Sample-based Planning

Introduction To Reinforcement Learning, Leiden University, The Netherlands

Thomas Moerland

Planning/Search

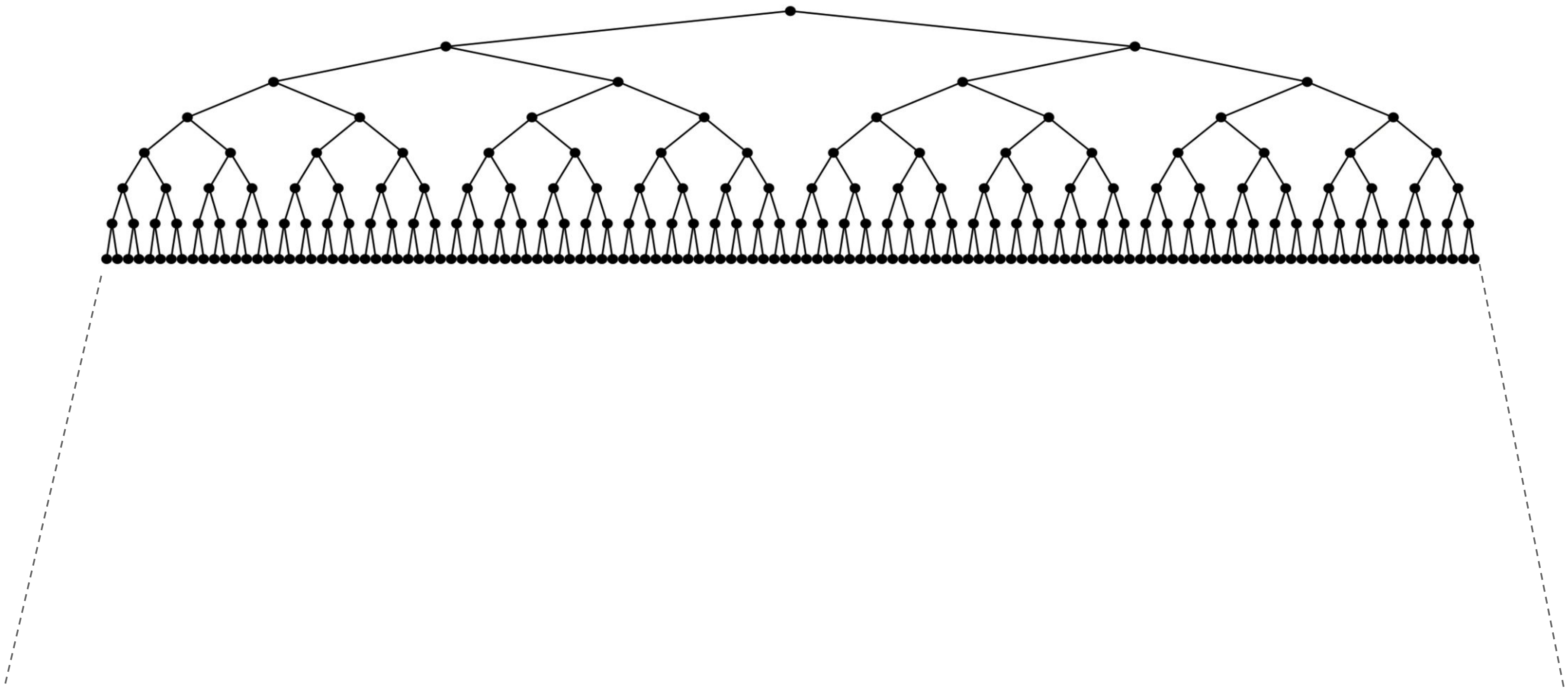
Planning/Search

Definition?

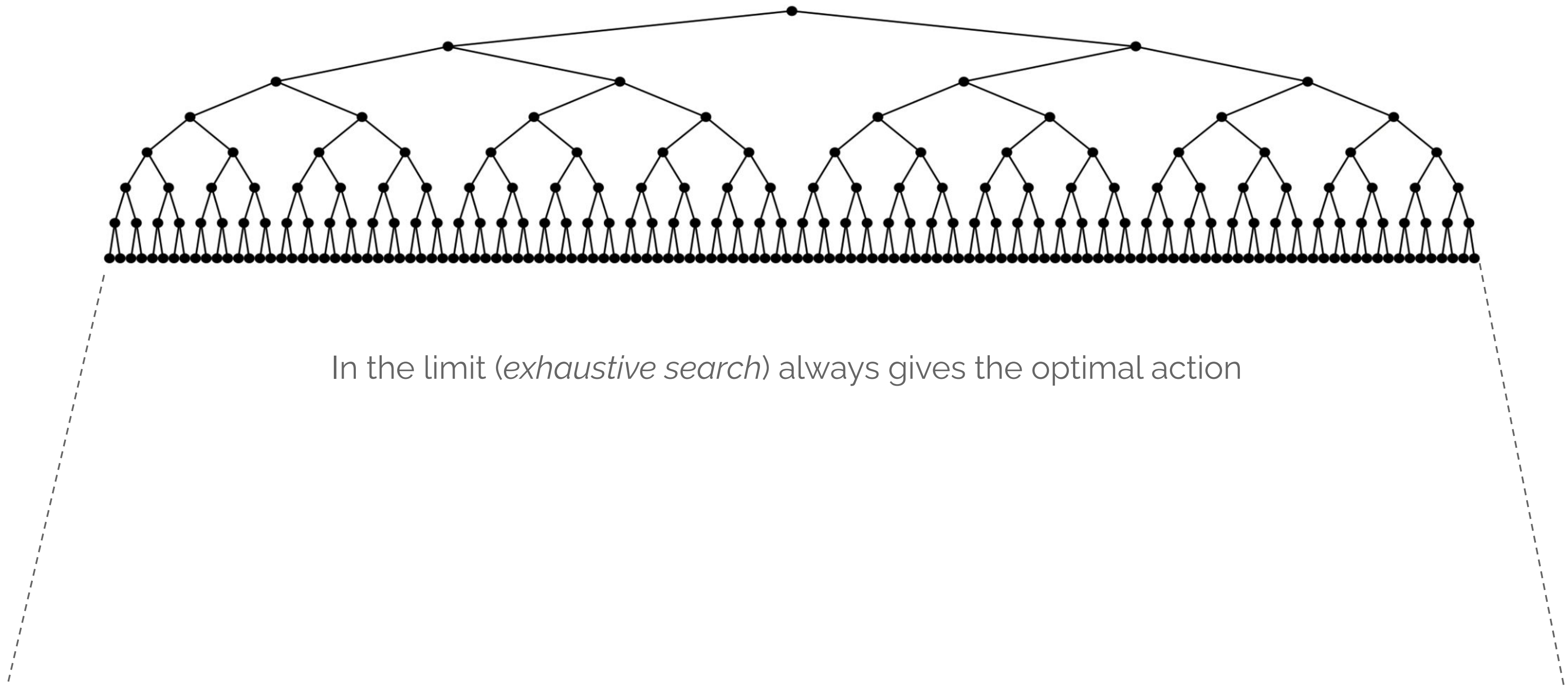
Planning/Search

Any type of lookahead search in a model to determine good actions

Planning/Search

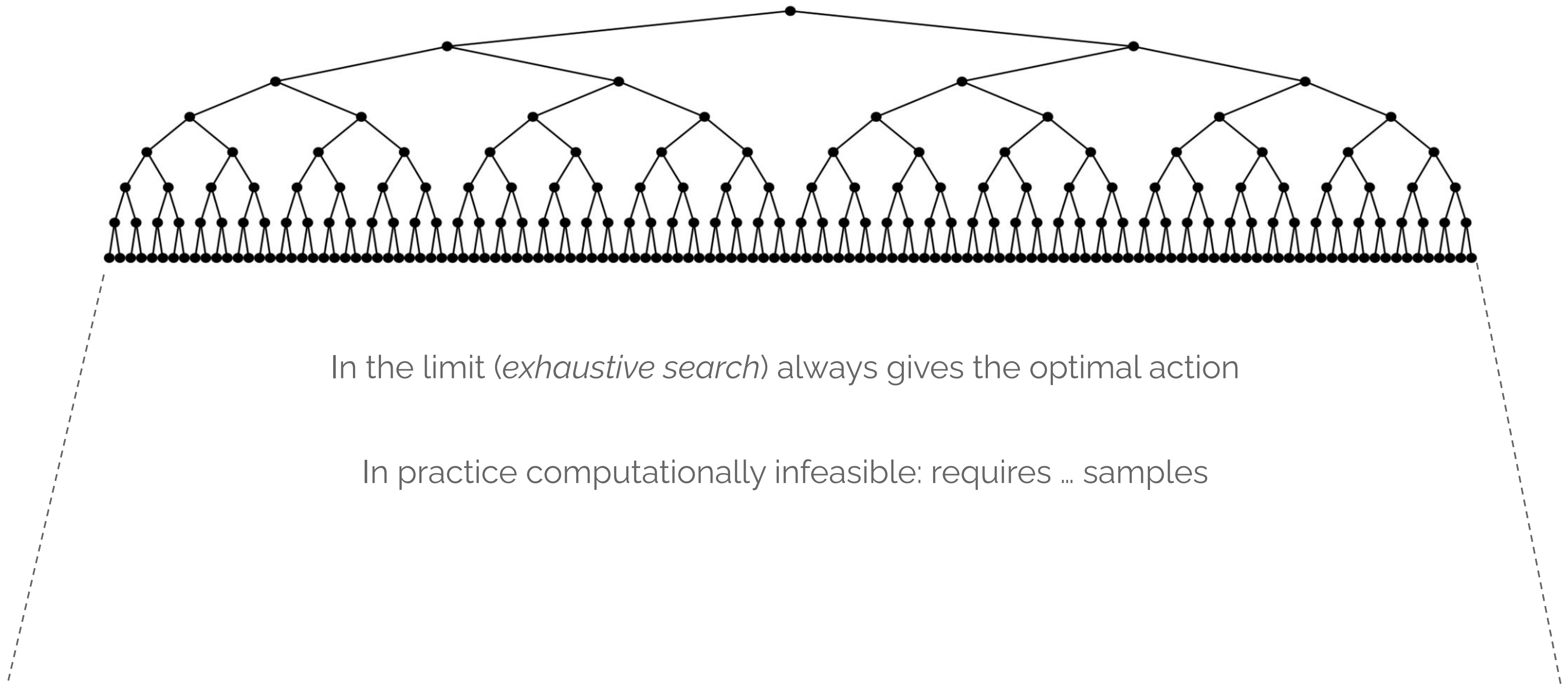


Planning/Search



In the limit (*exhaustive search*) always gives the optimal action

Planning/Search



In the limit (*exhaustive search*) always gives the optimal action

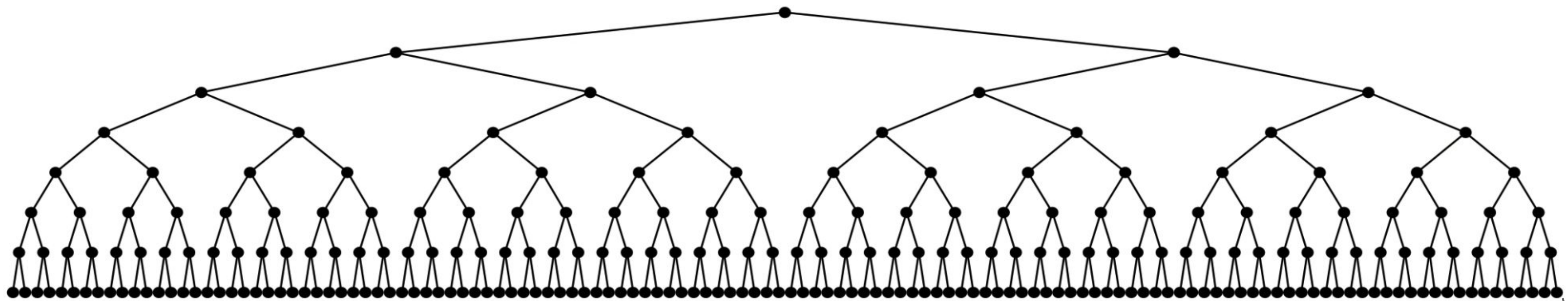
In practice computationally infeasible: requires … samples

Planning/Search

b=branching factor (# actions)

d= depth

(we for now ignore stochastic transitions)



In the limit (*exhaustive search*) always gives the optimal action

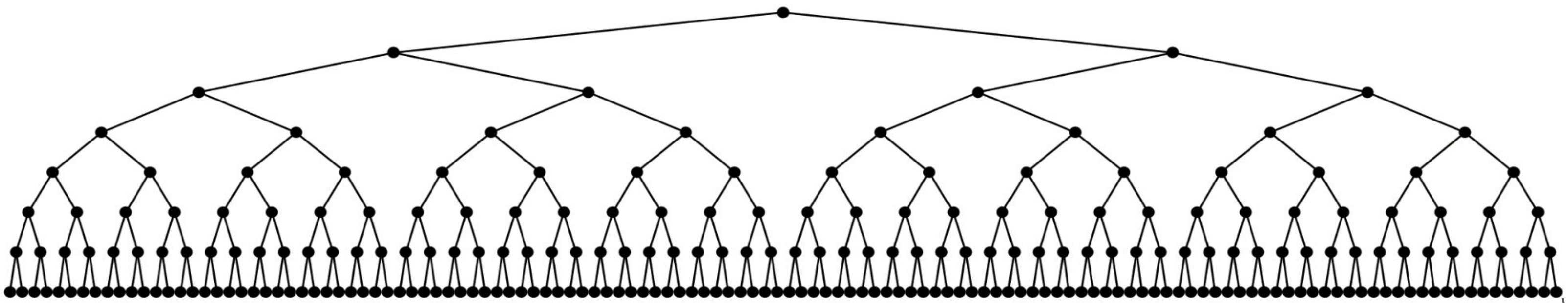
In practice computationally infeasible: requires $\mathbf{b}^{\mathbf{d}}$ samples

Planning/Search

b=branching factor (# actions)

d= depth

(we for now ignore stochastic transitions)



In the limit (*exhaustive search*) always gives the optimal action

In practice computationally infeasible: requires $\mathbf{b}^{\mathbf{d}}$ samples

All search algorithms try to improve the visitation order
(i.e., reduce the width and depth of the search)

Content

1. Types of planning (decision-time versus background)

2. Classic planning (uninformed & heuristic search)

Break

3. Sample-based planning (Monte Carlo Search, Sparse Sampling, Monte Carlo Tree Search)

4. Iterated planning and learning

1. Decision-time versus Background planning

Type of planning

Type of planning

Background planning

Decision-time planning

Type of planning

Background planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')

Decision-time planning

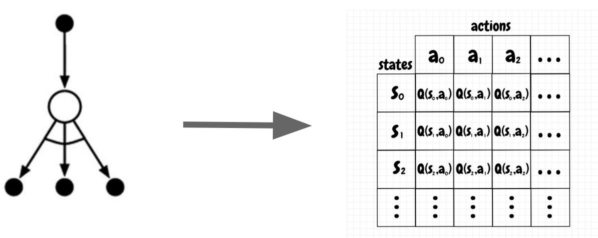
Type of planning

Background planning

Decision-time planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')



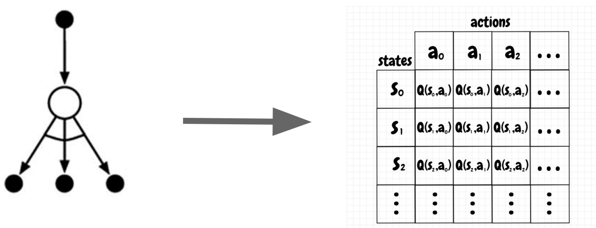
Type of planning

Background planning

Decision-time planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')



e.g. Dynamic Prog. (Ch. 4), Dyna (Ch. 8)

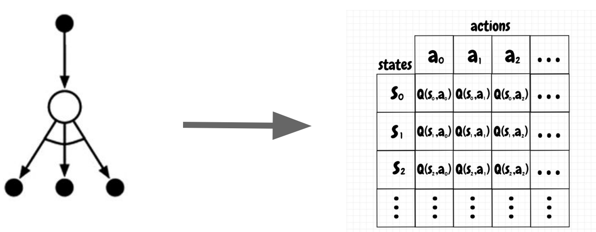
(Traditionally: smaller tree)

Type of planning

Background planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')



e.g. Dynamic Prog. (Ch. 4), Dyna (Ch. 8)

(Traditionally: smaller tree)

Decision-time planning

Use lookahead in model to find a good action for a current state s

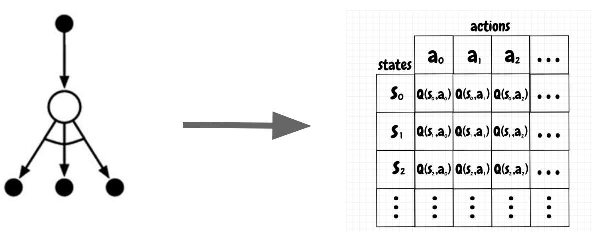
(focus all budget on current decision)

Type of planning

Background planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')



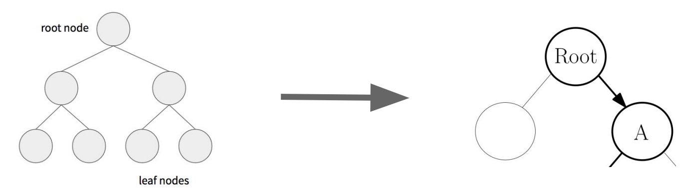
e.g. Dynamic Prog. (Ch. 4), Dyna (Ch. 8)

(Traditionally: smaller tree)

Decision-time planning

Use lookahead in model to find a good action for a current state s

(focus all budget on current decision)

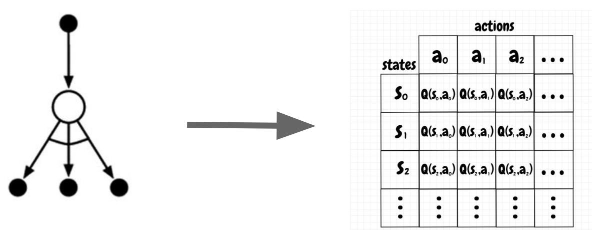


Type of planning

Background planning

Use lookahead in model to update a global (value/policy) solution

(improve overall solution – may be called 'learning')



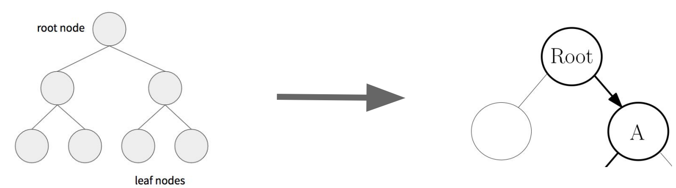
e.g. Dynamic Prog. (Ch. 4), Dyna (Ch. 8)

(Traditionally: smaller tree)

Decision-time planning

Use lookahead in model to find a good action for a current state s

(focus all budget on current decision)



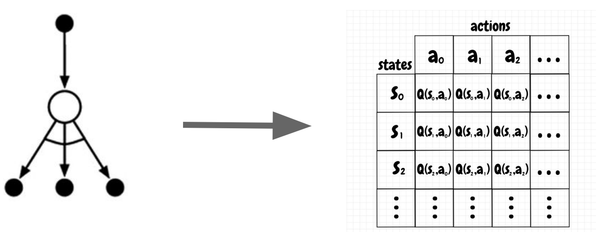
e.g. A*, MCTS (Ch. 8)

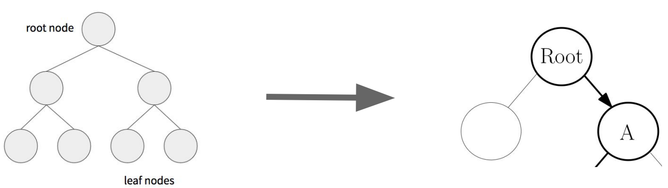
(Traditionally: larger tree, afterward discarded)

Type of planning

Main topic of today

Background planning	Decision-time planning
Use lookahead in model to update a global (value/policy) solution	Use lookahead in model to find a good action for a current state s
(improve overall solution – may be called ‘learning’)	(focus all budget on current decision)

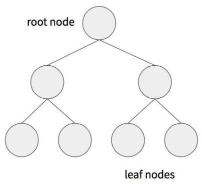




e.g. Dynamic Prog. (Ch. 4), Dyna (Ch. 8)	e.g. A*, MCTS (Ch. 8)
(Traditionally: smaller tree)	(Traditionally: larger tree, afterward discarded)

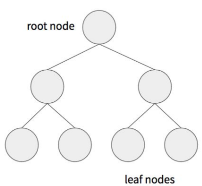
Both planning types can be combined

Both planning types can be combined

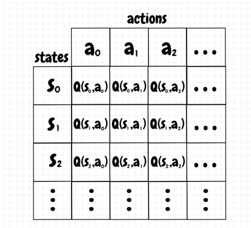


Arbitrarily small/large search

Both planning types can be combined

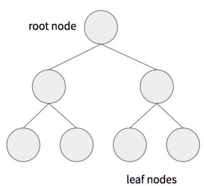


Arbitrarily small/large search



1. update overall solution (background/learning)

Both planning types can be combined



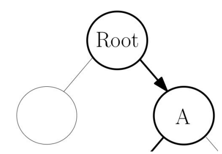
Arbitrarily small/large search

actions

states	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

1. update overall solution (background/learning)

and/or



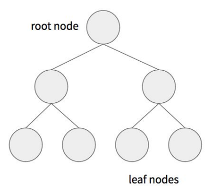
2. select an action (decision-time)

Both planning types can be combined

may also revert

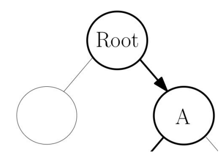
		actions		
states	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

1. update overall solution
(background/learning)



and/or

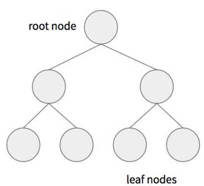
Arbitrarily small/large search



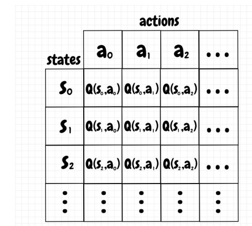
2. select an action
(decision-time)

Both planning types can be combined

may also revert



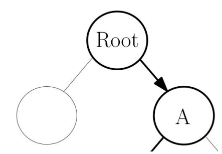
Arbitrarily small/large search



	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

1. update overall solution (background/learning)

and/or



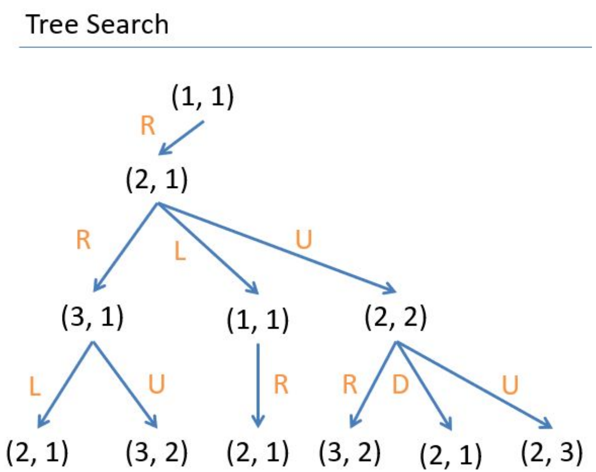
2. select an action (decision-time)

Discuss combined combined planning
& learning at end of this lecture

2. Classic Planning

Tree versus graph search

Tree versus graph search



Tree versus graph search

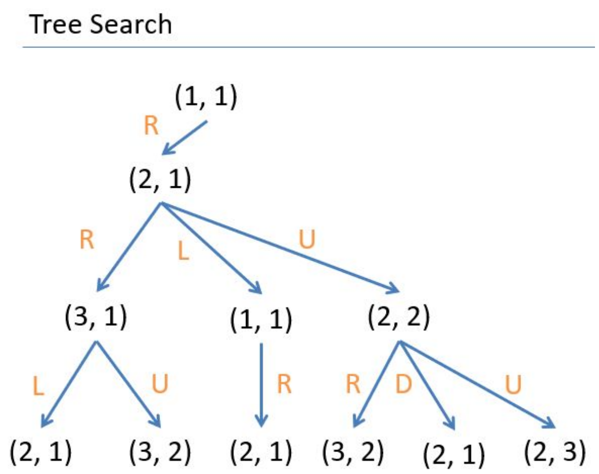
Q: Can a tree search spend useless compute?

Tree versus graph search

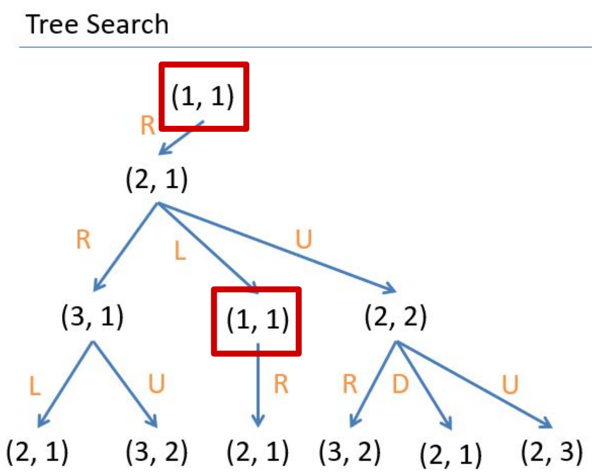
Q: Can a tree search spend useless compute?

A: Yes, because the same next state may appear in multiple directions

Tree versus graph search

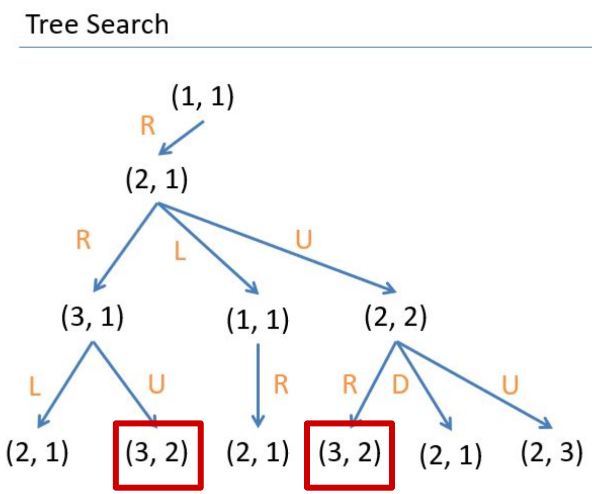


Tree versus graph search



'loop' : same state reappears in a path → only need to search from the first appearance

Tree versus graph search



redundant path: same state appears in different arms → only need to continue the search in the best path

Tree versus graph search

Q: Can a tree search spend useless compute?

A: Yes, because the same next state may appear in multiple directions

Q: What could be a solution?

Tree versus graph search

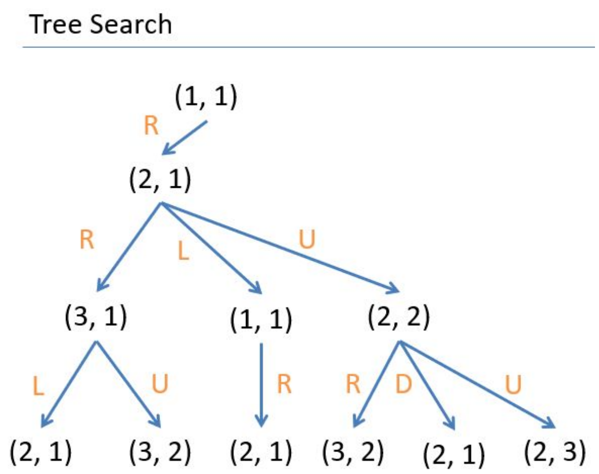
Q: Can a tree search spend useless compute?

A: Yes, because the same next state may appear in multiple directions

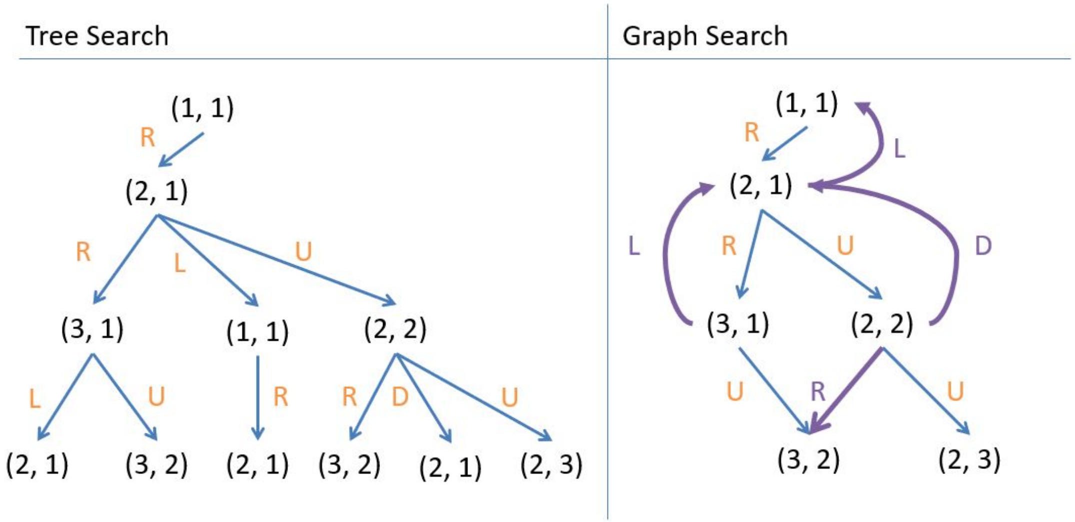
Q: What could be a solution?

A: Turn the tree search into a **graph search**

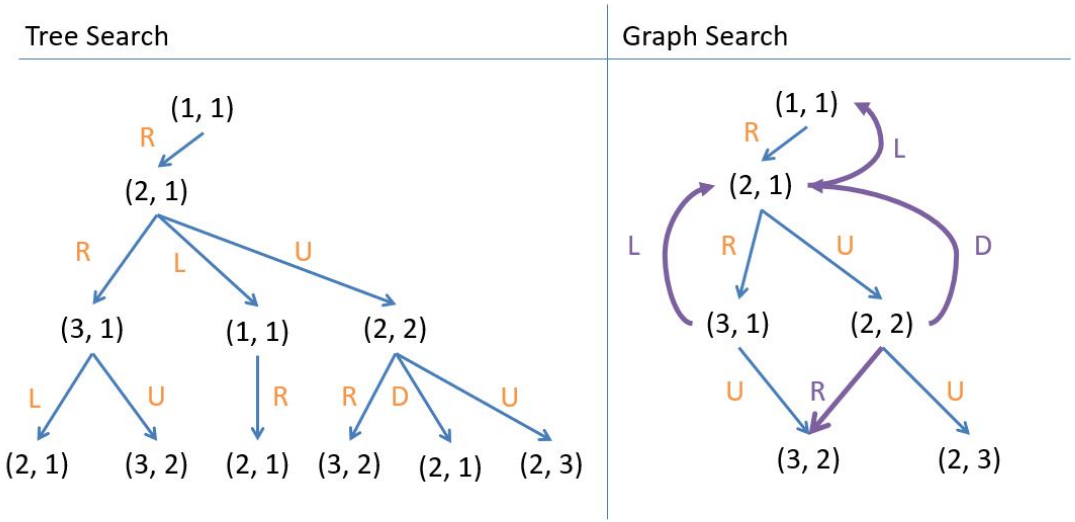
Tree versus graph search



Tree versus graph search



Tree versus graph search



Build a **graph**: only generate each unique state once, and build a search tree connecting them

Tree versus graph search

Q: Can a tree search spend useless compute?

A: Yes, because the same next state may appear in multiple directions

Q: What could be a solution?

A: Turn the tree search into a **graph search**

Q: What do we need to store/change for this?

Tree versus graph search

Q: Can a tree search spend useless compute?

A: Yes, because the same next state may appear in multiple directions

Q: What could be a solution?

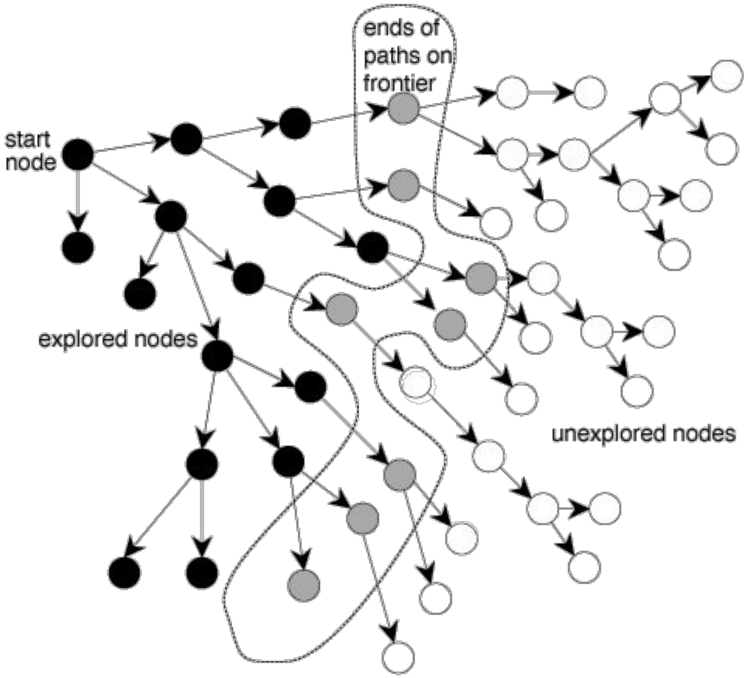
A: Turn the tree search into a **graph search**

Q: What do we need to store/change for this?

A: Track an **open list** (*frontier*) and **closed list** *(explored set)*

Closed & Open list

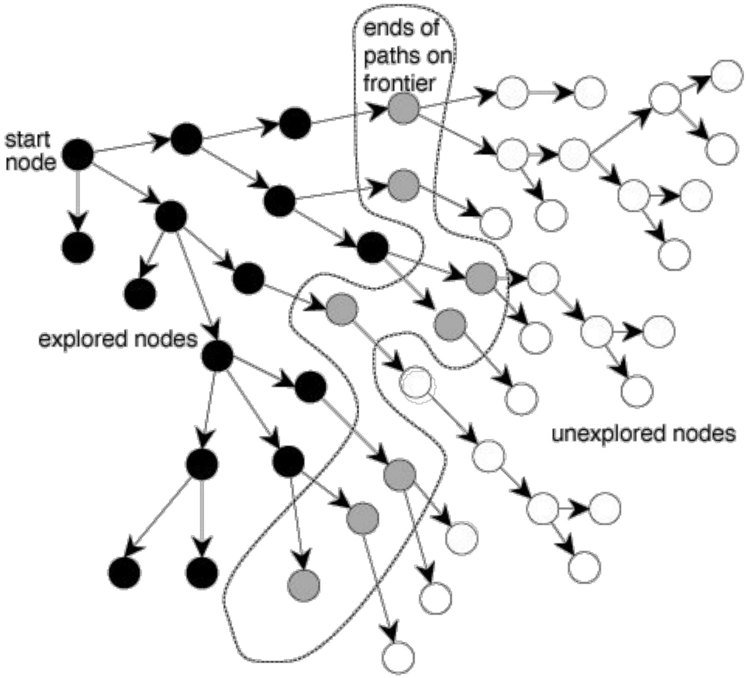
Closed & Open list



Closed & Open list

1. **Closed list**

   Fully expanded



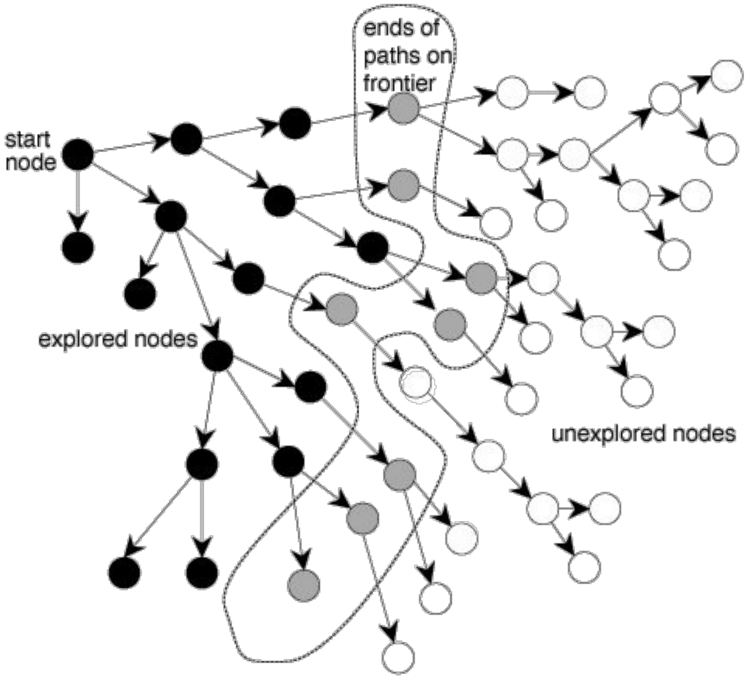
Closed & Open list

1. **Closed list**

   Fully expanded

2. **Open list = Frontier**

   Next candidates for expansion



Closed & Open list

1. **Closed list**

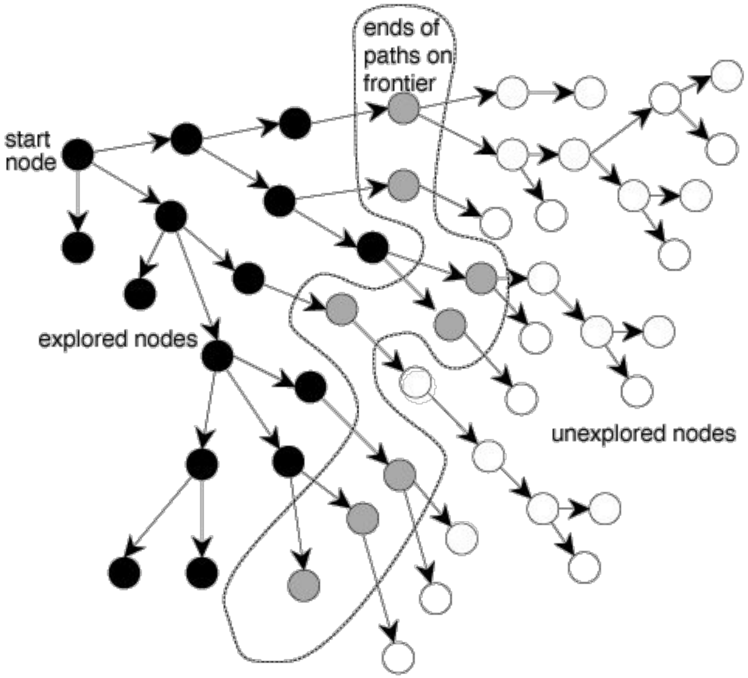
   Fully expanded

2. **Open list = Frontier**

   Next candidates for expansion

<u>Key idea</u>:

- Track every node in the graph (open/closed) and the optimal path towards is



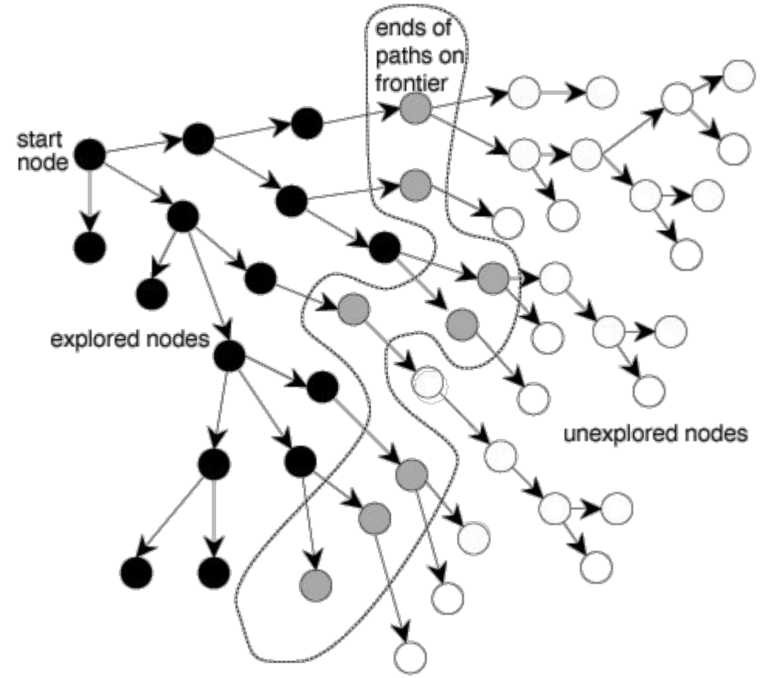
Closed & Open list

1. **Closed list**

   Fully expanded

2. **Open list = Frontier**

   Next candidates for expansion



<u>Key idea</u>:

- Track every node in the graph (open/closed) and the optimal path towards is
- Update these lists with every expansion (is this new expansion already in my closed or open list?)

Main challenge of planning

Main challenge of planning

In what order shall we visit state-actions?

Uninformed search

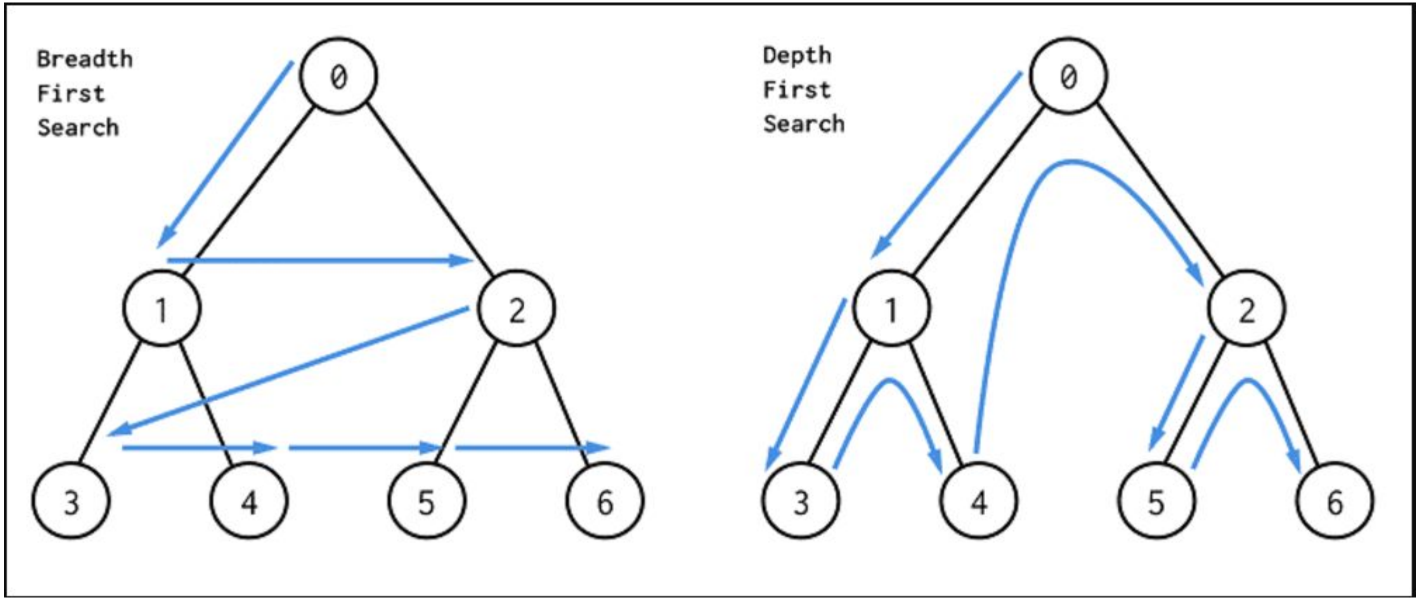
Can you give some example of uninformed search strategies?

Uninformed search

Can you give some example of uninformed search strategies?

- Breadth-first search
- Depth-first search
- Iterative deepening
- Uniform cost search / Dijkstra’s algorithm (weighted graphs)

Uninformed search



Uninformed search

Uninformed search

What is the downside of breadth/depth first search?

Uninformed search

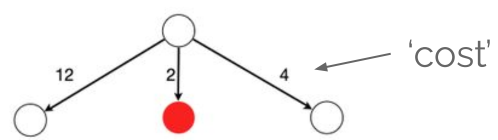
What is the downside of breadth/depth first search?

- Can be suboptimal if the <u>weight</u> (r*eward/cost*) per edge varies (ignored by DFS/BFS)

Uninformed search

What is the downside of breadth/depth first search?

- Can be suboptimal if the weight (*reward/cost*) per edge varies (ignored by DFS/BFS)



(BFS would expand the first action, but the second has lowest cost)

Reward versus Cost

Reward versus Cost

Reinforcement learning

Maximize the cumulative *reward*

Planning

Minimize the cumulative *cost*

Reward versus Cost

Reinforcement learning		**Planning**
Maximize the cumulative *reward*	=	*Minimize* the cumulative *cost*

same formulation

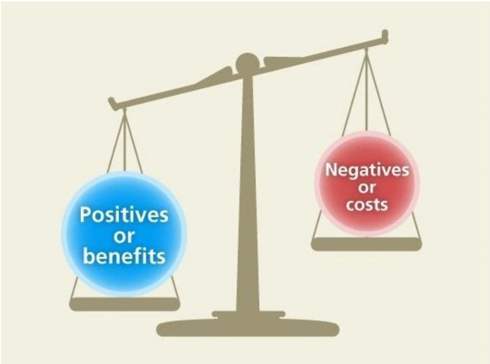
(cost = negative reward)

Reward versus Cost

Reinforcement learning		**Planning**
Maximize the cumulative *reward*	=	*Minimize* the cumulative *cost*

same formulation

(cost = negative reward)

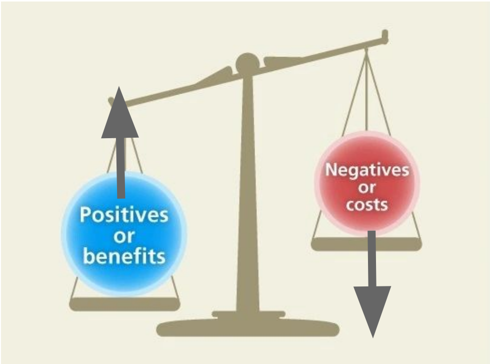


Reward versus Cost

Reinforcement learning		**Planning**
Maximize the cumulative *reward*	=	*Minimize* the cumulative *cost*

same formulation

(cost = negative reward)



Uninformed search

What is the downside of breadth/depth first search?

- Can be suboptimal if the <u>weight</u> (r*eward/cost*) per edge varies (ignored by DFS/BFS)

Uninformed search

What is the downside of breadth/depth first search?

- Can be suboptimal if the weight (*reward/cost*) per edge varies (ignored by DFS/BFS)

What is a potential solution?

Uninformed search

What is the downside of breadth/depth first search?

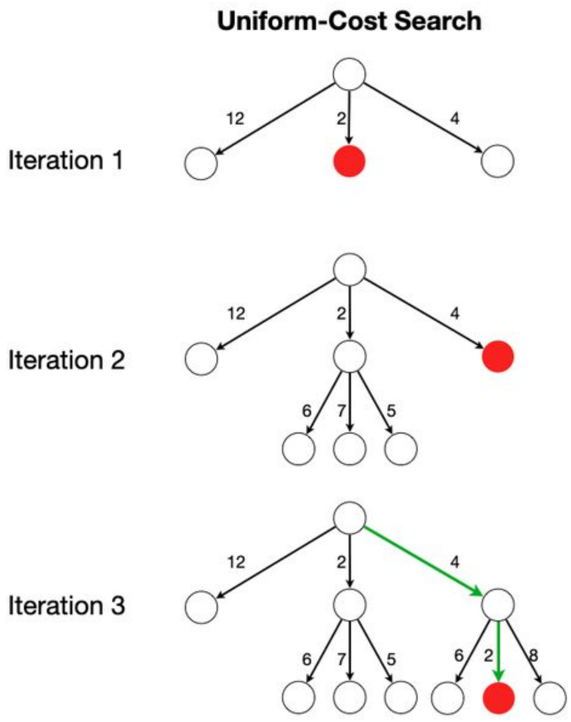
- Can be suboptimal if the weight (*reward/cost*) per edge varies (ignored by DFS/BFS)

What is a potential solution?

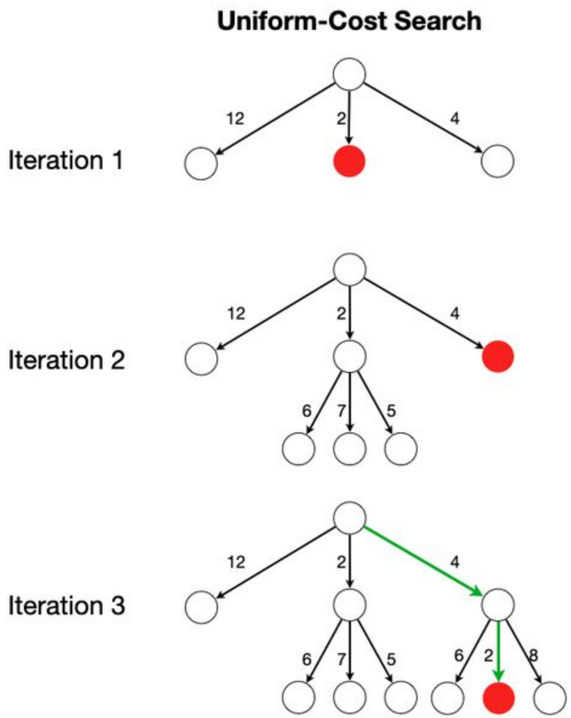
- Expand the node which currently looks most promising

Uniform cost search / Dijkstra's algorithm

Uniform cost search / Dijkstra's algorithm



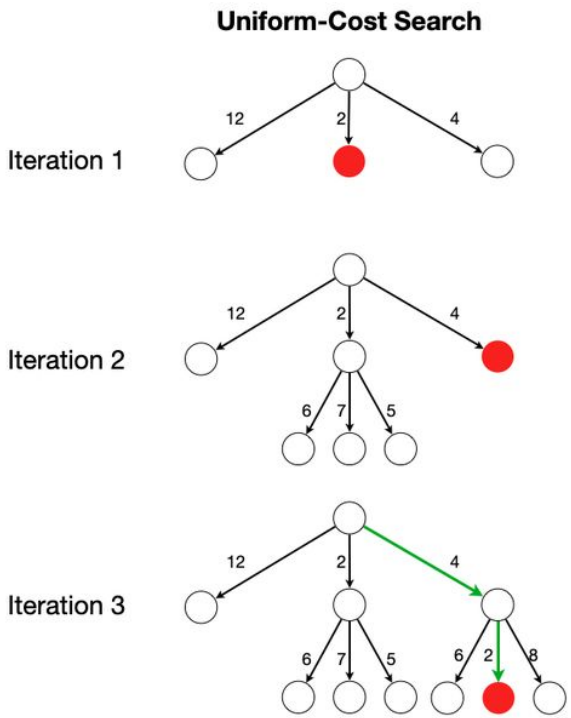
Uniform cost search / Dijkstra's algorithm



Note: only useful with non-uniform rewards/cost/edge weights

Q: What does this reduce to with uniform rewards?

Uniform cost search / Dijkstra's algorithm



Note: only useful with non-uniform rewards/cost/edge weights

Q: What does this reduce to with uniform rewards?
A: Breadth-first search

Uniform cost search / Dijkstra's algorithm

Uniform cost search / Dijkstra's algorithm

What is the downside of uniform cost search?

Uniform cost search / Dijkstra's algorithm

What is the downside of uniform cost search?

- We only look at the cost of the tree path g(s), but not at the remaining potential afterwards h(s)

Uniform cost search / Dijkstra's algorithm

What is the downside of uniform cost search?

- We only look at the cost of the tree path g(s), but not at the remaining potential afterwards h(s)

What is a potential solution?

Uniform cost search / Dijkstra's algorithm

What is the downside of uniform cost search?

- We only look at the cost of the tree path g(s), but not at the remaining potential afterwards h(s)

What is a potential solution?

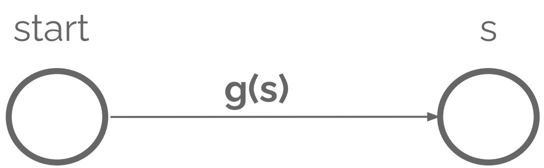
- Construct a heuristic function h(s) to predict the remaining potential

g(s) & h(s)

g(s) & h(s)

g(s)

g(s) = actual cumulative cost from start to state s

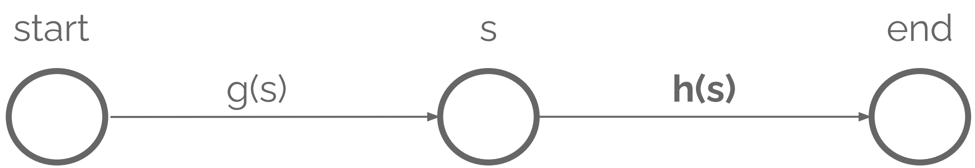


g(s) & h(s)

g(s) + **h(s)**

g(s) = actual cumulative cost from start to state s

h(s) = estimated cumulative cost from s to end



Best-first search

Best-first search

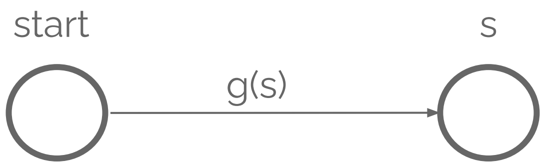
We want a general prioritization function **f(s)** to indicate what state to expand next

Best-first search

Best-first search

$f(s) = g(s)$

Dijkstra's algorithm/Uniform-cost search

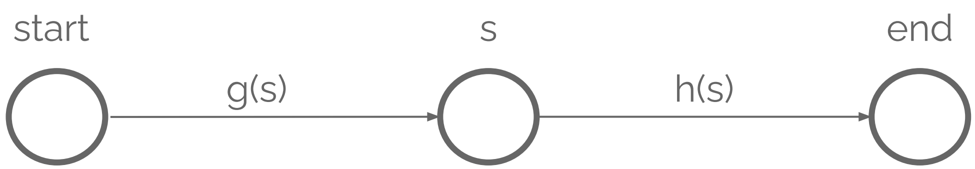


Best-first search

$f(s) = g(s)$	Dijkstra's algorithm/Uniform-cost search
$f(s) = h(s)$	Greedy best-first search

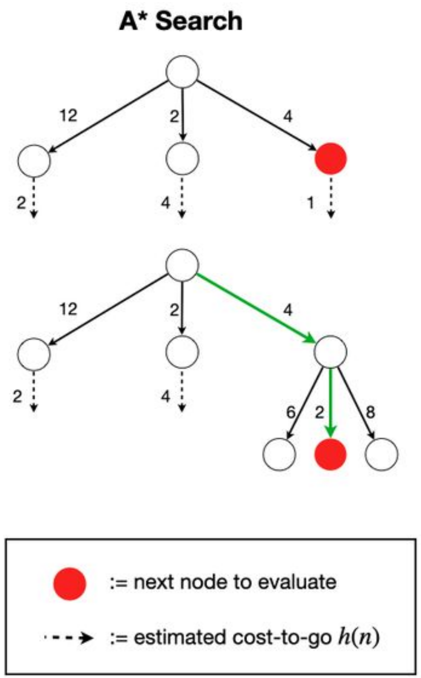
Best-first search

$f(s) = g(s)$	Dijkstra's algorithm/Uniform-cost search
$f(s) = h(s)$	Greedy best-first search
$f(s) = g(s) + h(s)$	**A* search**

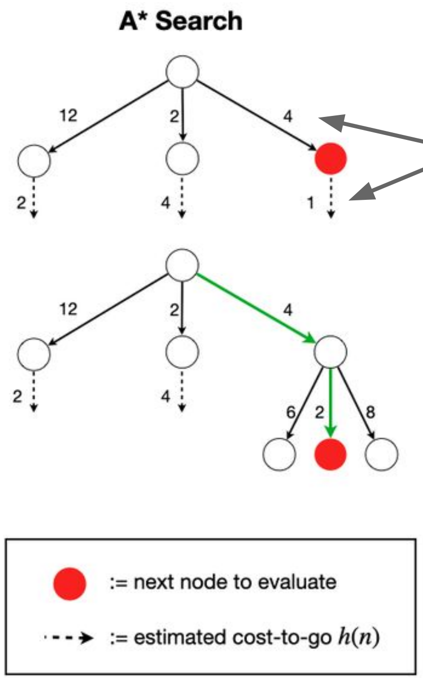


A* search

A* search


A* Search
12
2
4
2
4
1
12
2
4
2
4
6
2
8
:= next node to evaluate
:= estimated cost-to-go $h(n)$

A* search



Expand state with best g(s) + h(s)

Heuristic

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

A: This ensures optimality: we never skip an arm because of a too pessimistic heuristic.

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

A: This ensures optimality: we never skip an arm because of a too pessimistic heuristic.

Q: Why don't we then just initialize the heuristic h(s)=0 for every possible state?

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

A: This ensures optimality: we never skip an arm because of a too pessimistic heuristic.

Q: Why don't we then just initialize the heuristic h(s)=0 for every possible state?

A: This heuristic is completely uninformative: does not give any actual priority

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

A: This ensures optimality: we never skip an arm because of a too pessimistic heuristic.

Q: Why don't we then just initialize the heuristic h(s)=0 for every possible state?

A: This heuristic is completely uninformative: does not give any actual priority

Q: What is the perfect heuristic function?

Heuristic

Heuristic function h(s) is typically obtained from prior domain-knowledge or relaxations

Q: The heuristic h(s) needs to be *admissible*: it should always be equal to or optimistic about the remaining cumulative cost. Why?

A: This ensures optimality: we never skip an arm because of a too pessimistic heuristic.

Q: Why don't we then just initialize the heuristic h(s)=0 for every possible state?

A: This heuristic is completely uninformative: does not give any actual priority

Q: What is the perfect heuristic function?

A: The optimal value function: $h(s) = V^*(s)$! (The true optimal cumulative cost – You can see RL as learning the perfect heuristic – upon convergence eliminates the complete need for planning)

Reducing width

Reducing width

Heuristic are a way to reduce the depth of a search. Can we also reduce the width?

Reducing width

Heuristic are a way to reduce the depth of a search. Can we also reduce the width?

Forward pruning

(directly eliminate some of the available actions)

Reducing width

Heuristic are a way to reduce the depth of a search. Can we also reduce the width?

Forward pruning

(directly eliminate some of the available actions)

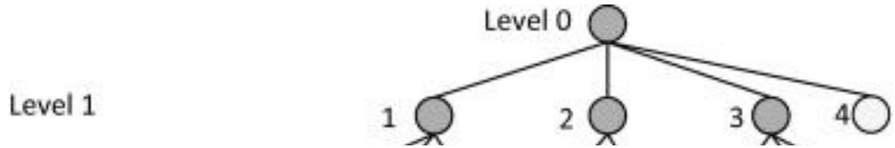
Simplest implementation: **beam search** (only keep best M candidates at every depth)

Beam search

- Expand children
- Select M best children

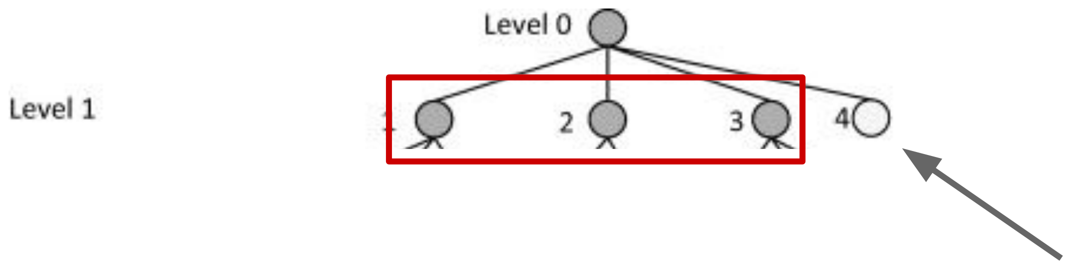
Beam search

- **Expand children**
- Select M best children



Beam search

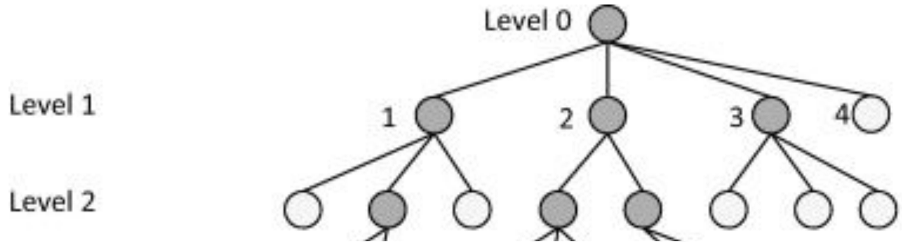
- Expand children
- **Select M best children**



‘forward pruning’

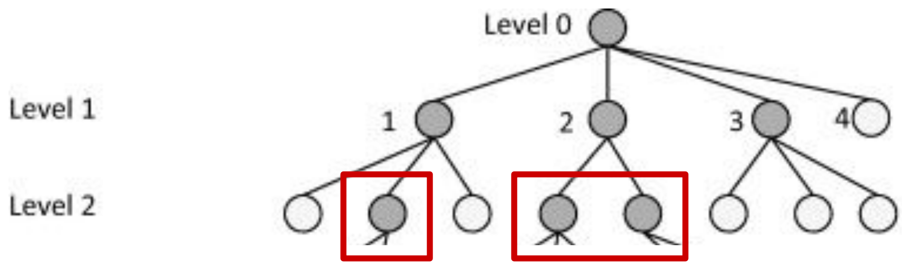
Beam search

- **Expand children**
- Select M best children



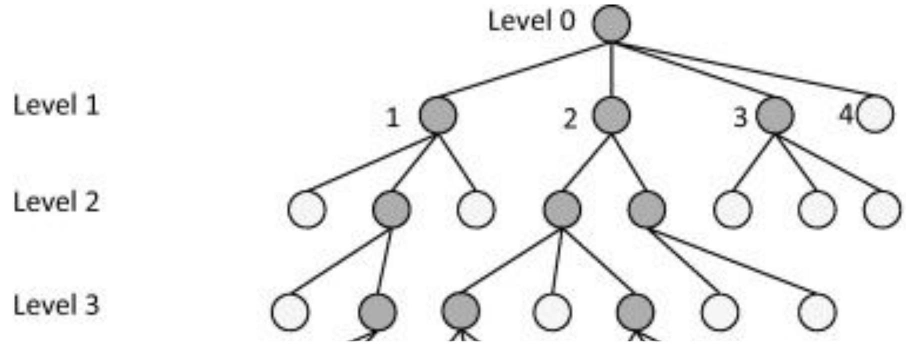
Beam search

- Expand children
- **Select M best children**



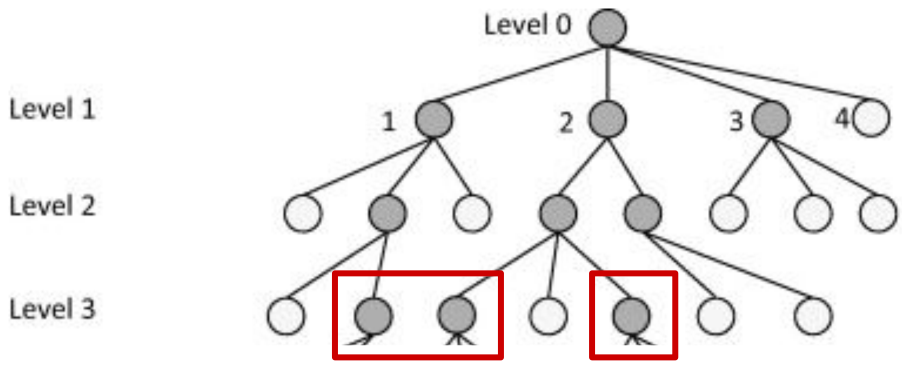
Beam search

- **Expand children**
- Select M best children



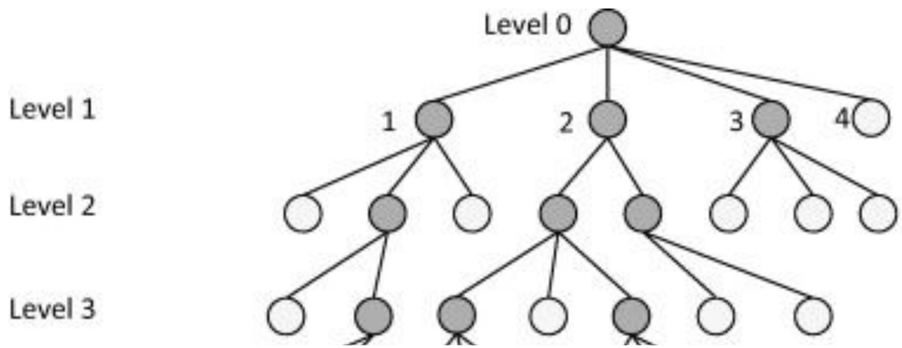
Beam search

- Expand children
- **Select M best children**



Beam search

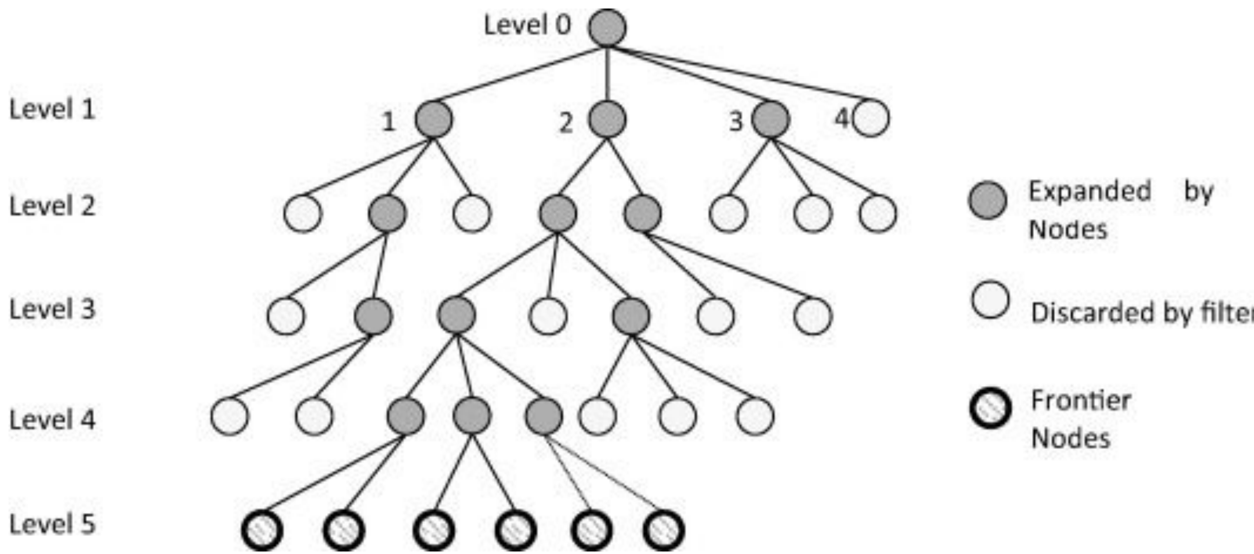
- Expand children
- Select M best children



etc.

Beam search

- Expand children
- Select M best children



Forward pruning

Forward pruning

Q: What is the risk of forward pruning?

Forward pruning

Q: What is the risk of forward pruning?

A: We may prune away optimal actions, and therefore we lose all optimality guarantees

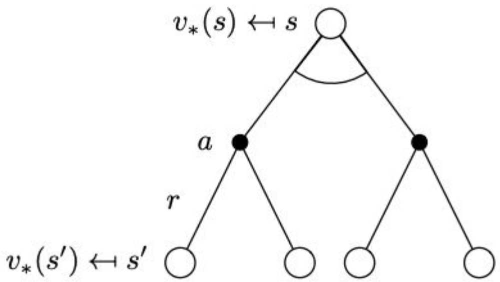
Stochastic dynamics

Classic planning primarily focused on deterministic settings.

Stochastic dynamics

Classic planning primarily focused on deterministic settings.

Can we also apply it to the full stochastic MDP setting?



AO*

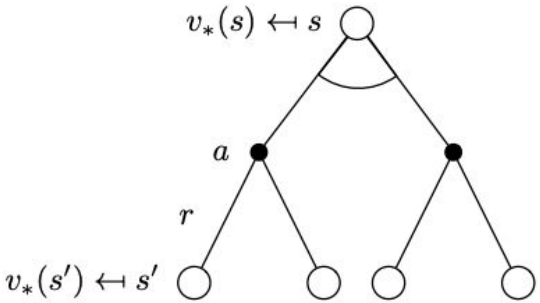
AO*

Yes, algorithms typically have their stochastic extension, where we unfold all possible states below an action (instead of only one state)

AO*

Yes, algorithms typically have their stochastic extension, where we unfold all possible states below an action (instead of only one state)

Example: A* —> **AO*** (*AND-OR*)

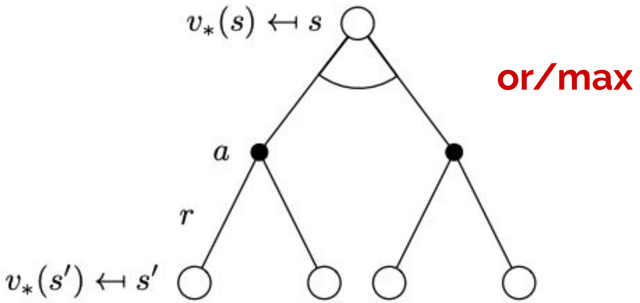


AO*

Yes, algorithms typically have their stochastic extension, where we unfold all possible states below an action (instead of only one state)

Example: A* —> **AO*** (*AND-OR*)

- OR = MAX = action selection



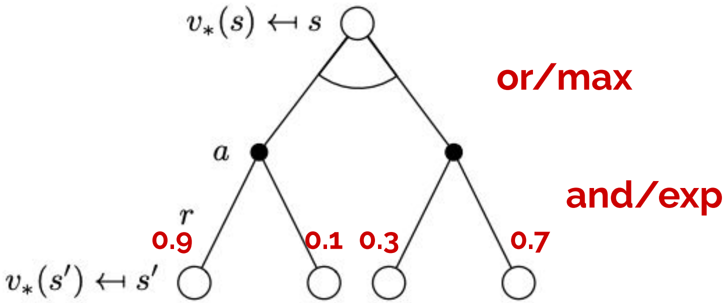
AO*

Yes, algorithms typically have their stochastic extension, where we unfold all possible states below an action (instead of only one state)

Example: A* —> **AO*** (*AND-OR*)

- OR = MAX = action selection

- AND = EXP expectation over dynamics

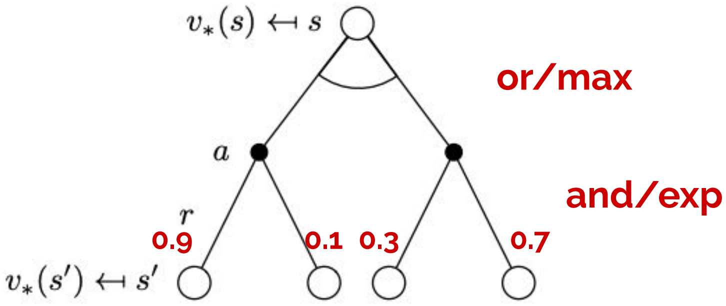


AO*

Yes, algorithms typically have their stochastic extension, where we unfold all possible states below an action (instead of only one state)

Example: A* —> **AO*** (*AND-OR*)

- OR = MAX = action selection
- AND = EXP expectation over dynamics



MDP = MAX-EXP graph = AND-OR graph

Stochastic dynamics

Q: What could be the problems of classic search in stochastic settings?

Stochastic dynamics

Q: What could be the problems of classic search in stochastic settings?

A:

1) Need an analytic model (often only a simulator is available, no exact probabilities)

2) Makes the search wide (since we need to expand all possible next states, which gives an extra/double branching factor)

Challenges of heuristic search

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available, but in many problems also faces it challenges:

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available, but in many problems also faces it challenges:

- **Required depth:** heuristic necessary to reduce depth, but often not available

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available, but in many problems also faces it challenges:

- **Required depth:** heuristic necessary to reduce depth, but often not available
- **Required width:** action pruning is risky, and stochastic dynamics make the search even wider

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available, but in many problems also faces it challenges:

- **Required depth:** heuristic necessary to reduce depth, but often not available
- **Required width:** action pruning is risky, and stochastic dynamics make the search even wider
- **Required model:** needs analytic transition probabilities, but often only a simulator is available

Challenges of heuristic search

Heuristic search is efficient in deterministic problems where a good heuristic is available, but in many problems also faces it challenges:

- **Required depth:** heuristic necessary to reduce depth, but often not available
- **Required width:** action pruning is risky, and stochastic dynamics make the search even wider
- **Required model:** needs analytic transition probabilities, but often only a simulator is available

Alternative solution: sample-based planning

Break

3. Sample-based Planning

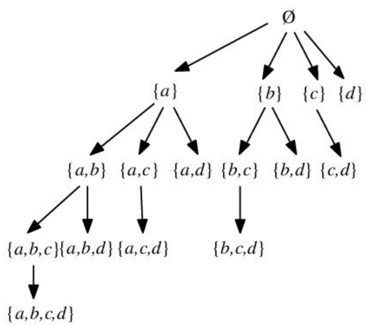
Sample-based planning

Sample-based planning

'Roll-out algorithms'

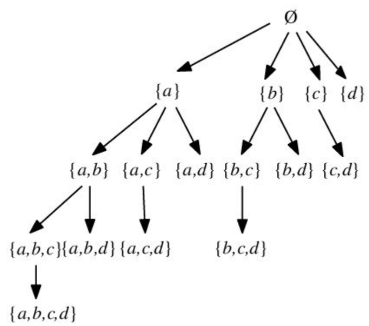
(Sutton & Barto)

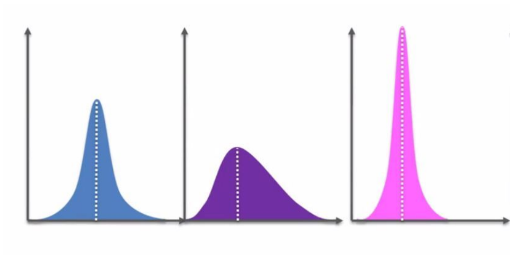
Sample-based planning



Replace the concept of systematic enumeration

Sample-based planning





Replace the concept of systematic enumeration

With statistical/probabilistic/Monte Carlo estimation of action values

Potential benefits of sample-based planning

Potential benefits of sample-based planning

Depth:

- No need for a heuristic (instead use a *Monte Carlo roll-out*)

Potential benefits of sample-based planning

Depth:

- No need for a heuristic (instead use a *Monte Carlo roll-out*)

Width:

- No need for forward pruning of actions (decide based on uncertainty principles)
- No need to expand all stochastic dynamics (simply sample one)

Potential benefits of sample-based planning

Depth:

- No need for a heuristic (instead use a *Monte Carlo roll-out*)

Width:

- No need for forward pruning of actions (decide based on uncertainty principles)
- No need to expand all stochastic dynamics (simply sample one)

Model:

- No need for exact transition probabilities (only needs a simulator)

Potential benefits of sample-based planning

Depth:

- No need for a heuristic (instead use a *Monte Carlo roll-out*)

Width:

- No need for forward pruning of actions (decide based on uncertainty principles)
- No need to expand all stochastic dynamics (simply sample one)

Model:

- No need for exact transition probabilities (only needs a simulator)

<u>Some algorithms still retain probabilistic convergence guarantees (in the limit)</u>

Monte Carlo Search

Monte Carlo Search

Main idea:

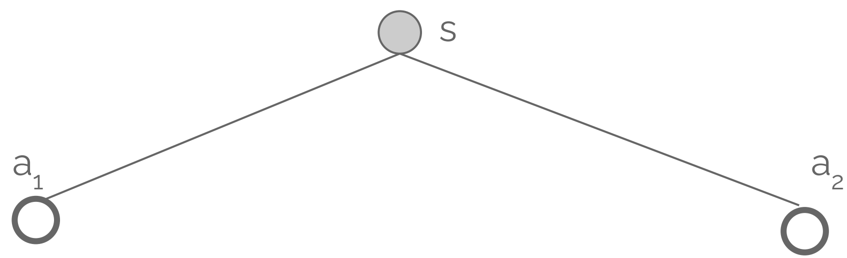
1) Use roll-outs to estimate of mean return of each action (‘*Monte Carlo estimation*’)

2) Select the action with the highest mean return (‘*Uniform bandit algorithm*’)

Monte Carlo Search

Monte Carlo Search

For each of
the A actions

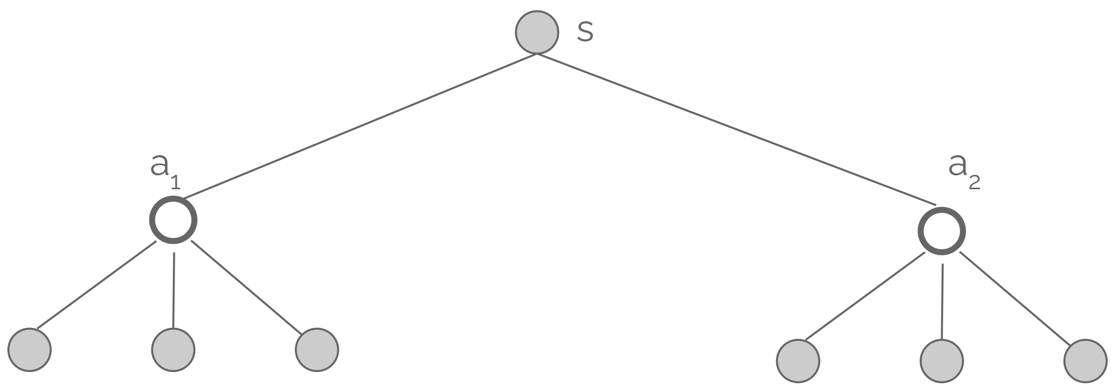


A=2

Monte Carlo Search

For each of
the A actions

Sample N
trajectories



A=2

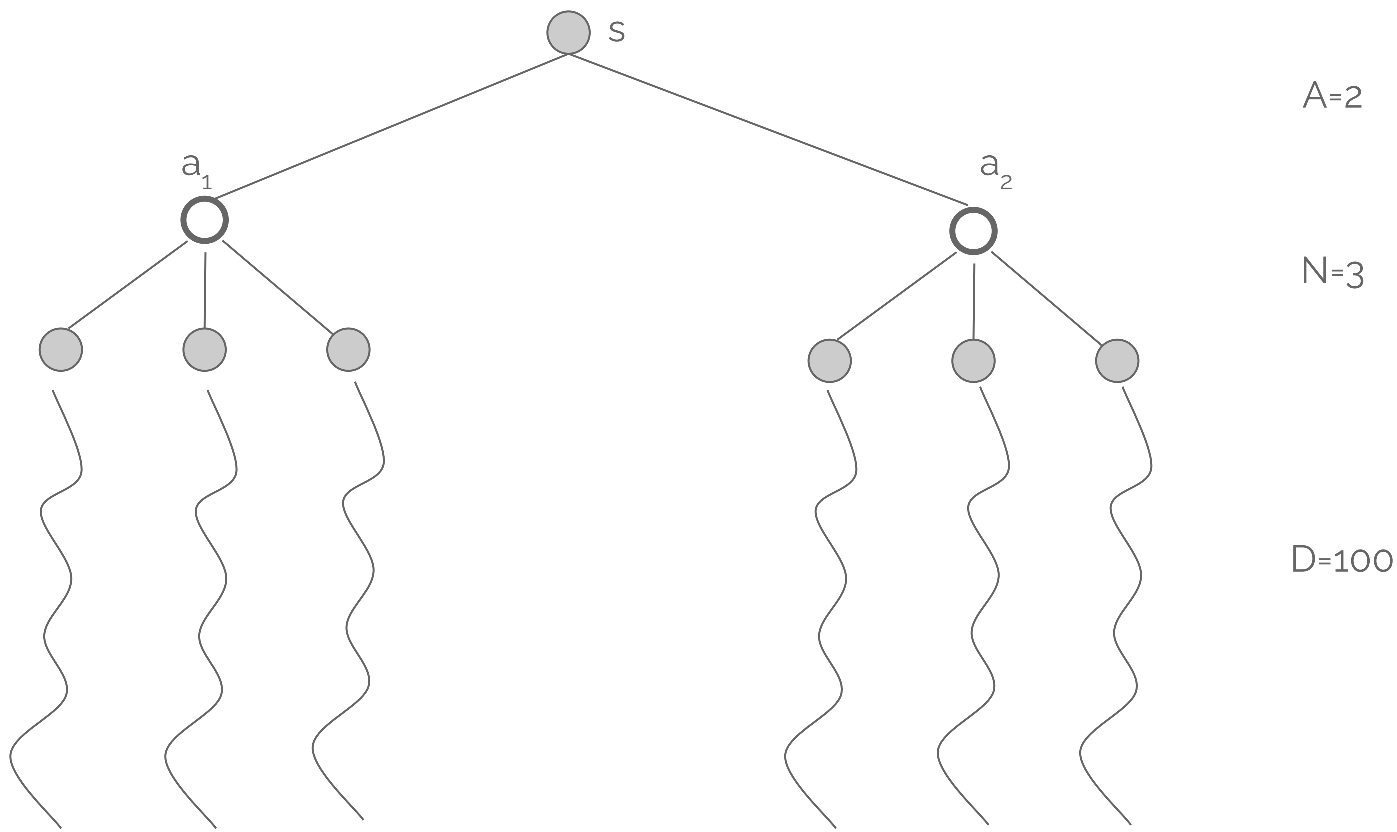
N=3

Monte Carlo Search

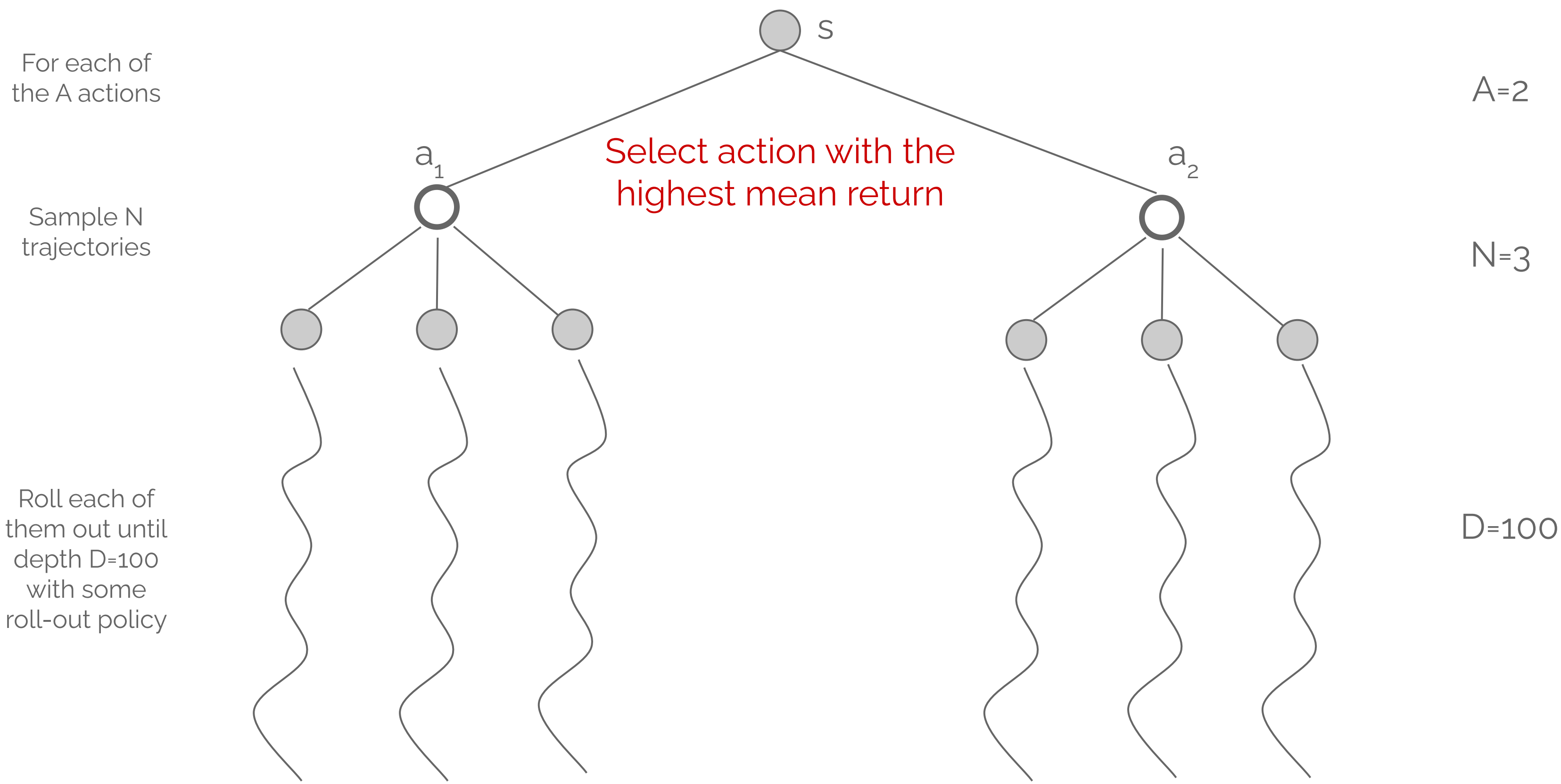
For each of the A actions

Sample N trajectories

Roll each of them out until depth D=100 with some roll-out policy



Monte Carlo Search



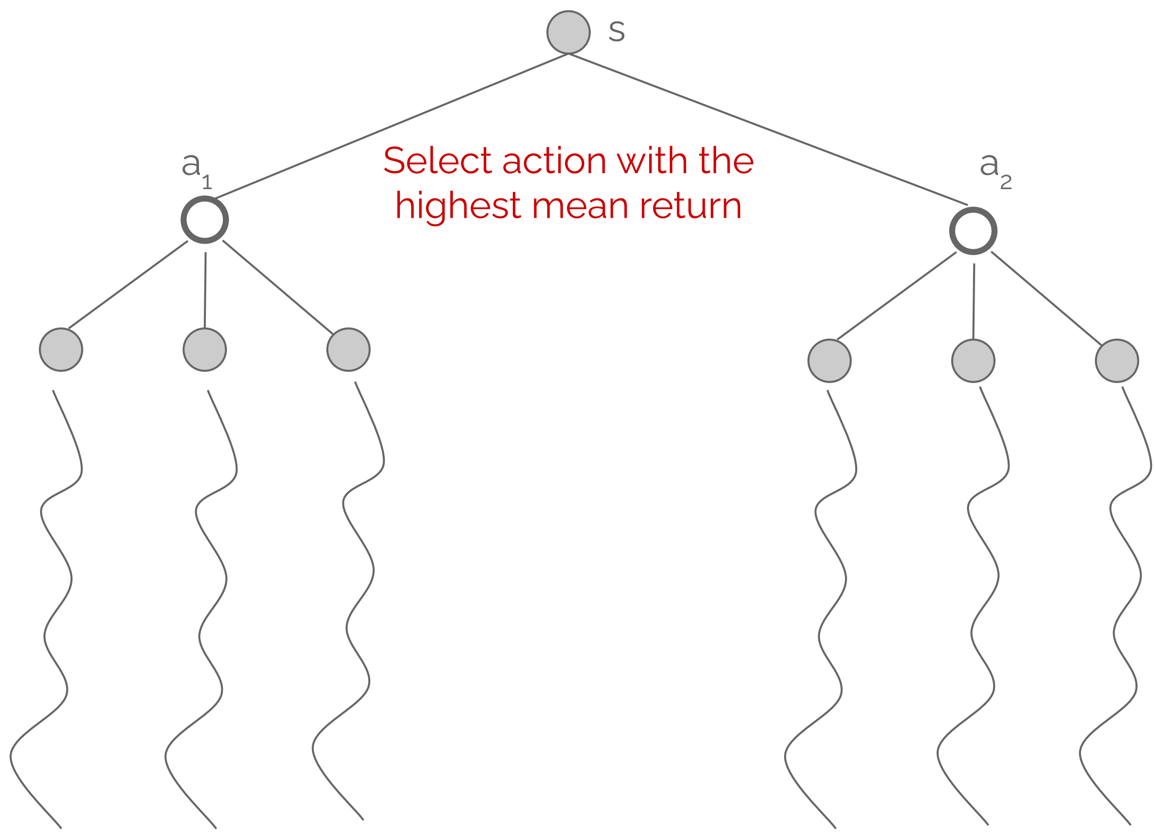
Monte Carlo Search

<u>Sample complexity</u>: ?

For each of the A actions

Sample N trajectories

Roll each of them out until depth D=100 with some roll-out policy



A=2

N=3

D=100

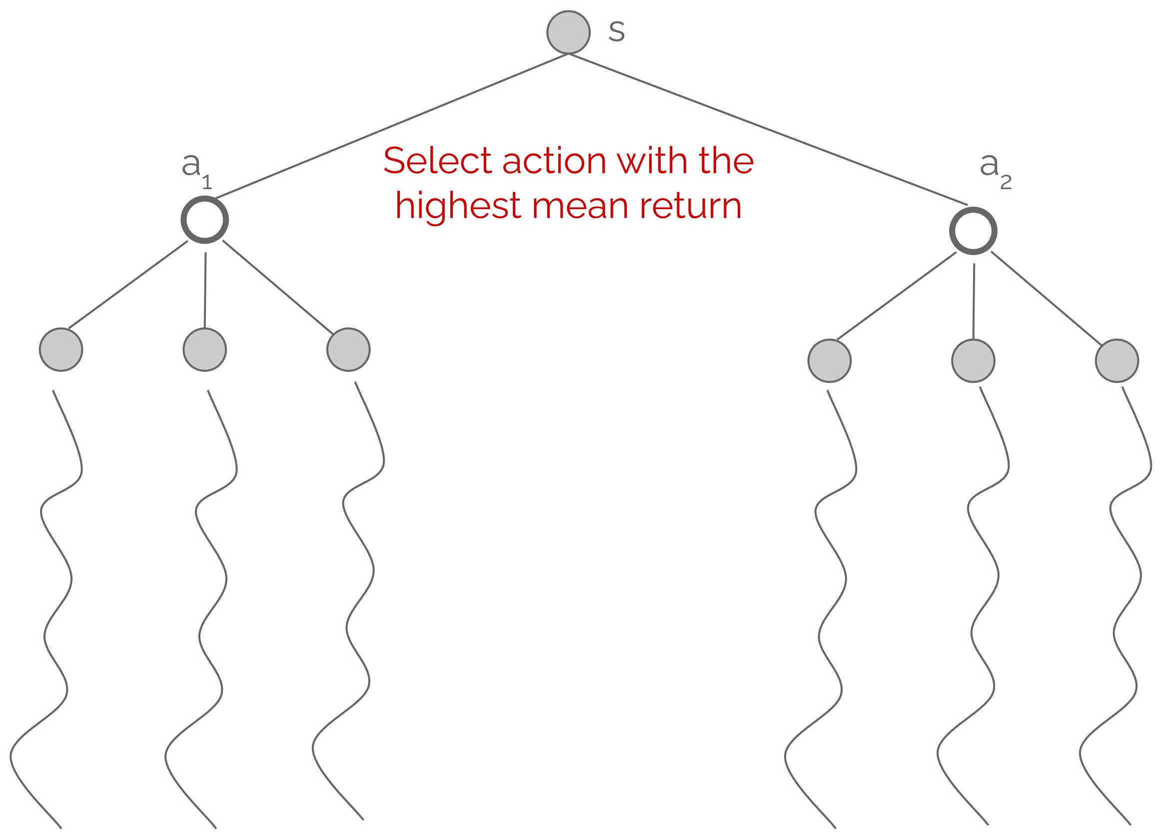
Monte Carlo Search

Sample complexity: A · N · D

For each of the A actions

Sample N trajectories

Roll each of them out until depth D=100 with some roll-out policy



A=2

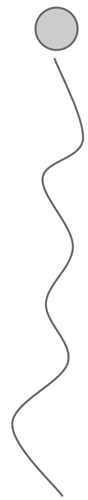
N=3

D=100

Monte Carlo Search: Roll-out policy

Monte Carlo Search: Roll-out policy

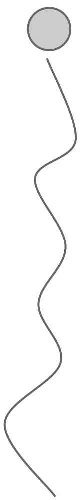
Performance of Monte Carlo search depends on the quality of the roll-out policy



Monte Carlo Search: Roll-out policy

Performance of Monte Carlo search depends on the quality of the roll-out policy

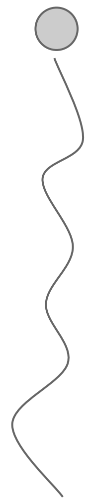
- Uninformed version: Random policy (default choice)



Monte Carlo Search: Roll-out policy

Performance of Monte Carlo search depends on the quality of the roll-out policy

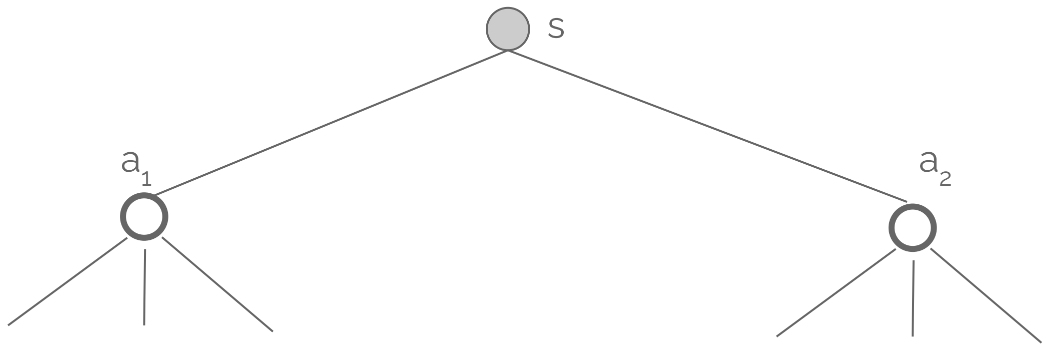
- Uninformed version: Random policy (default choice)
- Informed version: May use better prior roll-out policy when available



Monte Carlo Search: Interpretation

Monte Carlo Search: Interpretation

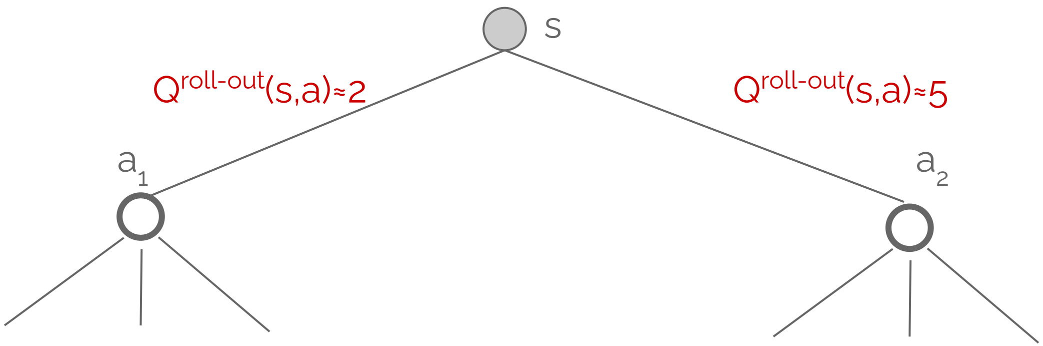
What does the mean return of each action in Monte Carlo Search actually estimate?



Monte Carlo Search: Interpretation

What does the mean return of each action in Monte Carlo Search actually estimate?

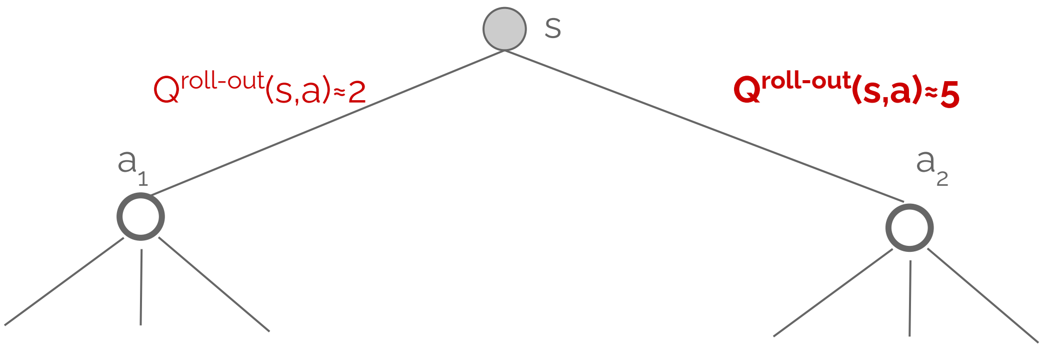
- The Q(s,a) value of that action under the roll-out policy



Monte Carlo Search: Interpretation

What does the mean return of each action in Monte Carlo Search actually estimate?

- The Q(s,a) value of that action under the roll-out policy

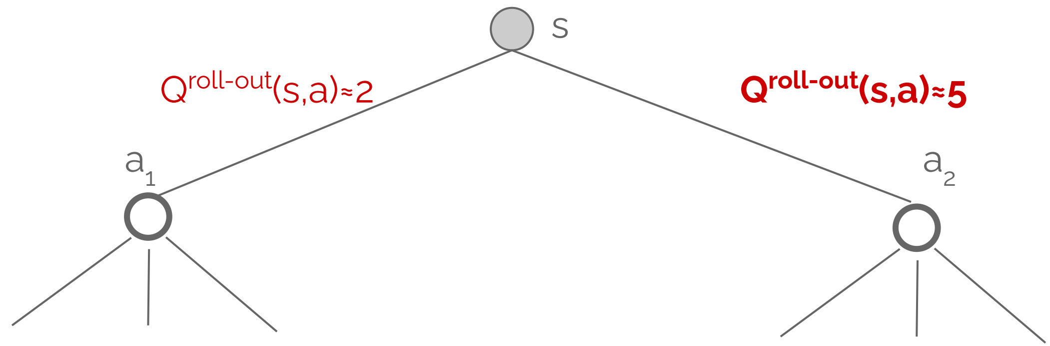


We then greedily select the action with the highest value

Monte Carlo Search: Interpretation

What does the mean return of each action in Monte Carlo Search actually estimate?

- The Q(s,a) value of that action under the roll-out policy



We then greedily select the action with the highest value

- A form of local, one-step policy improvement over the prior roll-out policy

Monte Carlo Search: Downside

Can you think of a downside of Monte Carlo Search?

Monte Carlo Search: Downside

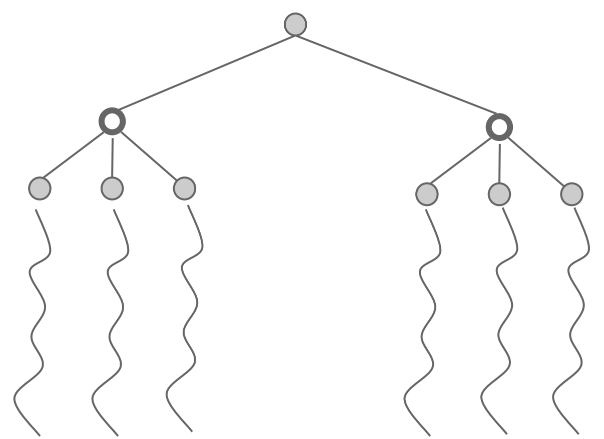
Can you think of a downside of Monte Carlo Search?

It does not store any statistics or do any policy improvement below depth 1

Monte Carlo Search: Downside

Can you think of a downside of Monte Carlo Search?

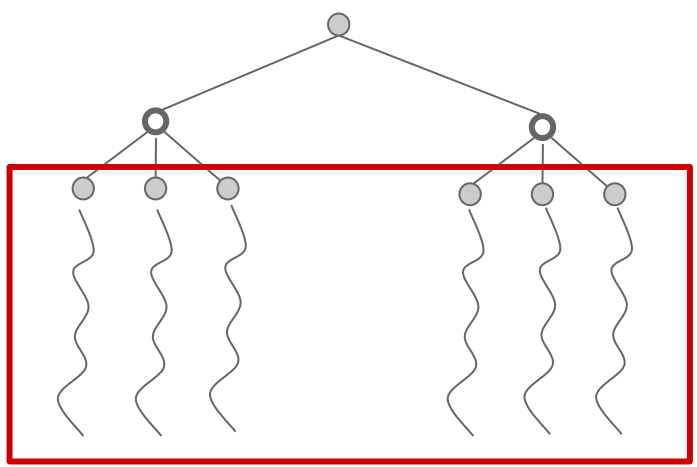
It does not store any statistics or do any policy improvement below depth 1



Monte Carlo Search: Downside

Can you think of a downside of Monte Carlo Search?

It does not store any statistics or do any policy improvement below depth 1



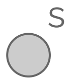
black box: not changing your policy based on the search

Sparse sampling

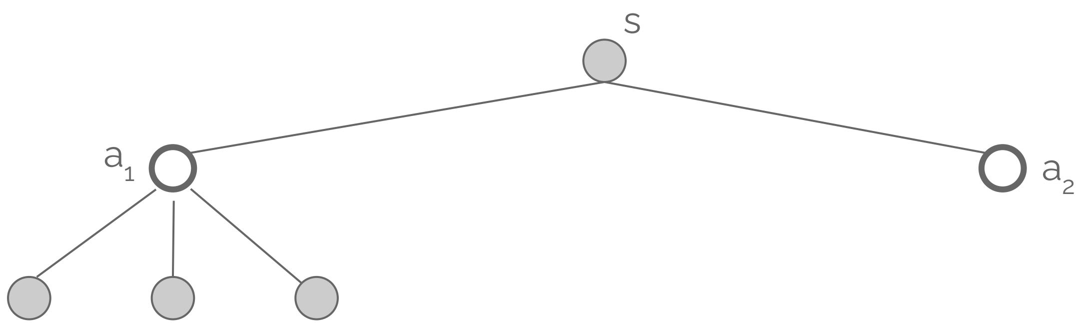
Sparse sampling

Main idea: Repeat the same process at deeper levels

Sparse sampling



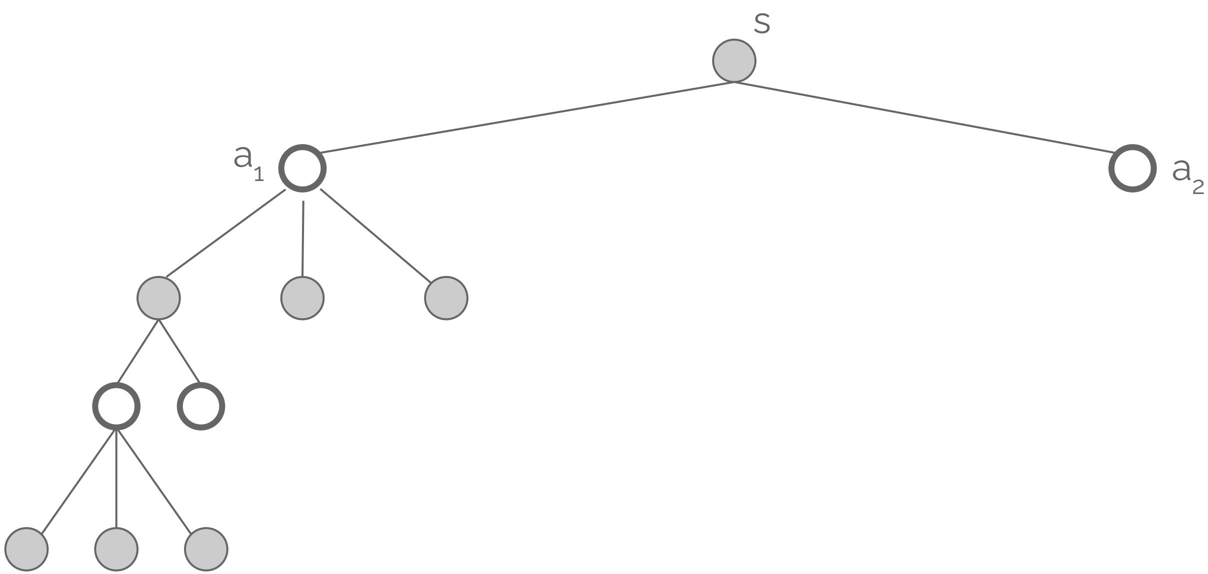
Sparse sampling



A=2

N=3

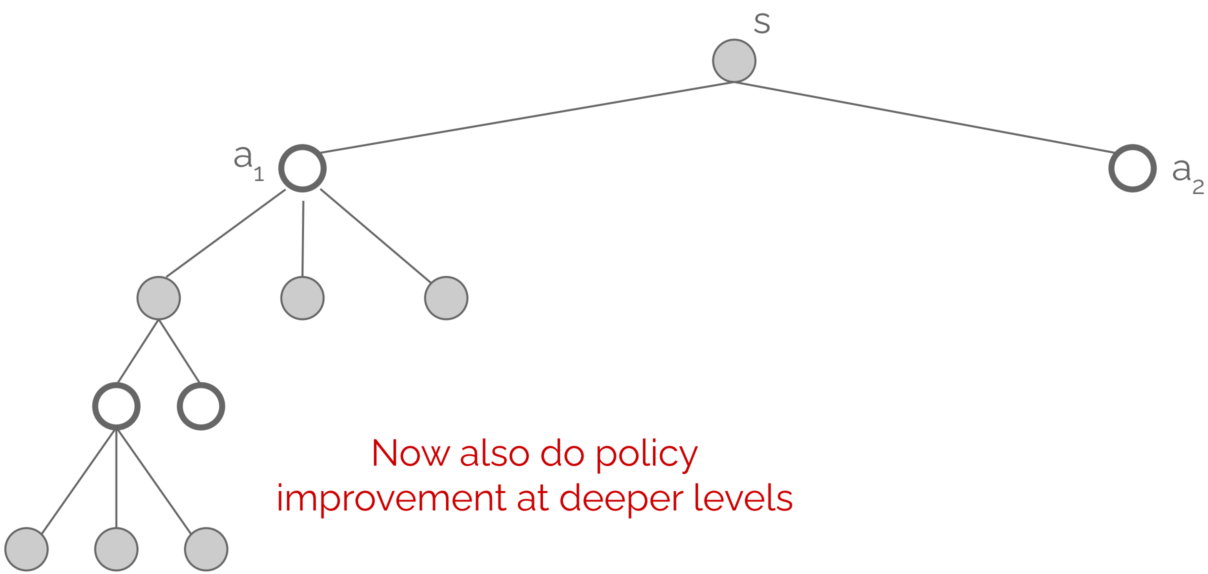
Sparse sampling



A=2

N=3

Sparse sampling

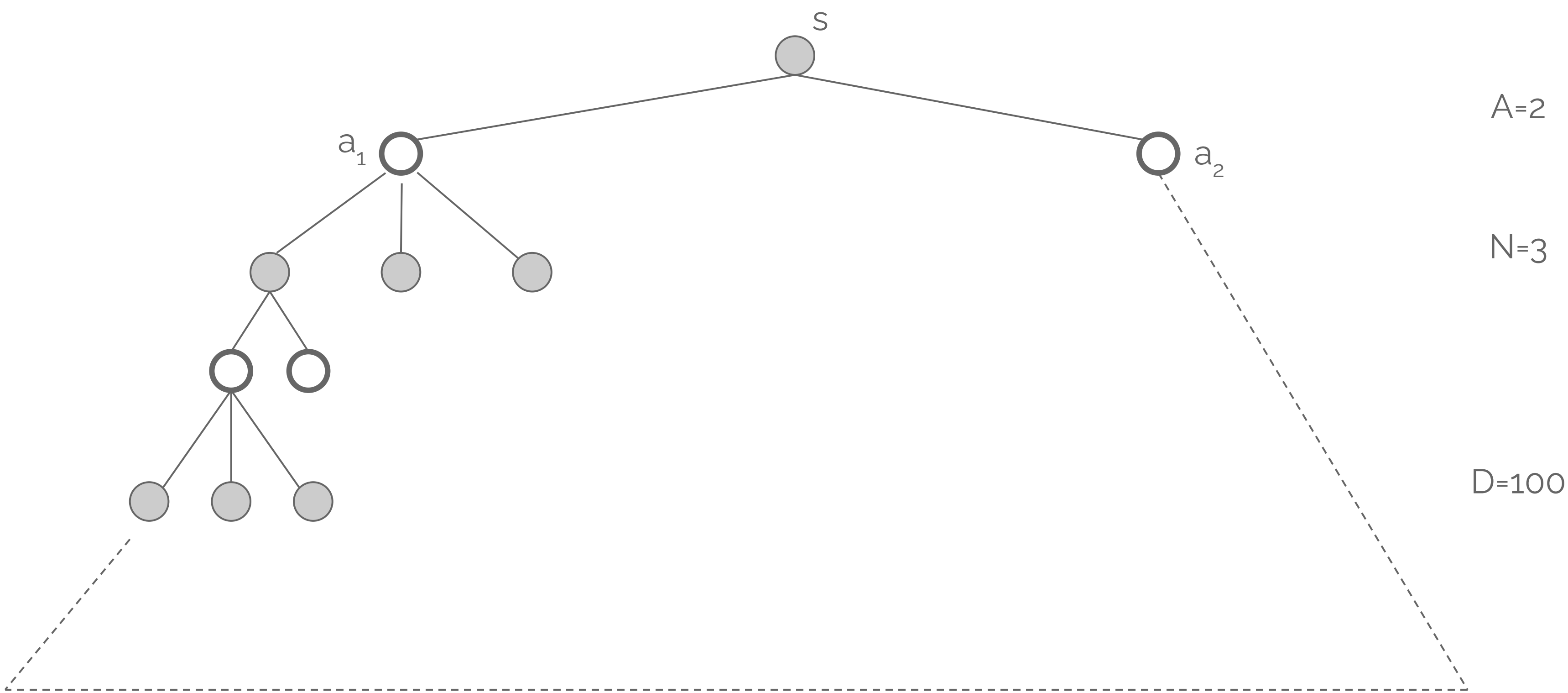


A=2

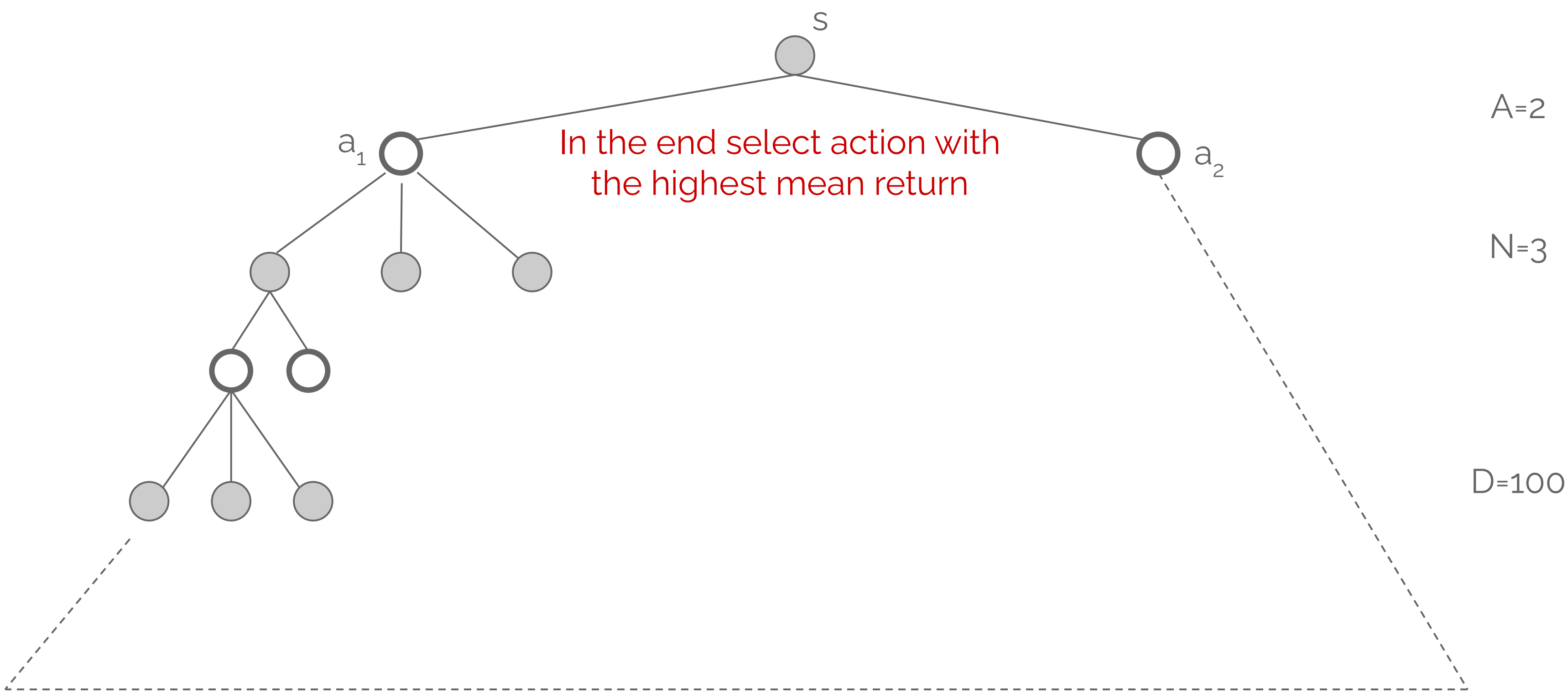
N=3

Now also do policy
improvement at deeper levels

Sparse sampling

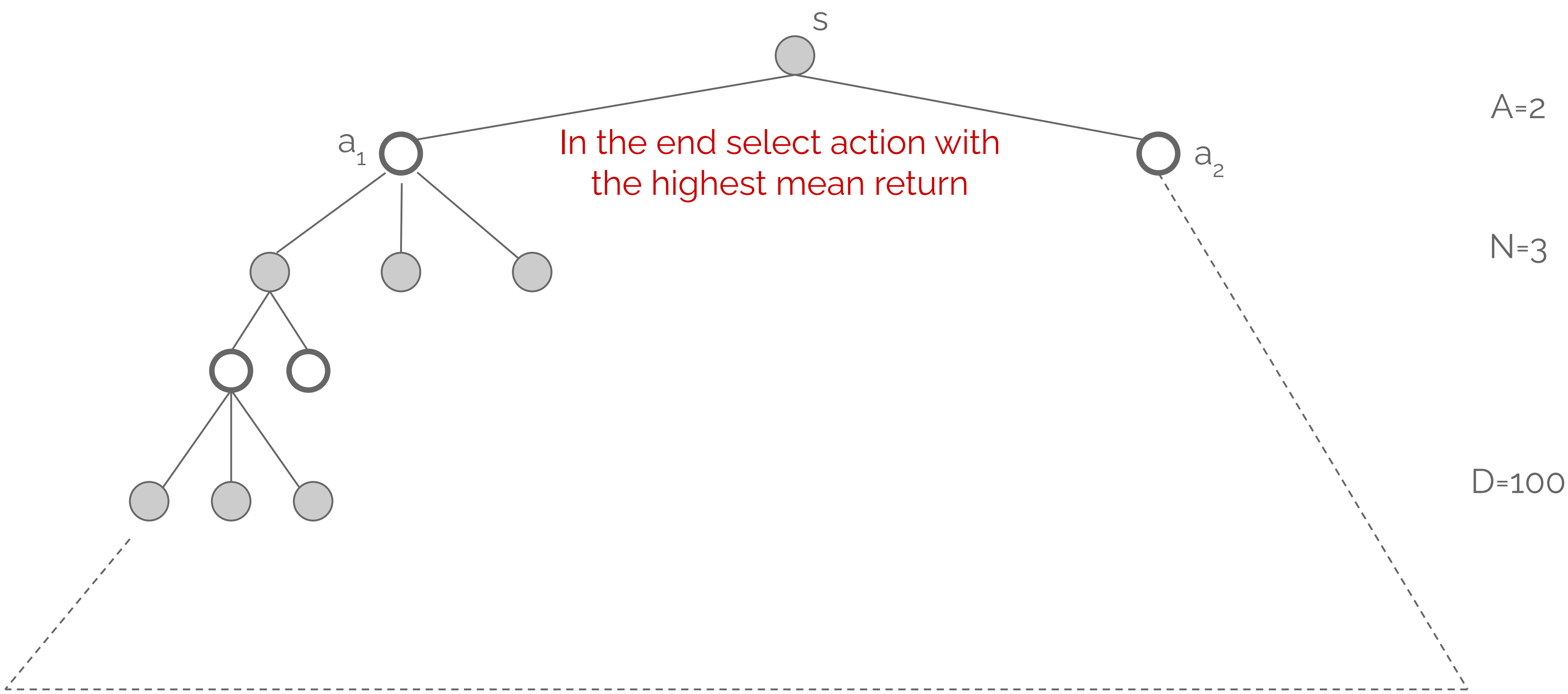


Sparse sampling



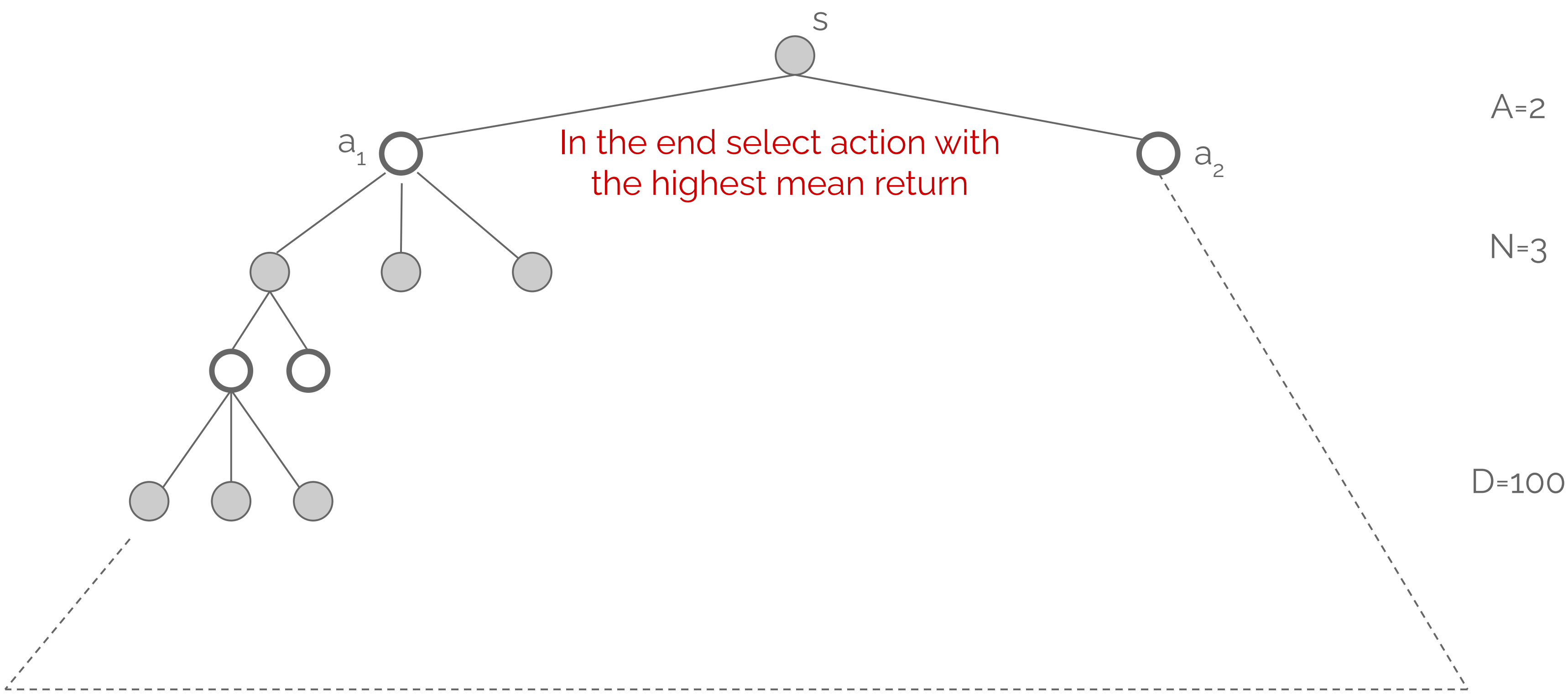
Sample complexity: ?

Sparse sampling



Sample complexity: $(A \cdot N)^D$

Sparse sampling



Sparse sampling

Sparse sampling

What is the problem with sparse sampling?

Sparse sampling

What is the problem with sparse sampling?

- It's very expensive (exponential in D). Not complete enumeration, but still spends much effort in poorly performing directions.

Sparse sampling

What is the problem with sparse sampling?

- It's very expensive (exponential in D). Not complete enumeration, but still spends much effort in poorly performing directions.

Can you think of a solution?

Sparse sampling

What is the problem with sparse sampling?

- It's very expensive (exponential in D). Not complete enumeration, but still spends much effort in poorly performing directions.

Can you think of a solution?

- Adaptive Monte Carlo methods → replace Uniform sampling with an adaptive bandit algorithm (Ch. 2) that focuses in directions where initial samples perform well (trading-off exploration & exploitation).

Monte Carlo Tree Search (MCTS)

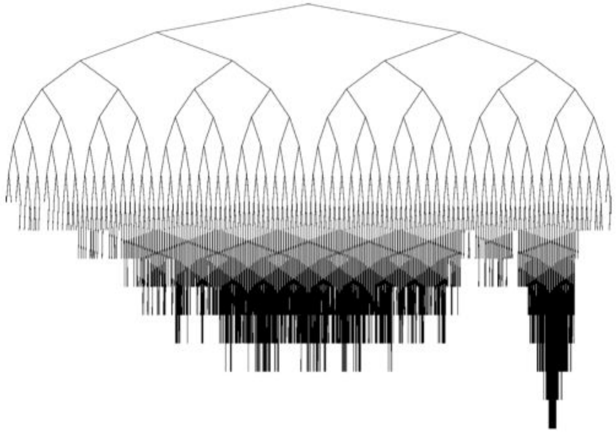
Monte Carlo Tree Search (MCTS)

Main idea: iteratively apply adaptive bandit algorithm at every depth

Monte Carlo Tree Search (MCTS)

Main idea: iteratively apply adaptive bandit algorithm at every depth

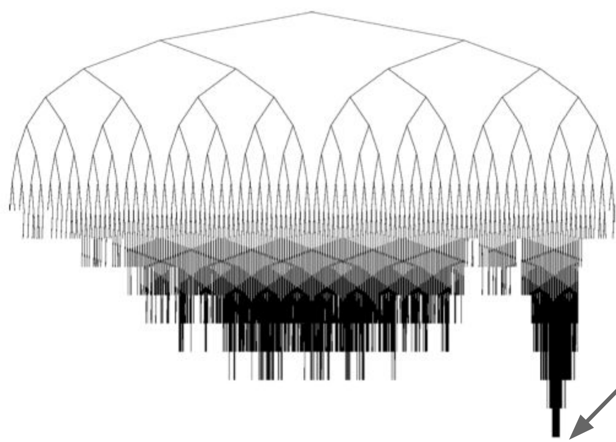
Naturally produces an asymmetric search tree



Monte Carlo Tree Search (MCTS)

Main idea: iteratively apply adaptive bandit algorithm at every depth

Naturally produces an asymmetric search tree



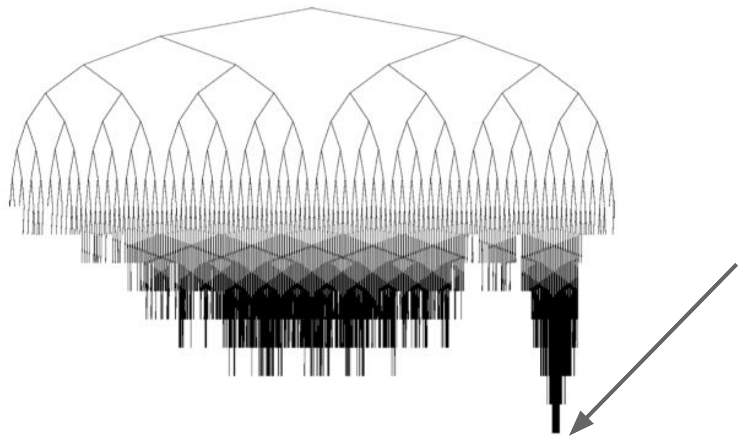
Extends deeper in directions where initial samples giving promising returns

(sample-based equivalent of prioritized search)

Monte Carlo Tree Search (MCTS)

Main idea: iteratively apply adaptive bandit algorithm at every depth

Naturally produces an asymmetric search tree



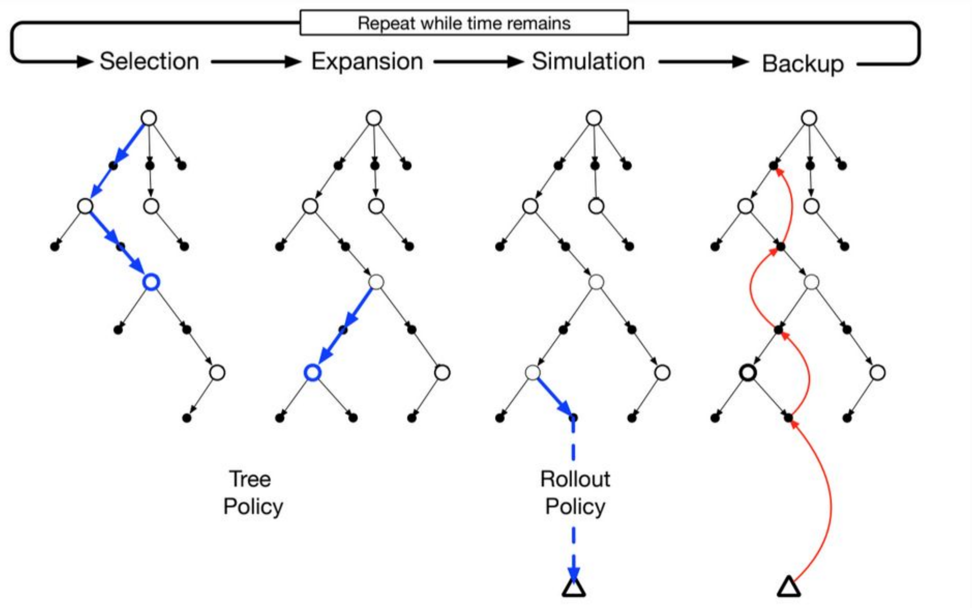
Extends deeper in directions where initial samples giving promising returns

(sample-based equivalent of prioritized search)

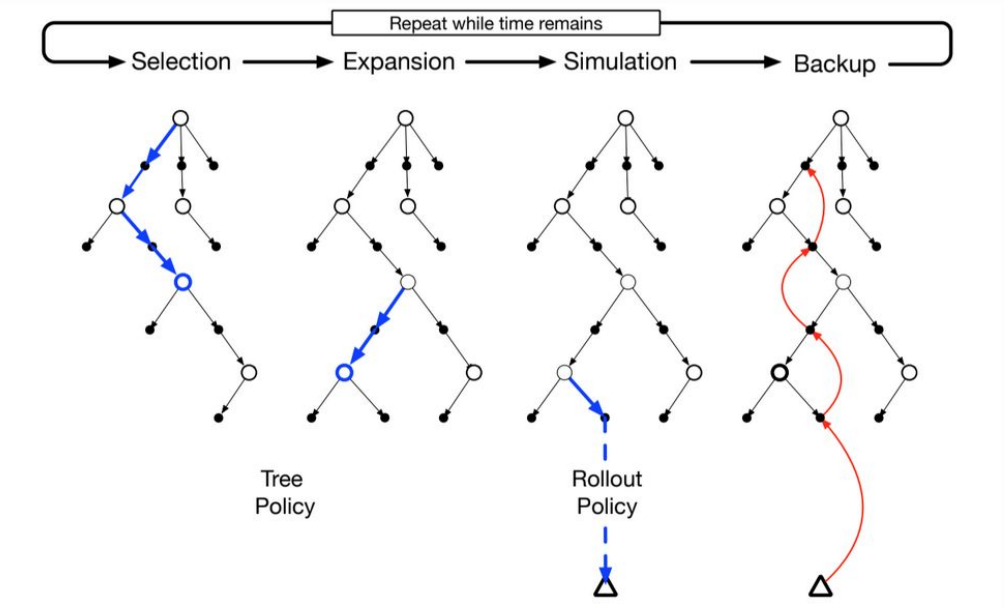
Breakthrough performance in the game of Go

Four phases of MCTS

Four phases of MCTS



Four phases of MCTS

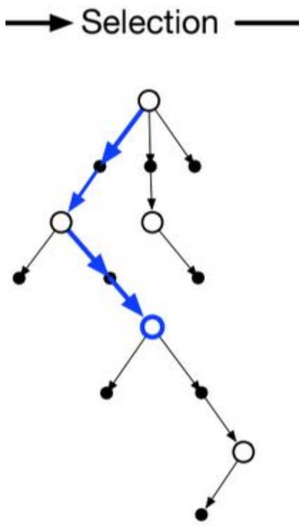


Each iteration makes one roll-out, that moves through four phases.

Selection

Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

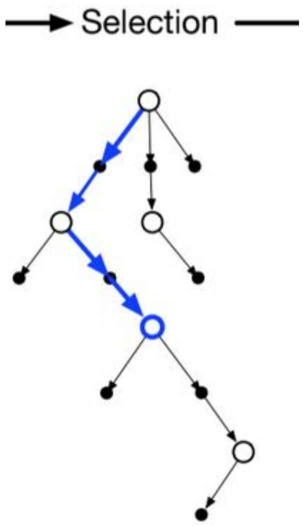


Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)

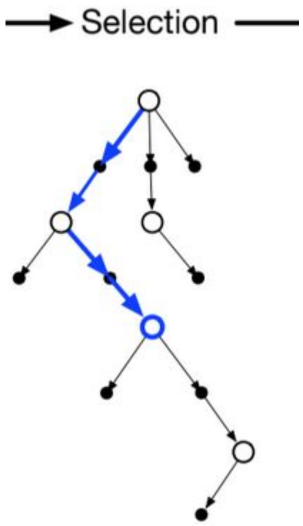
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$



Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Select the action with the highest

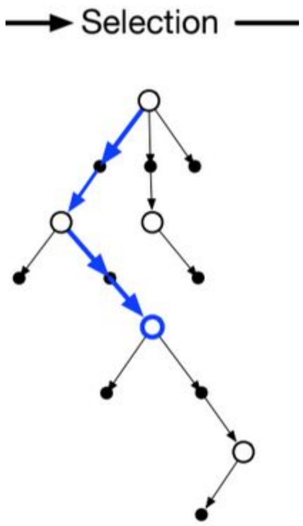
Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

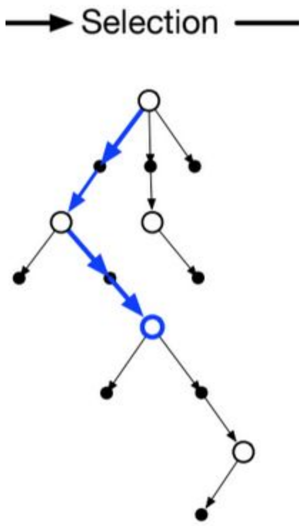
mean return of previous traces
(exploitation)



Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)



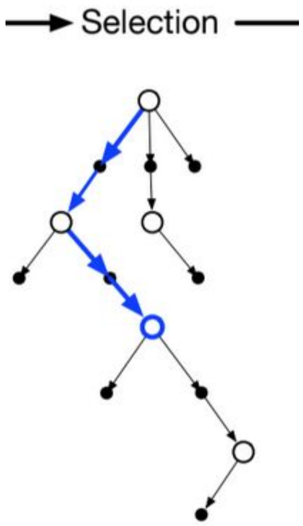
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

c = constant we empirically tune
(higher c → more exploration)

Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

n(s) = number of traces through state
n(s,a) = number of traces through state-action
(exploration: lower n(s,a) → higher second term)

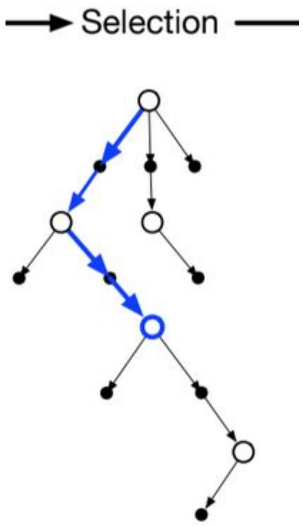
Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

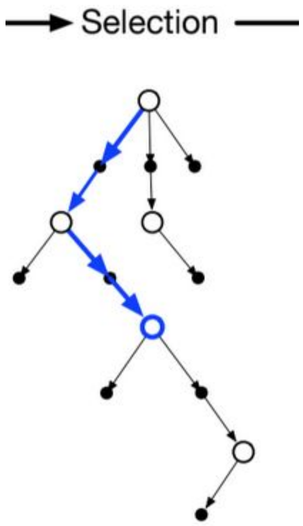
- What is the UCT value of an untried action [n(s,a)=0]?



Selection

- Apply a bandit algorithm to select the most promising action (balances exploration & exploitation)

- Most common choice:
  Upper Confidence Bounds applied to Trees (**UCT**)



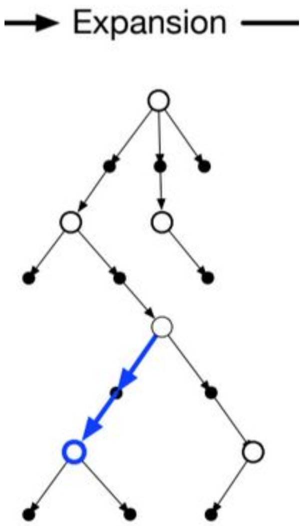
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

- What is the UCT value of an untried action [n(s,a)=0]?
  - We treat the second term as infinity (divide over 0) and therefore **always select an untried action when available (=expand)**

Expansion

Expansion

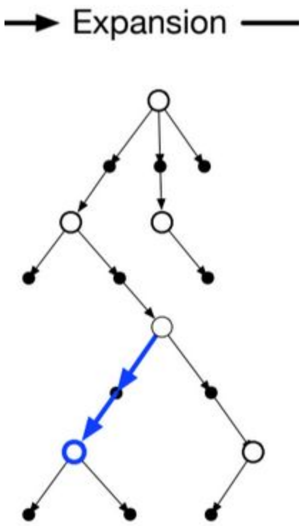
- Once we reach an unvisited action, expand it (i.e. add child state and its actions to the tree)



Expansion

- Once we reach an unvisited action, expand it (i.e. add child state and its actions to the tree)

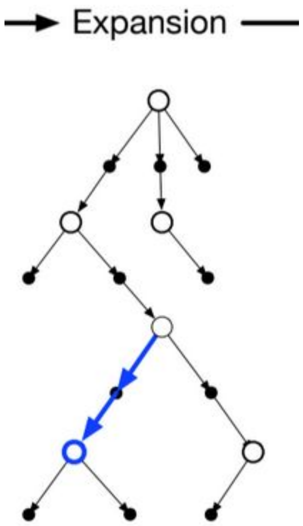
- Each iteration expands the tree with only one new state. Why?



Expansion

- Once we reach an unvisited action, expand it (i.e. add child state and its actions to the tree)

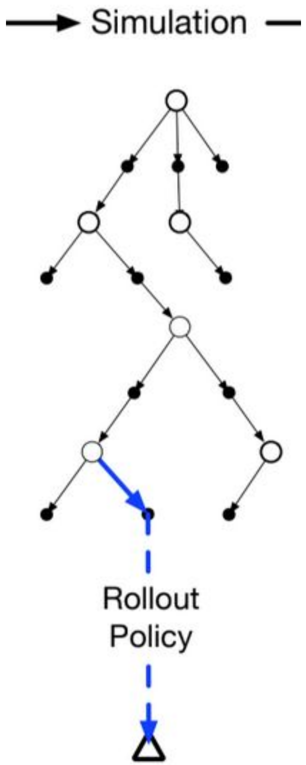
- Each iteration expands the tree with only one new state. Why?
    - Could store everything below but eats away memory and compute. We only start storing a deeper state once we repeatedly visited that direction.



Simulation

Simulation

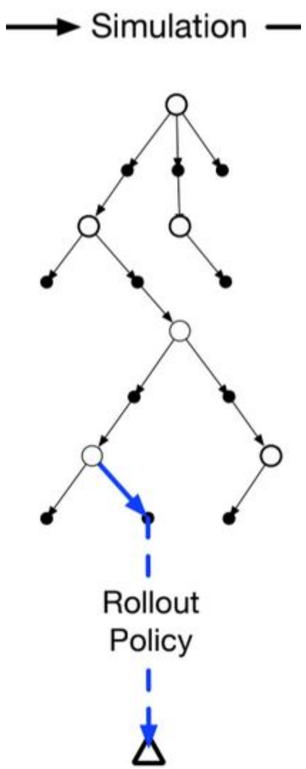
Again use Monte Carlo roll-out as an estimate of the value of the expanded state



Simulation

Again use Monte Carlo roll-out as an estimate of the value of the expanded state

- Default = random policy, can use better prior policy when available.

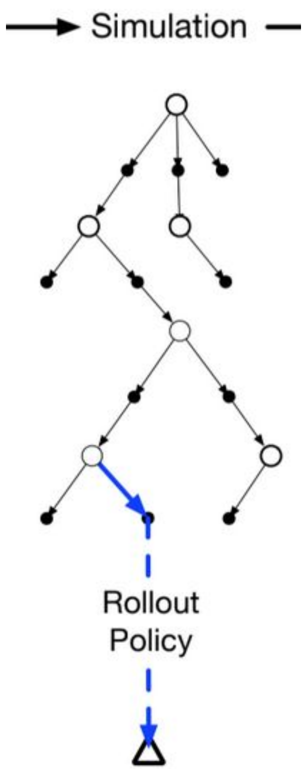


Simulation

Again use Monte Carlo roll-out as an estimate of the value of the expanded state

- Default = random policy, can use better prior policy when available.

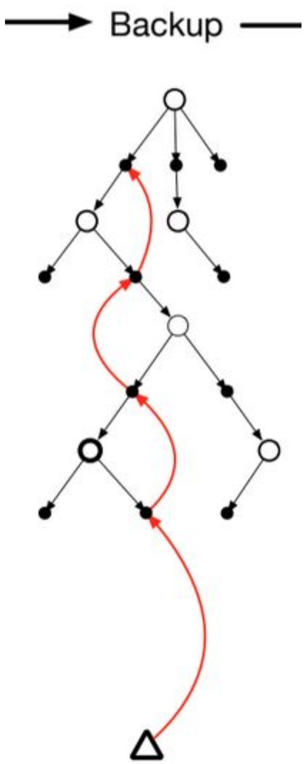
- Note max total depth D (so if current leaf at depth 5 with D=100, then you roll-out for length 95)



Backup

Backup

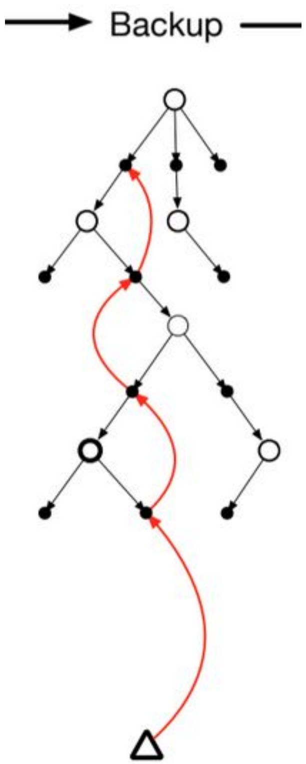
Update the statistics throughout the tree to direct the next iteration



Backup

Update the statistics throughout the tree to direct the next iteration

- Action nodes: store visit count **n(s,a)** and mean return **Q(s,a)**

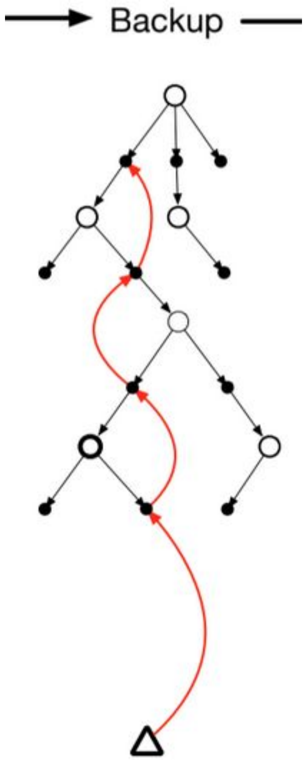


Backup

Update the statistics throughout the tree to direct the next iteration

- Action nodes: store visit count **n(s,a)** and mean return **Q(s,a)**

May also store the return sum of all traces through (s,a) as **R_{sum}(s,a)** and compute **Q(s,a) = R_{sum}(s,a)/n(s,a)**

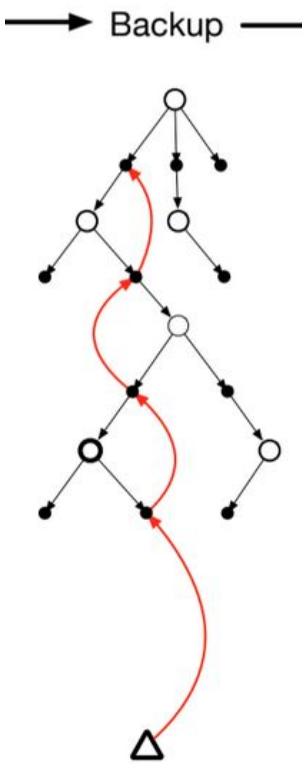


Backup

Update the statistics throughout the tree to direct the next iteration

- Action nodes: store visit count **n(s,a)** and mean return **Q(s,a)**
- State nodes: store visit count **n(s)**

May also store the return sum of all traces through (s,a) as **R_{sum}(s,a)** and compute **Q(s,a) = R_{sum}(s,a)/n(s,a)**



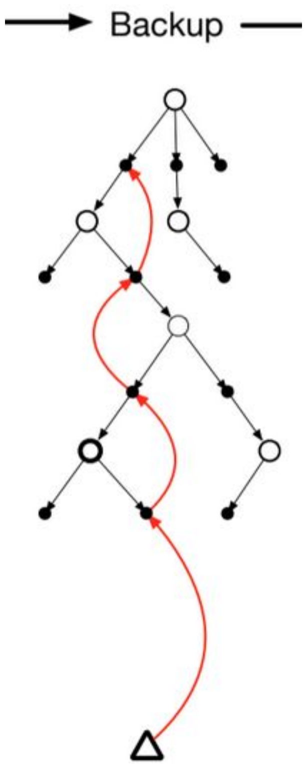
Backup

Update the statistics throughout the tree to direct the next iteration

- Action nodes: store visit count **n(s,a)** and mean return **Q(s,a)**
- State nodes: store visit count **n(s)**

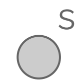
Or compute them as $\boldsymbol{\Sigma_a}$ **n(s,a)** once needed

May also store the return sum of all traces through (s,a) as $\mathbf{R_{sum}(s,a)}$ and compute $\mathbf{Q(s,a) = R_{sum}(s,a)/n(s,a)}$



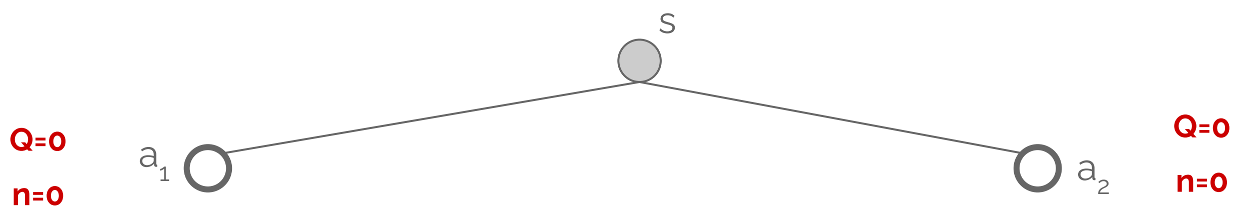
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up



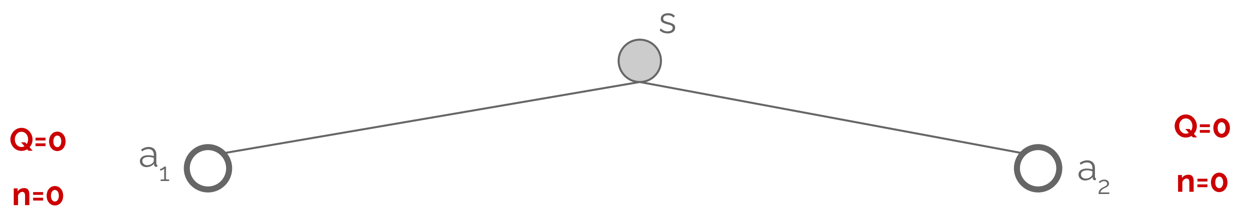
Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up



1. **Select**
2. Expand
3. Roll-out
4. Back-up

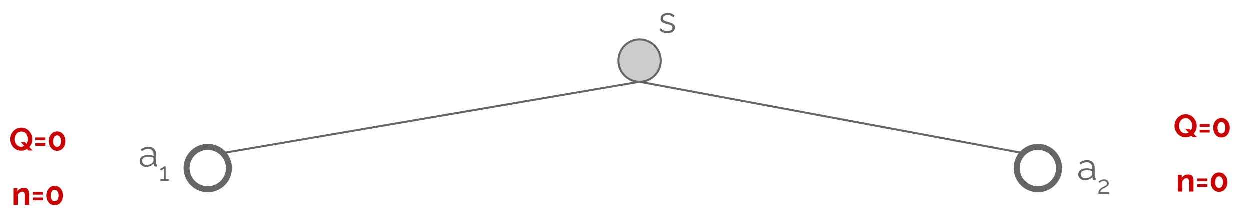
Monte Carlo Tree Search



Initialize mean action return (**Q(s,a)**) and count (**n(s,a)**) to 0

Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up



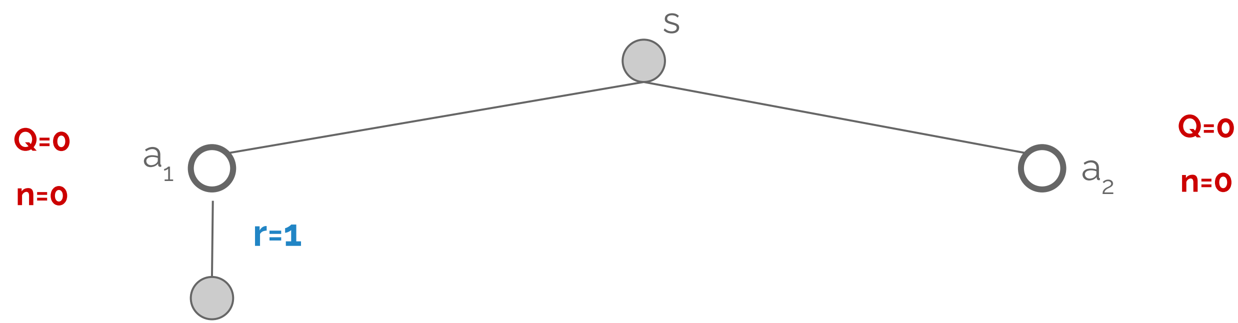
Select next action based on UCT rule:

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Both actions untried (n=0), randomly pick one

Monte Carlo Tree Search

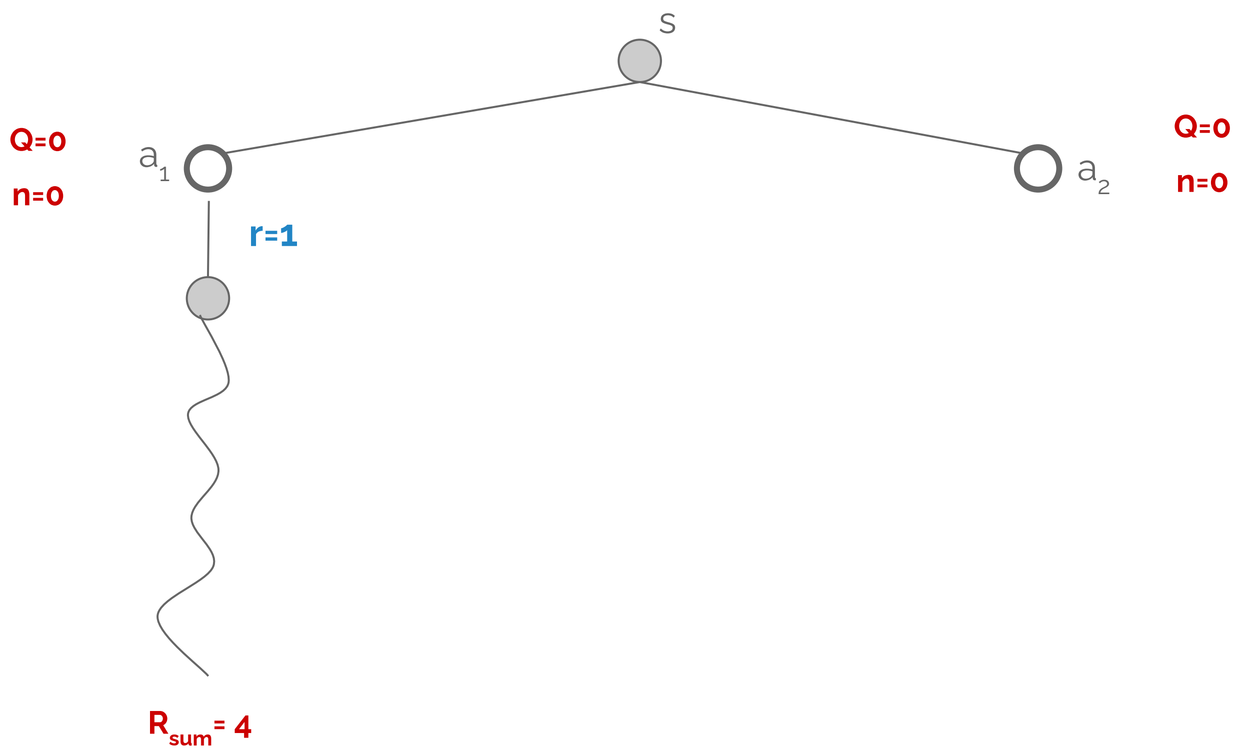
1. Select
2. **Expand**
3. Roll-out
4. Back-up



We expand the tree once we encounter an untried action

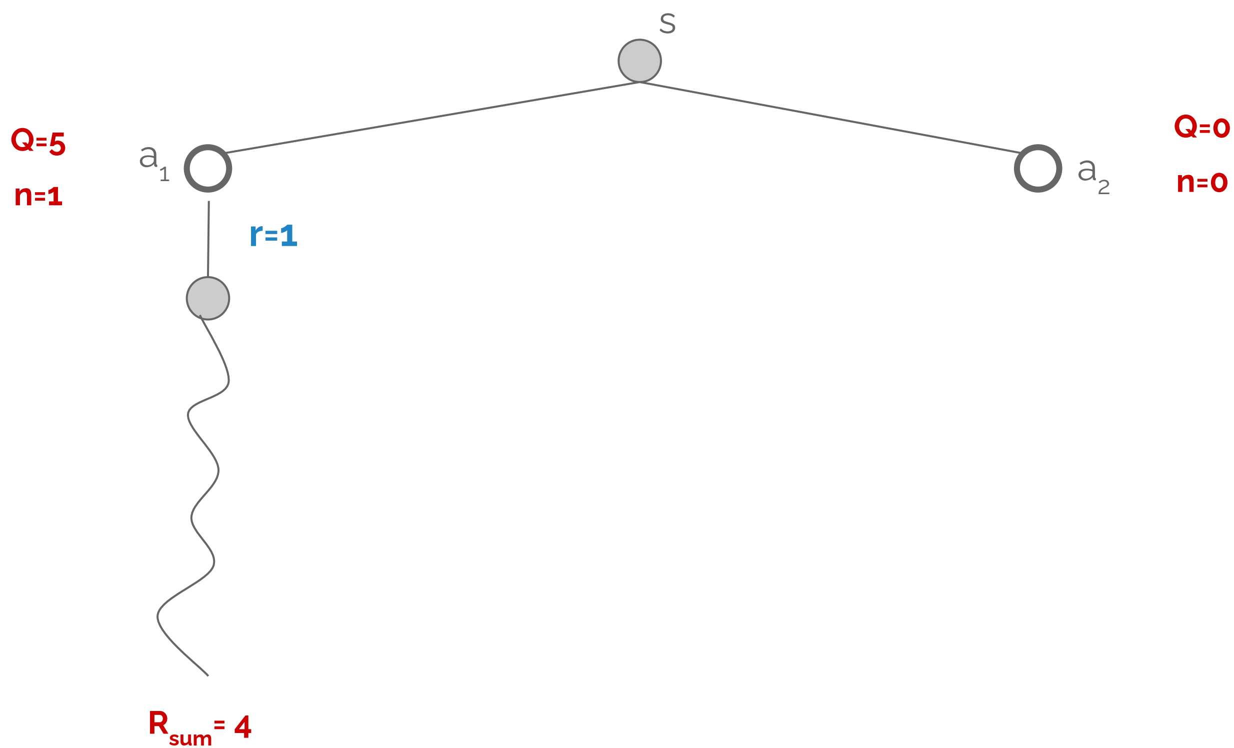
Monte Carlo Tree Search

1. Select
2. Expand
3. **Roll-out**
4. Back-up



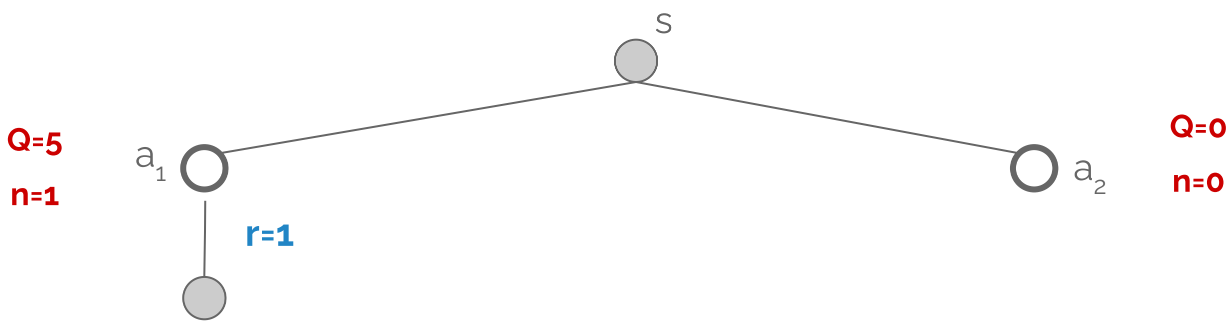
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



Monte Carlo Tree Search

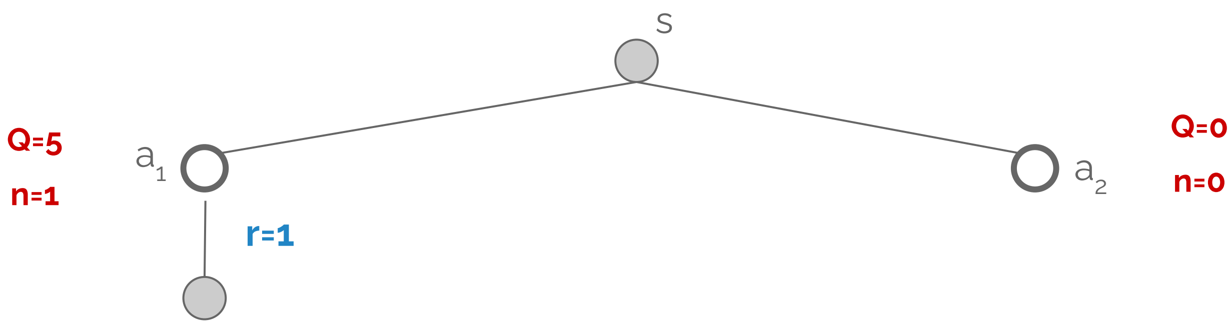
1. Select
2. Expand
3. Roll-out
4. Back-up



First iteration, repeat!

Monte Carlo Tree Search

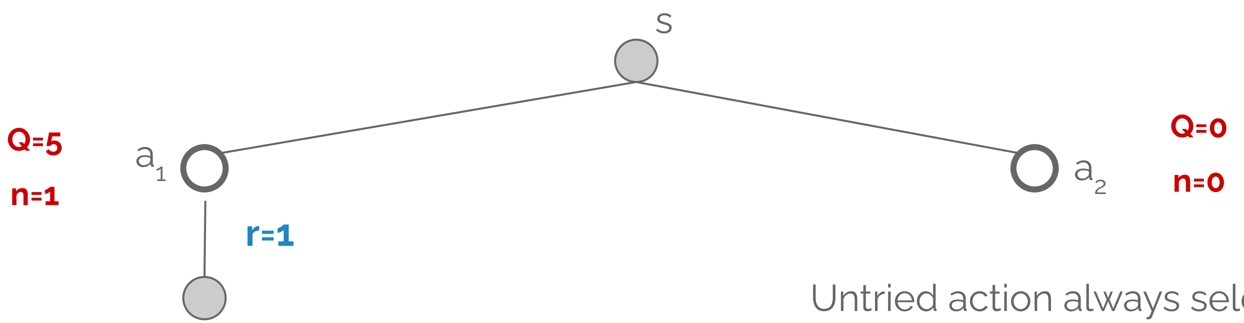
1. **Select**
2. Expand
3. Roll-out
4. Back-up



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up

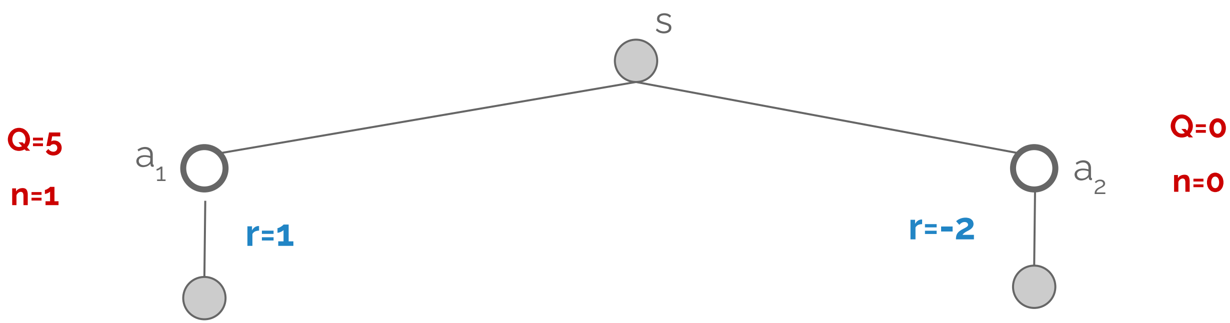


Untried action always selected
(=expand a_2)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

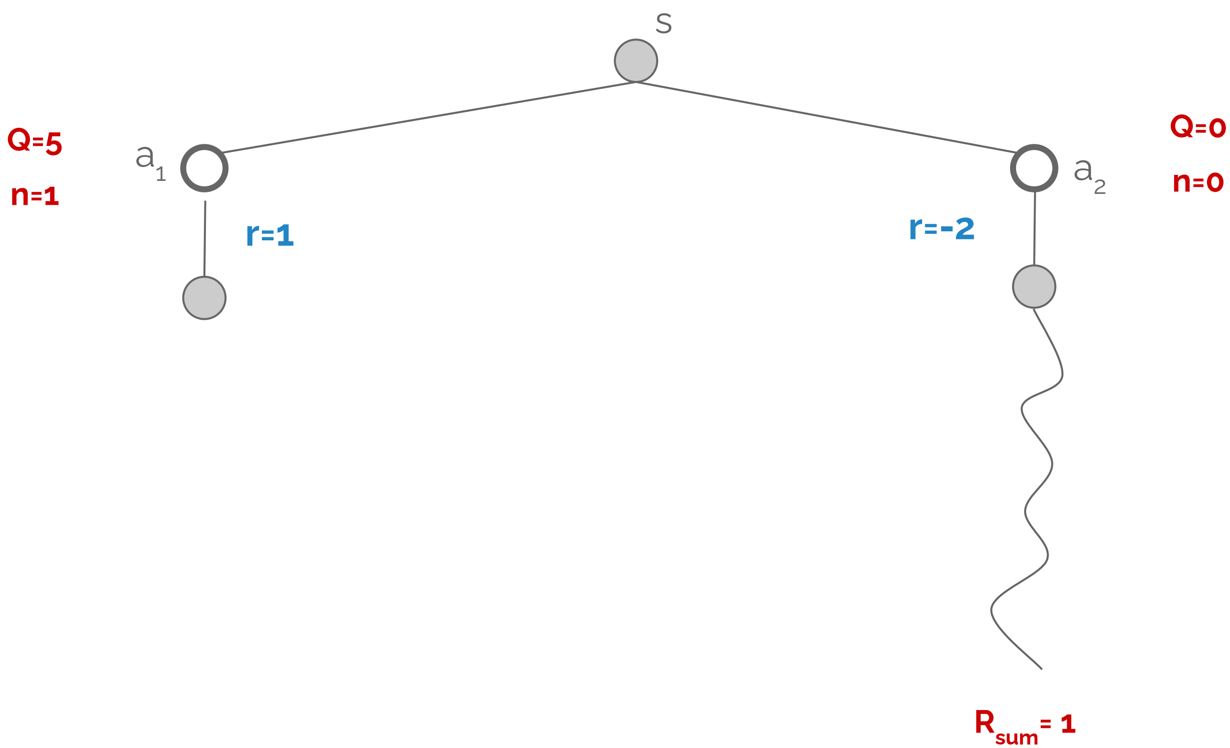
Monte Carlo Tree Search

1. Select
2. **Expand**
3. Roll-out
4. Back-up



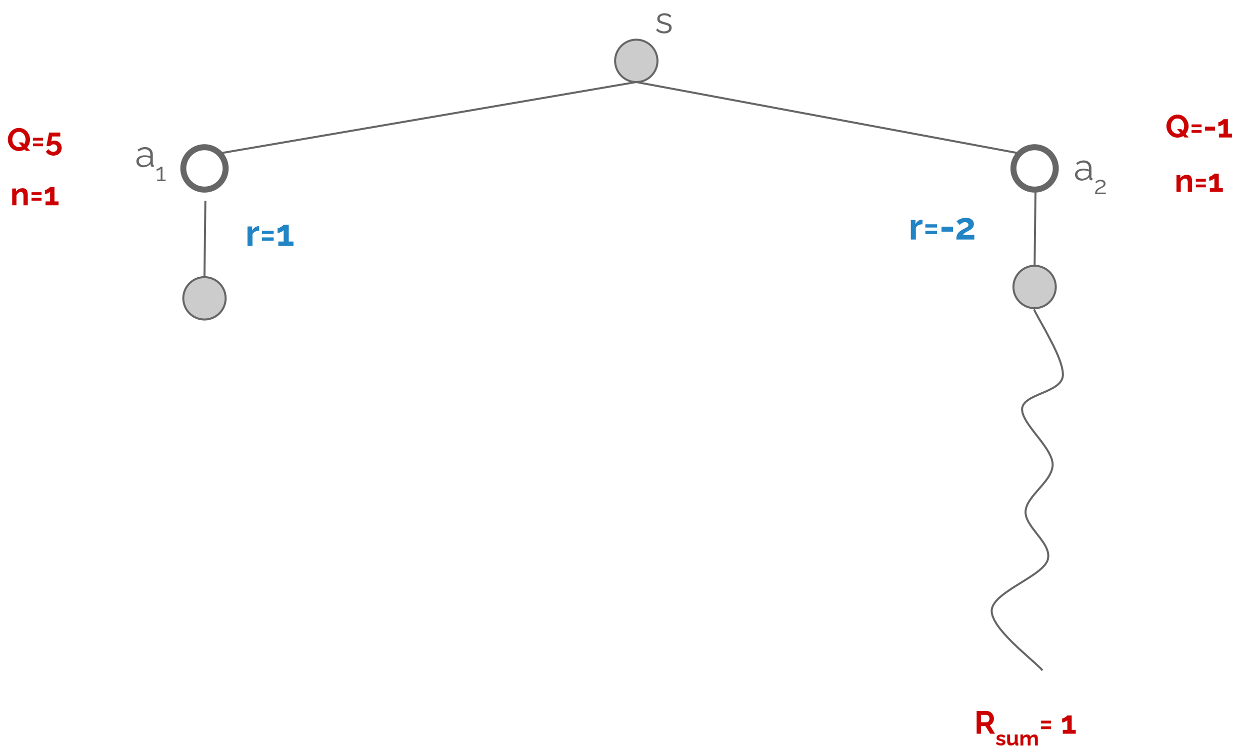
Monte Carlo Tree Search

1. Select
2. Expand
3. **Roll-out**
4. Back-up



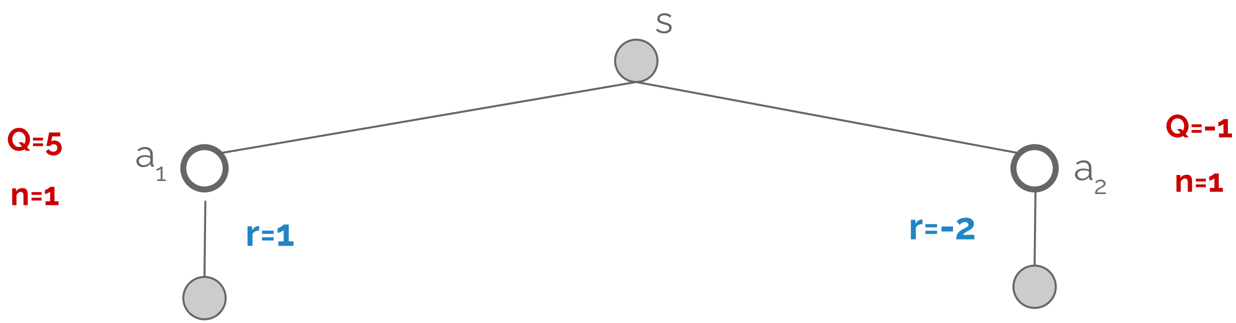
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



Monte Carlo Tree Search

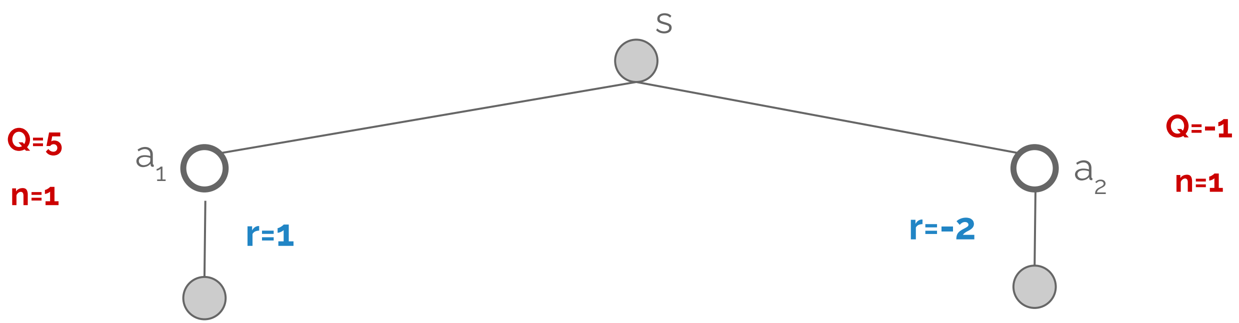
1. Select
2. Expand
3. Roll-out
4. Back-up



Second iteration, repeat!

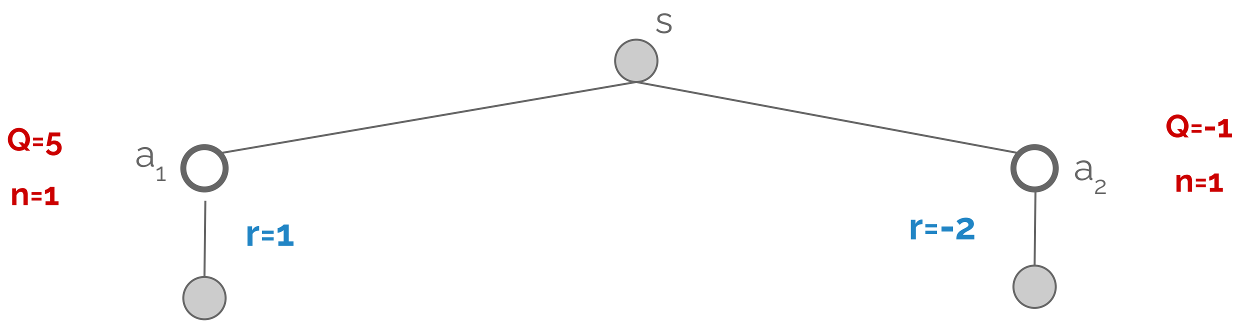
Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up



Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up

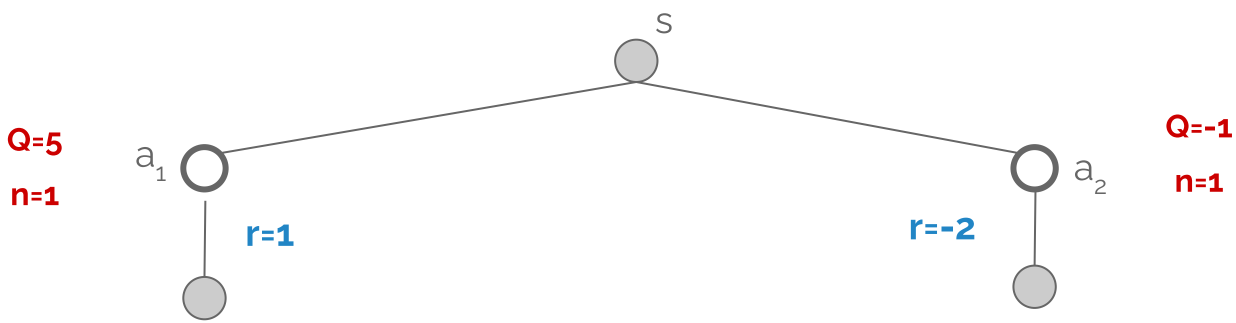


No untried action left, so now we really use the UCT select rule (assume c=1.0)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up



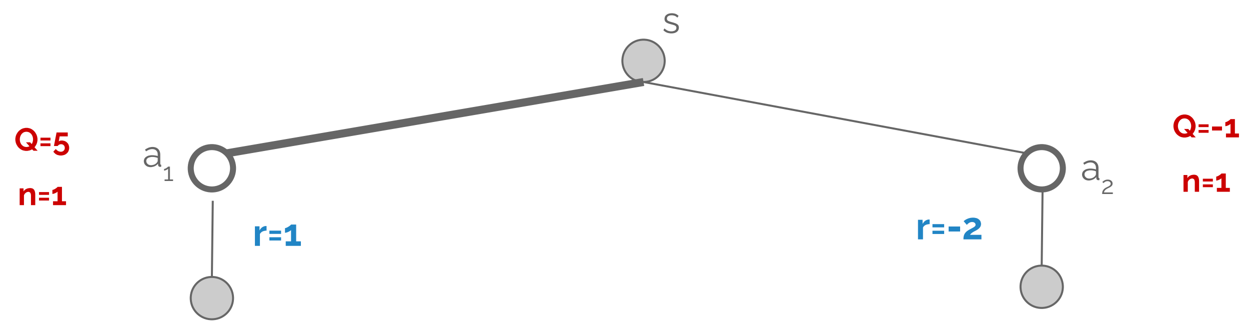
No untried action left, so now we really use the UCT select rule (assume c=1.0)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Q=5, n(s,a)=1, n(s)=2	Q=-1, n(s,a)=1, n(s)=2
UCT = 5.8	UCT = -0.2

Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up



No untried action left, so now we really use the UCT select rule (assume c=1.0)

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Q=5, n(s,a)=1, n(s)=2

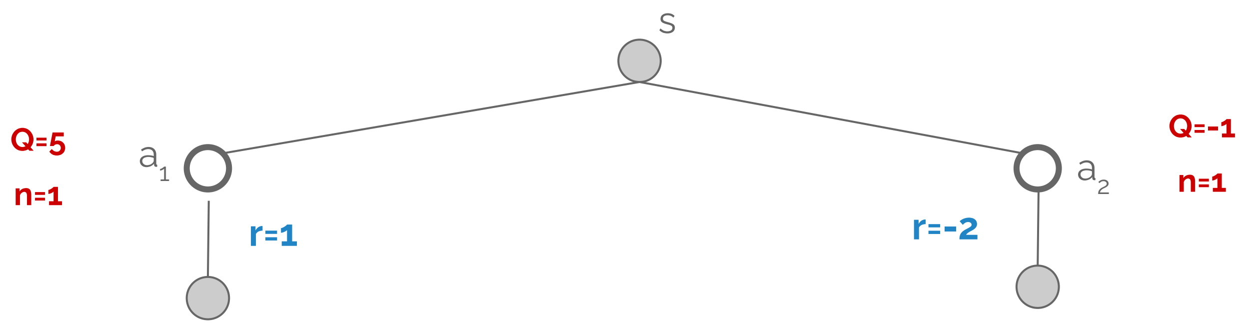
UCT = 5.8

Q=-1, n(s,a)=1, n(s)=2

UCT = -0.2

Monte Carlo Tree Search

1. **Select**
2. Expand
3. Roll-out
4. Back-up

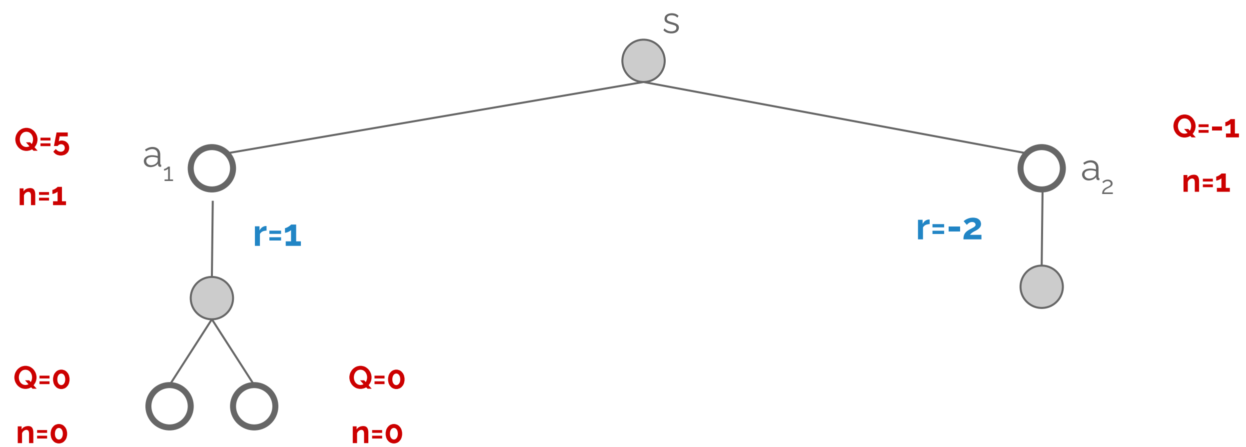


Now we need to select at this state at depth 1.

What will happen?

Monte Carlo Tree Search

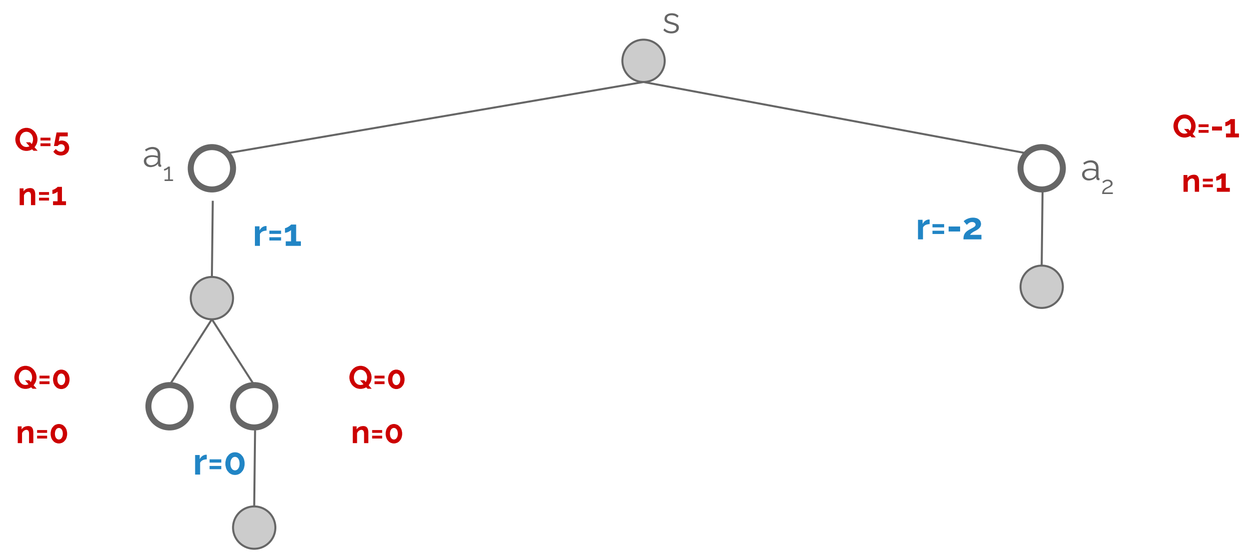
1. **Select**
2. Expand
3. Roll-out
4. Back-up



Unvisited actions, need to expand,
randomly pick one of the available actions

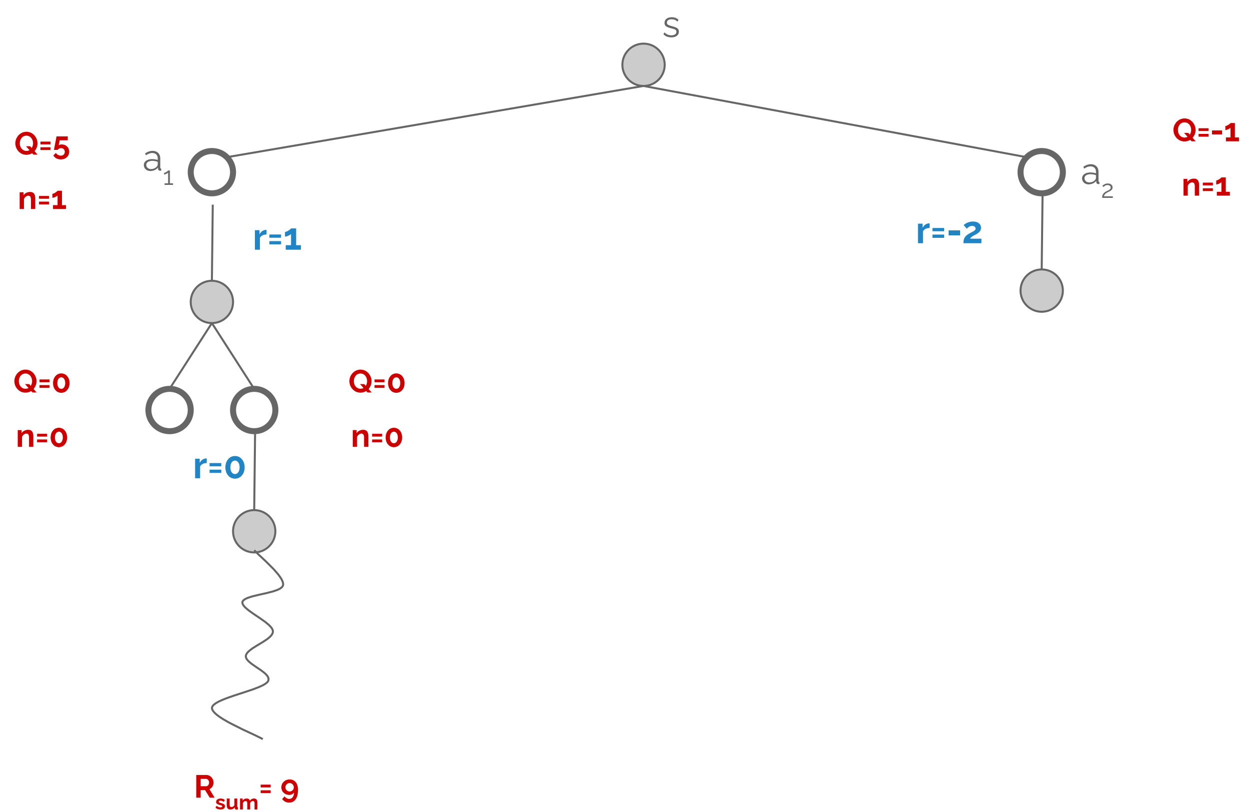
Monte Carlo Tree Search

1. Select
2. **Expand**
3. Roll-out
4. Back-up



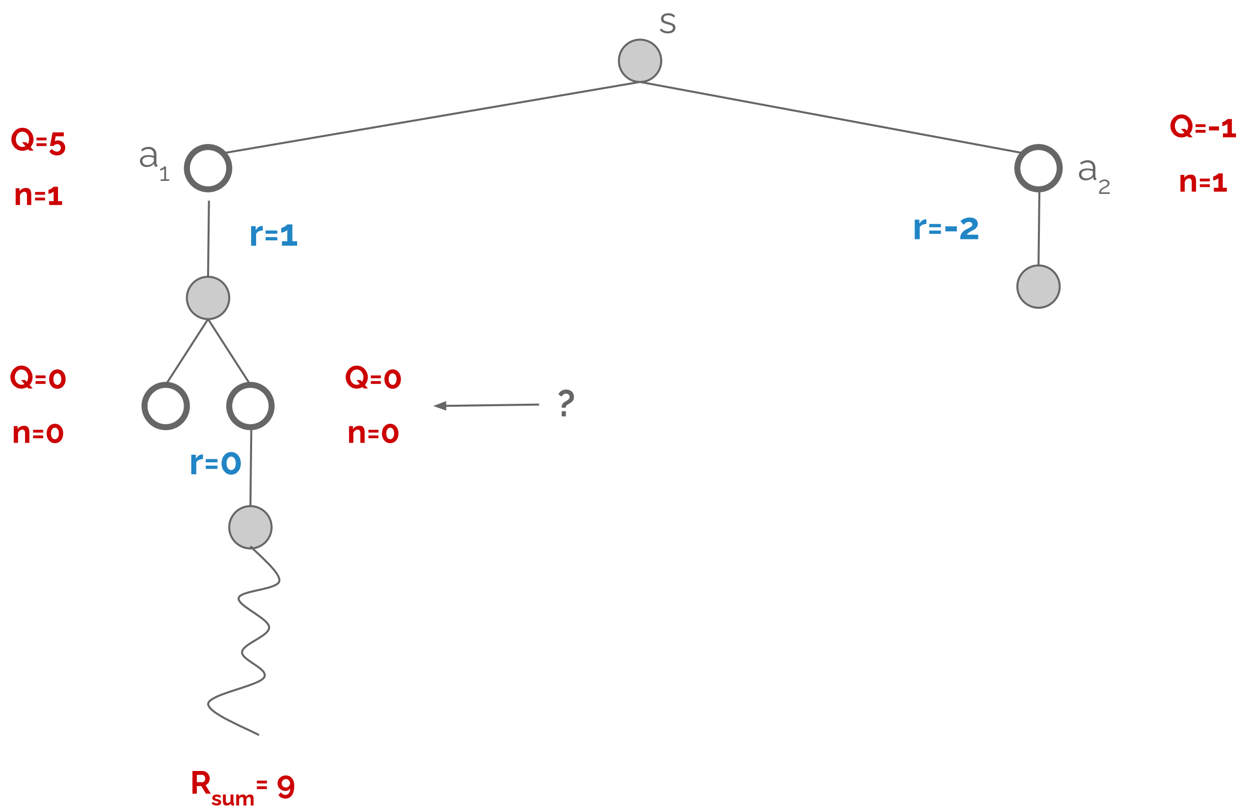
Monte Carlo Tree Search

1. Select
2. Expand
3. **Roll-out**
4. Back-up



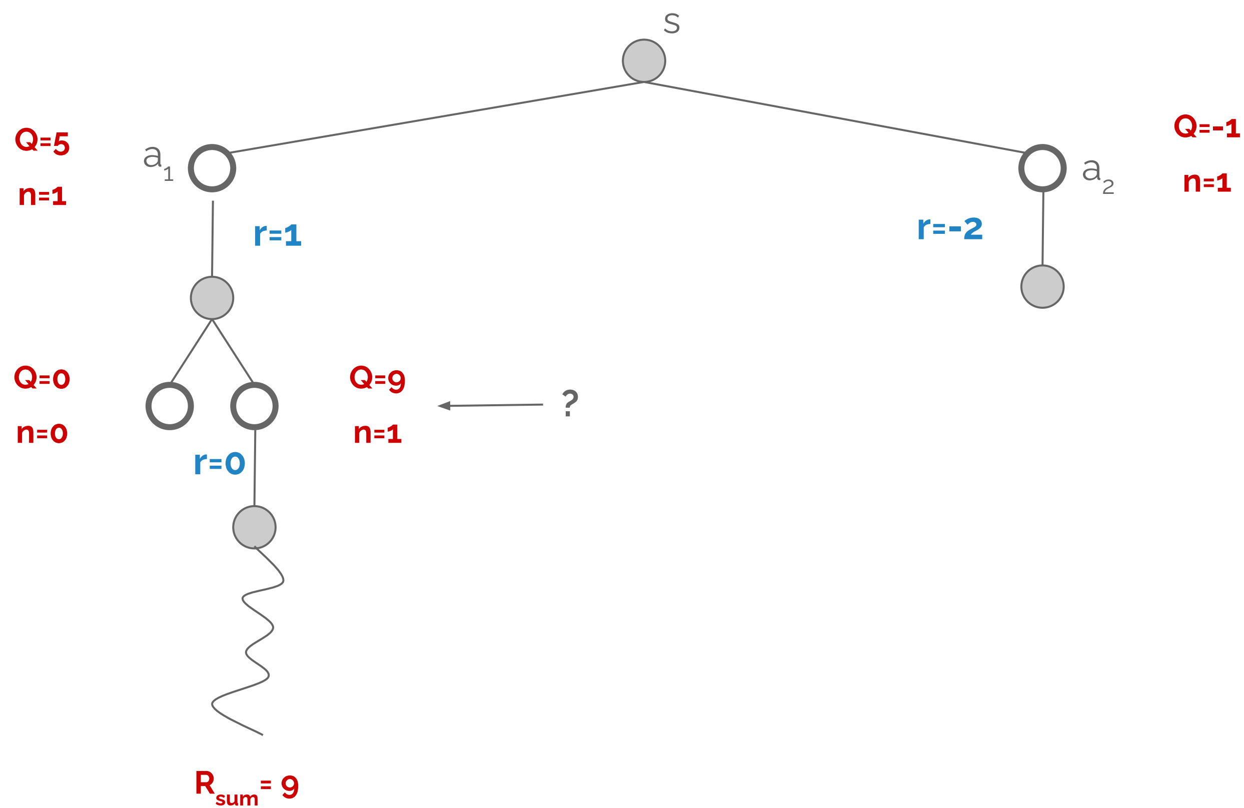
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



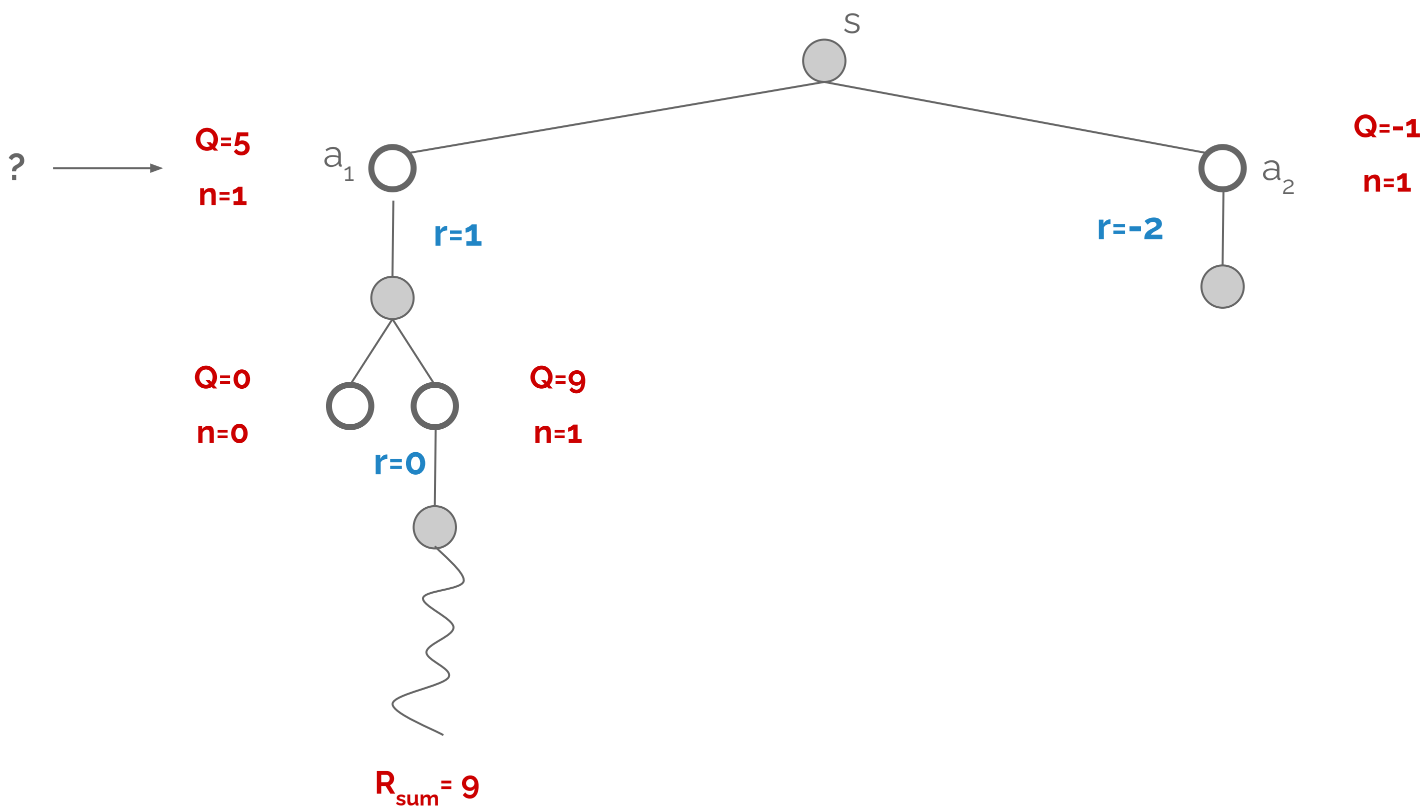
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



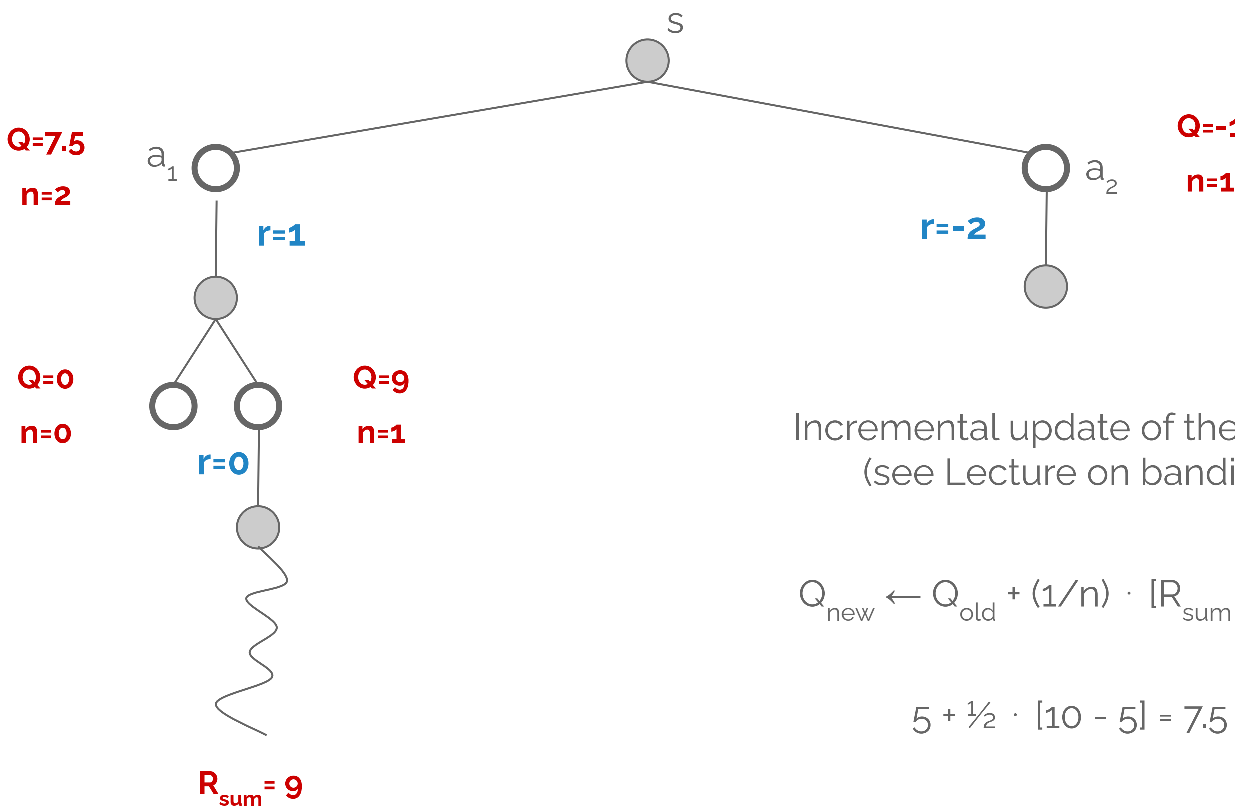
Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. **Back-up**



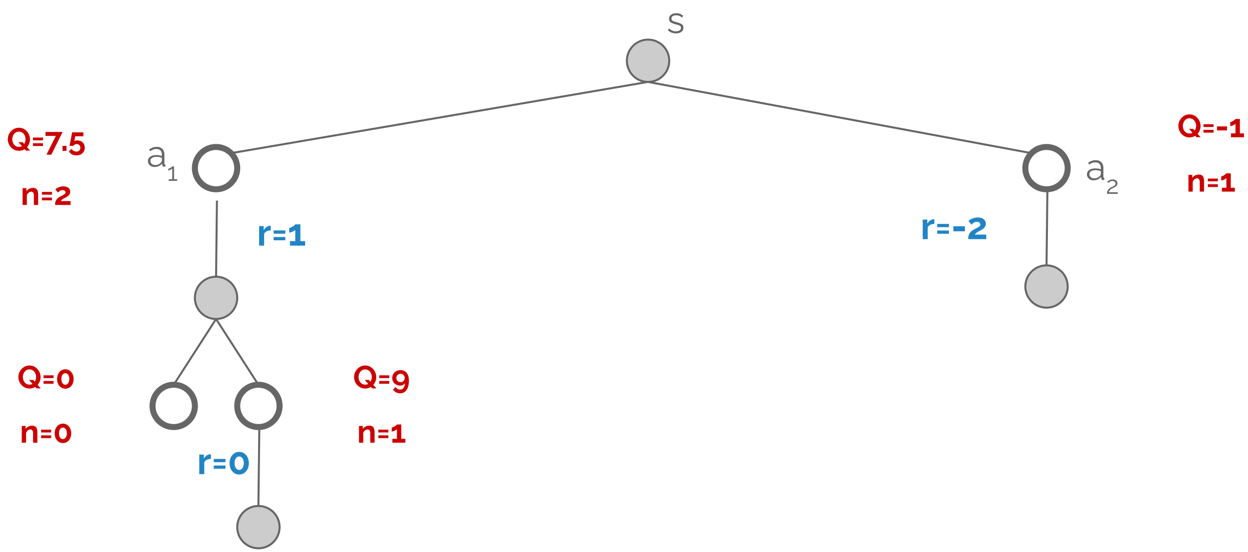
Incremental update of the mean
(see Lecture on bandits)

$$Q_{new} \leftarrow Q_{old} + (1/n) \cdot [R_{sum} - Q_{old}]$$

$$5 + ½ \cdot [10 - 5] = 7.5$$

Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up

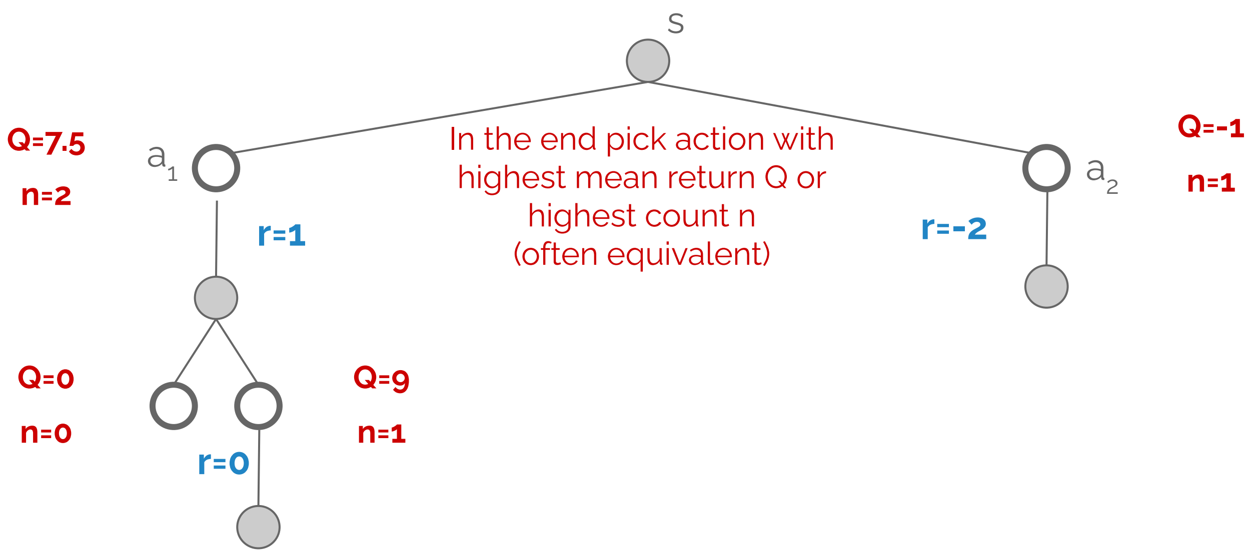


Third iteration,

repeat until trace budget M is up.

Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up

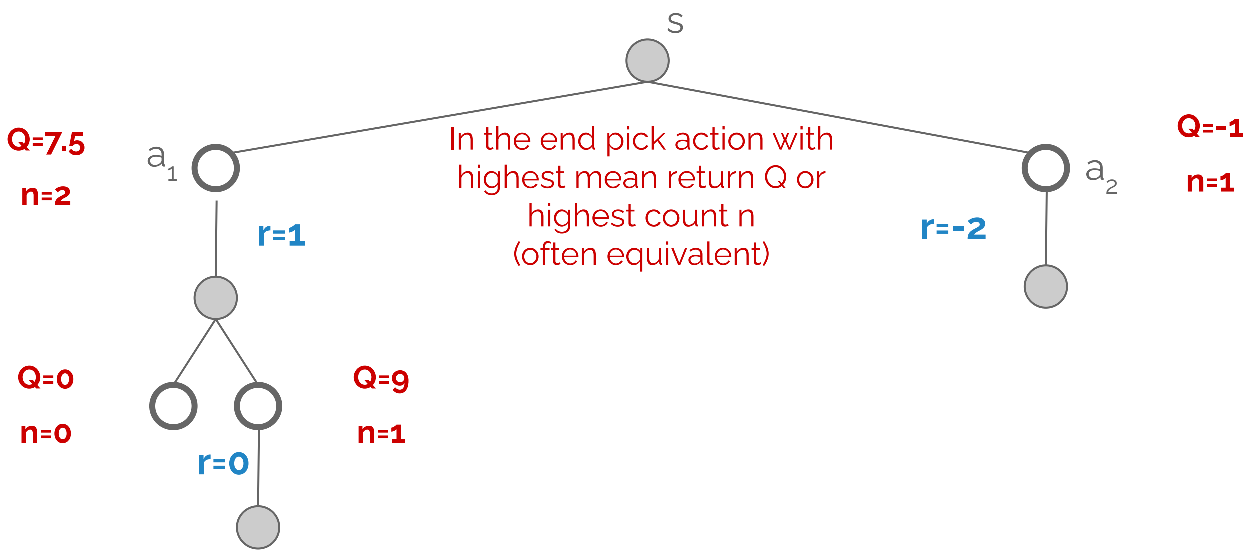


Third iteration,

repeat until trace budget M is up.

Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up



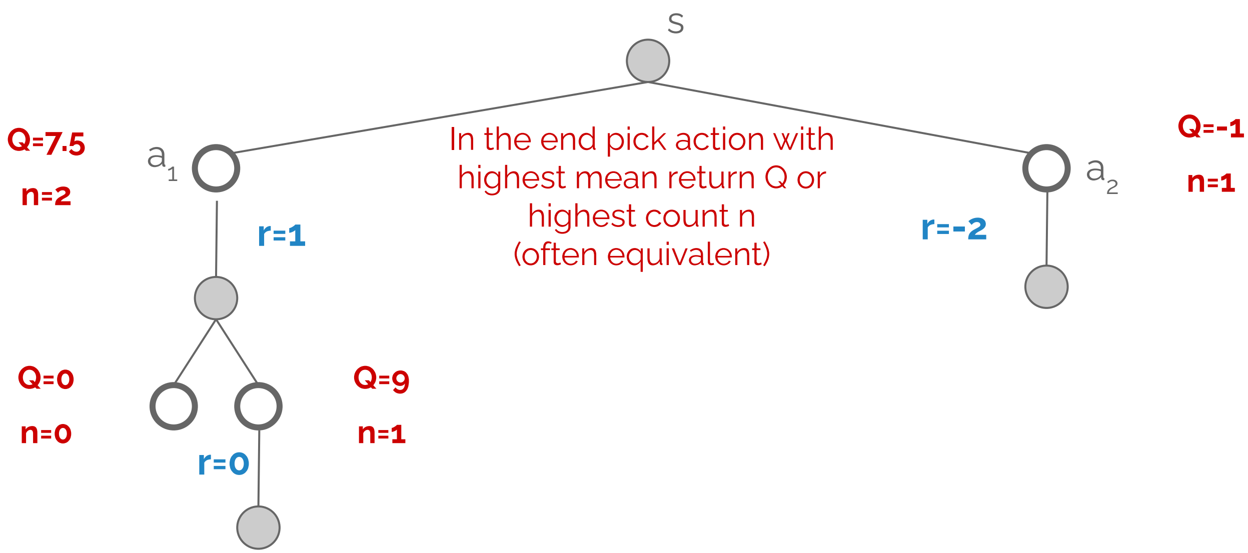
Third iteration,

repeat until trace budget M is up.

Sample complexity: ?

Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up



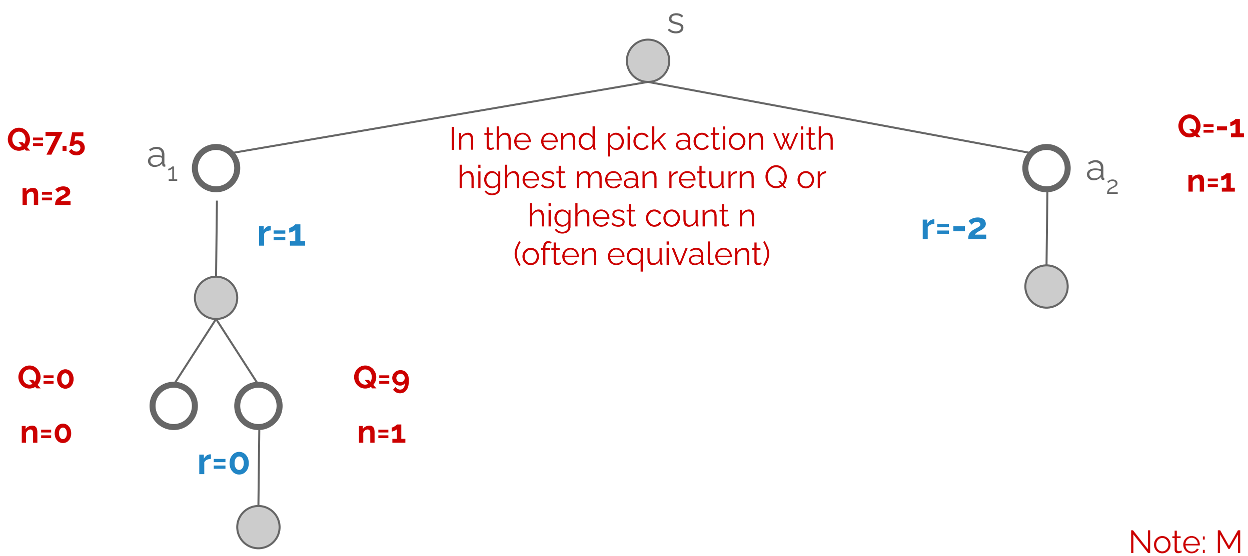
Third iteration,

repeat until trace budget M is up.

Sample complexity: $M \cdot D$

Monte Carlo Tree Search

1. Select
2. Expand
3. Roll-out
4. Back-up



In the end pick action with highest mean return Q or highest count n (often equivalent)

Third iteration,

repeat until trace budget M is up.

Note: M in MCTS (total # of traces) is not the same as N (# of traces per action) in MCS and SS

Sample complexity: $M \cdot D$

Summary: Monte Carlo Tree Search

Summary: Monte Carlo Tree Search

- Very powerful search paradigm: adaptively focuses search budget based on statistical uncertainty measures.

Summary: Monte Carlo Tree Search

- Very powerful search paradigm: adaptively focuses search budget based on statistical uncertainty measures.

- One of the most popular algorithms in problems without a good heuristic.

4. Iterated planning & learning

Planning versus learning

Planning versus learning

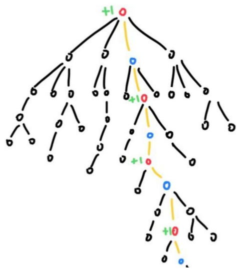
Pure planning is often suboptimal

Planning versus learning

Pure planning is often suboptimal

-

Uninformed (sample-based) search is too expensive & we lack good heuristics

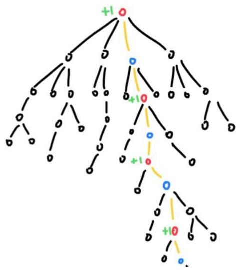


Planning versus learning

Pure planning is often suboptimal

-

Uninformed (sample-based) search is too expensive & we lack good heuristics



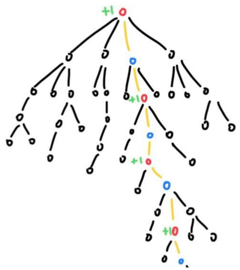
Pure learning is often suboptimal

Planning versus learning

Pure planning is often suboptimal

-

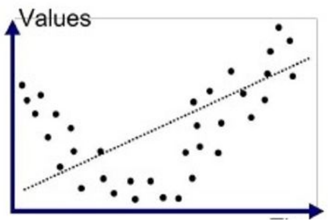
Uninformed (sample-based) search is too expensive & we lack good heuristics



Pure learning is often suboptimal

-

Learned approximate value/policy typically has remaining errors

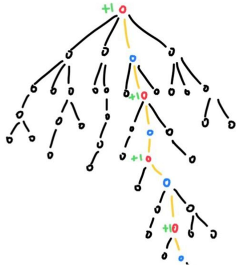


Planning versus learning

Pure planning is often suboptimal

-

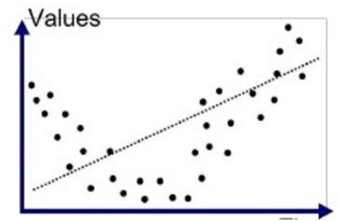
Uninformed (sample-based) search is too expensive & we lack good heuristics



Pure learning is often suboptimal

-

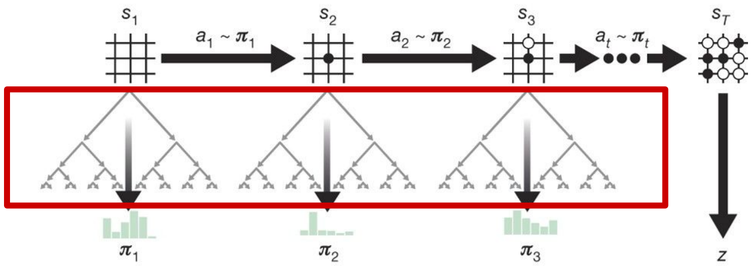
Learned approximate value/policy typically has remaining errors



<u>But both approaches can be combined!</u>

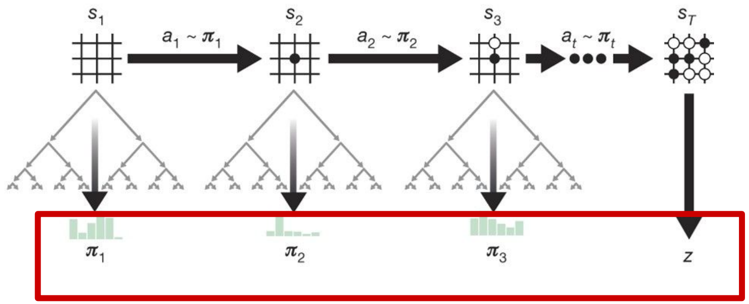
AlphaGo

AlphaGo



Planning..

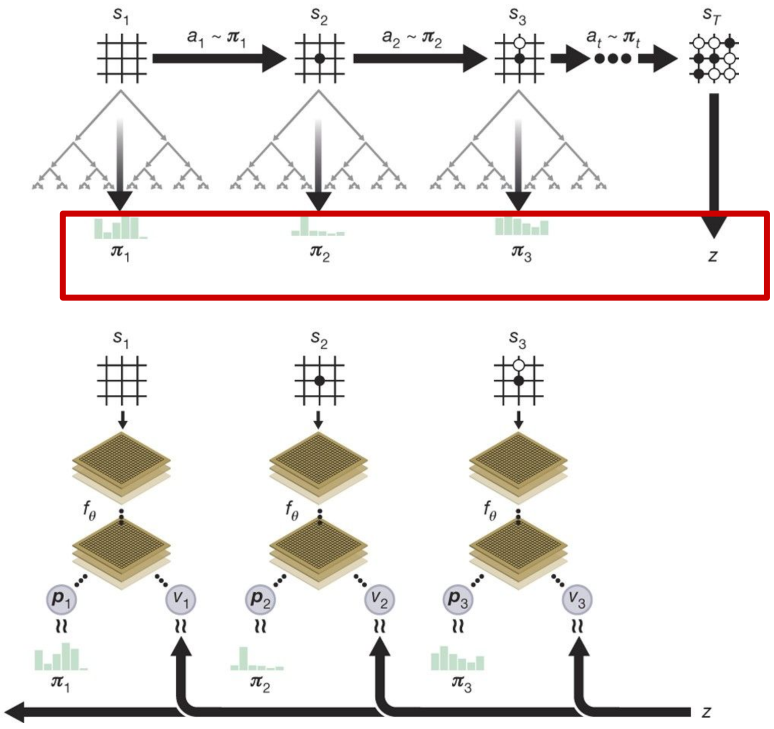
AlphaGo



Planning..

..generates statistics..

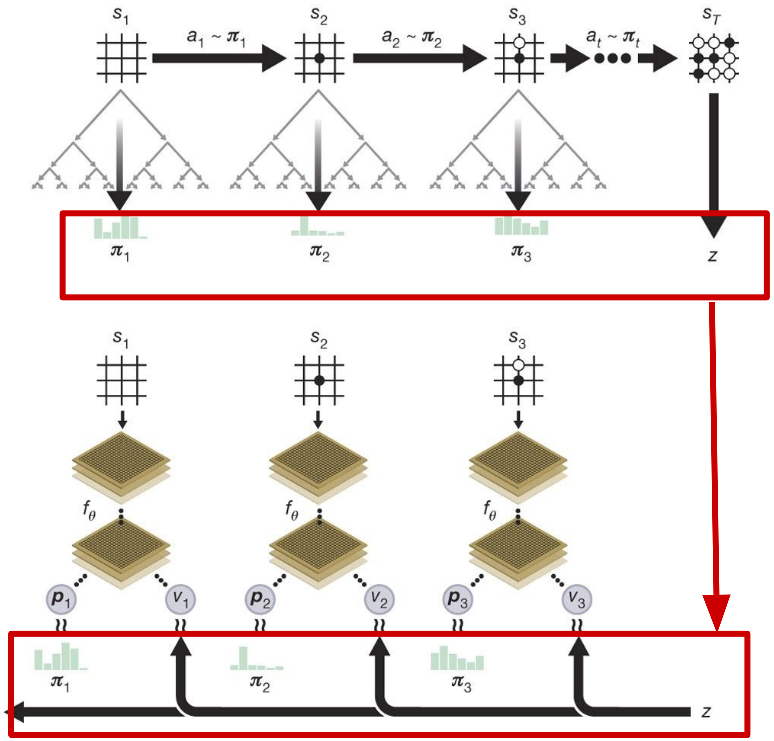
AlphaGo



Planning..

..generates statistics..

AlphaGo

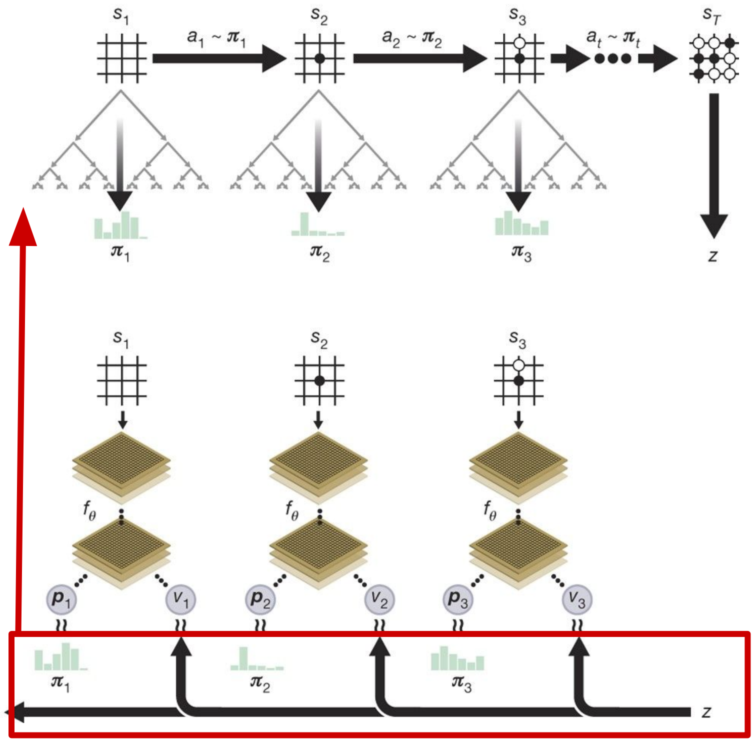


Planning..

..generates statistics..

..that are used to train (approximate) value and policy functions..

AlphaGo

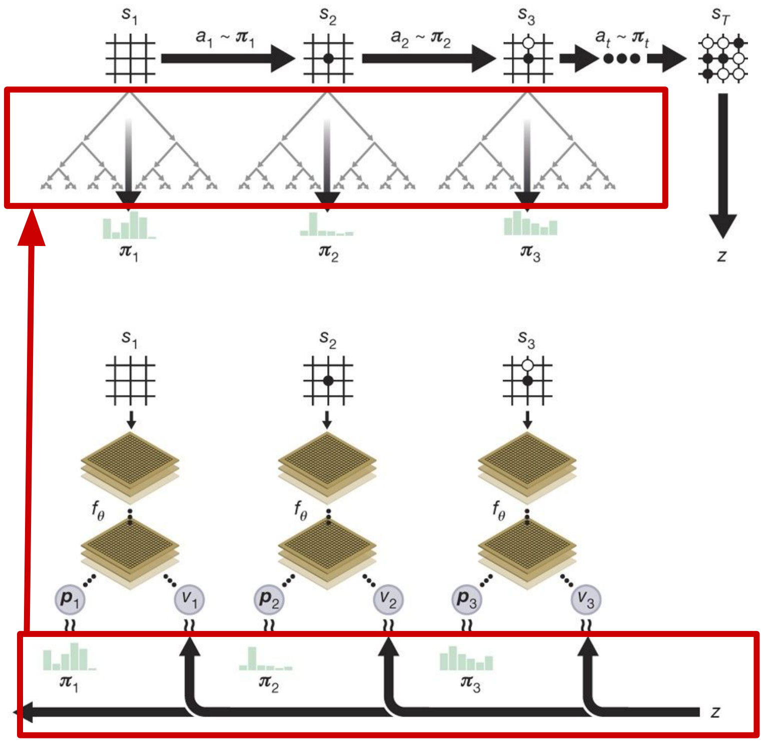


..which we may itself use..

AlphaGo

..to steer new planning iterations.

..which we may itself use..



Iterated planning and learning

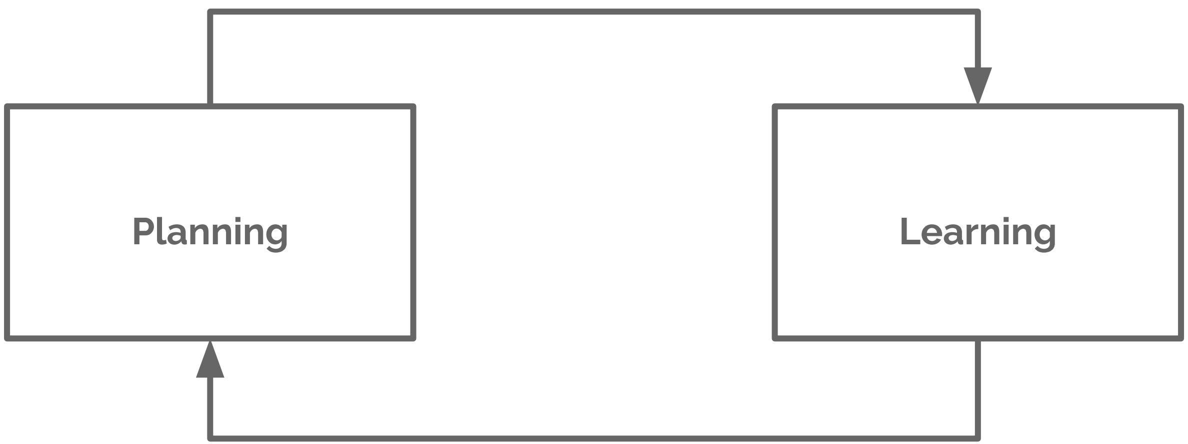
Iterated planning and learning

Iterated planning and learning

… use learned value/policy function to steer new planning iterations.

Iterated planning and learning

.. use planning to **1)** correct errors in learned solution (*'decision-time planning'*) and/or
2) generate training data for learning (*'background planning'*).

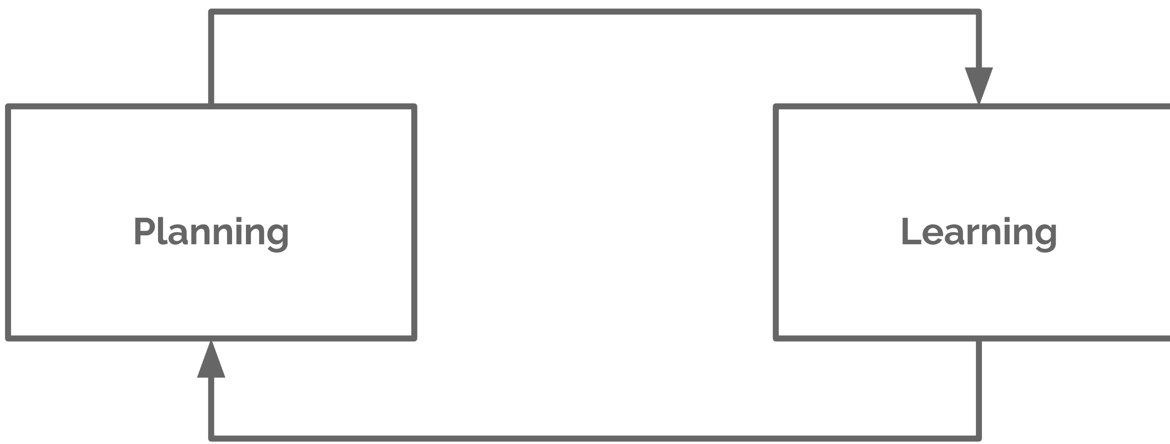


... use learned value/policy function to steer new planning iterations.

Iterated planning and learning

.. use planning to **1)** correct errors in learned solution (*'decision-time planning'*) and/or **2)** generate training data for learning (*'background planning'*).

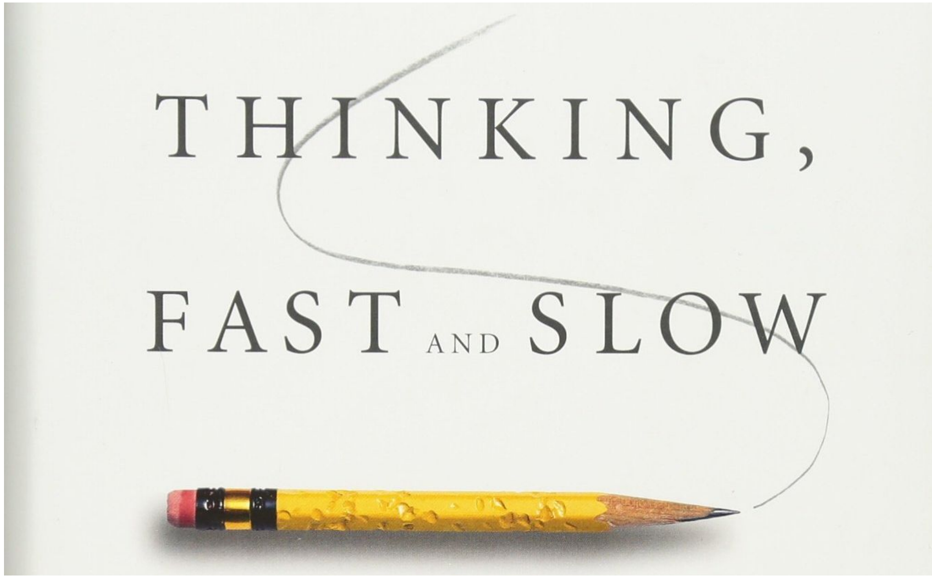
Both types of planning are useful/combined in this iterated scheme



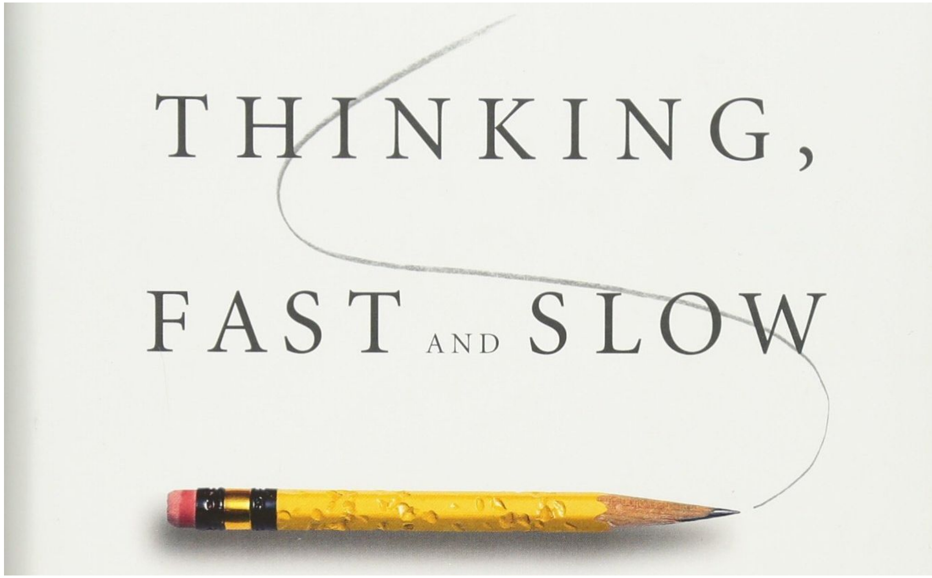
... use learned value/policy function to steer new planning iterations.

Thinking fast & slow

Thinking fast & slow



Thinking fast & slow



Psychology research, but well interpretable in terms of AI

Thinking fast & slow

Thinking fast & slow

Learned (approximate) value function

=

‘Thinking fast’

Thinking fast & slow

Learned (approximate) value function

=

'Thinking fast'

(reactive behaviour based on pattern recognition in known situations)

Thinking fast & slow

Learned (approximate) value function

=

'Thinking fast'

(reactive behaviour based on pattern recognition in known situations)

Decision-time planning

=

'Thinking slow'

Thinking fast & slow

Learned (approximate) value function

=

'Thinking fast'

(reactive behaviour based on pattern recognition in known situations)

Decision-time planning

=

'Thinking slow'

(putting local effort in current decision to overcome errors in the learned value function)

Thinking fast & slow

Learned (approximate) value function

=

'Thinking fast'

(reactive behaviour based on pattern recognition in known situations)

Decision-time planning

=

'Thinking slow'

(putting local effort in current decision to overcome errors in the learned value function)

Both have their role in optimal decision-making!

Thinking fast & slow

Learned (approximate) value function

=

'Thinking fast'

(reactive behaviour based on pattern recognition in known situations)

Decision-time planning

=

'Thinking slow'

(putting local effort in current decision to overcome errors in the learned value function)

Both have their role in optimal decision-making!

(more in later lecture on AlphaGo)

Summary

1. Decision-time versus background planning
2. Classic planning
3. Monte Carlo search
4. Iterated planning & learning

Summary

1. Decision-time versus background planning

2. Classic planning

3. Monte Carlo search

4. Iterated planning & learning

Questions?