. to decide where to expand the tree

- Backup updates all parents of v_l as
  $n(v) \leftarrow n(v) + 1$ (count how often has it been played)
  $Q(v) \leftarrow Q(v) + \Delta$ (sum of rewards received)

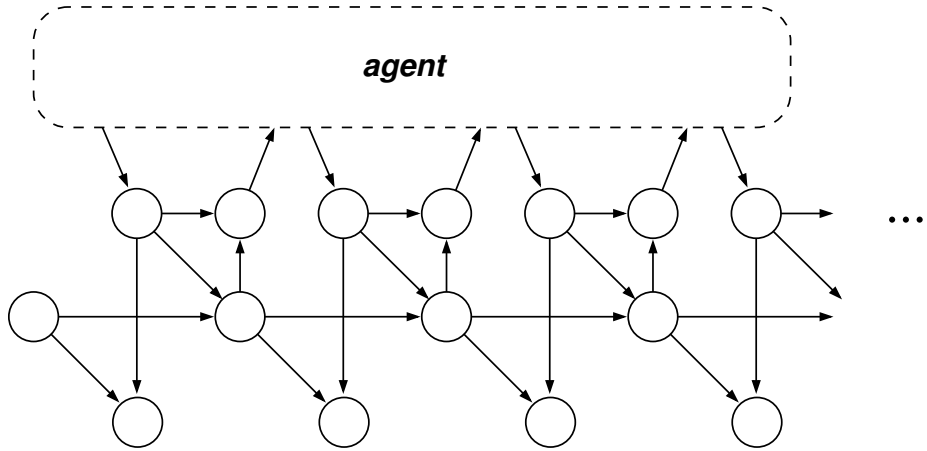
- TreePolicy chooses child nodes based on UCB:

$$\operatorname*{argmax}_{v' \in \partial(v)} \frac{Q(v')}{n(v')} + \beta \sqrt{\frac{2 \ln n(v)}{n(v')}}$$

  or choose v' if $n(v') = 0$

MCTS applied to POMDPs**

Recall POMDPs



- initial state distribution $P(s_0)$
- transition probabilities $P(s'|s,a)$
- observation probabilities $P(y'|s',a)$
- reward probabilities $P(r|s,a)$

- An optimal agent maps the history to an action, $(y_{0:t}, a_{0:t\text{-}1}) \mapsto a_t$

Issues when applying MCTS ideas to POMDPs

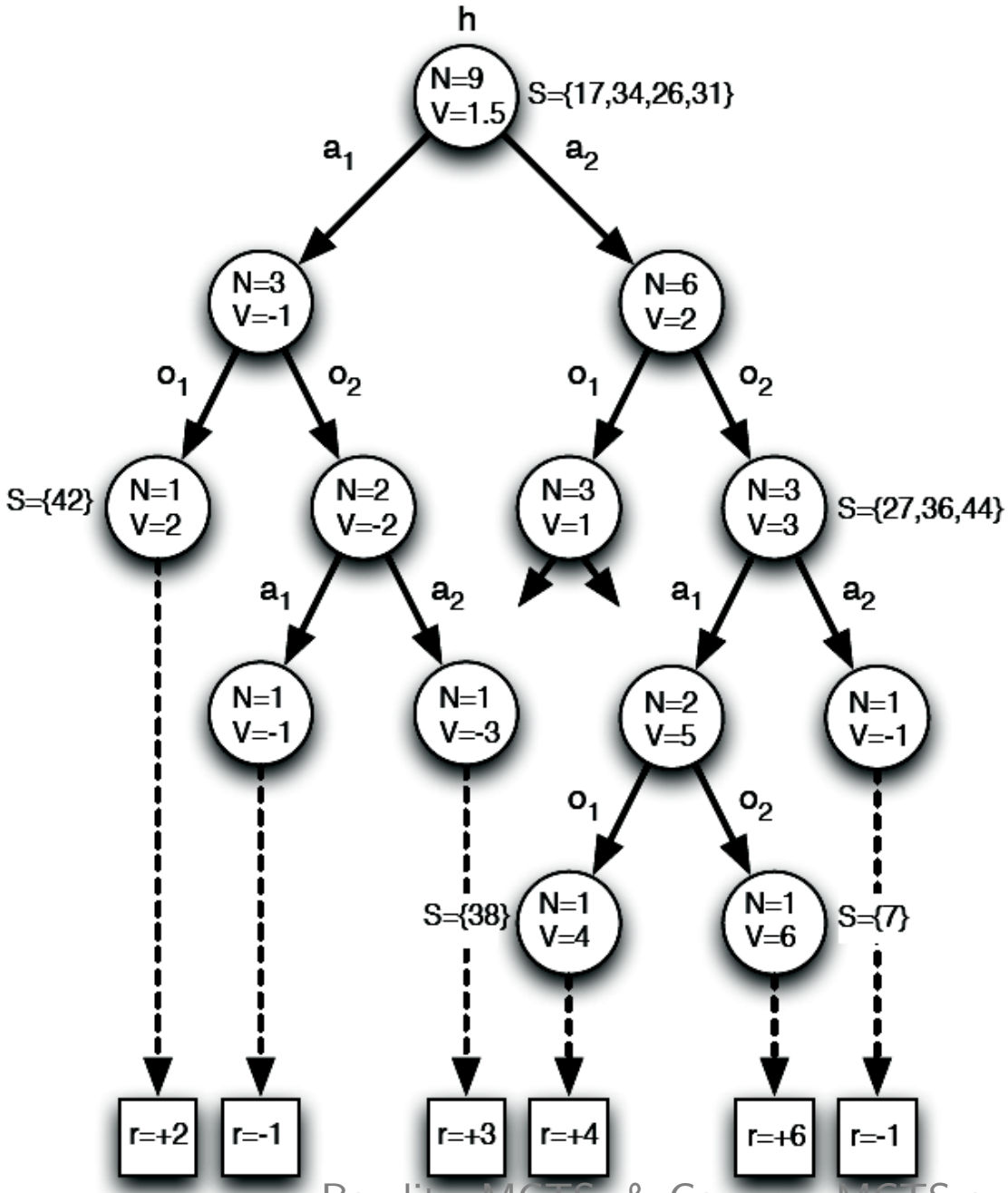
- key paper:
  Silver & Veness: *Monte-Carlo Planning in Large POMDPs*, NIPS 2010

- MCTS is based on generating rollouts using a simulator
  – Rollouts need to start at a specific *state* s_t
  $\rightarrow$ Nodes in our tree need to have states associated, to start rollouts from

- At any point in time, the agent has only the history $h_t = (y_{0:t}, a_{0:t\text{-}1})$ to decide on an action
  – The agent wants to estimate the Q-funcion $Q(h_t, a_t)$
  $\rightarrow$ Nodes in our tree need to have a history associated

  $\rightarrow$ Nodes in the search tree will
  - maintain $n(v)$ and $Q(v)$ as before
  - have a history $h(v)$ attached
  - have a *set* of states $\mathcal{S}(v)$ attached

MCTS applied to POMDPs



from Silver & Veness

MCTS applied to POMDPs

- For each rollout:
  - Choose a *random* world state $s_0 \sim \mathcal{S}(v_0)$ from the set of states associated to the root v_0; initialize the simulator with this s_0
  - Use a TREEPOLICY to traverse the current tree; during this, update the state sets $\mathcal{S}(v)$ to contain the world state simulated by the simulator
  - Use a ROLLOUTPOLICY to simulate a full rollout
  - Append a new leaf v_l with novel history $h(v_l)$ and a single state $\mathcal{S}(v_l)$ associated

Monte Carlo Tree Search

- MCTS combines forward information (starting simulations from s_0) with backward information (accumulating $Q(v)$ at tree nodes)

- UCT uses an optimistic estimate of return to decide on how to expand the tree – this is the stochastic analogy to the A^* heuristic

table	PDDL	NID	MDP	POMDP	DEC-POMDP	Games	control
y	y	y	y	y	?	y	

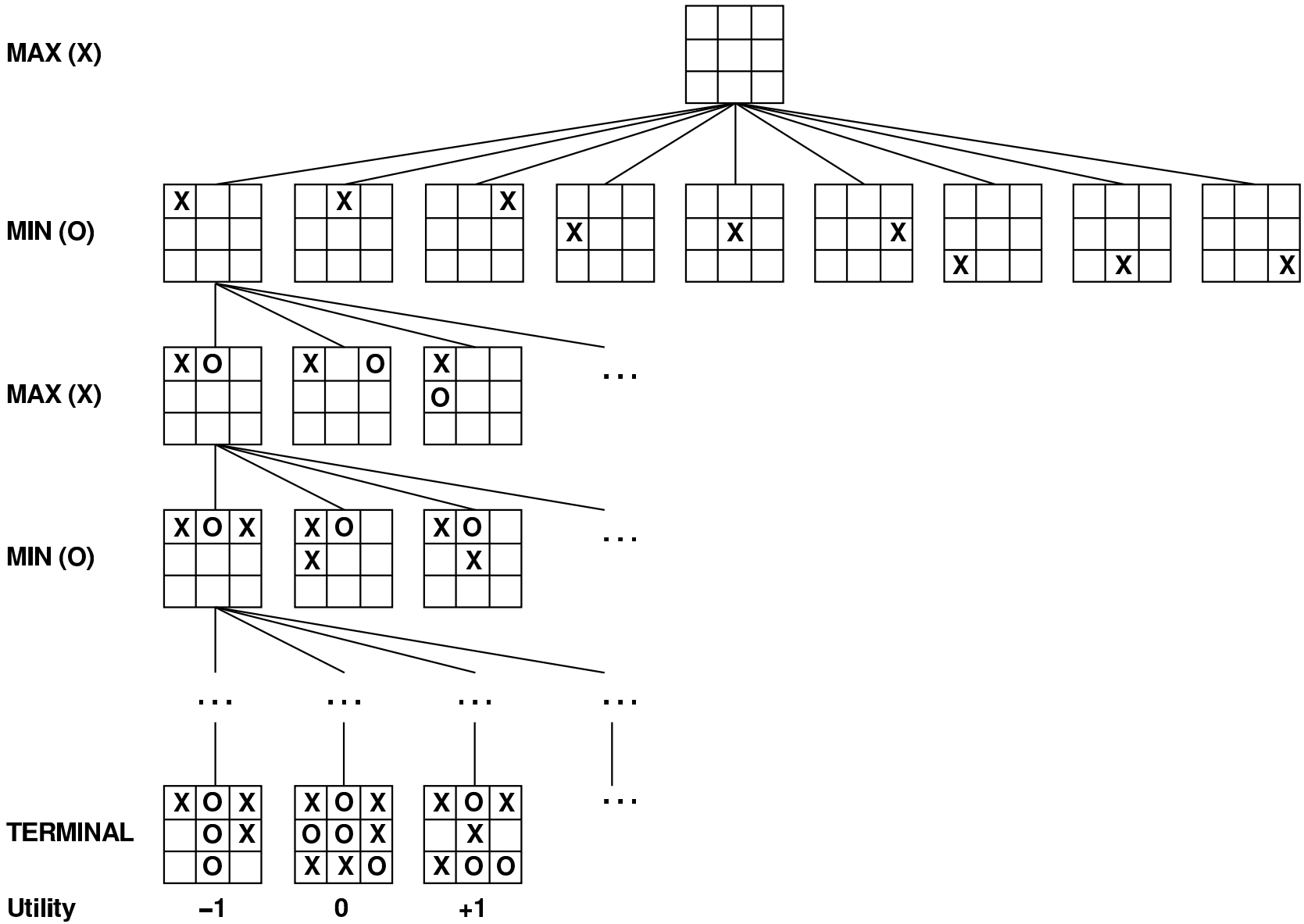
- *Conclusion:* MCTS is a very generic and often powerful planning method. For many many samples it converges to correct estimates of the Q-function. However, the Q-function can be estimated also using other methods.

Game Playing

Outline

- Minimax
- α–β pruning
- UCT for games

Game tree (2-player, deterministic, turns)

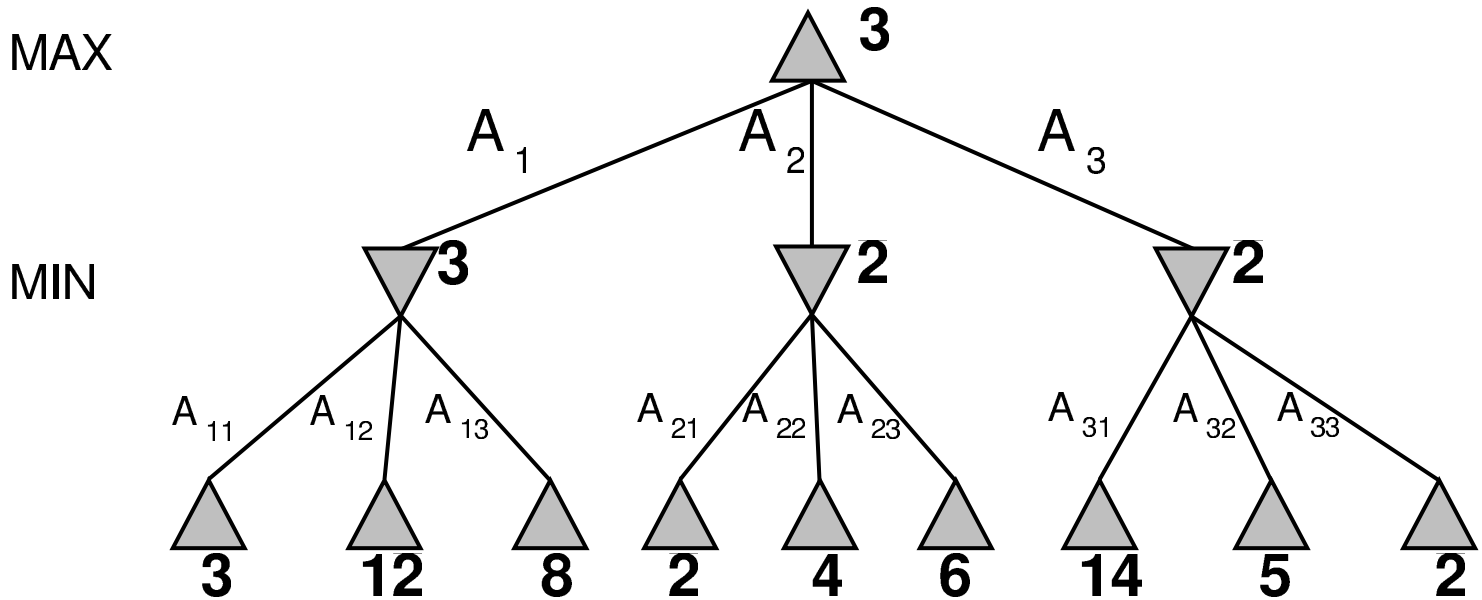


Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

= best achievable payoff against best play



Minimax algorithm

- Computation by direct recursive function calls, which effectively does DFS

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(*s*))
  return *v*

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(*s*))
  return *v*

Properties of minimax

Complete??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Optimal?? Yes, against an optimal opponent. Otherwise??
Time complexity?? $O(b^m)$
Space complexity??

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

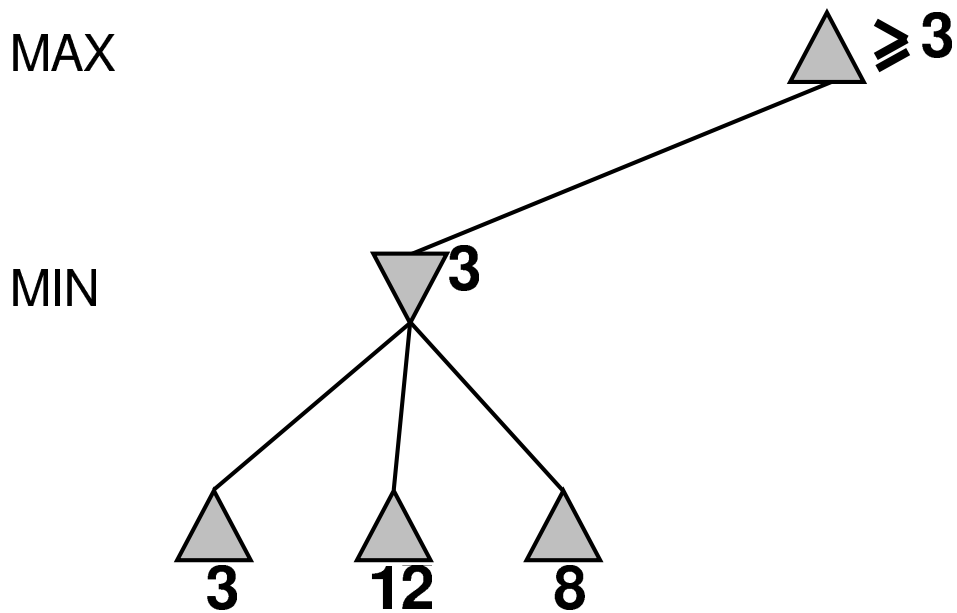
Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games

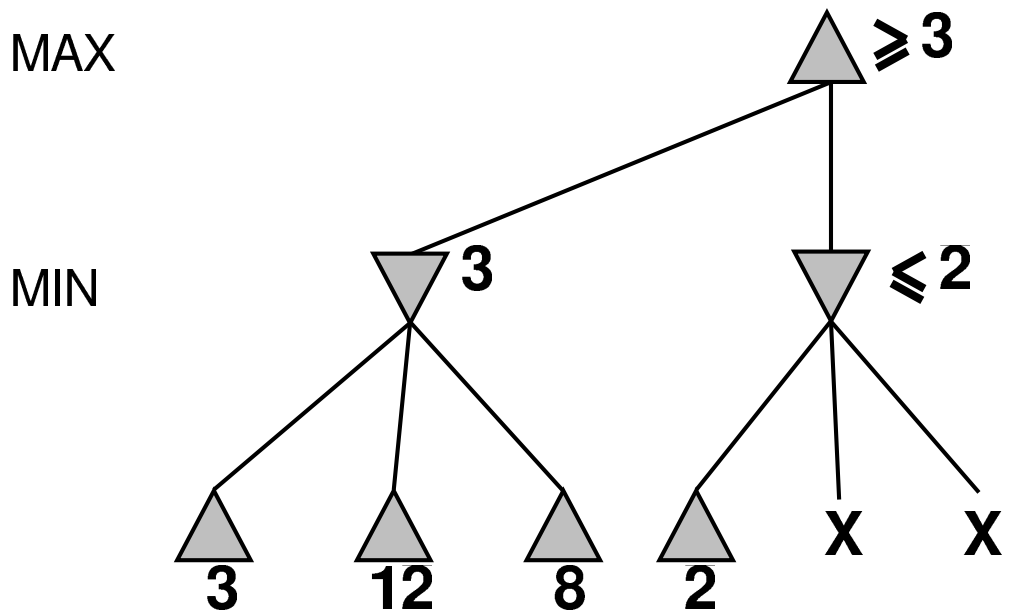
$\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?

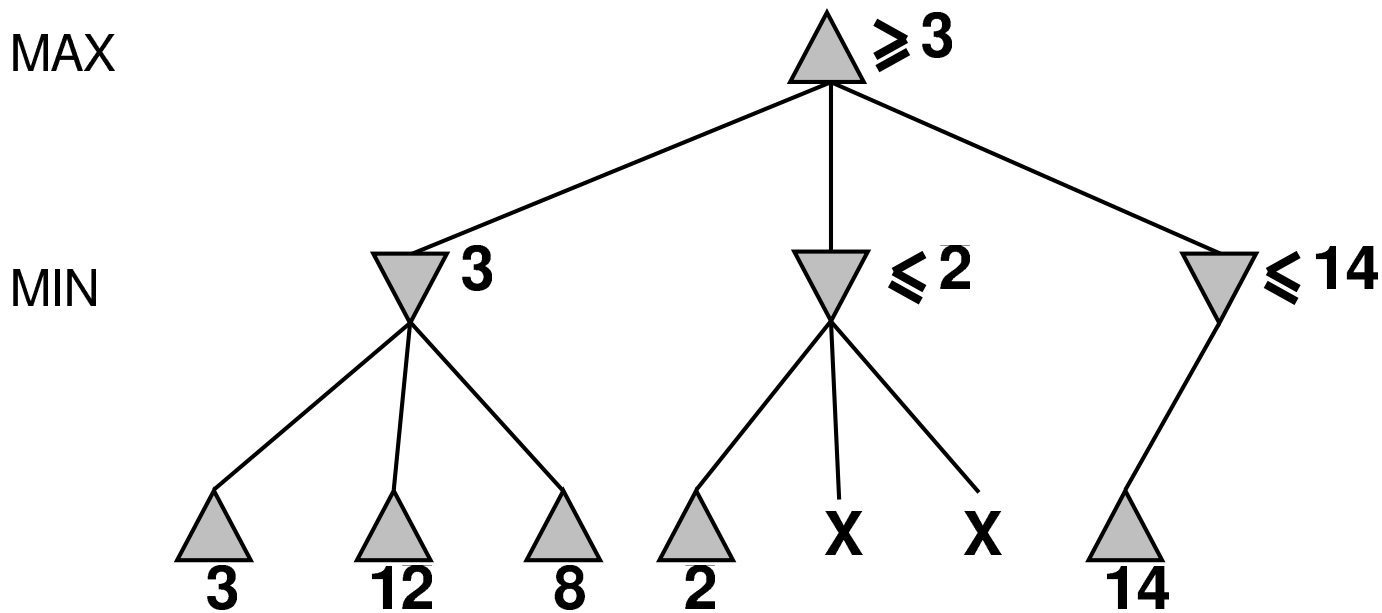
α–β pruning example



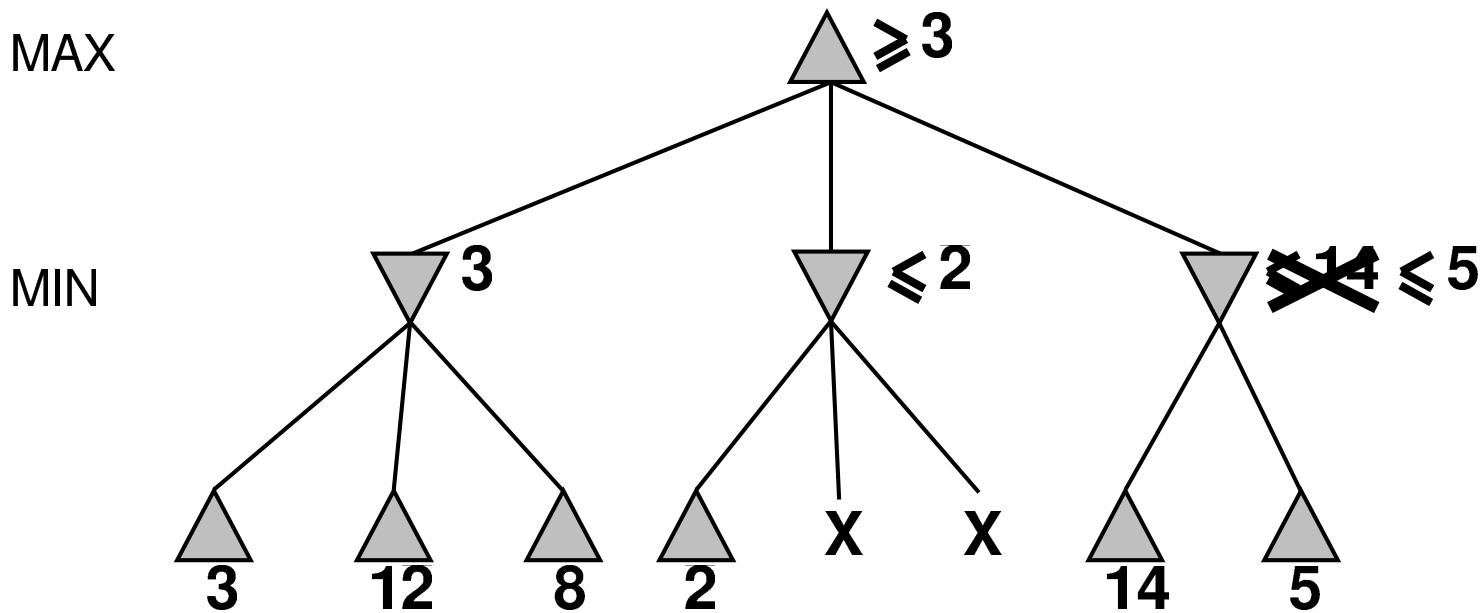
α–β pruning example



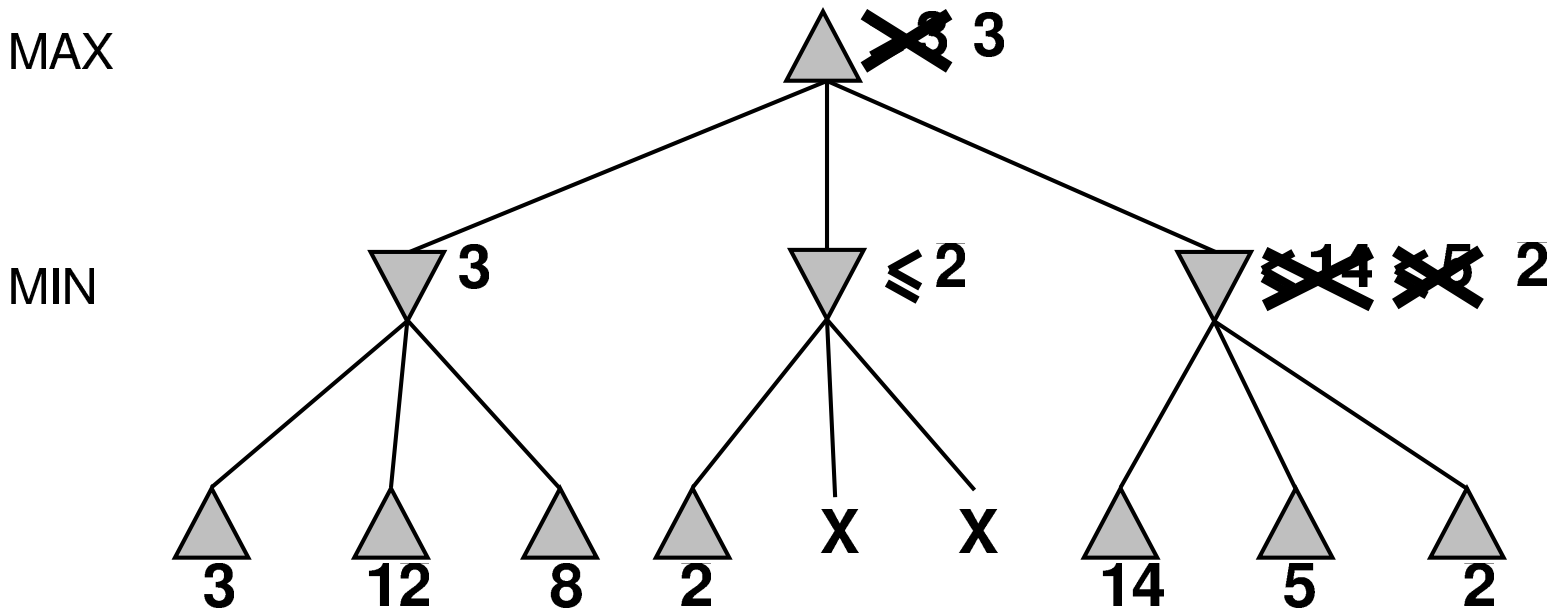
α–β pruning example



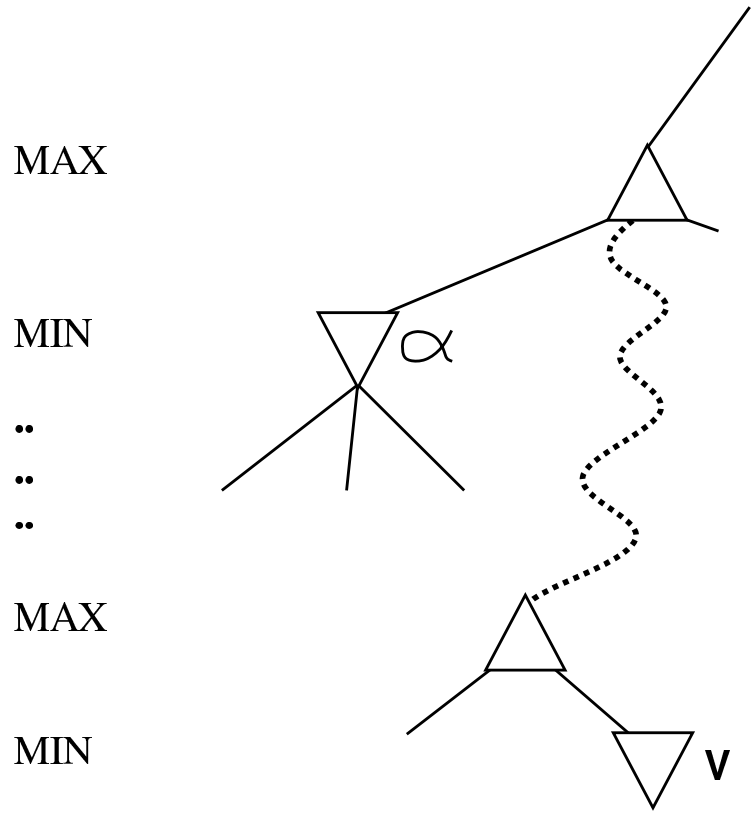
α–β pruning example



α–β pruning example



α–β pruning as instance of branch-and-bound



- α is the best value (to $\textsc{max}$) found so far off the current path
- If V is worse than α, $\textsc{max}$ will avoid it $\Rightarrow$ prune that branch
- Define β similarly for $\textsc{min}$

The α–β algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(*s*, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Properties of α–β

- Pruning *does not* affect final result
- Good move ordering improves effectiveness of pruning!
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Resource limits

Standard approach:

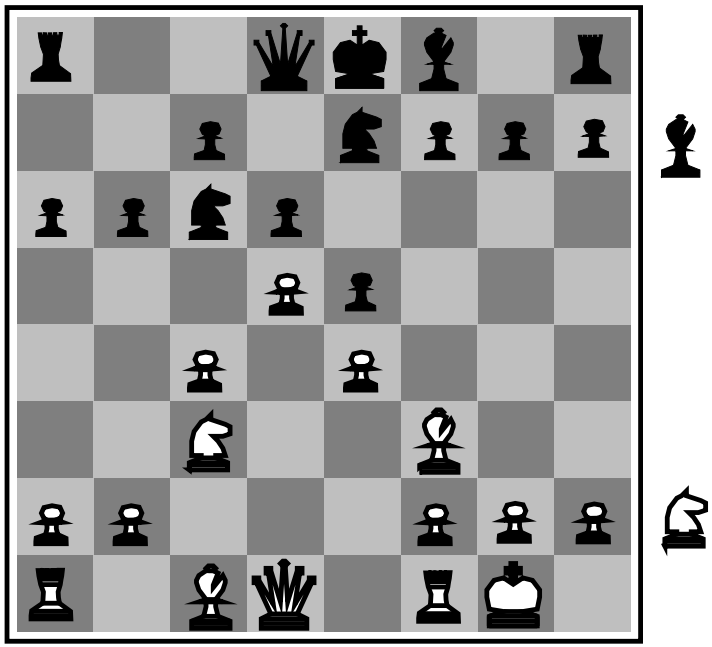
- Use CUTOFF-TEST instead of TERMINAL-TEST
  e.g., depth limit
- Use EVAL instead of UTILITY
  i.e., evaluation function that estimates desirability of position

Suppose we have 100 seconds, explore 10^4 nodes/second

$\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$

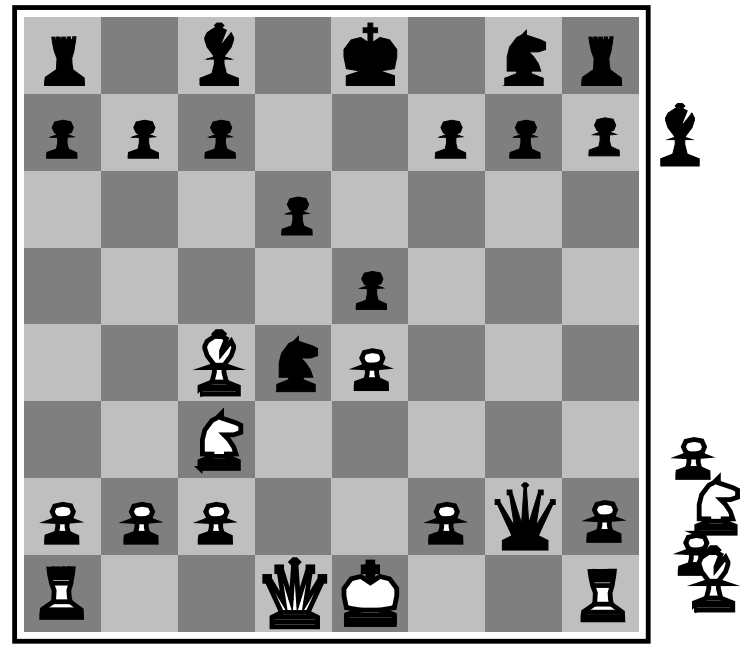
$\Rightarrow \alpha$–β reaches depth 8 $\Rightarrow$ pretty good chess program

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically linear weighted sum of features

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Upper Confidence Tree (UCT) for games

- Standard backup updates all parents of v_l as
  $n(v) \leftarrow n(v) + 1$ (count how often has it been played)
  $Q(v) \leftarrow Q(v) + \Delta$ (sum of rewards received)

- In games use a "negamax" backup: While iterating upward, flip sign $\Delta \leftarrow -\Delta$ in each iteration

- Survey of MCTS applications:
  Browne et al.: A Survey of Monte Carlo Tree Search Methods, 2012.

	Go	Phantom Go	Blind Go	NoGo	Multi-player Go	Hex	Y, Star, Renkula!	Havannah	Lines of Action	P-Game	Clobber	Othello	Amazons	Arimaa	Khet	Shogi	Mancala	Blokus Duo	Focus	Chinese Checkers	Yavalath	Connect Four	Tic Tac Toe	Sum of Switches	Chess	LeftRight Games	Morpion Solitaire	Crossword	SameGame	Sudoku, Kakuro	Wumpus World	Mazes,Tigers,Grids	CADIAPLAYER ARY
Flat MC/UCB	+										+		+																				
BAST																																	
TDMC(λ)												+																					
BB Active Learner																																	
UCT	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+			+	?							+	+
SP-MCTS																												+					
FUSE																																	
MP-MCTS																			+	+													
Coalition Reduction																																	
Multi-agent MCTS	+																																
Ensemble MCTS	+																																
HOP																																	
Sparse UCT																																	
Info Set UCT																																	
Multiple MCTS																							+										
UCT+																																	
$MC_{\alpha\beta}$									+																?								
MCCFR																																	
Reflexive MC																											+						
Nested MC																										+	+		+	+			+
NRPA																											+	+					
HGSTS																											+		+				
FSSS, BFS3																															+		
TAG																																	
UNLEO																																	
UCTSAT																																	
ρUCT																							+									+	
MRW																																	
MHSP																																	
UCB1-Tuned																																	
Bayesian UCT																																	
EXP3																																	
HOOT																																	
First Play Urgency	+																																
(Anti)Decisive Moves	+			+				+													+												
Move Groups	+									+																							
Move Ordering	+											+				+																	
Transpositions	+									+				+	+											+							
Progressive Bias	+																																
Opening Books	+												+																				
MCPG																																	
Search Seeding	+																																
Parameter Tuning	+																																
History Heuristic	+													+		+			+	+												+	
AMAF	+	+						+																		+							
RAVE	+	+	+	+	+	+		+						+										+								+	
Killer RAVE								+																									
RAVE-max																								+									
PoolRAVE	+							+																									
MCTS-Solver									+																								
MC-PNS	+																																
Score Bounded MCTS	+																					+											
Progressive Widening	+												+																				
Pruning	+					+							+									+											
Contextual MC	+							+																									
Fill the Board	+					+	+																										
MAST, PAST, FAST																																+	
Simulation Balancing	+																																
Last Good Reply	+							+																									
Patterns	+					+	+	+																									
Score Bonus																								+									
Decaying Reward										+																							
Leaf Parallelisation	+													+			+															+	
Root Parallelisation	+													+																				
Tree Parallelisation	+																																+
UCT-Treesplit	+																																+

TABLE 3
Summary of MCTS variations and enhancements applied to combinatorial games.

	Tron	Ms. Pac-Man	Pocman, Battleship	Dead-End	Wargus	ORTS	Skat	Bridge	Poker	Dou Di Zhu	Klondike Solitaire	Magic: The Gathering	Phantom Chess	Urban Rivals	Backgammon	Settlers of Catan	Scotland Yard	Roshambo	Thurn and Taxis	OnTop	Security	Mixed Integer Prog.	TSP, CTP	Sailing Domain	Physics Simulations	Function Approx.	Constraint Satisfaction	Schedul. Benchmarks	Printer Scheduling	Rock-Sample Problem	PMPs	Bus Regulation	Large State Spaces	Feature Selection	PCG
Flat MC/UCB		+	+	+			+		+			+														+									
BAST																										+									
TDMC(λ)																																			
BB Active Learner																																			
UCT	+	+			+	+			+		+		+	+	+				+	+	+	+	+	+	+		+	+		+		+			+
SP-MCTS																														+					
FUSE																																	+		
MP-MCTS		+														+																			
Coalition Reduction																	+																		
Multi-agent MCTS																																			
Ensemble MCTS																																			
HOP											+																								
Sparse UCT													+																						
Info Set UCT							+			+																									
Multiple MCTS																																			
UCT+									+																										
$MC_{\alpha\beta}$																																			
MCCFR									+																										
Reflexive MC																																			
Nested MC																						+			+						+				
NRPA																																			
HGSTS																																			
FSSS, BFS3																																	+		
TAG																									+										
UNLEO																																			
UCTSAT																										+									
ρUCT			+						+									+																	
MRW																									+										
MHSP																											+								
UCB1-Tuned	+																																		
Bayesian UCT																																			
EXP3													+																						
HOOT																									+										
First Play Urgency																																			
(Anti)Decisive Moves																																			
Move Groups																																			
Move Ordering																																			
Transpositions																																			
Progressive Bias	+																																		
Opening Books																																			
MCPG																																			+
Search Seeding																																			
Parameter Tuning																																			
History Heuristic																																			
AMAF													+																						
RAVE													+																						
Killer RAVE																																			
RAVE-max																																			
PoolRAVE																																			
MCTS-Solver	+																																		
MC-PNS																																			
Score Bounded MCTS																																			
Progressive Widening																														+					
Pruning				+																															
Contextual MC																																			
Fill the Board																																			
MAST, PAST, FAST																																			
Simulation Balancing																																			
Last Good Reply																																			
Patterns																																			
Score Bonus																																			
Decaying Reward																							+												
Leaf Parallelisation																																			
Root Parallelisation																																			
Tree Parallelisation																																			
UCT-Treesplit																																			

TABLE 4
Summary of MCTS variations and enhancements applied to other domains.

Brief notes on game theory

- Zero-sum games can be represented by a payoff matrix
- U_{ji} denotes the utility of player 1 if she chooses the *pure* (=deterministic) strategy i and player 2 chooses the pure strategy j.

  Zero-sum games: $U_{ji} = -U_{ij}\ , \quad U^T = -U$
- Fining a minimax optimal *mixed strategy* p is a Linear Program

$$\max_w w \quad \text{s.t.} \quad Up \geq w\ , \quad \sum_i p_i = 1\ , \quad p \geq 0$$

  Note that $Up \geq w$ implies $\min_j (Up)_j \geq w$.
- Gainable payoff of player 1: $\max_p \min_q q^T U p$

  Minimax-Theorem: $\max_p \min_q q^T U p = \min_q \max_p q^T U p$

  Minimax-Theorem $\leftrightarrow$ optimal p with $w \geq 0$ exists

Beyond bandits**

- Perhaps have a look at the tutorial: *Bandits, Global Optimization, Active Learning, and Bayesian RL – understanding the common ground*

Global Optimization

- Let $x \in \mathbb{R}^n$, $f:\ \mathbb{R}^n \to \mathbb{R}$, find

$$\min_x\ f(x)$$

(I neglect constraints $g(x) \leq 0$ and $h(x) = 0$ here – but could be included.)

- Blackbox optimization: find optimium by sampling values $y_t = f(x_t)$
  No access to ∇f or $\nabla^2 f$
  Observations may be noisy $y \sim \mathcal{N}(y \,|\, f(x_t), \sigma)$

Global Optimization $=$ infinite bandits

- In global optimization $f(x)$ defines a reward for every $x \in \mathbb{R}^n$
  – Instead of a finite number of actions a_t we now have x_t

- The unknown "world property" is the function $\theta = f$
- Optimal Optimization could be defined as: find $\pi : \ h_t \mapsto x_t$ that

$$\min \langle \sum_{t=1}^{T} f(x_t) \rangle$$

or

$$\min \langle f(x_T) \rangle$$

Gaussian Processes as belief

- If all the infinite bandits would be uncorrelated, there would be no chance to solve the problem $\rightarrow$ *No Free Lunch Theorem*
- One typically assumes that nearby function values $f(x), f(x')$ are correlated as described by a covariance function $k(x, x') \rightarrow$ Gaussian Processes

Greedy 1-step heuristics

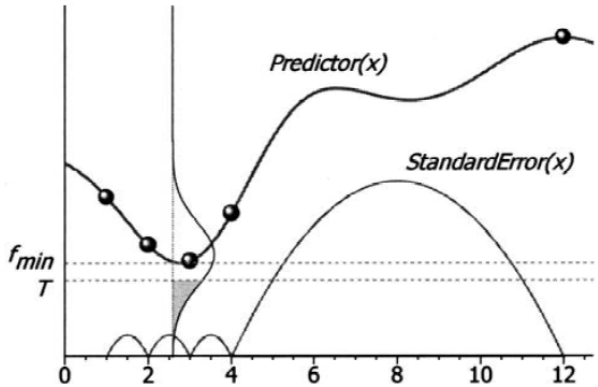


Figure 14. Using kriging, we can estimate the probability that sampling at a given point will 'improve' our solution, in the sense of yielding a value that is equal or better than some target T.

from Jones (2001)

- Maximize Probability of Improvement (MPI)

$$x_t = \operatorname*{argmax}_x \int_{-\infty}^{y^*} \mathcal{N}(y|\hat{f}(x), \hat{\sigma}(x))$$

- Maximize Expected Improvement (EI)

$$x_t = \operatorname*{argmax}_x \int_{-\infty}^{y^*} \mathcal{N}(y|\hat{f}(x), \hat{\sigma}(x))\ (y^* - y)$$

- Maximize UCB

$$x_t = \operatorname*{argmin}_x \hat{f}(x) - \beta_t \hat{\sigma}(x)$$

(Often, $\beta_t = 1$ is chosen. UCB theory allows for better choices. See Srinivas et al. citation below.)

From Srinivas et al., 2012:

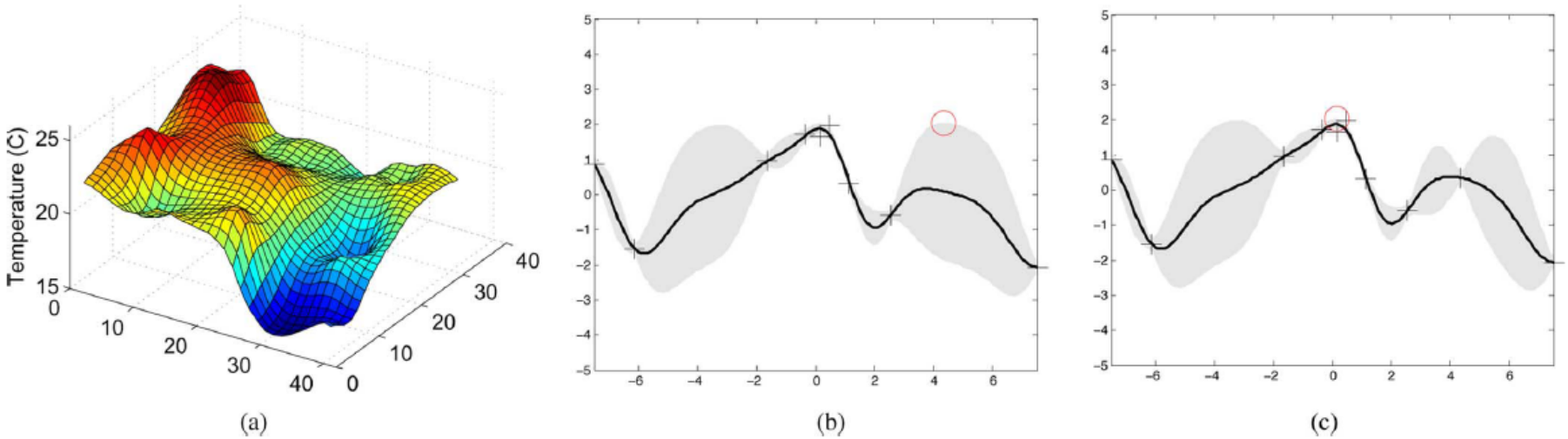


Fig. 2. (a) Example of temperature data collected by a network of 46 sensors at Intel Research Berkeley. (b) and (c) Two iterations of the GP-UCB algorithm. The dark curve indicates the current posterior mean, while the gray bands represent the upper and lower confidence bounds which contain the function with high probability. The "+" mark indicates points that have been sampled before, while the "o" mark shows the point chosen by the GP-UCB algorithm to sample next. It samples points that are either (b) uncertain or have (c) high posterior mean.

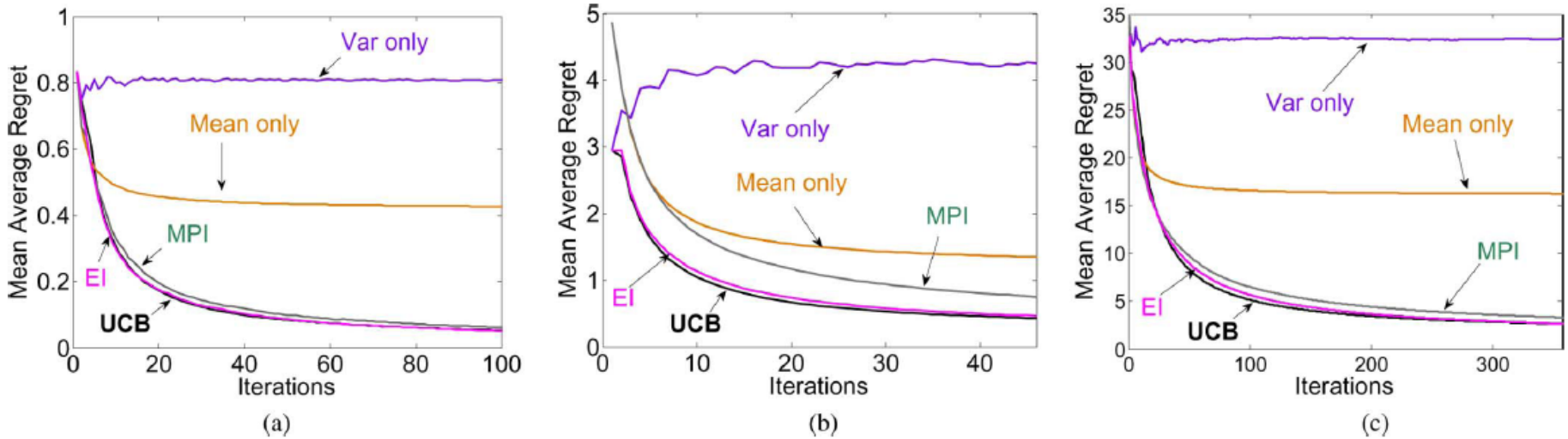


Fig. 6. Mean average regret: GP-UCB and various heuristics on (a) synthetic and (b, c) sensor network data.

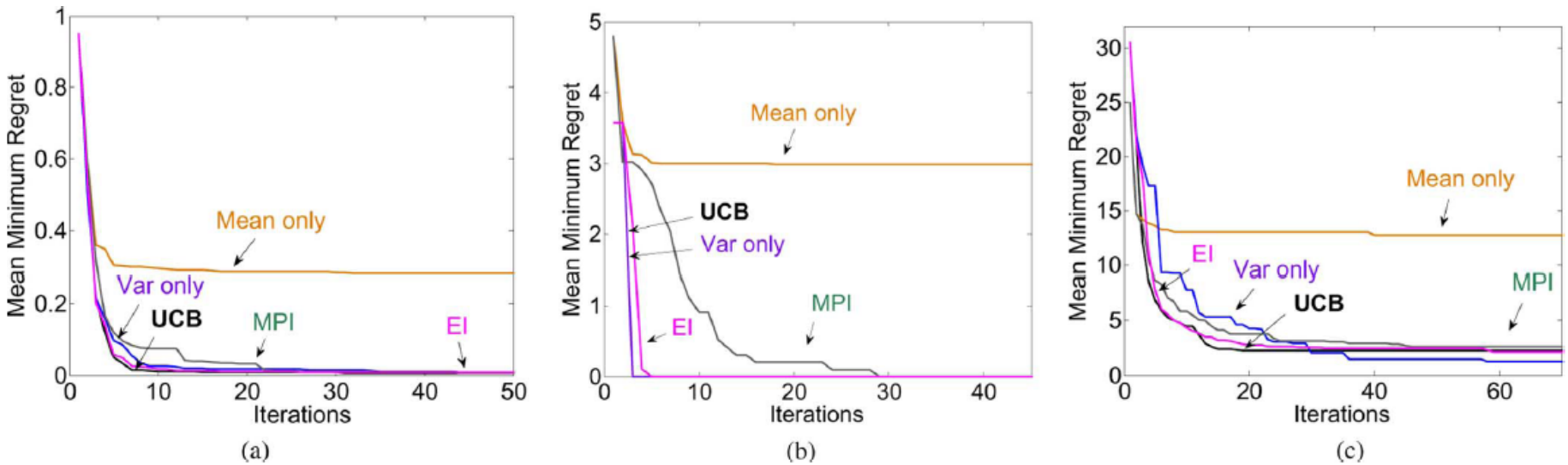


Fig. 7. Mean minimum regret: GP-UCB and various heuristics on (a) synthetic, and (b, c) sensor network data.

Further reading

- Classically, such methods are known as *Kriging*

- *Information-theoretic regret bounds for gaussian process optimization in the bandit setting* Srinivas, Krause, Kakade & Seeger, Information Theory, 2012.

- *Efficient global optimization of expensive black-box functions.* Jones, Schonlau, & Welch, Journal of Global Optimization, 1998.
- *A taxonomy of global optimization methods based on response surfaces* Jones, Journal of Global Optimization, 2001.
- *Explicit local models: Towards optimal optimization algorithms*, Poland, Technical Report No. IDSIA-09-04, 2004.

Active Learning**

Active Learning

- In standard ML, a data set $D_t = \{(x_s, y_s)\}_{s=1}^{t\text{-}1}$ is given.
  In active learning, the learning agent sequentially decides on each x_t – where to collect data
- Generally, the aim of the learner should be to learn as fast as possible, e.g. minimize predictive error

- Again, the unknown "world property" is the function $\theta = f$
- Finite horizon T predictive error problem:
  Given $P(x^*)$, find a policy $\pi : \ D_t \mapsto x_t$ that

$$\min \langle -\log P(y^*|x^*, D_T) \rangle_{y^*, x^*, D_T; \pi}$$

  This also can be expressed as *predictive entropy*:

$$\begin{aligned} \langle -\log P(y^*|x^*, D_T) \rangle_{y^*, x^*} &= \langle -\textstyle\int_{y^*} P(y^*|x^*, D_T) \ \log P(y^*|x^*, D_T) \rangle_{x^*} \\ &= \langle H(y^*|x^*, D_T) \rangle_{x^*} =: H(f|D_T) \end{aligned}$$

- Find a policy that $\min \mathsf{E}\{D_T; \pi\} H(f|D_T)$

Greedy 1-step heuristic

- The simplest greedy policy is 1-step Dynamic Programming:
  Directly maximize immediate expected reward, i.e., minimizes $H(b_{t+1})$.

$$\pi : \ b_t(f) \mapsto \operatorname*{argmin}_{x_t} \int_{y_t} P(y_t|x_t, b_t) \ H(b_t[x_t, y_t])$$

- For GPs, you reduce the entropy most if you choose x_t where the current predictive variance is highest:

$$\text{Var}(f(x)) = k(x,x) - \boldsymbol{\kappa}(x)(\boldsymbol{K} + \sigma^2 \mathbf{I}_n)^{-1} \boldsymbol{\kappa}(x)$$

  This is referred to as *uncertainty sampling*

- Note, if we fix hyperparameters:
  - This variance is *independent* of the observations y_t, only the set D_t matters!
  - The order of data points also does not matter
  - You can pre-optimize a set of "grid-points" for the kernel – and play them in any order

Further reading

- *Active learning literature survey.* Settles, Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2009.
- *Bayesian experimental design: A review.* Chaloner & Verdinelli, Statistical Science, 1995.
- *Active learning with statistical models.* Cohn, Ghahramani & Jordan, JAIR 1996.
- ICML 2009 Tutorial on *Active Learning*, Sanjoy Dasgupta and John Langford `http://hunch.net/~active_learning/`