MDP Algorithms

Part II

Thomas Keller

University of Basel

June 20, 2018

Outline of this lecture

- Markov decision processes
- Planning via determinization
- Monte-Carlo methods
- Monte-Carlo Tree Search
- Heuristic Search
- Trial-based Heuristic Tree Search

Please ask questions at any time!

Reminder: First Part

- presented several algorithms for probabilistic planning:
  - Determinizations
  - Hindsight Optimization
  - Policy Sampling
  - Sparse Sampling
- there are applications for all where they perform well
- but all are suboptimal even if provided with unlimited resources

Foreword

- there is a lot of research in MCTS $\Rightarrow$ I can only cover the basics
- definition of MCTS controversial in the literature
- I only consider algorithms as MCTS that incorporate concepts of Monte-Carlo and Tree Search

Browne et al.
A Survey of Monte Carlo Tree Search Methods
2012

Idea

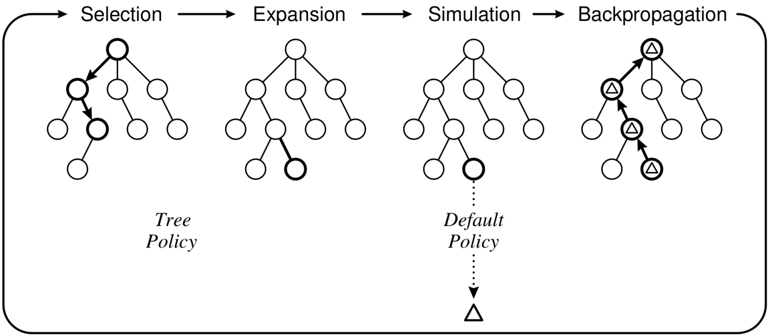
- perform iterations as long as resources (deliberation time, memory) allow:
  - build a partial MDP tree, where nodes n are annotated with
    - average reward $\hat{Q}^{MCTS}k(n.\text{state}, a)$ for action a after k iterations
    - visit counter $N_k(n)$ and $N_k(n, a)$ for action a after k iterations
  - initially, the tree contains only the root node
  - each iteration adds one node to the tree
- execute the action with the highest reward estimate

Iterations

Each iteration consists of (up to) four phases:

- selection: traverse the tree by sampling the execution of the tree policy until
  - a terminal node is reached (skip next two phases and proceed with $r = 0$ in this case), or
  - a state s is reached that is not yet in the tree
- expansion: add a node for s to the tree
- simulation: from s, apply default policy until a terminal state is reached, yielding reward r
- backpropagation: is a has been selected in n, update n:
  - increase visit counter $N_k(n)$ and $N_k(n,a)$
  - set $r \leftarrow r + R(n.\text{state}, a)$
  - update current average reward $\hat{Q}_k^{MCTS}(n,a)$ based on r

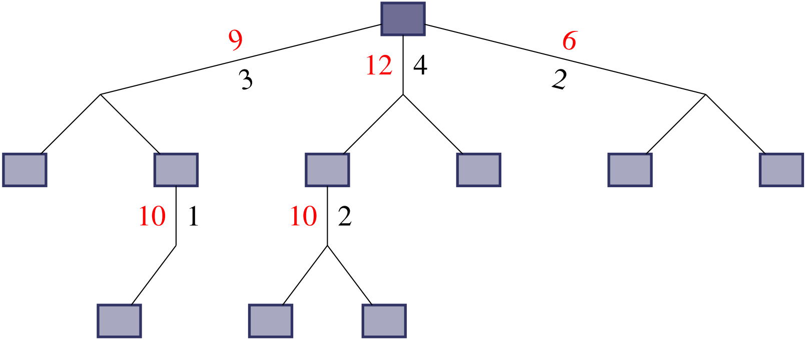
Iterations



Taken from Browne et al., “A Survey of Monte Carlo Tree Search Methods”, 2012

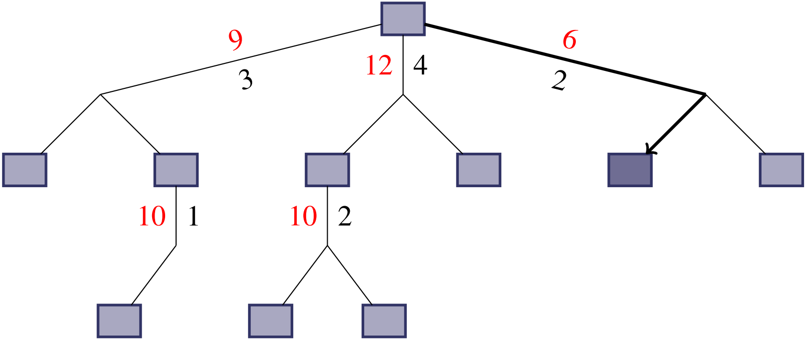
Selection Phase

apply tree policy to traverse tree



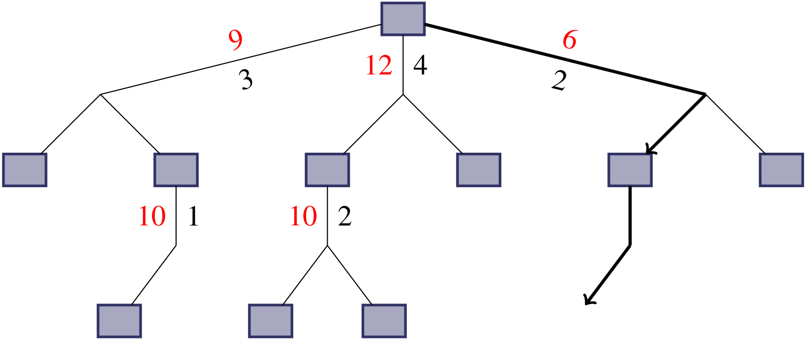
Selection Phase

apply tree policy to traverse tree



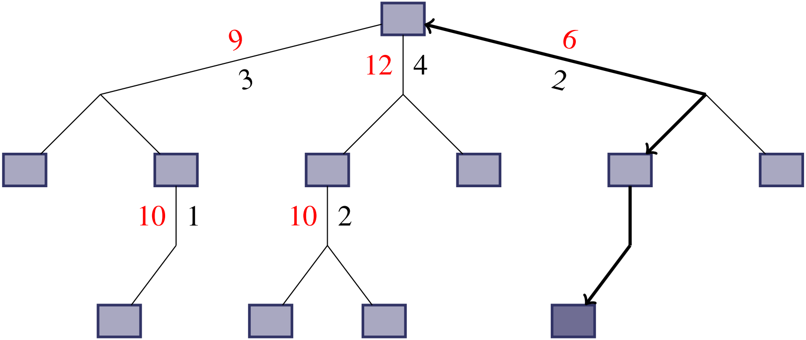
Selection Phase

apply tree policy to traverse tree



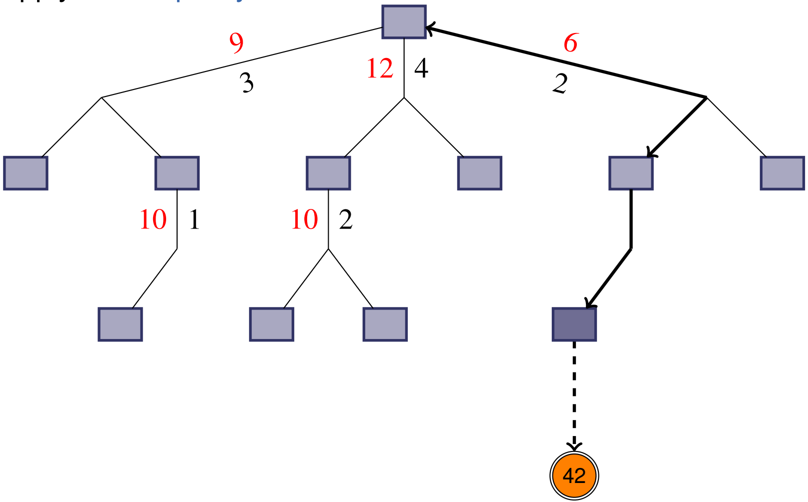
Expansion Phase

create a node for first state not in tree



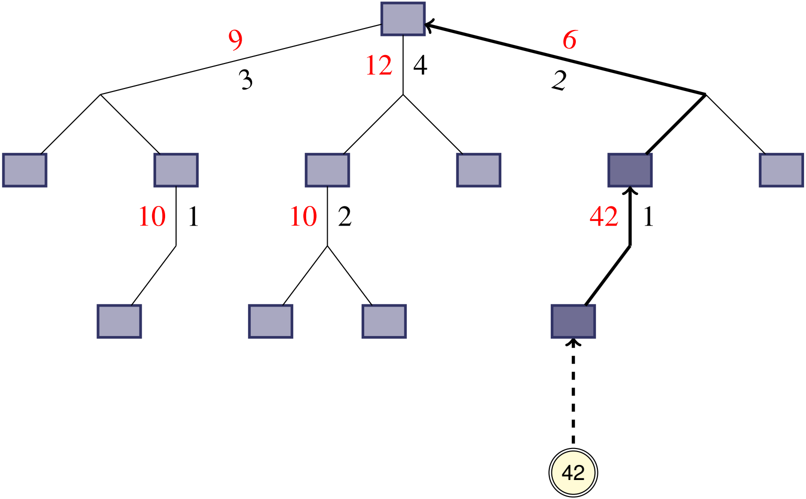
Simulation Phase

apply default policy until terminal state is reached



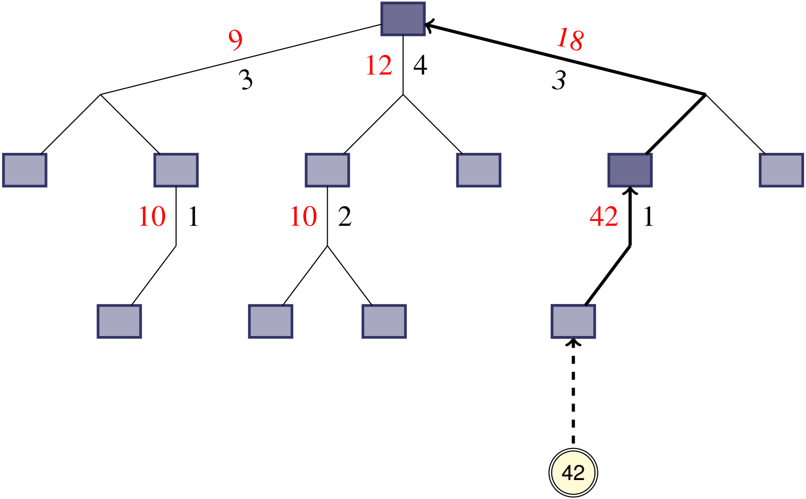
Backpropagation Phase

update visited nodes



Backpropagation Phase

update visited nodes



MCTS Configurations

- MCTS is a framework of algorithms
- concrete algorithms are specified in terms of:
  - tree policy
  - default policy
- for many probabilistic planning problems, there is a well-suited MCTS configuration
- However: there are also many poorly suited MCTS configurations
- and for every MCTS configuration that works well in one problem, there is another problem where it performs poorly

$\Rightarrow$ Are there properties we can analyse theoretically?

Asymptotic Optimality

Asymptotic Optimality

An MCTS algorithm is asymptotically optimal if $Q_k^{MCTS}(n)$ converges to optimal action-value $Q^\star(n.\text{state})$ for all $n \in \text{succ}(n_0)$ with $k \to \infty$.

Asymptotic Optimality

Asymptotic Optimality

An MCTS algorithm is asymptotically optimal if $Q_k^{MCTS}(n)$ converges to optimal action-value $Q^\star(n.\text{state})$ for all $n \in \text{succ}(n_0)$ with $k \to \infty$.

Note: there are MCTS instantiations that act optimally although the values do not converge in this way (e.g., if all $Q_k^{MCTS}(n)$ converge to $\ell \cdot Q^\star(n.\text{state})$ for a constant $\ell > 0$)

Asymptotic Optimality

A tree policy is asymptotically optimal if

- it explores forever:
  - every position is expanded eventually and visited infinitely often
- and it is greedy in the limit:
  - the probability that an optimal move action selected converges to 1

$\rightsquigarrow$ in the limit, backups based on iterations where only an optimal policy is followed dominate suboptimal backups

Tree Policy: Objective

tree policies have two contradictory objectives:

- explore parts of the game tree that have not been investigated thoroughly
- exploit knowledge about good moves to focus search on promising areas

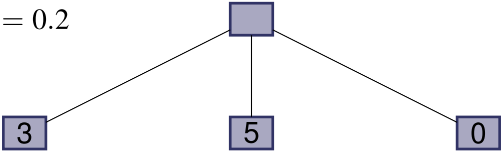
central challenge: balance exploration and exploitation

ε-greedy: Idea

- tree policy with constant parameter ε
- with probability $1 - \varepsilon$, pick a greedy move (i.e., one that leads to a successor node with the best action-value estimate)
- otherwise, pick a non-greedy successor uniformly at random

ε-greedy: Example

$\varepsilon = 0.2$



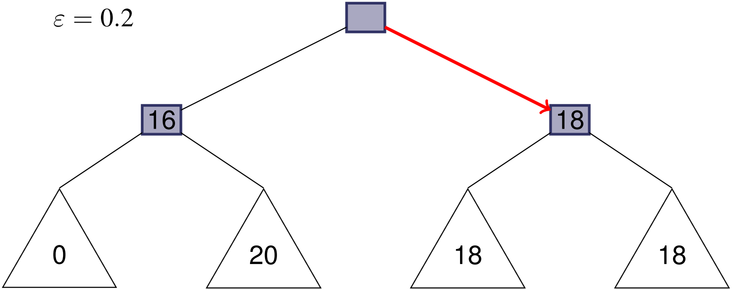
$\mathbb{P}(n_1) = 0.1$ $\mathbb{P}(n_2) = 0.8$ $\mathbb{P}(n_3) = 0.1$

ε-greedy: Asymptotic Optimality

Asymptotic Optimality of ε-greedy

- explores forever
- not greedy in the limit

$\rightsquigarrow$ not asymptotically optimal



ε-greedy: Asymptotic Optimality

Asymptotic Optimality of ε-greedy

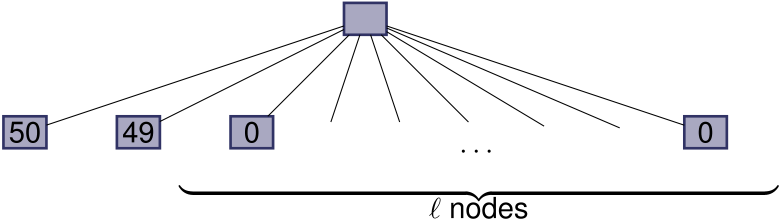
- explores forever
- not greedy in the limit

$\rightsquigarrow$ not asymptotically optimal

asymptotically optimal variant: use decaying ε, e.g. $\varepsilon = \frac{1}{k}$

ε-greedy: Weakness

Problem:
when ε-greedy explores, all non-greedy moves are treated equally



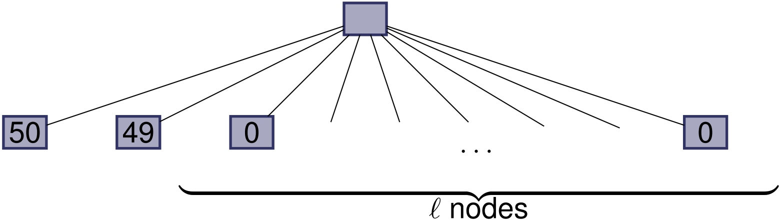
e.g., $\varepsilon = 0.2, \ell = 9$: $\mathbb{P}(n_1) = 0.8$, $\mathbb{P}(n_2) = 0.02$

Softmax: Idea

- tree policy with constant parameter τ
- select moves proportionally to their action-value estimate
- Boltzmann exploration selects moves proportionally to

$$\mathbb{P}(n) \propto e^{\frac{\hat{Q}_k^{\text{MCTS}}(n)}{\tau}}$$

Softmax: Example



e.g., $\tau = 10, \ell = 9$: $\mathbb{P}(n_1) \approx 0.51$, $\mathbb{P}(n_2) \approx 0.46$

Boltzmann Exploration: Asymptotic Optimality

Asymptotic Optimality of Boltzmann Exploration

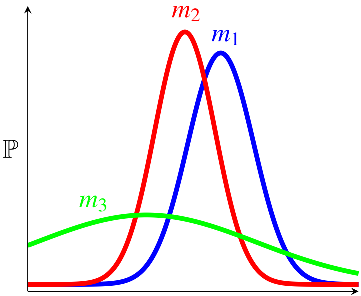
- explores forever
- not greedy in the limit
  (probabilities converge to positive constant)

$\rightsquigarrow$ not asymptotically optimal

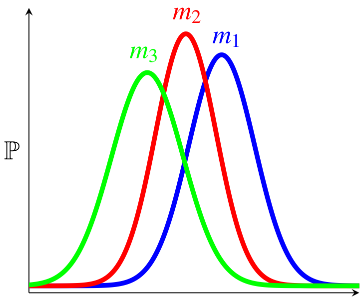
asymptotically optimal variants: use decaying τ

careful: τ must not decay faster than logarithmically
(i.e., must have $\tau \geq \frac{\text{const}}{\log k}$) to explore infinitely

Boltzmann Exploration: Weakness



scenario 1: high variance for m_3



scenario 2: low variance for m_3

- Boltzmann exploration only considers mean of sampled action-values for the given moves
- as we sample the same node many times, we can also gather information about variance (how reliable the information is)
- Boltzmann exploration ignores the variance, treating the two scenarios equally

Upper Confidence Bounds: Idea

balance exploration and exploitation by preferring moves that

- have been successful in earlier iterations (exploit)
- have been selected rarely (explore)

Upper Confidence Bounds: Idea

Upper Confidence Bounds

- select successor n' of n that maximizes $\hat{Q}_k^{MCTS}(n') + B(n')$
- based on action-value estimate $\hat{Q}^{\text{MCTS}}(n')$
- and a bonus term $B(n')$
- select $B(n')$ such that $Q^\star(n'.\text{state}) \leq \hat{Q}_k^{MCTS}(n') + B(n')$ with high probability
- idea: $\hat{Q}_{(}^{\text{MCTS}}n') + B(n')$ is an upper confidence bound on $Q^\star(n')$ under the collected information

Upper Confidence Bounds: UCB1

- use $B(a) = \sqrt{\frac{2 \cdot \ln N_k(n)}{N_k(n',a)}}$ as bonus term
- bonus term is derived from Chernoff-Hoeffding bound:
  - gives the probability that a sampled value (here: $\hat{Q}_k^{\text{MCTS}}(n')$)
  - is far from its true expected value (here: $Q^\star(n'.\text{state})$)
  - in dependence of the number of samples (here: $N_k(n', n'.\text{action})$)
- picks the optimal move exponentially more often

Upper Confidence Bounds: Asymptotic Optimality

Asymptotic Optimality of UCB1

- explores forever
- greedy in the limit

$\rightsquigarrow$ asymptotically optimal

Upper Confidence Bounds: Asymptotic Optimality

Asymptotic Optimality of UCB1

- explores forever
- greedy in the limit

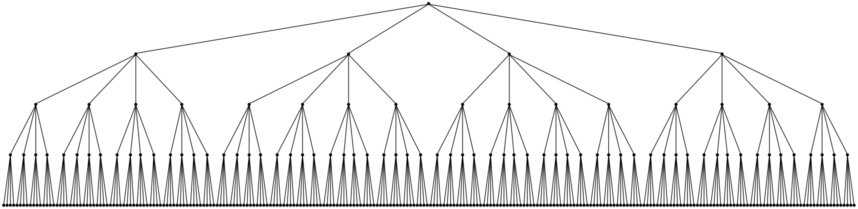
$\rightsquigarrow$ asymptotically optimal

However:

- no theoretical justification to use UCB1
  in trees or planning scenarios
- development of tree policies active research topic

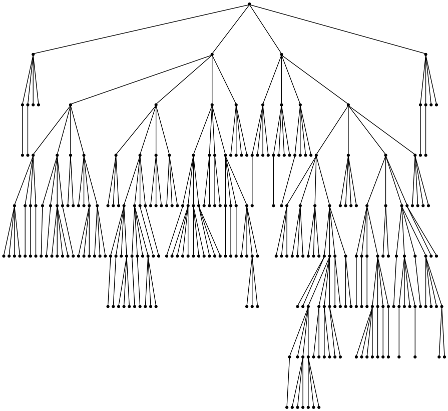
Tree Policy: Asymmetric Game Tree

full tree up to depth 4



Tree Policy: Asymmetric Game Tree

UCT tree (equal number of search nodes)



Default Policy

- Most often used: Monte-Carlo random walk
- Very successful: learned default policies, (e.g., for Go) Requires large amounts of data and/or computational power

AO*: Idea

AO* for acyclic MDPs (Nilsson, 1971)

- add s_0 to open list and to greedy policy graph
- repeat until greedy policy graph has no fringe
  - expand best state s on the fringe of the greedy graph
  - initialize all new states with heuristic
  - backup all states from where s is reachable
  - recompute the greedy graph
- output the greedy graph as the final policy

AO* Tree Search

- perform iterations as long as resources (deliberation time, memory) allow:
- build a partial MDP tree, where nodes n are annotated with
  - average reward
  - a solve label
- initially, the tree contains only the root node
- each iteration expands one node in the tree

AO* Tree Search: Iterations

Iterations proceed as follows:

- starting from the root node, select a successor node with maximum action-value estimate
- until a node is reached without successors
- expand that node and add all children to the tree
- initialize action-value estimate of all children with heuristic
- if a child is a terminal node, set its solve label to true
- if a has been selected in n, update n:
  - derive successor n' with maximum action value
  - set $\hat{Q}_k(n) = \hat{Q}_k(n') + R(n.\text{state}, n'.\text{action})$
  - set the solve label of n to the value of the solve label of n'

Real-Time Dynamic Programming

- AO* Tree Search very similar to (Trial-based) Real-Time Dynamic Programming (Barto, Bradtke, and Singh 1995)
- Difference: RTDP expands states until a terminal state is reached

Trial-based Heuristic Tree Search

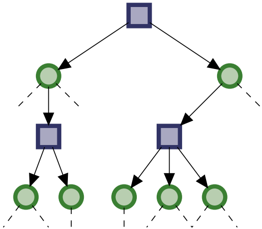
asymptotically optimal MDP algorithms:

- Monte-Carlo Tree Search (e.g., UCT)
- Dynamic Programming (e.g., RTDP)
- Heuristic Search (e.g., AO*)

all have complementary strengths
common framework that allows to describe them all Trial-based Heuristic Tree Search (THTS)

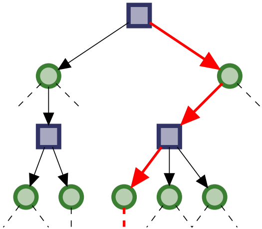
Trial-based Heuristic Tree Search

- initialize search tree with initial state
- extend search tree iteratively in trials



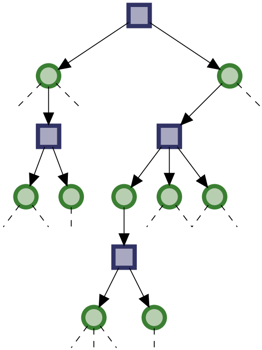
Trial-based Heuristic Tree Search

- initialize search tree with initial state
- extend search tree iteratively in trials
- 6 variable ingredients:
  - action selection
  - outcome selection



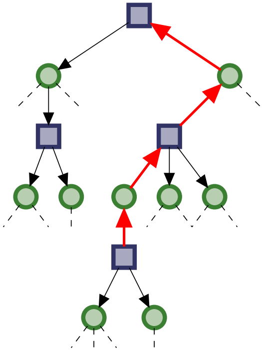
Trial-based Heuristic Tree Search

- initialize search tree with initial state
- extend search tree iteratively in trials
- 6 variable ingredients:
  - action selection
  - outcome selection
  - initialization
  - trial length



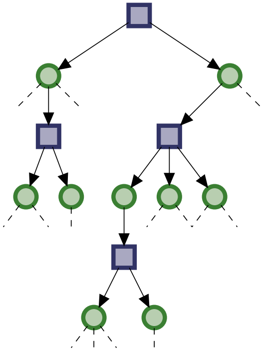
Trial-based Heuristic Tree Search

- initialize search tree with initial state
- extend search tree iteratively in trials
- 6 variable ingredients:
  - action selection
  - outcome selection
  - initialization
  - trial length
  - backup function



Trial-based Heuristic Tree Search

- initialize search tree with initial state
- extend search tree iteratively in trials
- 6 variable ingredients:
  - action selection
  - outcome selection
  - initialization
  - trial length
  - backup function
  - recommendation function



MCTS in the THTS Framework

- trial length: terminate trial once node is added to tree
- action selection: tree policy (e.g., UCB1 for MCTS)
- outome selection: sample
- initialization: add single node to the tree and initialize with heuristic that simulates the default policy
- backup function: Monte-Carlo backups
- recommendation function: expected best arm

AO* in the THTS Framework

- trial length: terminate trial once node is added to tree
- action selection: greedy
- outome selection: sample (or whatever is done in your version of AO*)
- initialization: add all successors, including all outcomes, and initialize with admissible heuristic
- backup function: Bellman backups
- recommendation function: expected best arm

RTDP in the THTS Framework

- trial length: finish trials only in terminal states
- action selection: greedy
- outome selection: sample
- initialization: add all successors, including all outcomes, and initialize with admissible heuristic
- backup function: Bellman backups
- recommendation function: expected best arm

Other Ingredients

- recommendation function:
  - most played arm (MCTS variants)
- initialization:
  - tree expansion: add all (action) successors (UCT*)
  - state-action heuristic, e.g., IDS-based lookahead heuristic of PROST (UCT*)

Other Ingredients: backup functions

- Temporal Differences
- Q-Learning
- MaxMonte-Carlo (DP-UCT)
- Partial Bellman (UCT*)

Considered ingredients in my dissertation

- 1 trial length, 1 outcome selection, 1 initialization
- 2 different recommendation functions
- 9 different backup functions
- 9 different action selections

$\Rightarrow$ 162 different THTS algorithms

115 asymptotically optimal

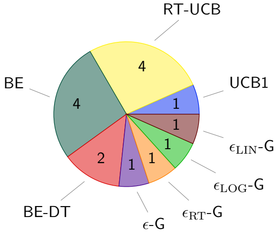
Experimental Evaluation

- most played arm recommendation function often better than same configuration with expected best arm

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	**Total**
$\mathrm{MC}^{\mathrm{UCB1}}_{\mathrm{MPA}}$	27	65	78	86	45	92	77	89	86	71	46	84	**70**
Prost 2011	26	62	49	84	42	90	69	88	83	60	49	85	**66**

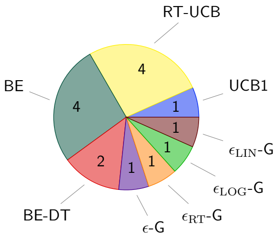
Experimental Evaluation

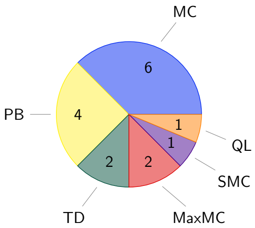
- most played arm recommendation function often better than same configuration with expected best arm
- Boltzman exploration and root-valued UCB1 perform best in most domains



Experimental Evaluation

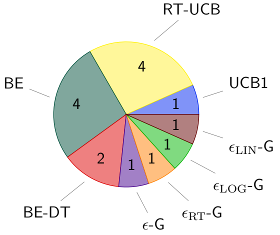
- most played arm recommendation function often better than same configuration with expected best arm
- Boltzman exploration and root-valued UCB1 perform best in most domains
- Monte-Carlo and Partial Bellman backups perform best in most domains

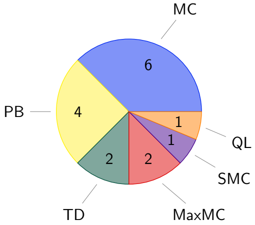




Experimental Evaluation

- most played arm recommendation function often better than same configuration with expected best arm
- Boltzman exploration and root-valued UCB1 perform best in most domains
- Monte-Carlo and Partial Bellman backups perform best in most domains
- almost all action selections and backup functions perform best in at least one domain





Implementation: PROST

- The PROST planner is an implementation of the MCTS framework
- mixing and matching of ingredients very simple
- to add new ingredients, just inherit from the corresonding class

https://bitbucket.org/tkeller/prost/

Summary

- MCTS asymptotically optimal with right components
- AO* and RTDP have complementary strength
- THTS allows to combine ideas from MCTS, DP and Heuristic Search
- almost all ingredients perform well in at least one domain