UNIVERSITÀ DI PAVIA

Deep Learning

12 – Monte Carlo Tree Search (MCTS)

Marco Piastra

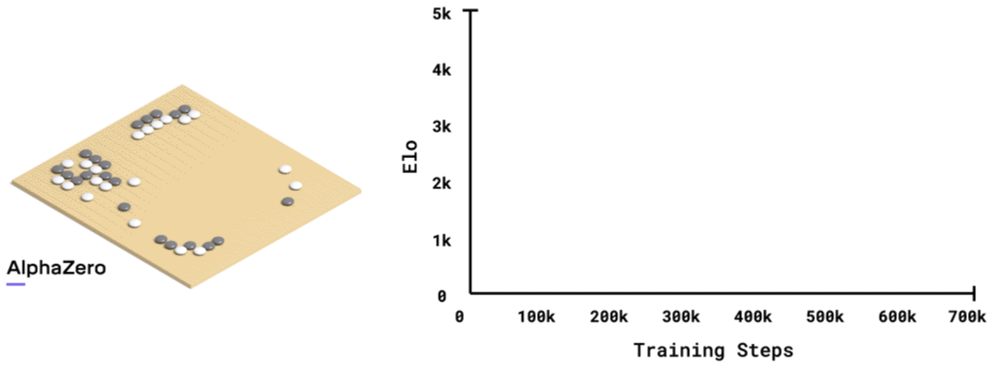
This presentation can be downloaded at:
http://vision.unipv.it/DL

Prologue:
Playing Games better than Humans

Beyond Emulating Humans: AlphaZero (2018)

Image from: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

AlphaGo is heavily reliant
on the experience of human players



- **AlphaZero learns by itself**
  [2018, D. Silver, et al. (13 authors), https://science.sciencemag.org/content/362/6419/1140.full]

Basic Knowledge Only

It just knows the basic rules of the games

Learning via Self-Play

It plays against a (frozen) copy of itself

MCTS and DCNN in a closed loop

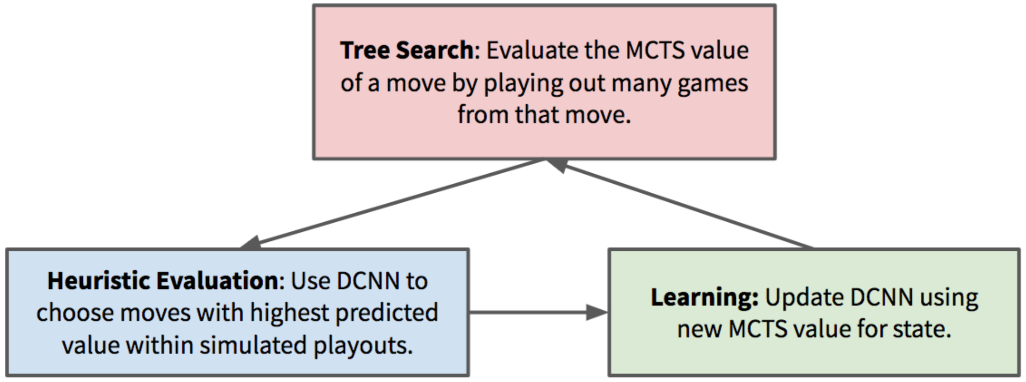
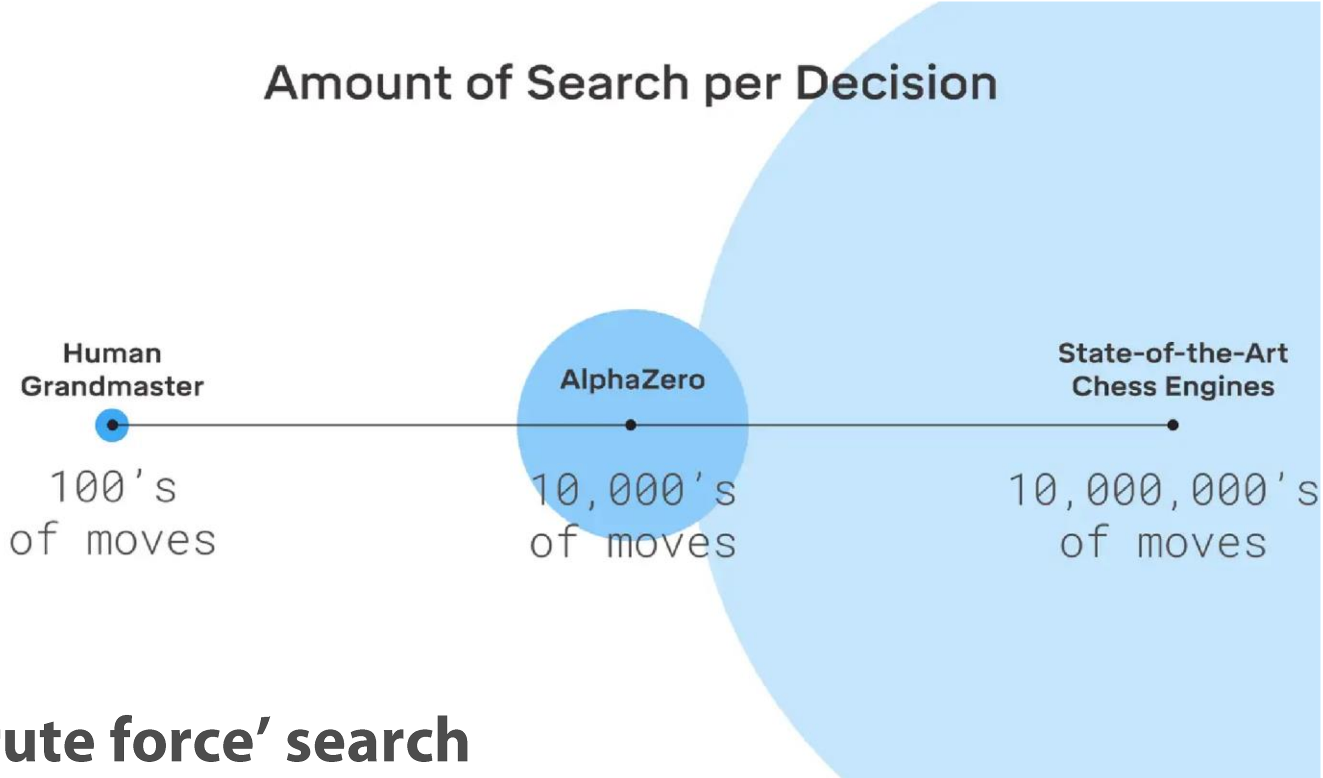


Image from: https://nikcheerla.github.io/deeplearningschool/2018/01/01/AlphaZero-Explained/

Beyond Emulating Humans: AlphaZero (2018)

Image from: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go



- **AlphaZero uses much less 'brute force' search**
  - When playing, the search process is driven by its neural network
    - *It acts like a memory of past experiences*
  - While training, it learns through a huge amount of self-playing
    - *But it is a faster learner than Alpha Go*

Playing Games with Trees

Tree representation

- **Game Tree (*simplest case*):**

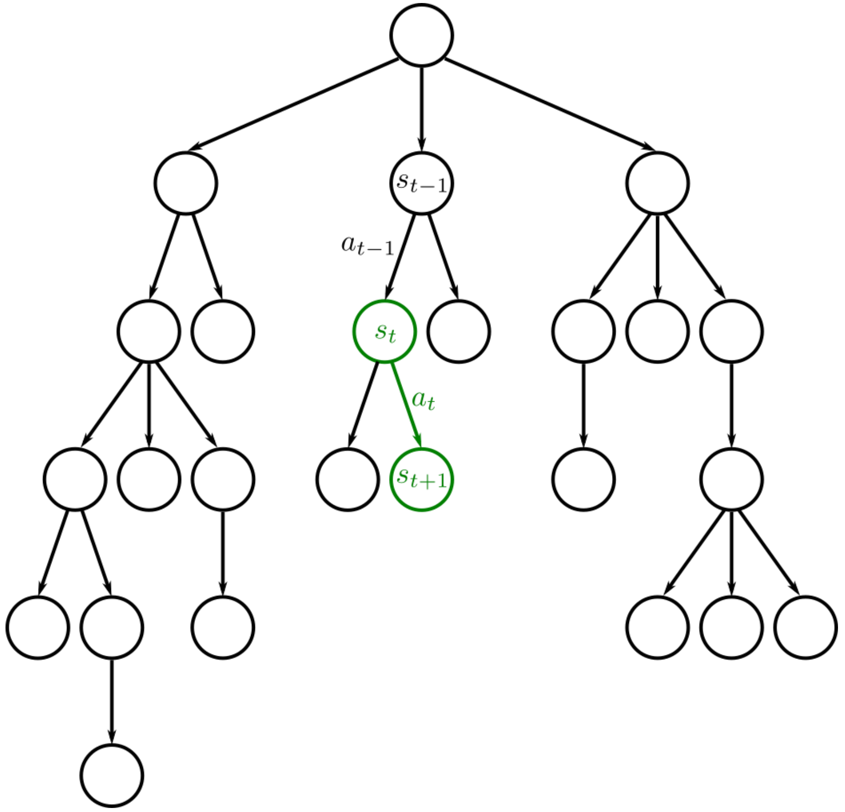
  The <u>*current state*</u> s_t at time t is a ***node*** with depth t

  Any admissible <u>*action*</u> a_t is an ***edge*** of the tree

  (<u>branching factor</u> = number of admissible actions in a state)

  State s_{t+1} obtained from s_t after executing a_t is determined by a <u>*transition function*</u>

$$\tau : \ (s_t, a_t) \mapsto s_{t+1}$$



Tree representation

- **Game Tree (*simplest case*):**

The *current state* s_t at time t is a ***node*** with depth t

Any admissible *action* a_t is an ***edge*** of the tree

(branching factor = number of admissible actions in a state)

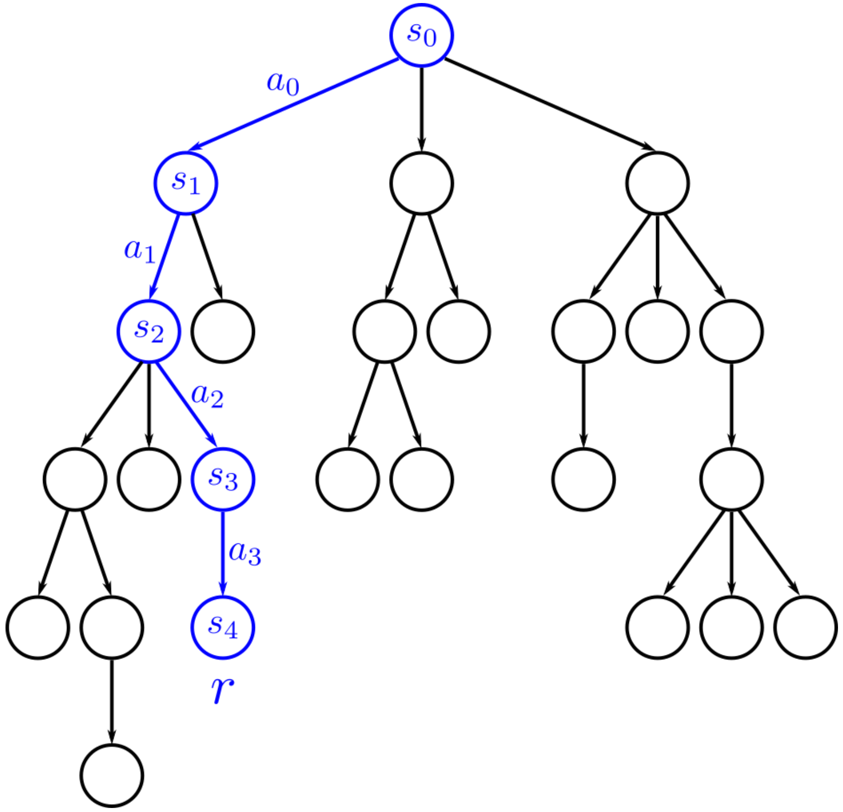
State s_{t+1} obtained from s_t after executing a_t
is determined by a *transition function*

$$\tau : \ (s_t, a_t) \mapsto s_{t+1}$$

A *playout* is a **path** $\langle s_0, a_0, s_1, \ldots, a_{T-1}, s_T \rangle$
from the *initial state* s_0 to a *terminal state* s_T

A *reward* r is the outcome of a playout

A *policy* is a map $\pi : \ s \mapsto a$ which selects action a to be executed in state s



Policy optimization

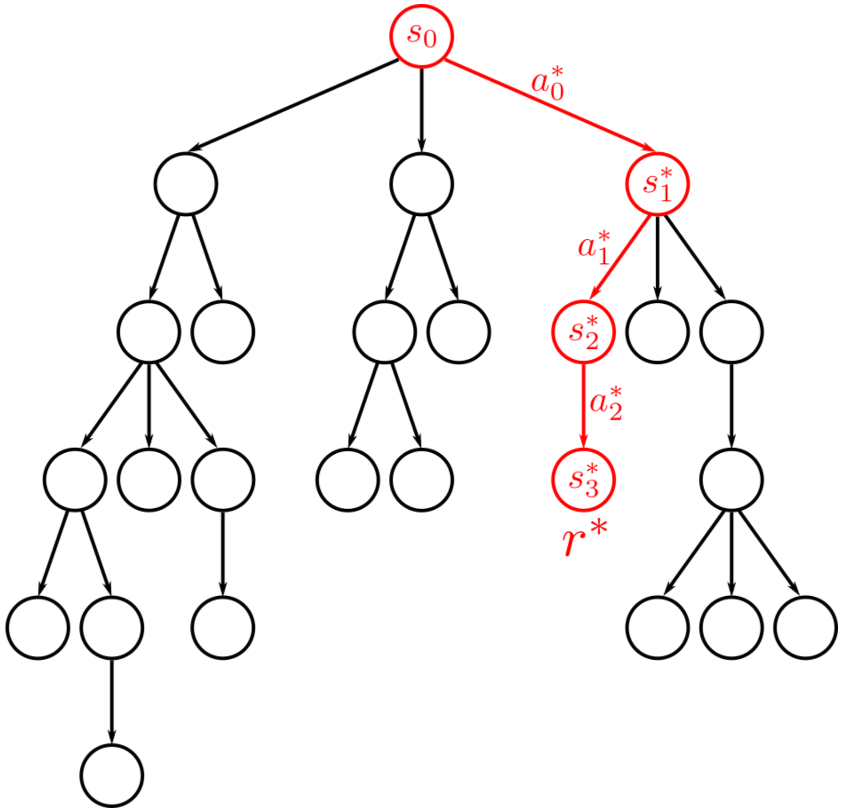
- **Goal:** finding the *best policy* π^*

  such that the reward r^* of playout

  $$\langle s_0, a_0^*, s_1^*, \ldots, a_{T-1}^*, s_T^* \rangle$$

  with $a_t^* := \pi^*(s_t^*)$ and $s_{t+1}^* := \tau(s_t^*, a_t^*)$

  is ***maximal***



"Brute Force": a simple (bad) policy optimization

- Goal: finding the <u>*best policy*</u> π^*

- ***"Brute Force"***:
  1. explore the entire tree by following **all** possible paths
  2. select the path(s) with the best outcome (and randomly choose one of them)
  3. play by following the policy associated with that path

  Possible problems:

  - ***Huge game tree*** making full exploration unfeasible
    (branching factor in Go is around 200)
  - ***Infinitely many*** admissible actions
  - Intrinsic ***stochasticity*** and/or ***uncertainty*** of execution

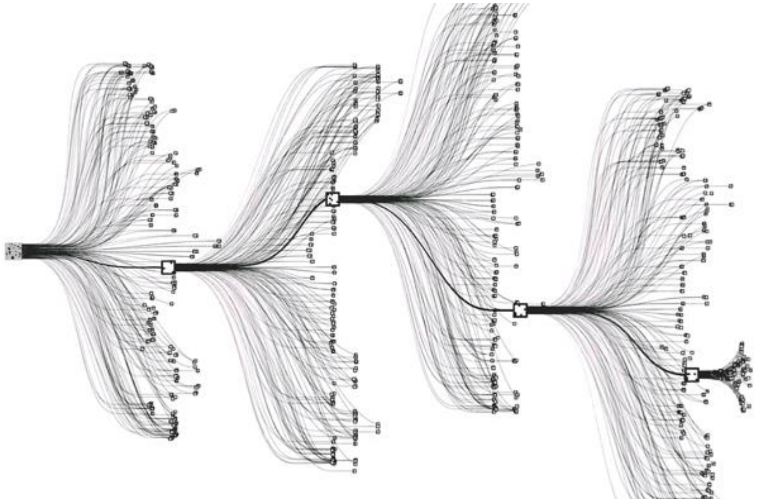


Image from https://thenewstack.io/google-ai-beats-human-champion-complex-game-ever-invented/

Stochasticity and Uncertainty: examples

- ***Multi-armed bandits***

  The reward after each action is *stochastic*

  i.e. which arm to play

  random variable

  probability of reward r for action a

$$Q(s,a) := \mathbb{E}[R \mid s,a] = \sum_r r\,P(r \mid s,a)$$

Q-value (expected reward of action a performed in state s)

- ***Games with two players (White and Black):***

  White plays action a_t in state s_t
  but *her* next state s_{t+1} depends on Black's next action

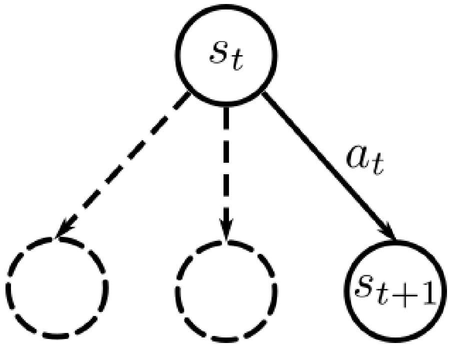
  Uncertainty of execution:
  nondeterministic $\tau : (s_t, a_t) \mapsto s_{t+1}$ with $P(s_{t+1} \mid s_t, a_t)$

  transition function

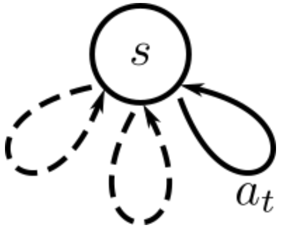
  probability transition distribution

Stochasticity and Uncertainty: tree representation

- ***Simplest case scenario***
  - deterministic transition
  - deterministic reward

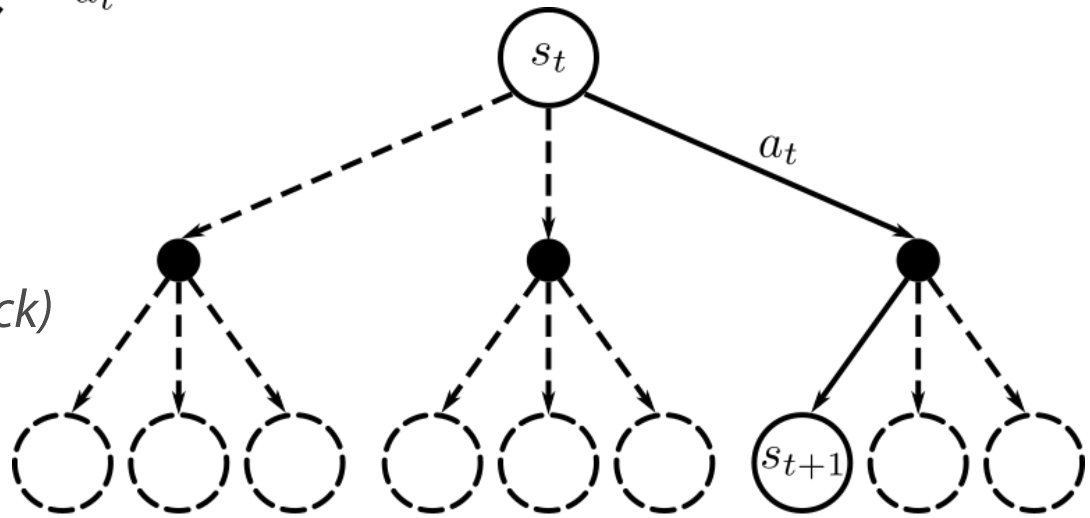


- ***Multi-armed bandits***
  - deterministic transition
  - stochastic reward



Actually, this is not a tree! (but it can be expanded and became one)

- ***Uncertainty of execution:***
  - stochastic transition
  - either deterministic *(White vs Black)* or stochastic reward



Monte Carlo method: step by step simulations

Monte Carlo (MC) step

- Goal: finding the <u>*best policy*</u> π^* *(avoiding brute-force approach)*

  It can be done iteratively, by focusing on the single best action $a^* =: \pi^*(s)$ in the current state s

- ***Monte Carlo (MC)*** *step:* [Abramson 1990]

repeat n times {
1) perform a <u>*random*</u> *playout* from current state s
2) compute and save the reward r obtained at the end of the playout
}

3) for each admissible action a in state s compute the mean of the rewards

$$\hat{Q}(s,a) := \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} r_{a,i}$$

estimates $Q(s,a)$

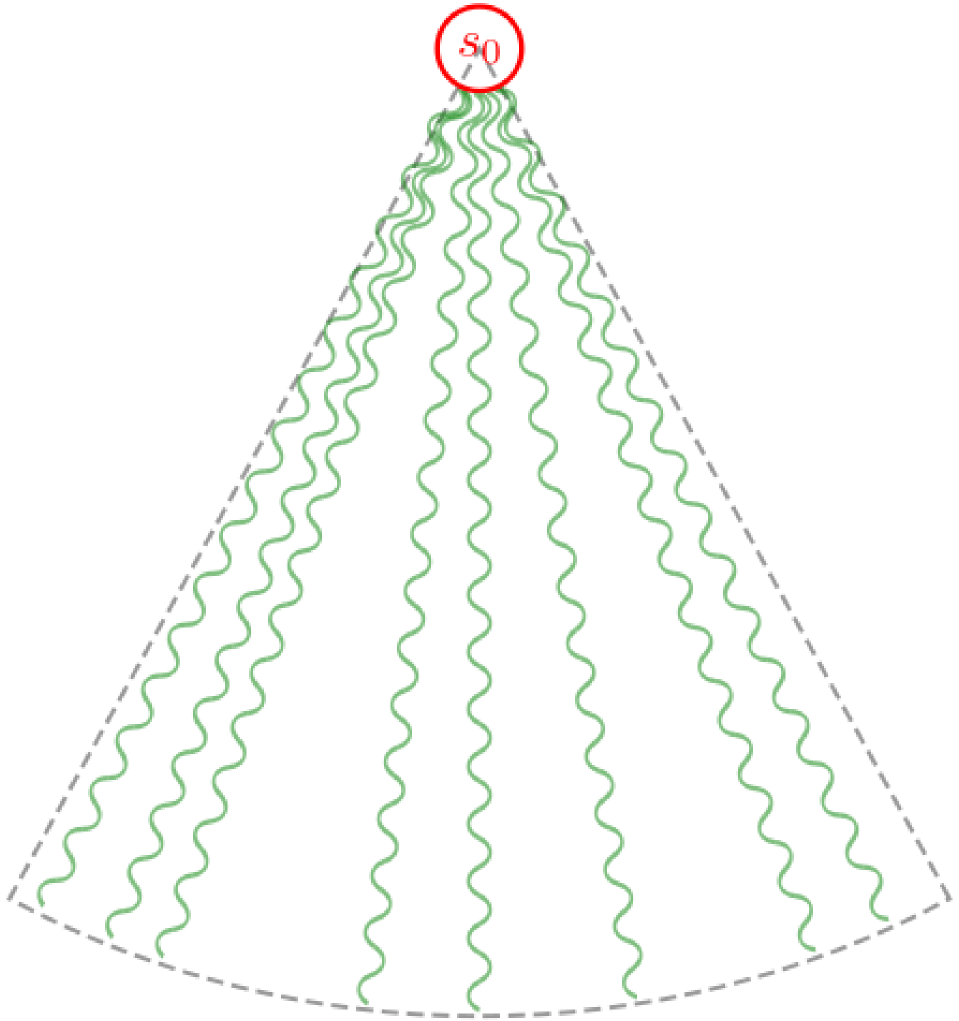
number of playouts with first action a

reward of i^{th} playout with first action a

4) $a^* := \text{argmax}_a \hat{Q}(s,a)$ is the action with the highest mean

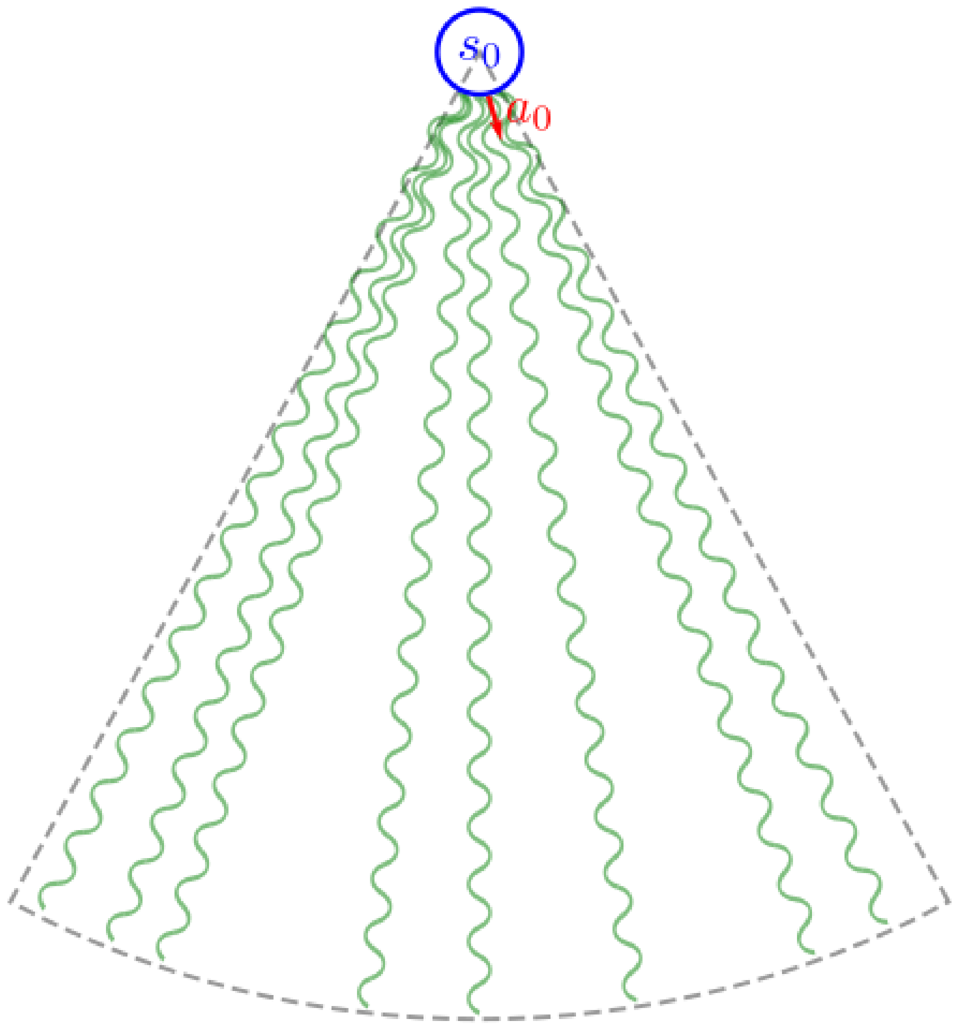
Monte Carlo episode

- ***Monte Carlo*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t + 1$
  6) repeat 2) to 5) until end game



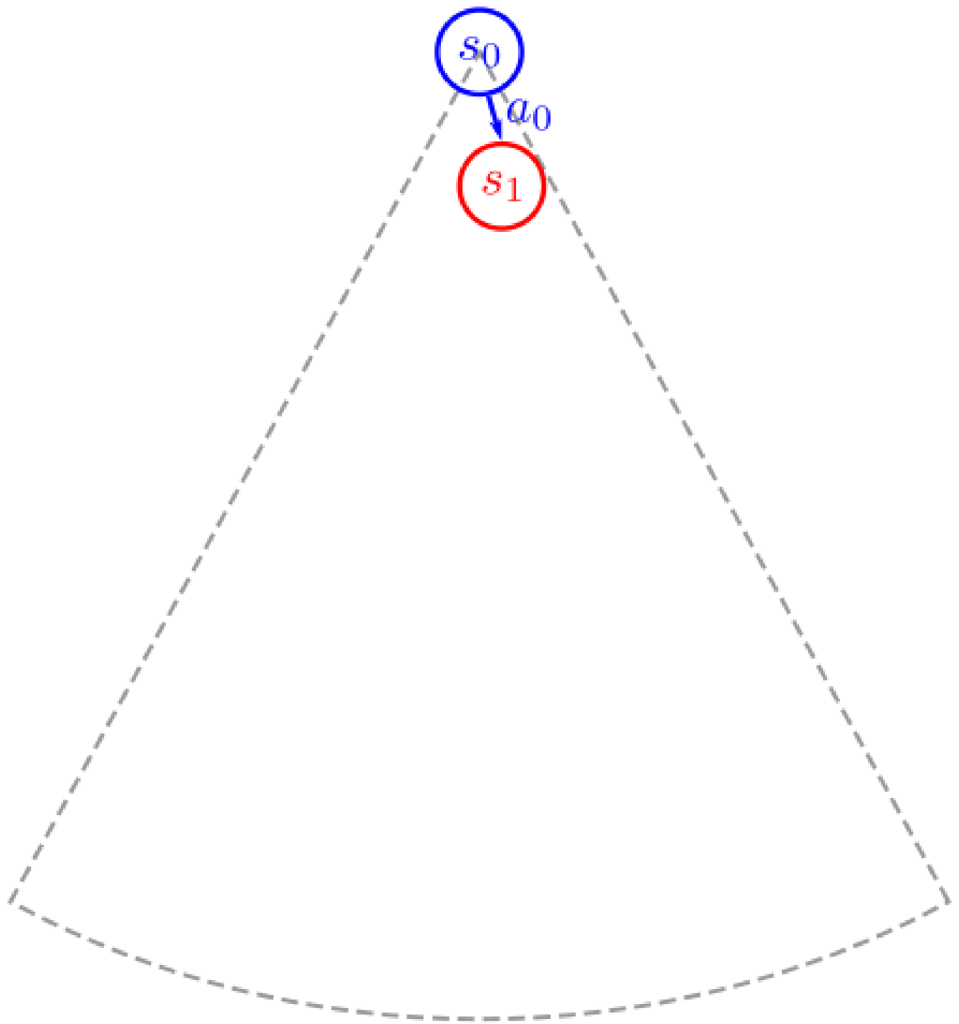
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2) to 5) until end game



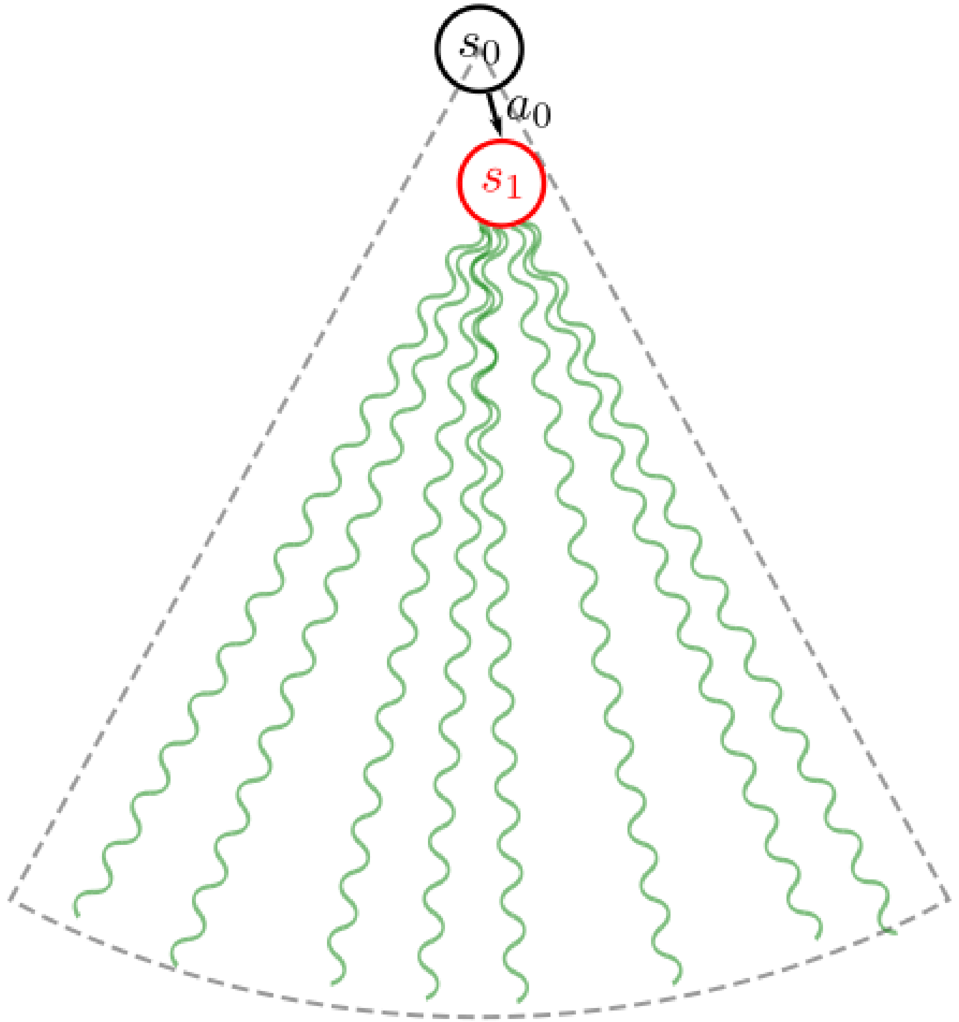
Monte Carlo episode

- ***Monte Carlo** episode:*
    1) set $t{:=}0$
    2) current state $s{:=}s_t$
    3) use *MC step* to decide a_t
    4) compute $s_{t+1} := \tau(s_t, a_t)$
    5) set $t{:=}t+1$
    6) repeat 2) to 5) until end game



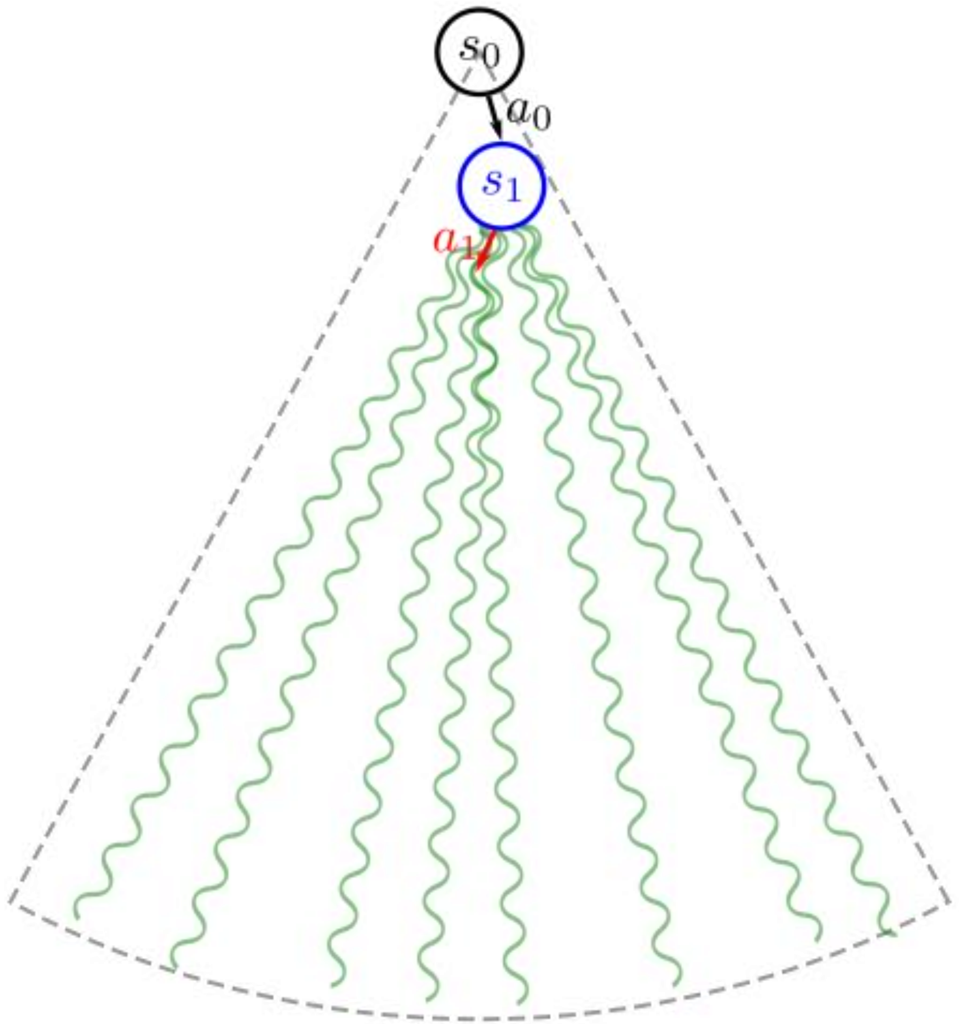
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2) to 5) until end game



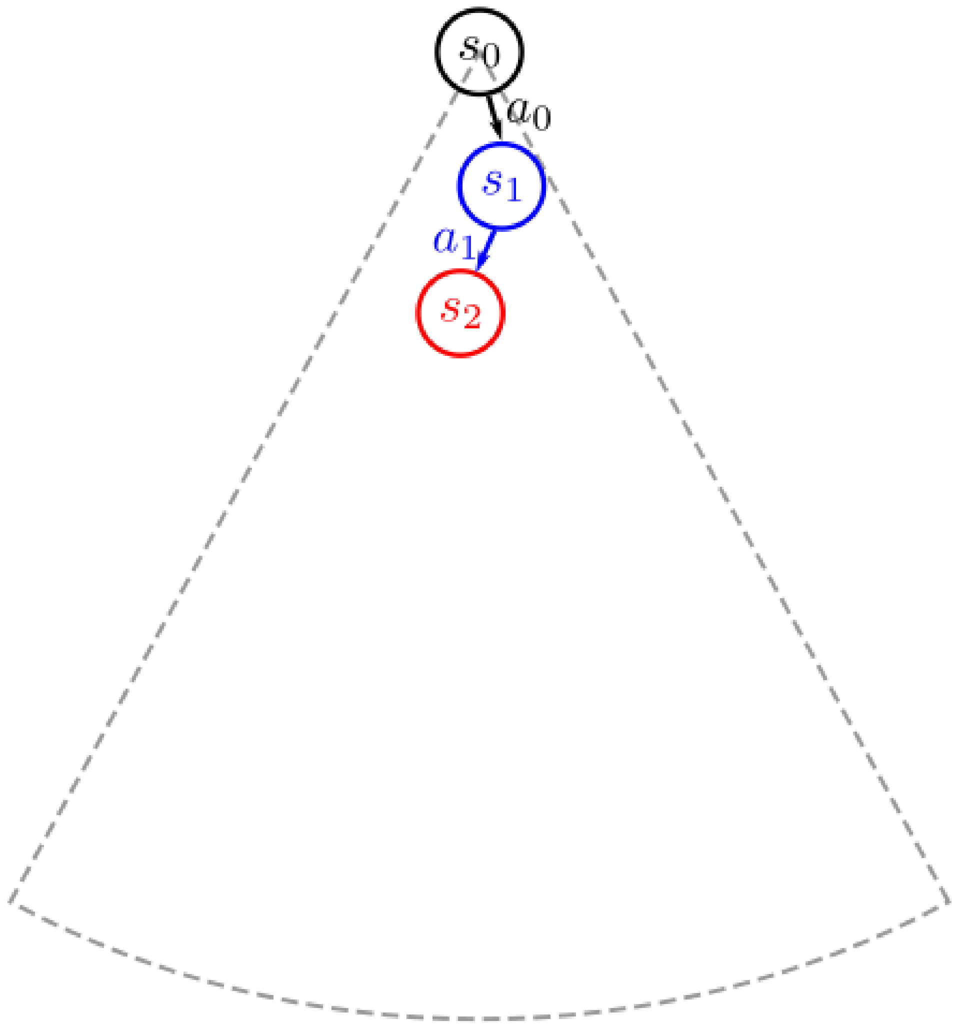
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set t:=0
  2) current state s:=s_t
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set t:=t+1
  6) repeat 2) to 5) until end game



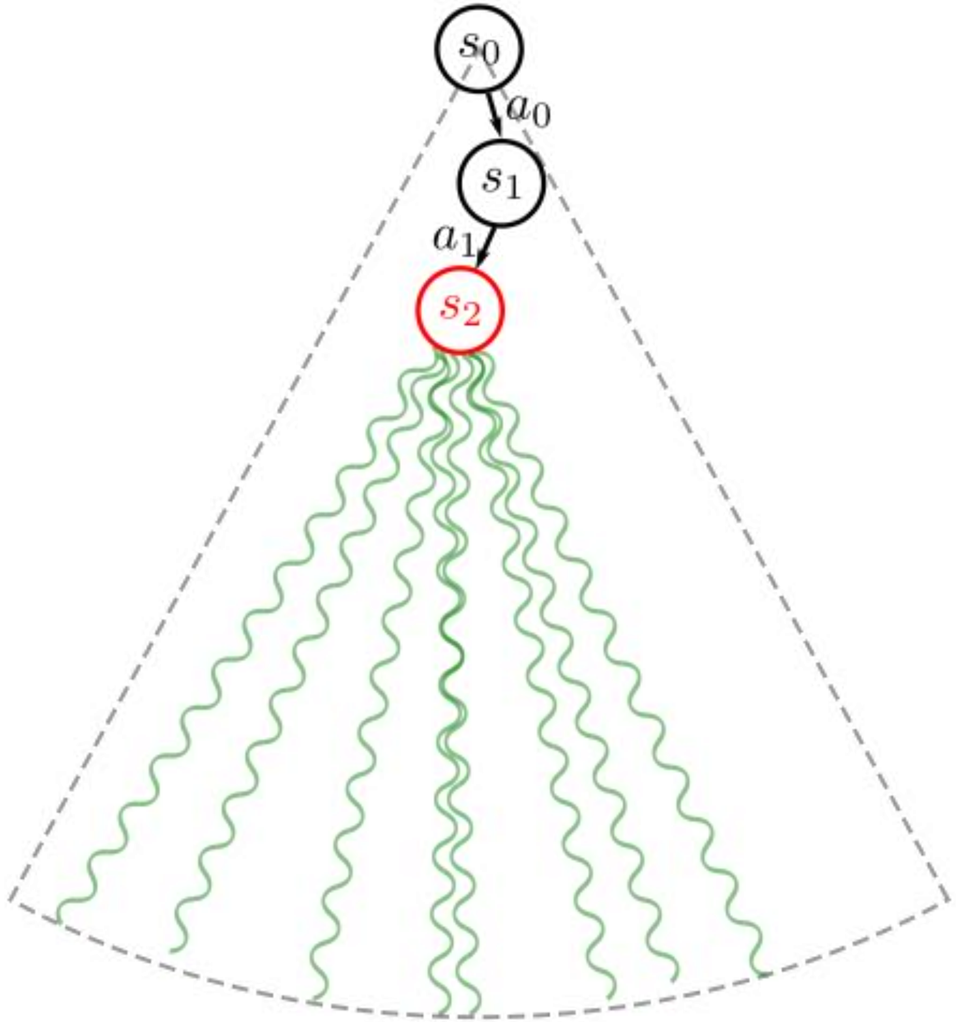
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2) to 5) until end game



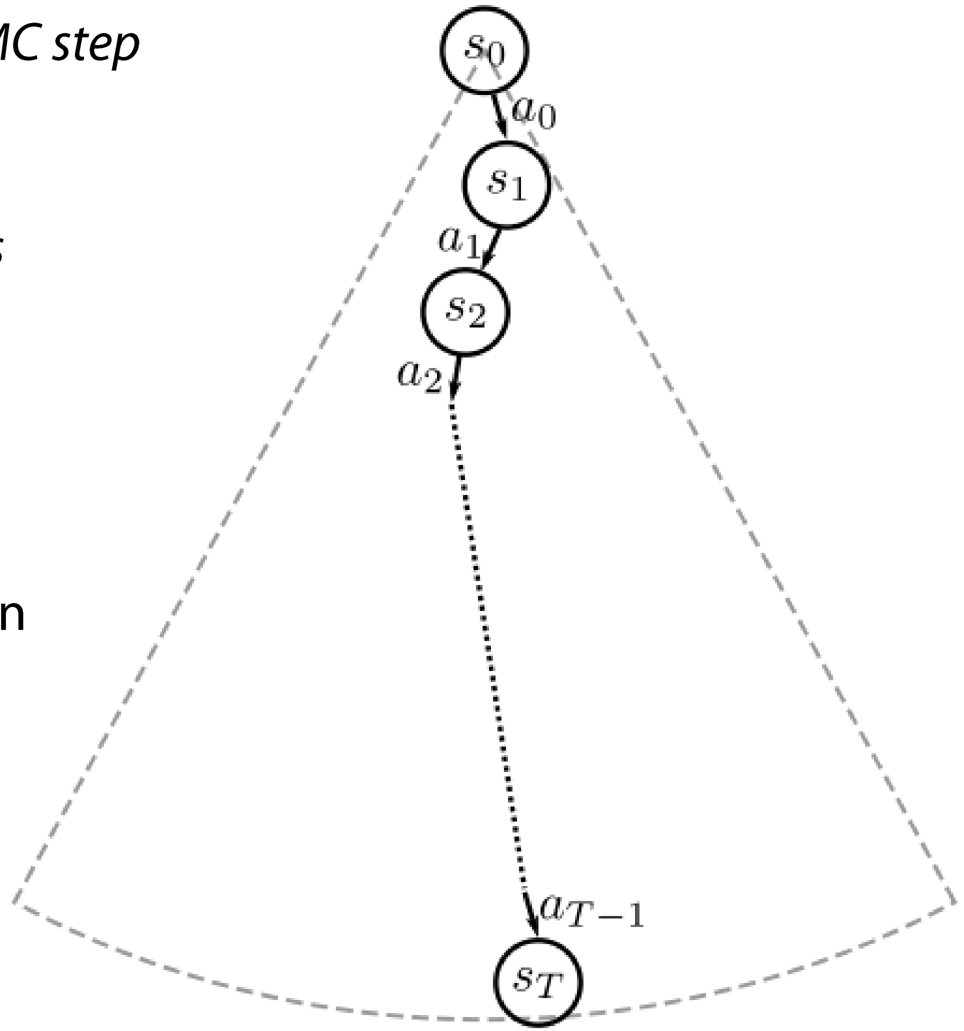
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t\!:=\!0$
  2) current state $s\!:=\!s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t\!:=\!t\!+\!1$
  6) repeat 2) to 5) until end game



Monte Carlo method

- ***Monte Carlo*** *method:*
  - *no memory* of past playouts in a single *MC step*
    (only the reward is saved)
  - no *transfer knowledge* between *MC steps*
  - *no construction* of game subtree
  - optimal policy only *partially* defined
    (on actually computed states)
  - *intrinsically stochastic* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - *no knowledge transfer*
    between *MC episodes*

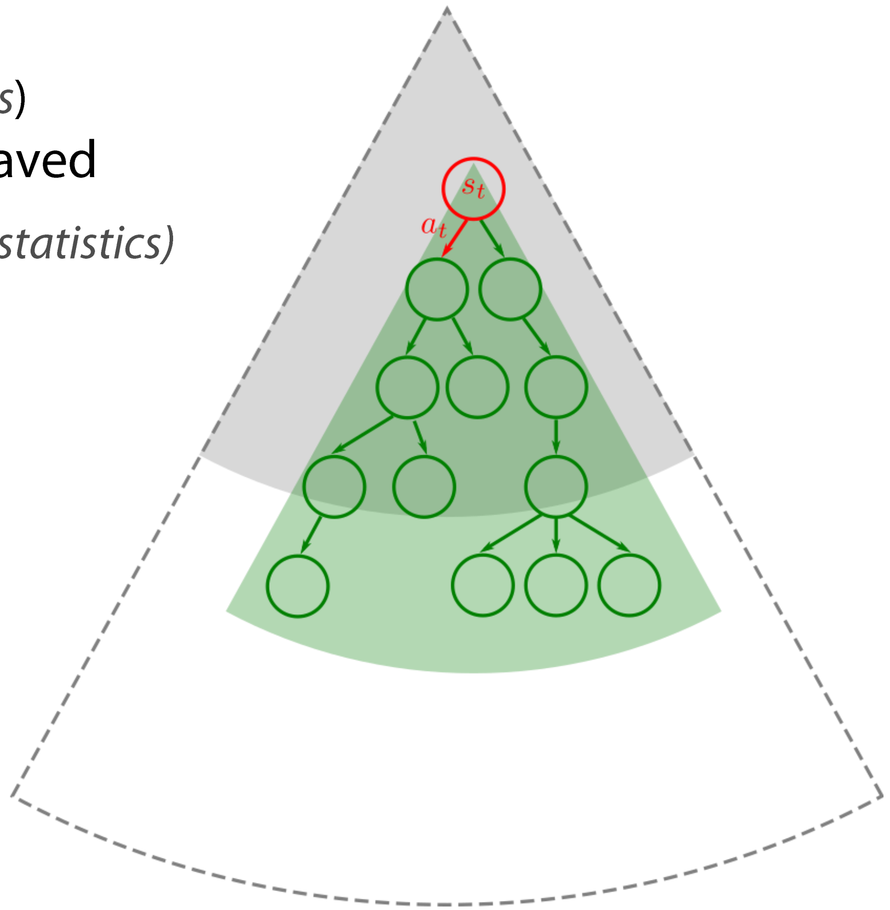


Monte Carlo Tree Search (MCTS):

simulation + incremental expansion

MCTS episode: basic idea

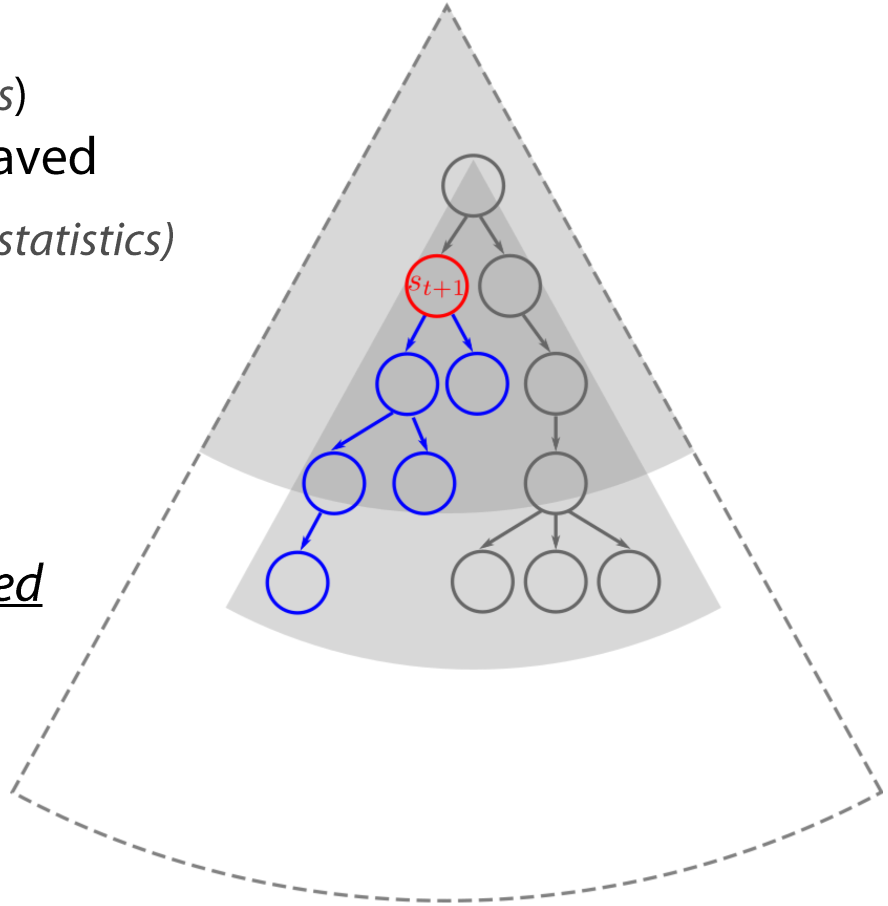
- *At each step (with current state s_t):*
  - a <u>*subgraph*</u> G_t with root s_t is created
  - <u>*statistics*</u> (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed



MCTS episode: basic idea

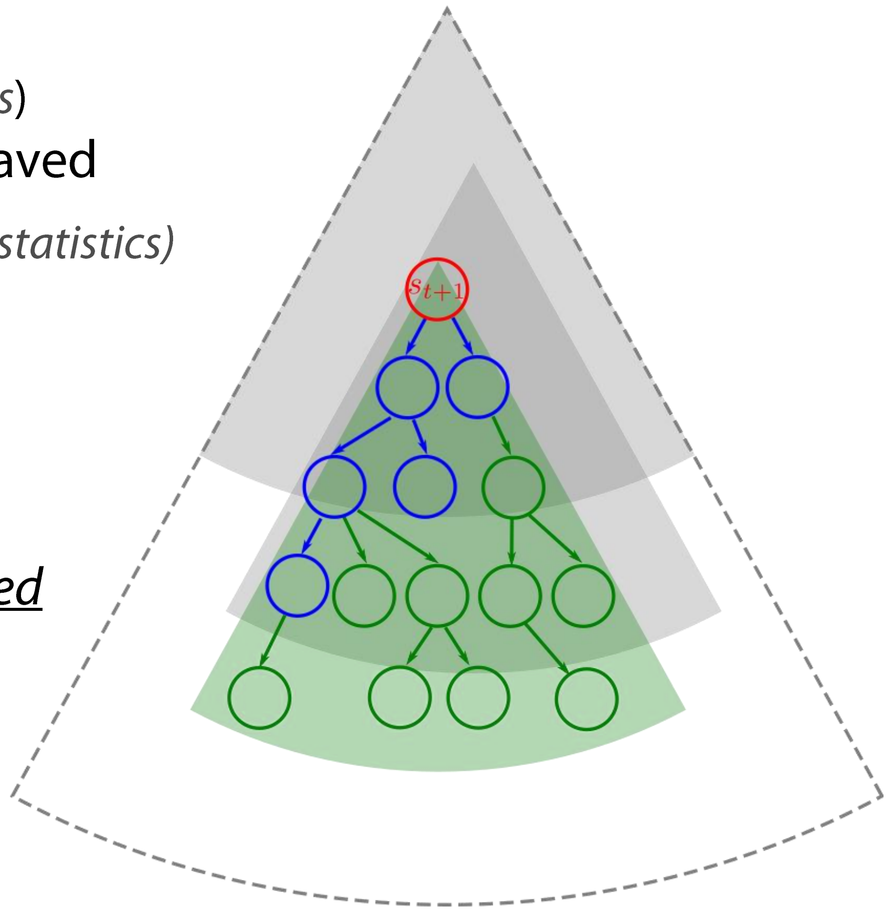
- *At each step (with current state s_t):*
  - a *subgraph* G_t with root s_t is created
  - *statistics* (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed

- *In the next step (with current state s_{t+1}):*
  - the subgraph of G_t with root s_{t+1} is *expanded* to create G_{t+1}
  - the statistics are *updated* and saved
  - best action a_{t+1} is decided
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed



MCTS episode: basic idea

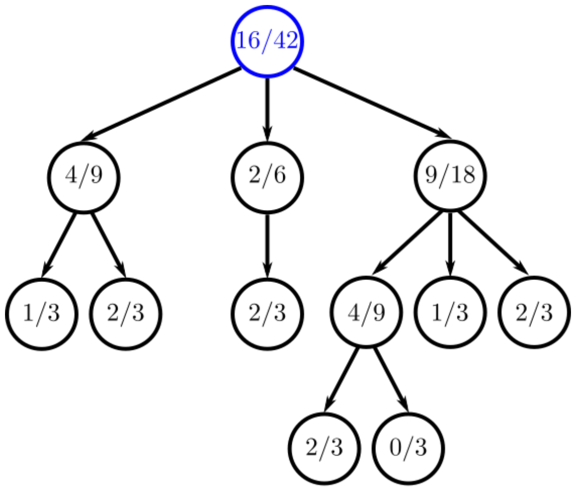
- *At each step (with current state s_t):*
  - a ***subgraph*** G_t with root s_t is created
  - ***statistics*** (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed



- *In the next step (with current state s_{t+1}):*
  - the subgraph of G_t with root s_{t+1} is ***expanded*** to create G_{t+1}
  - the statistics are ***updated*** and saved
  - best action a_{t+1} is decided
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed

Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]
    1) start from current state s *(and the –possibly empty– stored tree with root s)*



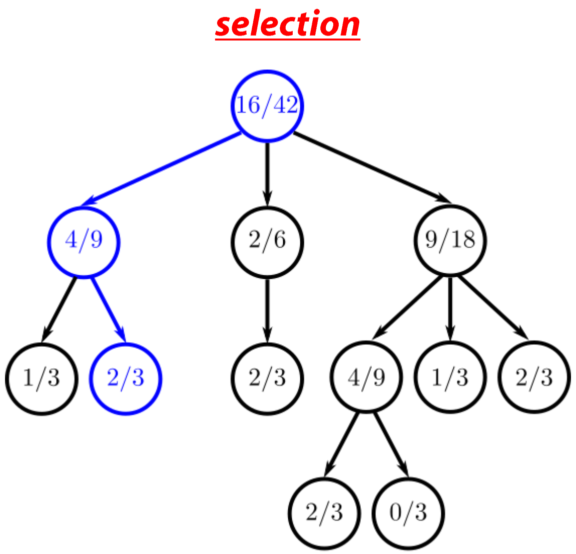
Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)** step:* [Coulom 2006]

  1) start from current state s *(and the –possibly empty– stored tree with root s)*
  2) traverse the tree by following the *selection policy*

$$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

  until encountering a *leaf node* s_L (i.e. a state not stored in the tree)



Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

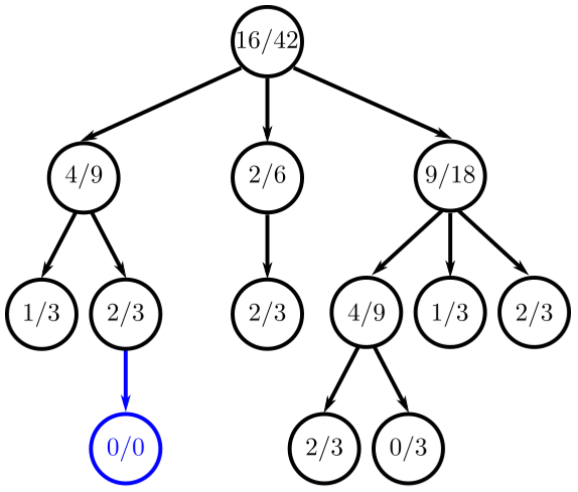
  1) start from current state s *(and the –possibly empty– stored tree with root s)*
  2) traverse the tree by following the *selection policy*

  $$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

  until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

  3) *expand* the tree by adding s_L

expansion



Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the *selection policy*

$$\pi^{\text{sel}} \; : \; s_t \mapsto a_t$$

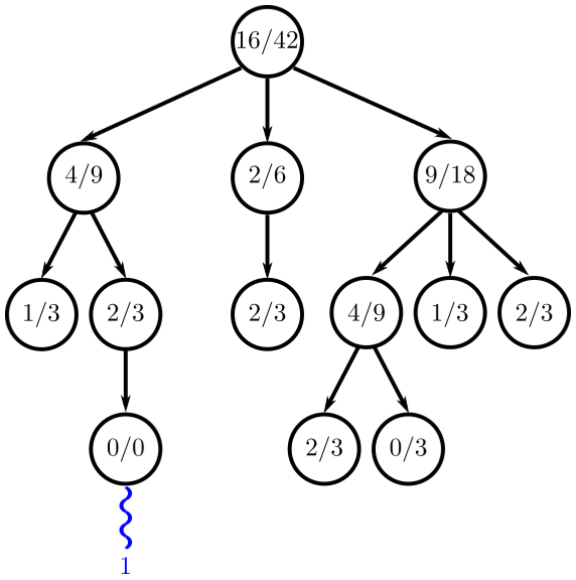
until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

3) *expand* the tree by adding s_L
4) play one random playout from state s_L by following the *simulation policy*

$$\pi^{\text{sym}} \; : \; s_t \mapsto a_t$$

and obtain the reward r

simulation



- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the *selection policy*

$$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

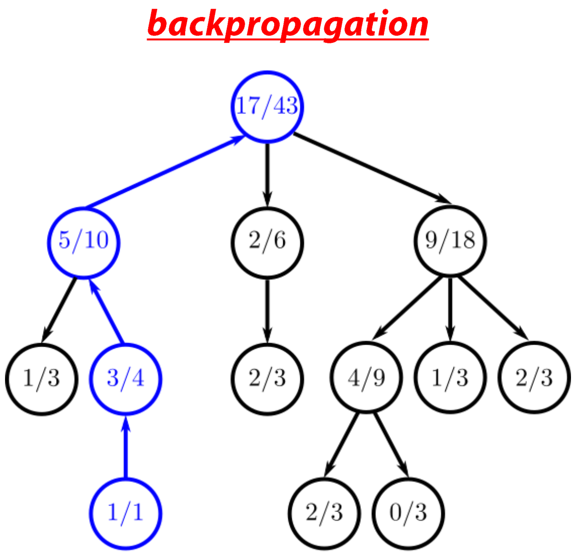
   until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

3) *expand* the tree by adding s_L
4) play one random playout from state s_L by following the *simulation policy*

$$\pi^{\text{sym}} : \; s_t \mapsto a_t$$

   and obtain the reward r

5) *backpropagate* r (and update the statistics of each encountered state and action)



Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

repeat m times:

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the ***selection policy***
$$\pi^{\mathrm{sel}} : \; s_t \mapsto a_t$$
until encountering a *leaf node* s_L (i.e. a state not stored in the tree)
3) ***expand*** the tree by adding s_L

repeat n times:

4) play one random playout from state s_L by following the ***simulation policy***
$$\pi^{\mathrm{sym}} : \; s_t \mapsto a_t$$
and obtain the reward r
5) ***backpropagate*** r (and update the statistics of each encountered state and action)

Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)** step:* [Coulom 2006]

repeat m times:

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the *selection policy*

$$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

3) *expand* the tree by adding s_L

repeat n times:

4) play one random playout from state s_L by following the *simulation policy*

$$\pi^{\text{sym}} : \; s_t \mapsto a_t$$

and obtain the reward r

5) *backpropagate* r (and update the statistics of each encountered state and action)

6) decide the *best* action to be performed in s with the *greedy policy*

$$\pi^{\text{gre}} : \; s \mapsto a$$

MCTS statistics: expansion and backpropagation

- ***MCTS** statistics* for state s and action a:

  $N(s)$ = total number of times state s has been visited

  $N(s,a)$ = number of times action a has been selected in state s

  $\hat{Q}(s,a)$ = estimated outcome of action a when selected in state s

- *Expansion initialization:* $N(s) := 0, \quad N(s,a) := 0, \quad \hat{Q}(s,a) := 0$

- *Backpropagation update* after a single playout with reward r:

$$N(s) := N(s) + 1$$
$$N(s,a) := N(s,a) + 1$$
$$\hat{Q}(s,a) := \hat{Q}(s,a) + \frac{r - \hat{Q}(s,a)}{N(s,a)}$$

MCTS: greedy, selection and simulation policies

- *Greedy policy:*

$$\pi^{\text{gre}}(s) := \underset{N(s,a)>0}{\operatorname{argmax}} \hat{Q}(s,a)$$

- *Selection policy:* ***Upper Confidence Bound applied to Trees (UCT)***

parameter
(default=1)

$$\pi^{\text{sel}}(s) := \pi^{\text{UCT}}(s) := \underset{N(s,a)>0}{\operatorname{argmax}} \left\{ \hat{Q}(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}} \right\}$$

exploitation
of actions
that look currently the best

exploration
of currently suboptimal-looking actions
(no good alternatives are missed
because of early estimation errors)

Convergence [Kocsis 2006]: for the first state s of a single *MCTS* episode

$$\pi^{\text{UCT}}(s) \to a^* := \pi^*(s) \quad \text{for } n \to +\infty$$

MCTS: greedy, selection and simulation policies

- Greedy policy:

$$\pi^{\text{gre}}(s) := \operatorname*{argmax}_{N(s,a)>0} \hat{Q}(s,a)$$

- Selection policy: **Upper Confidence Bound applied to Trees (UCT)**

$$\pi^{\text{sel}}(s) := \pi^{\text{UCT}}(s) := \operatorname*{argmax}_{N(s,a)>0} \left\{ \hat{Q}(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}} \right\}$$

- Simulation policy: **Random Uniform Policy**

$$\pi^{\text{sym}}(s) := a \qquad \text{with } P(s,a) = \frac{1}{|\mathcal{A}(s)|}$$

set of admissible actions in state s

Monte Carlo Tree Search (MCTS) step

Algorithm 2 UCT

```
procedure UCTSEARCH(s_0)
    while time remaining do
        {s_0, ..., s_T}, R = SIMULATE(s_0)
        BACKUP({s_0, ..., s_T}, R)
    end while
    return argmax_{a∈A} Q(s_0, a)
end procedure

procedure SIMULATE(s_0)
    t = 0
    R = 0
    repeat
        if s_t ∈ T then
            a = UCB1(s_t)
        else
            NEWNODE(s_t)
            a_t = DEFAULTPOLICY(s_t)
        end if
        s_{t+1} = SAMPLETRANSITION(s_t, a_t)
        r_{t+1} = SAMPLEREWARD(s_t, a_t, s_{t+1})
        R = R + r_{t+1}
        t += 1
    until Terminal(s_t)
    return {s_0, ..., s_t}, R
end procedure

procedure UCB1(s)
    a* = argmax_a Q(s, a) + c sqrt(2 log N(s) / N(s,a))
    return a*
end procedure

procedure BACKUP({s_0, ..., s_T}, R)
    for t = 0 to T − 1 do
        N(s_t) += 1
        N(s_t, a_t) += 1
        Q(s_t, a_t) += (R − Q(s_t,a_t)) / N(s_t,a_t)
    end for
end procedure

procedure NEWNODE(s)
    N(s) = 0
    for all a ∈ A do
        N(s, a) = 0
        Q(s, a) = ∞
    end for
    T.Insert(s)
end procedure
```

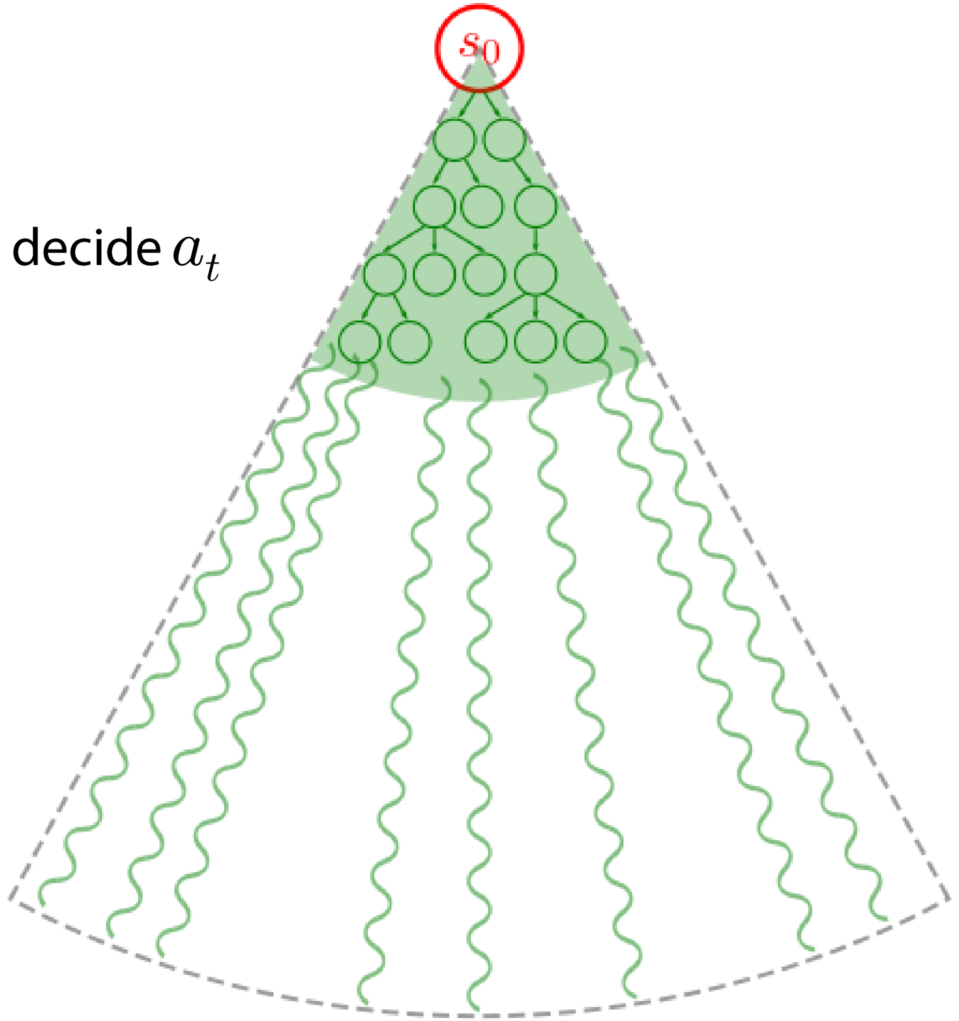
$$a^* = \underset{a}{\operatorname{argmax}}\ Q(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}}$$

$$Q(s_t,a_t) \mathrel{+}= \frac{R - Q(s_t,a_t)}{N(s_t,a_t)}$$

From: D. Silver, Reinforcement Learning and Simulation-Based Search in Computer Go, PhD Thesis, 2009

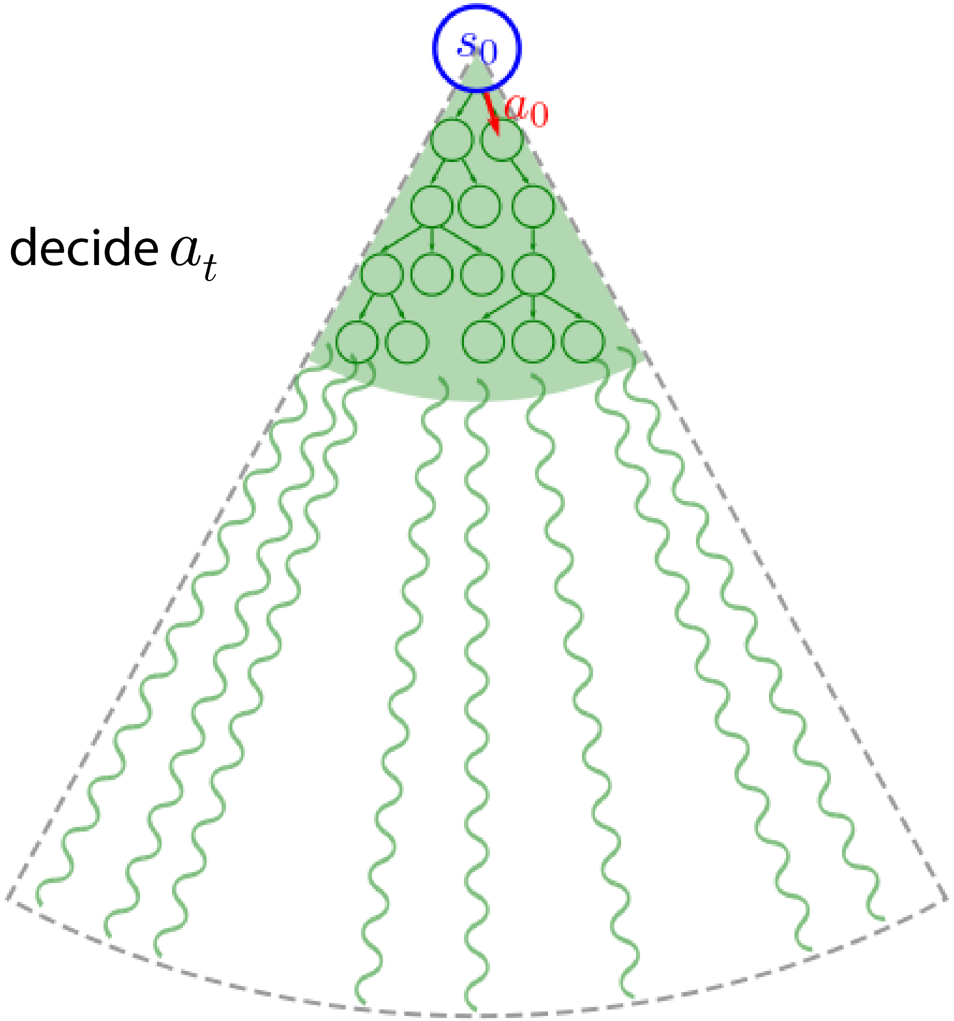
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t + 1$
  6) repeat steps 2-5 until end game



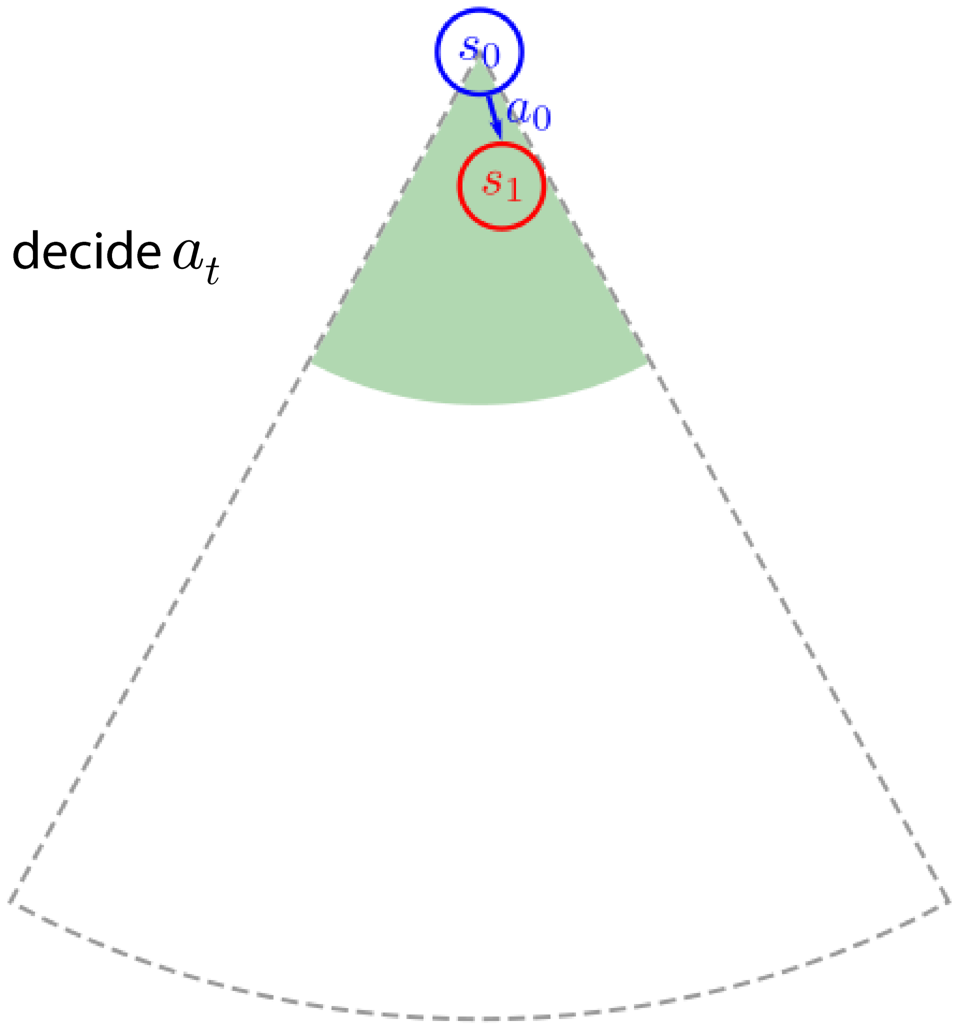
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat steps 2-5 until end game



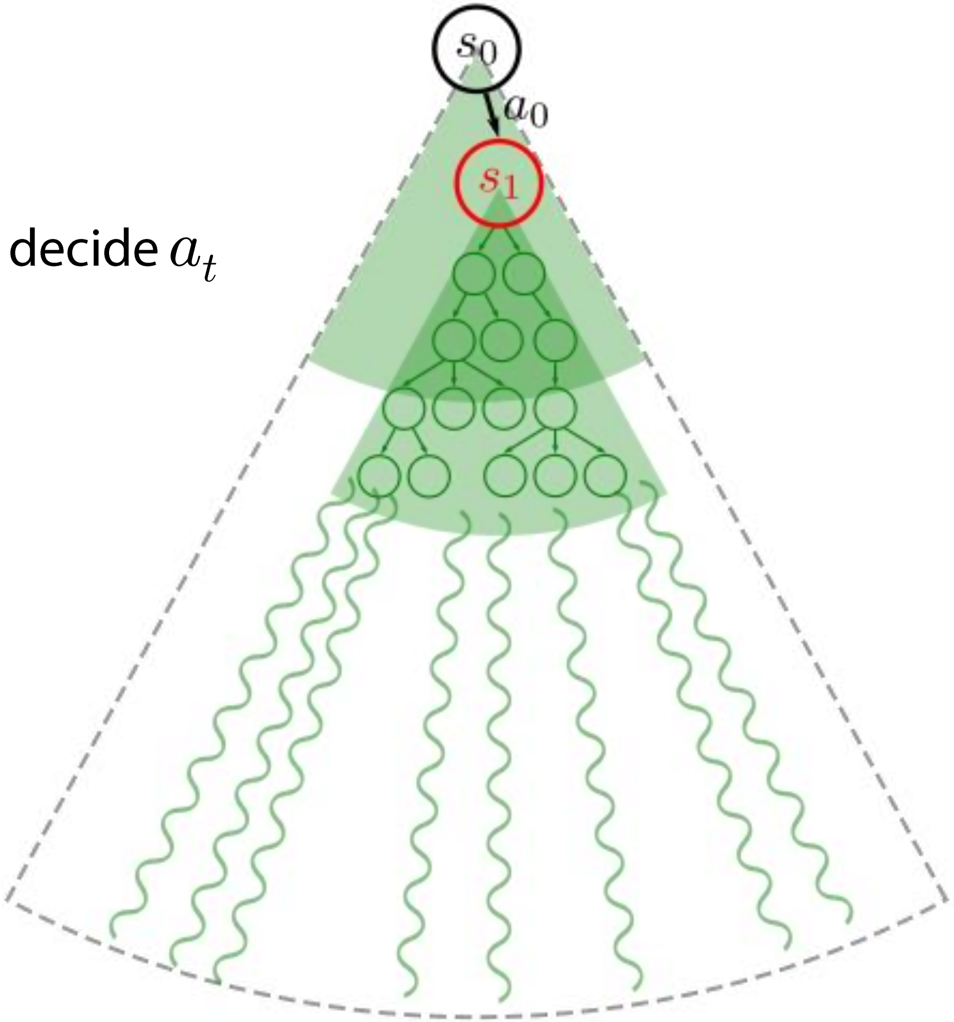
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t + 1$
  6) repeat steps 2-5 until end game



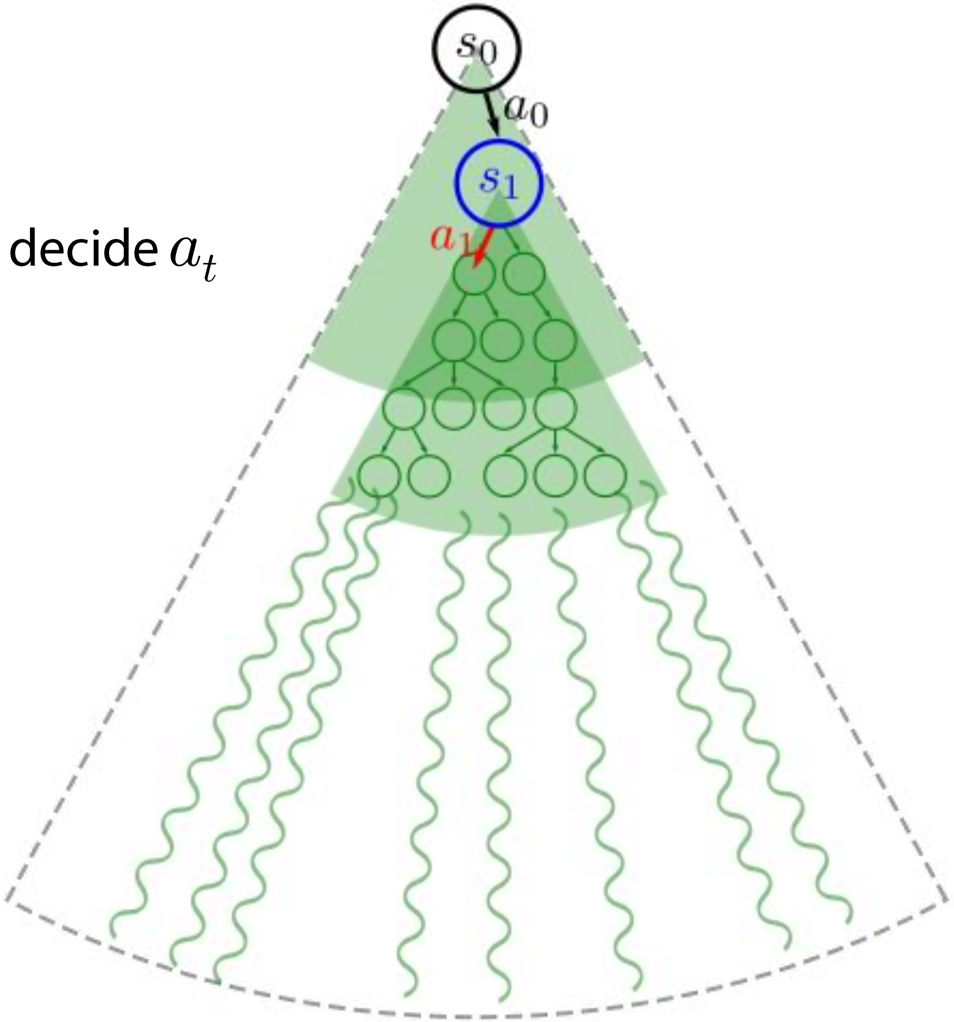
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat steps 2-5 until end game



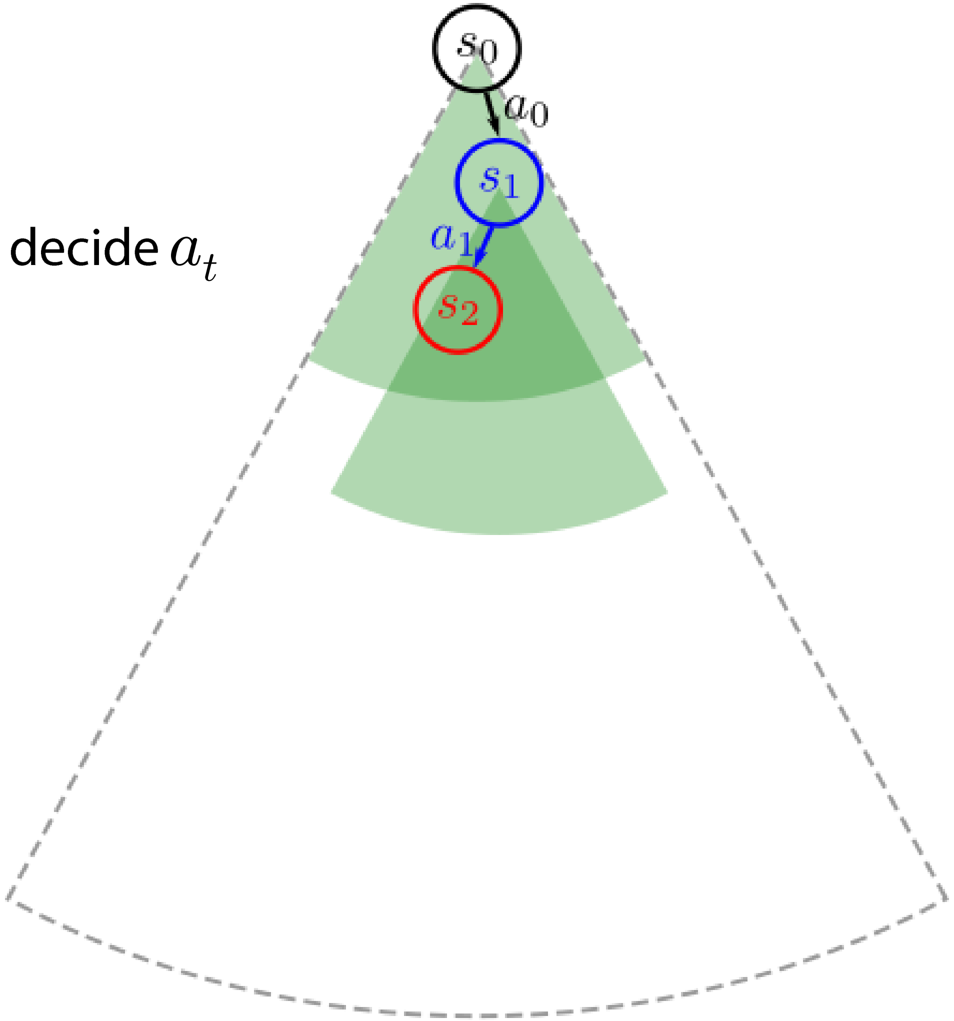
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat steps 2-5 until end game



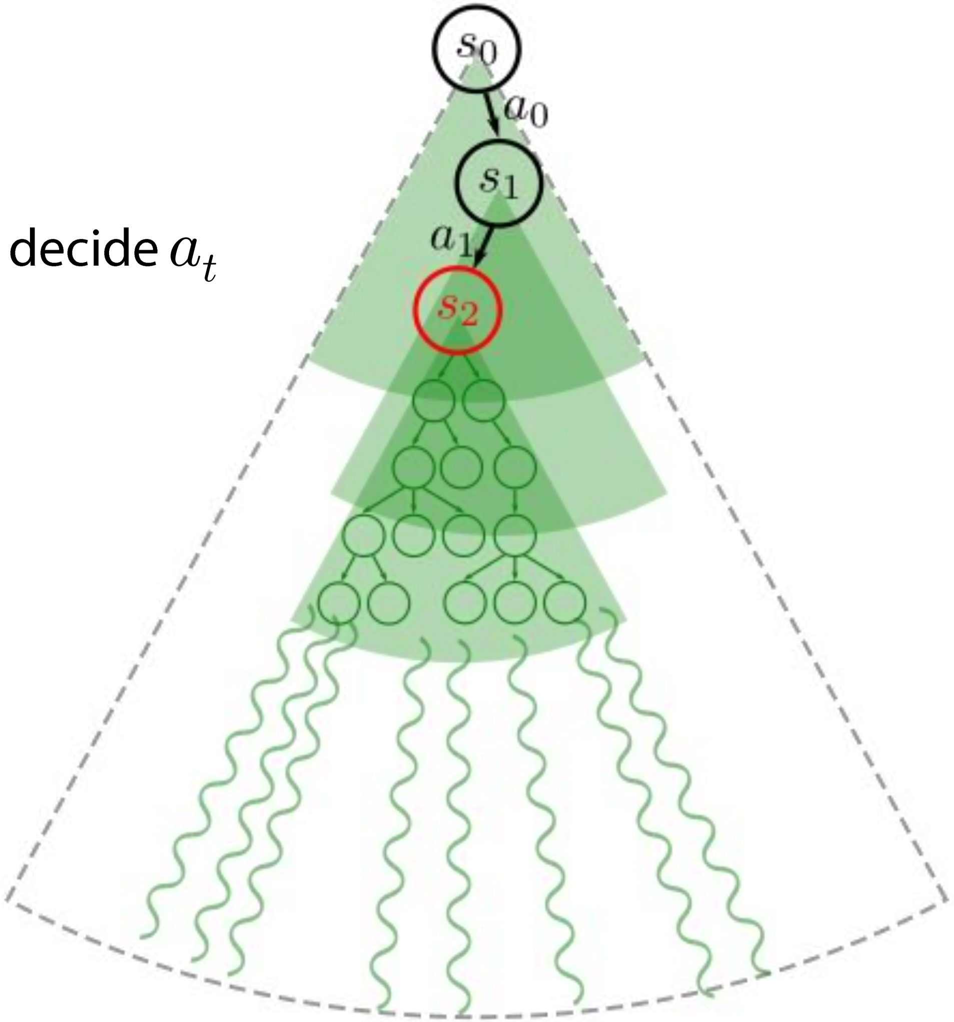
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat steps 2-5 until end game



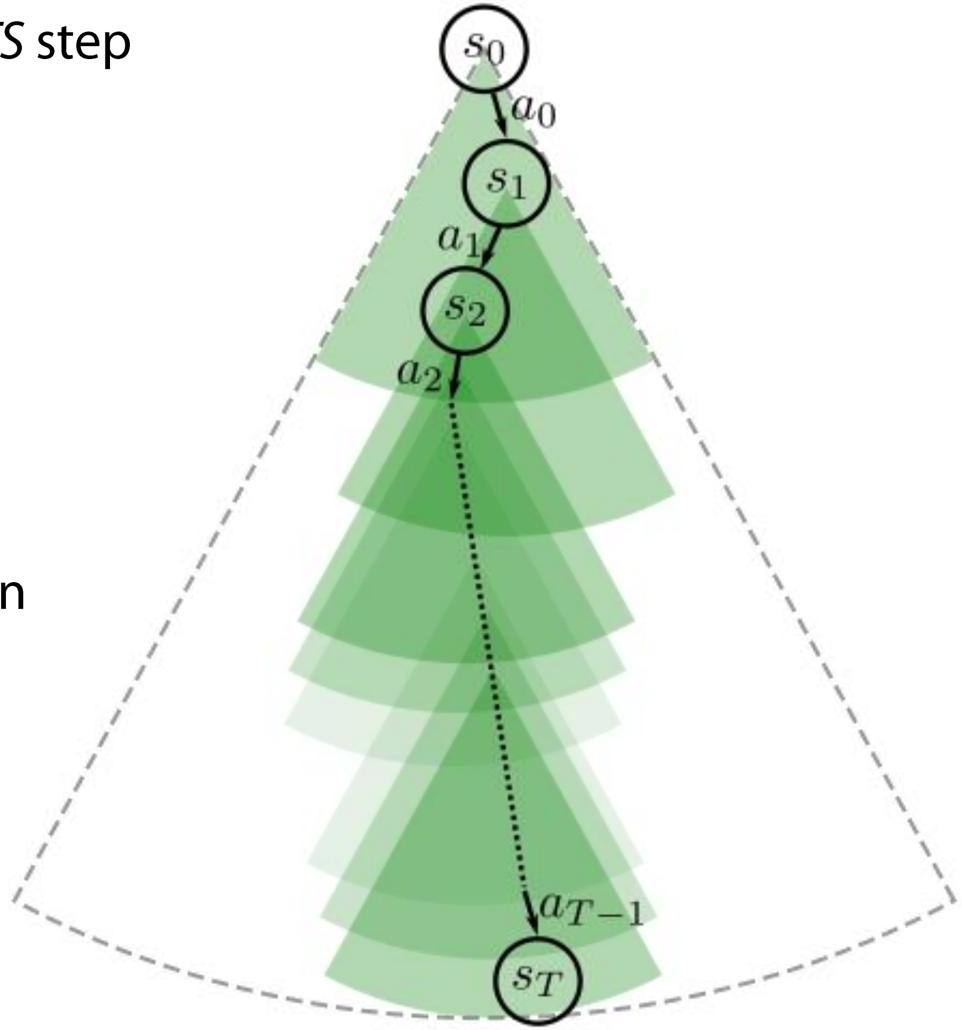
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat steps 2-5 until end game



Monte Carlo Tree Search (MCTS) method

- ***Monte Carlo Tree Search*** *method:*
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - optimal policy only *partially* defined
    (on actually computed states)
  - *intrinsically stochastic* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - What about *knowledge transfer*
    between *MCTS episodes*?
    transferring the entire MCTS tree
    would rapidly cause its explosive growth…



Dealing with

Stochasticity and Uncertainty

Stochasticity and Uncertainty: general setting

- Stochastic reward:
  - *immediate reward* $r(s_t, a_t)$ is obtained when performing action a_t in state s_t
  - *delayed reward* is obtained only at the end of the game

$$r(s_t) := \begin{cases} 0 & \text{if } s_t \text{ is not a terminal state} \\ r & \text{otherwise} \end{cases}$$

  possibly with $P(r \mid s_t, a_t)$ or $P(r \mid s_t)$ respectively

- Stochastic policy:

  policy $\pi(s, a) := P(a \mid s)$ is a *probability distribution*

- Uncertainty of execution:

  stochastic transition function $\tau : (s_t, a_t) \mapsto s_{t+1}$ with $P(s_{t+1} \mid s_t, a_t)$

Reinforcement Learning (RL)

- *Value function:*

$$V^{\pi}(s) := \mathbb{E}_{\pi}[R \mid s_0 = s]$$

mean over the trajectories following policy π

Optimal value: $V^*(s) := \max_{\pi} V^{\pi}(s) \ \forall s$

- *Action-value function:*

$$Q^{\pi}(s_t, a) := \mathbb{E}_{\pi}[R \mid s_0 = s, a_0 = a]$$

Optimal action-value: $Q^*(s, a) := \max_{\pi} Q^{\pi}(s, a) \ \forall s, a$

Optimal policy: $a^*(s) = \operatorname{argmax}_a [Q^*(s, a)]$

Connection: $V^{\pi}(s) = \mathbb{E}_{\pi}[Q^{\pi}(s, a)]$ and $V^*(s) = \max_a [Q^*(s, a)]$