An Introduction to Monte-Carlo Tree Search

Antoine Cornuéjols

AgroParisTech – INRAe Paris-Saclay

antoine.cornuejols@agroparistech.fr

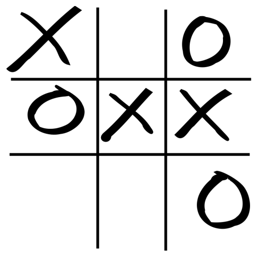
Plan

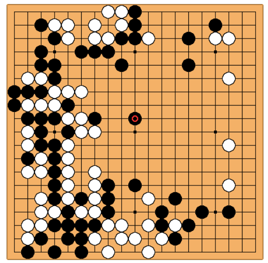
1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

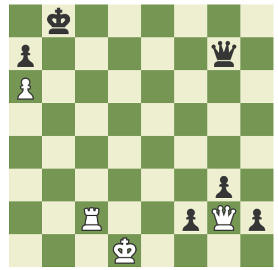
Conventional game tree search

1. **Perfect-information** games

   All aspects of the states are **fully observable**



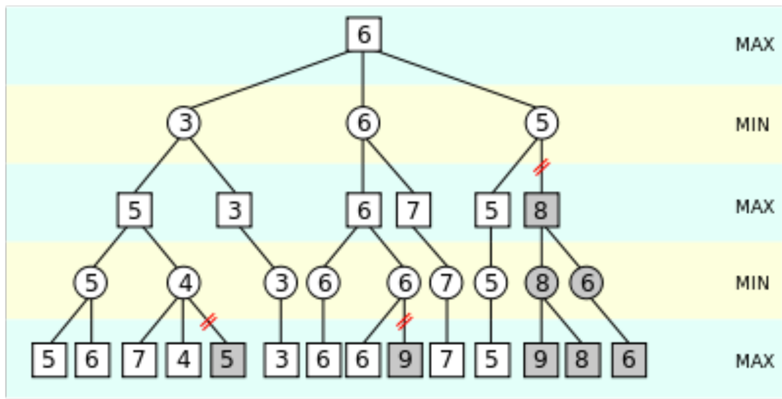




2. Technique: **MinMax** algorithm (with Alpha-Beta pruning)

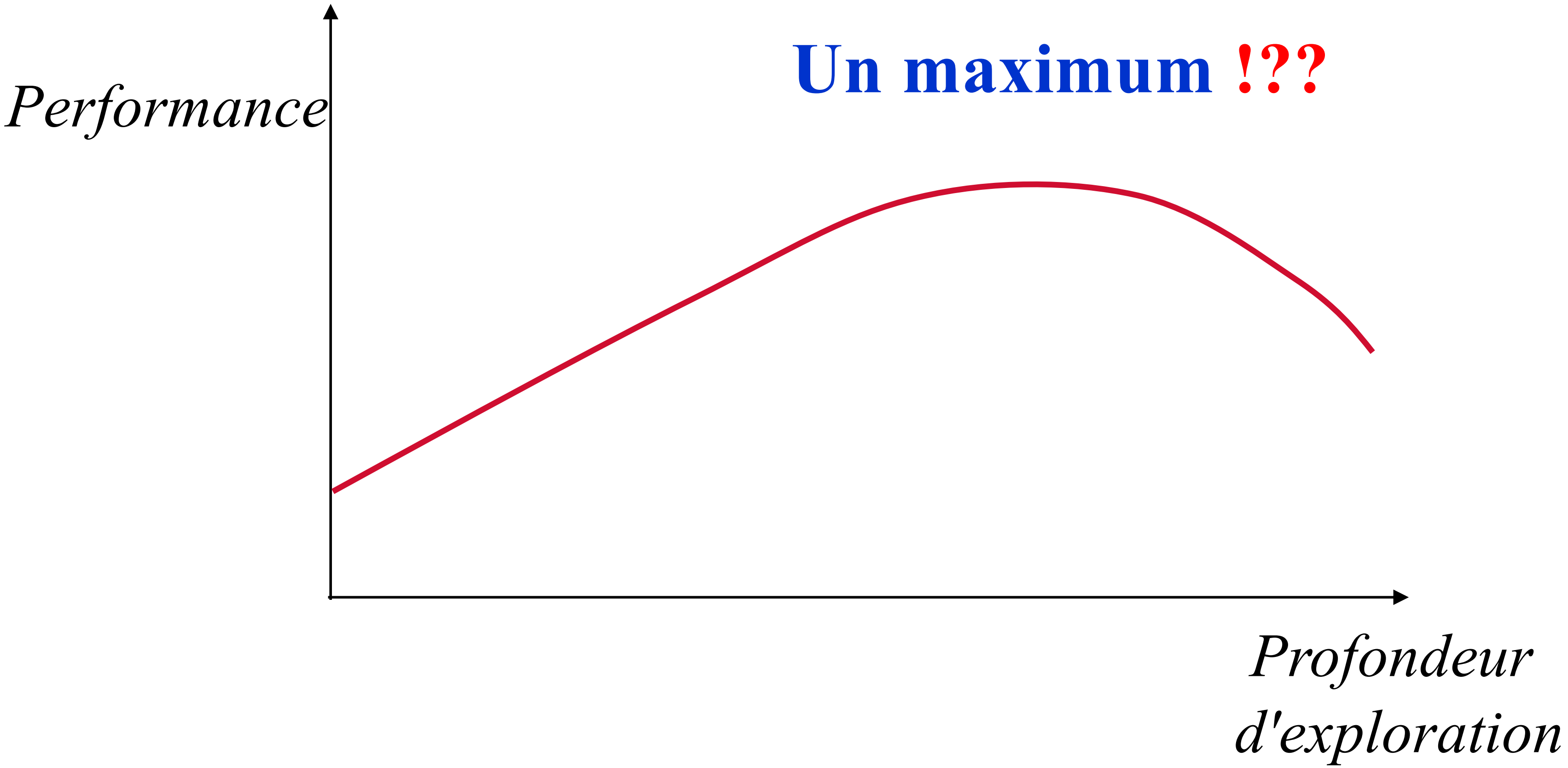
   Effective for

   - **Modest branching** factor
   - When a **good evaluation function** is available



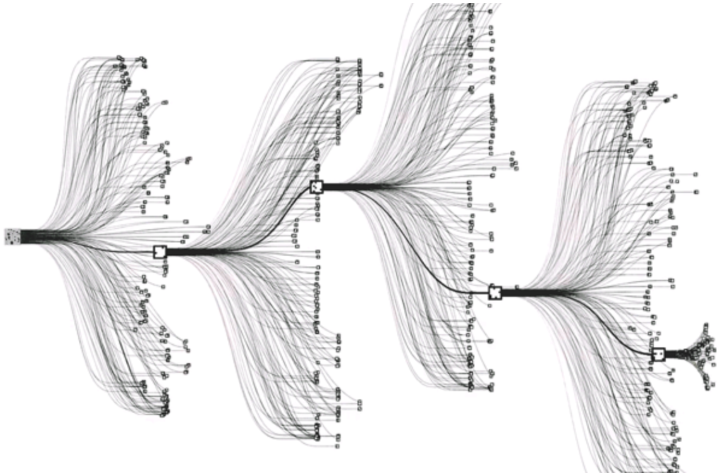
L' algorithme MinMax. Des jeux pathologiques ?!

- Analyse des raisons du succès de MinMax (et de ses limites)
- Un **comportement bizarre**



→ Importance de la fonction d’évaluation

Problems with Go



- The **branching factor is large**

  $b \approx 250$ on average ; depth > 200 moves

- We do **not** know a **good evaluation function**

  Similar looking positions may have completely different outcomes

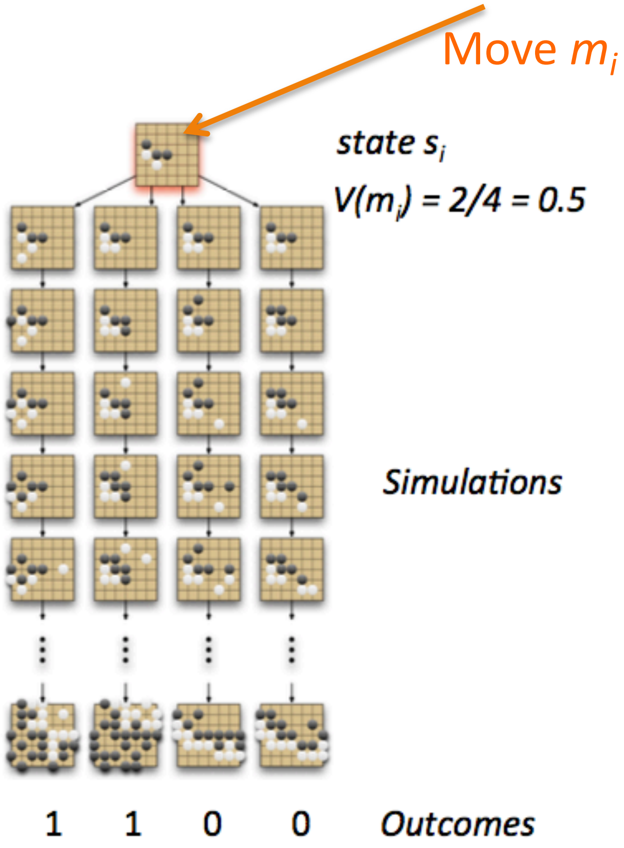
→ Alpha-Beta gives **weak to intermediate level** of play

Plan

Naïve approach: but interesting

Basic Monte-Carlo simulations

1. Simulate games using **random** moves
2. **Score** each game at the end
3. Store winning **statistics**
4. **Play** move with best winning percentage
5. Repeat



Use this a the **evaluation function**,

hoping it will ***preserve difference** between a good position and a bad one*

Limits

- **Use simulations directly as an evaluation function** for $\alpha\beta$

- Problems
  - Single simulation gives **very noisy evaluation**: 0/1 signal
  - **Running many simulations is required** for each position (move to consider)
  - Illustration
    - Typical speed of **chess** programs = 1 million eval/sec ; 30 moves to consider
      => 33,000 eval/move
    - **Go**: 1 million eval/sec ; 250 moves to consider ;
      => 4,000 eval/move.sec (not a lot for a complicated game)

→ Monte-Carlo was ignored for over 10 years in Go

Limits

- **Does not allocate search and evaluation wisely**:
  based on promise of the positions
- Evaluation is costly
  - We would like to have **precise evaluation in the promising regions** of the search space. It does not matter to be precise elsewhere as long as we are certain it is worthless.

→ The exploration vs. exploitation tradeoff

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

Idea

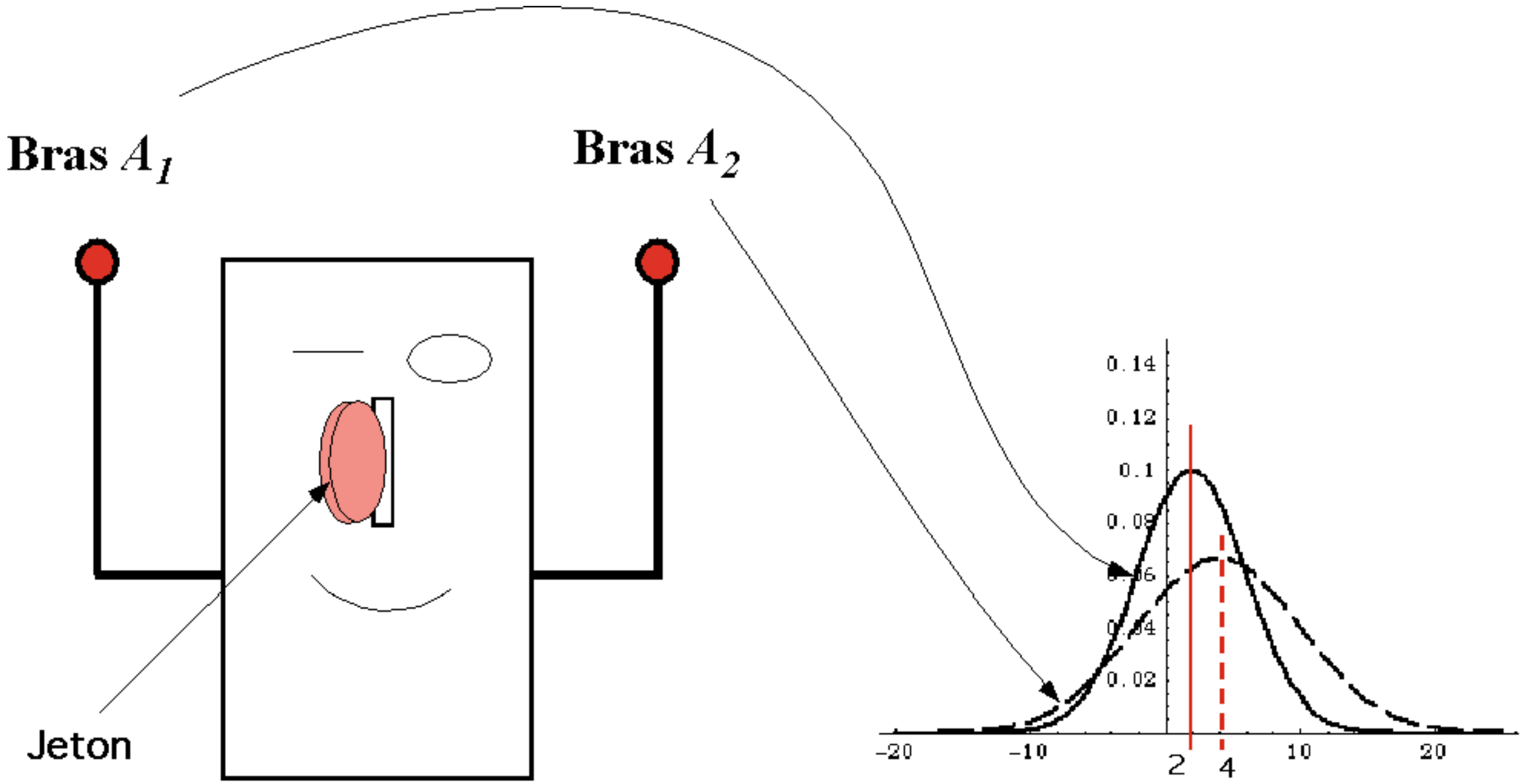
Use results of previous simulations to guide growth of the game tree

Politique de « sélection »

- **Exploitation**: play the **seemingly best move**
- **Exploration**: explore moves where the **uncertainty** is **highest**

→ Theory of bandits

Multi-Armed Bandit problem: illustration



Figure

Multi-Armed Bandit problem

Assumptions

- Choice among several arms
- Each arm pull is **independent** of other pulls
- Each arm pull is determined by a **distribution of unknown mean**

Which arm has the best average payoff?

Multi-Armed Bandit problem: illustration

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

Consider a row with *three slot machines*

- Each pull of an arm is either
  - A **win**: payoff **1**
  - A **loss**: payoff **0**
- Here A is the best arm → but we do not know that

Plan

Multi-Armed Bandit problem: illustration

A	B	C
		
P(A wins) = 60%	P(B wins) = 55%	P(C wins) = 40%

We want to **minimize** the "**regret**" = loss wrt. **the best sequence of pulls** (*if we had known* the best arm at the start)

→ Need to **balance** exploration and exploitation

Uniform *policy* ***Greedy*** *policy*

The ε-greedy algorithm

- **By default**, actions are chosen greedily.

  The action with the highest estimated reward is the selected action.

- However, at each time step, an action **may instead be selected at random** from the set of all possible actions.

  A random action is chosen with a probability 'ε' (Epsilon).

The ε-greedy algorithm

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Q(a) : estimate of the value of arm a

N(a) : number of draws of arm a

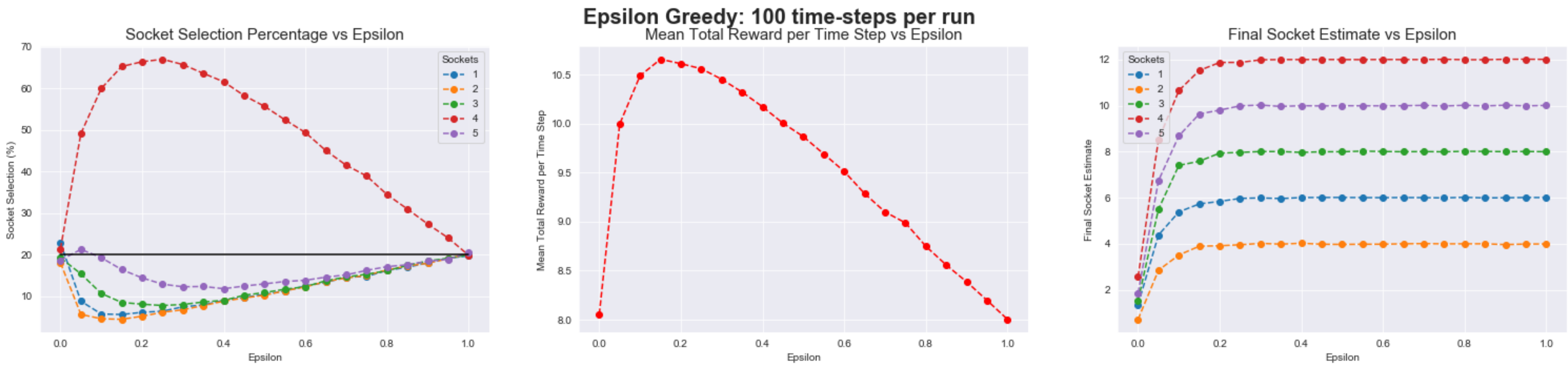
R : reward

From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

The ε-greedy algorithm

Illustration

- Suppose 5 arms with mean rewards :
  - Arm 1 : **6**   Arm 2 : **4**   Arm : 3 : **8**   Arm 4 : **12**   Arm 5 : **10**



The optimal value for ε seems to be ~ **0.2**

The ε-greedy algorithm

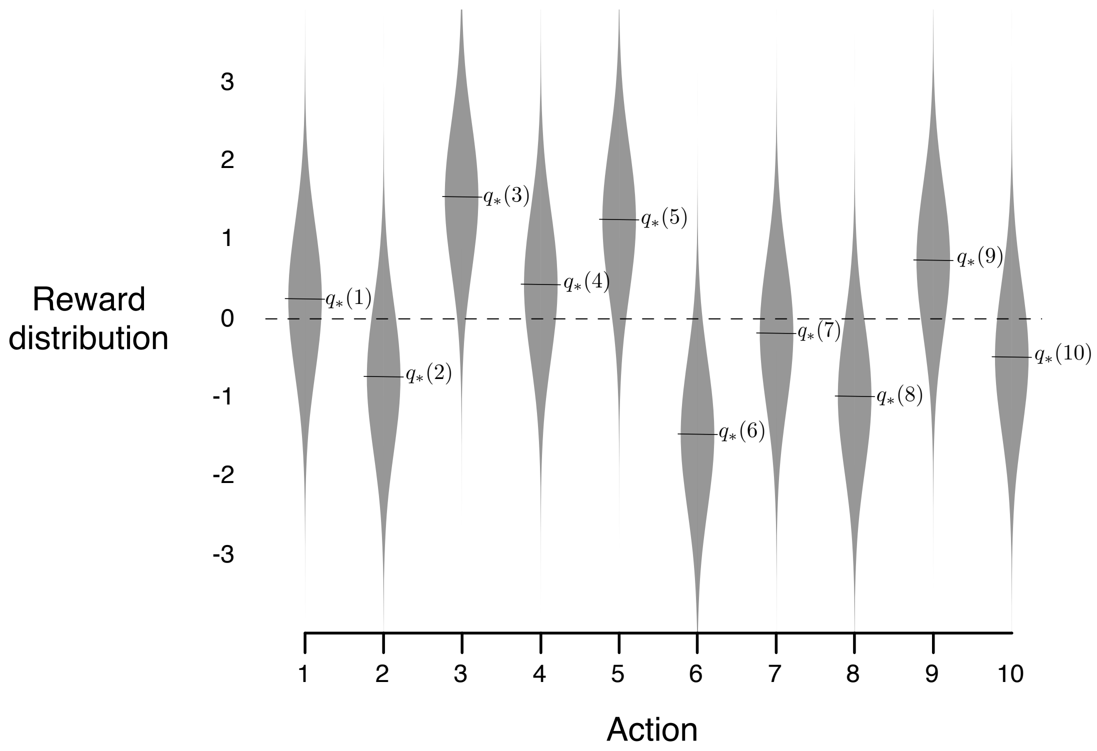
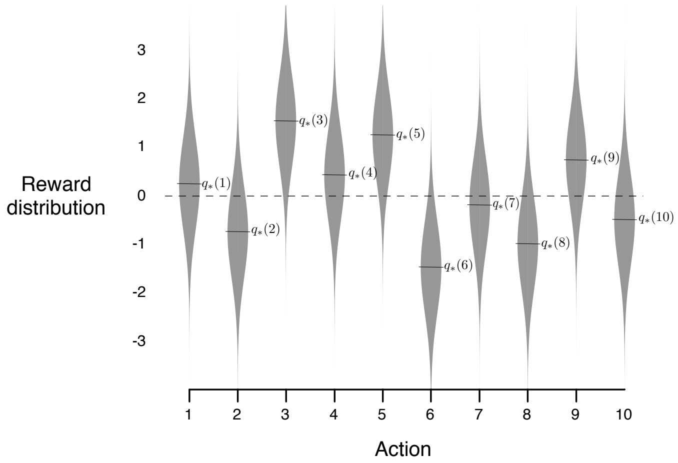


Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$, unit-variance normal distribution, as suggested by these gray distributions.

From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

The ε-greedy algorithm : choice for ε



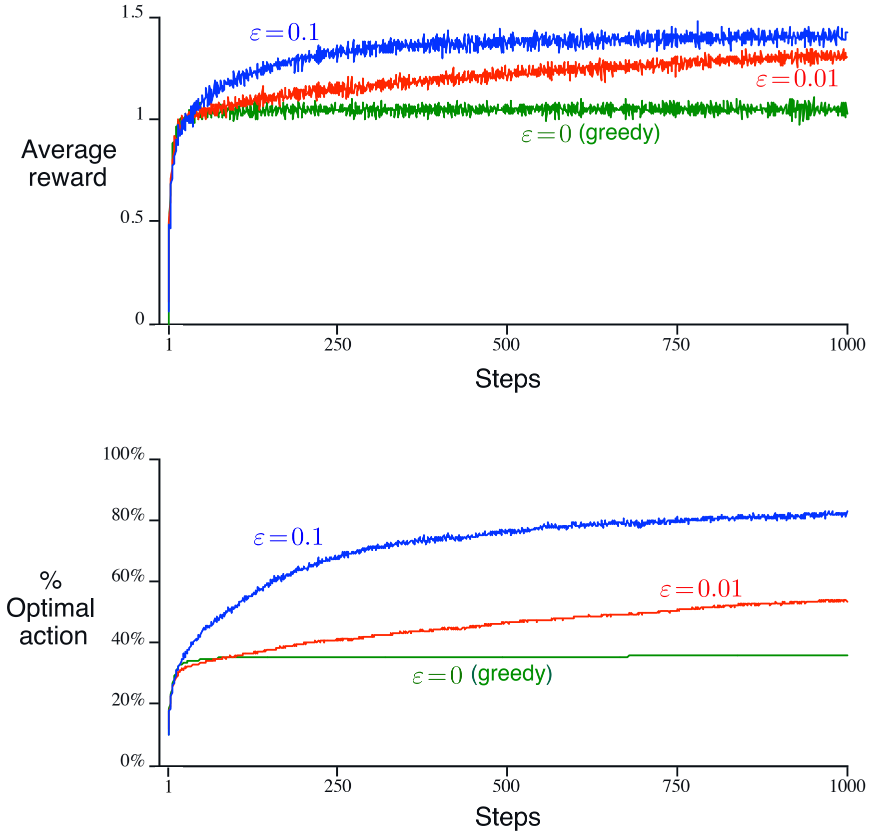


Figure 2.2: Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

…

" Regret "

- **Optimal** action $$a_* = \arg\max_a \mathbb{E}[R_t | A_t = a]$$

- **Regret** *L* after *T* time steps

$$L = \underbrace{T.\,\mathbb{E}[R_t | A_t = a_*]}_{\text{If I had known the best arm}} - \underbrace{\sum_{t=1}^{T} \mathbb{E}[R_t | A_t = a]}_{\text{The choices I actually made}}$$

$$L = \underbrace{T.\mathbb{E}[R_t|A_t = a_*]}_{\text{If I had known the best arm}} - \underbrace{\sum_{t=1}^{T} \mathbb{E}[R_t|A_t = a]}_{\text{The choices I actually made}}$$

$$\boxed{\varepsilon T \frac{(k-1)}{k} \Delta \leq L \leq \varepsilon T \frac{(k-1)}{k} \Delta'}$$

Linear in time *T*

k arms

Δ = difference between **best** arm and the **next** one

Δ' = difference between **best** arm and the **worst** one

PROOF : Over *T* time steps, εT of the actions will have been chosen randomly.

Only one of the *k* arms gives maximal reward.

The remaining *k*-1 arms give sub-optimal reward.

Therefore, there are $\varepsilon T(k-1)/k$ rounds in which a sub-optimal arm is selected.

If at each time, this is the second best arm, then the left inequality follows.

" Regret "

- **Regret** *L* after *T* time steps

$$\varepsilon\, T\, \frac{(k-1)}{k}\, \Delta \;\leq\; L \;\leq\; \varepsilon\, T\, \frac{(k-1)}{k}\, \Delta'$$

$$\boxed{\varepsilon\, T\, \frac{(k-1)}{k}\, \Delta \;\leq\; L \;\leq\; \varepsilon\, T\, \frac{(k-1)}{k}\, \Delta'}$$

Linear in time *T*

k arms

Δ = difference between **best** arm and the **next** one

Δ' = difference between **best** arm and the **worst** one

Good for a naïve approach. But we can do better

See below the **UCB algorithm**

Plan

1. Limites de l'approche classique
2. Évaluation par Monte-Carlo
3. Le compromis Exploration vs. Exploitation : algorithmes de bandits
4. Approche ε-greedy
5. UCT = MCTS + UCB
6. Illustrations
7. AlaphaGo Zero

The **Upper Confidence Bound** algorithm

Principle: **Optimism** in the face of uncertainty

- **Choose the action** as if the environment is as nice as is **plausibly** possible

  The unknown mean payoff of each arm is as large as plausibly possible based on the data that has been observed

Intuition. One of two things can happen

1. The **optimism was justified**: the learner is acting optimally
2. The optimism was **not justified**: the agent takes some action that he believes might give large reward when in fact it does not.

   If this happens sufficiently often, then **the learner will learn what is the true payoff of this action** and **not choose it in the future**

The Upper Confidence Bound algorithm

- Optimism in the face of uncertainty

– Estimated mean of arm B > estimated mean of arm A

But – Estimated upper bound of arm B < estimated upper bound of arm A

Choose arm A

What "plausible" means

If $X_1, X_2, \ldots, X_n$ are independent and 1-subgaussian (which means that $\mathbb{E}(X_i) = 0$), and $\hat{\mu} = \sum_{i=1}^{n} X_t / n$, then:

$$\mathbf{P}(\hat{\mu} \geq \varepsilon) \; \leq \; \exp(-n\varepsilon^2/2)$$

Equating the right-hand side with δ and solving for ε leads to

$$\mathbf{P}\left\{\hat{\mu} \; \geq \; \sqrt{\frac{2}{n}\log\left(\frac{1}{\delta}\right)}\right\} \; \leq \; \delta$$

Suggests a definition of "as large as plausibly possible":

Take $\quad \hat{\mu} \; + \; \sqrt{\frac{2}{n}\log\left(\frac{1}{\delta}\right)} \quad$ as the plausible value

Choice of a good action

When the agent is deciding what to do in round t, it has observed $T_i(t-1)$ samples from arm i and observed rewards with an empirical mean of $\hat{\mu}(t-1)$ for it. Then **the agent should choose the action** i **that maximizes:**

$$i = \underset{j \in \text{actions}}{\text{ArgMax}} \left\{ \hat{\mu}_j(t-1) + \sqrt{\frac{2}{T_j(t-1)} \log\left(\frac{1}{\delta}\right)} \right\}$$

Explore more the arms with highest mean
and not well explored

δ is called the "confidence level" and different choices lead to different algorithms. We will take: $1/\delta = f(t) = 1 + t\log^2(t)$

...

Upper Confidence Bound (UCB)

UCB1 formula [Auer et al. 2002]

- The "true value" of arm *i* is expected to be in some **confidence interval** around v_i

w_i : **nb de parties gagnées** à partir du nœud *i*

n_i **: nb de de fois** où le nœud *i* a été **visité**

$$\frac{w_i}{n_i}$$

$$\underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(\overbrace{N}^{\text{total number of trials}})}{\underbrace{n_i}_{\text{num trials for arm i}}}}$$

Upper Confidence Bound (UCB)

UCB1 formula [Auer et al. 2002]

- Policy

  1. First, try each arm once
  2. Then **at each time step**:

     Choose arm *i* that maximizes the UCB1 formula

$$\underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(N)}{n_i}}$$

N: total number of trials so far

n_i: num trials for arm i

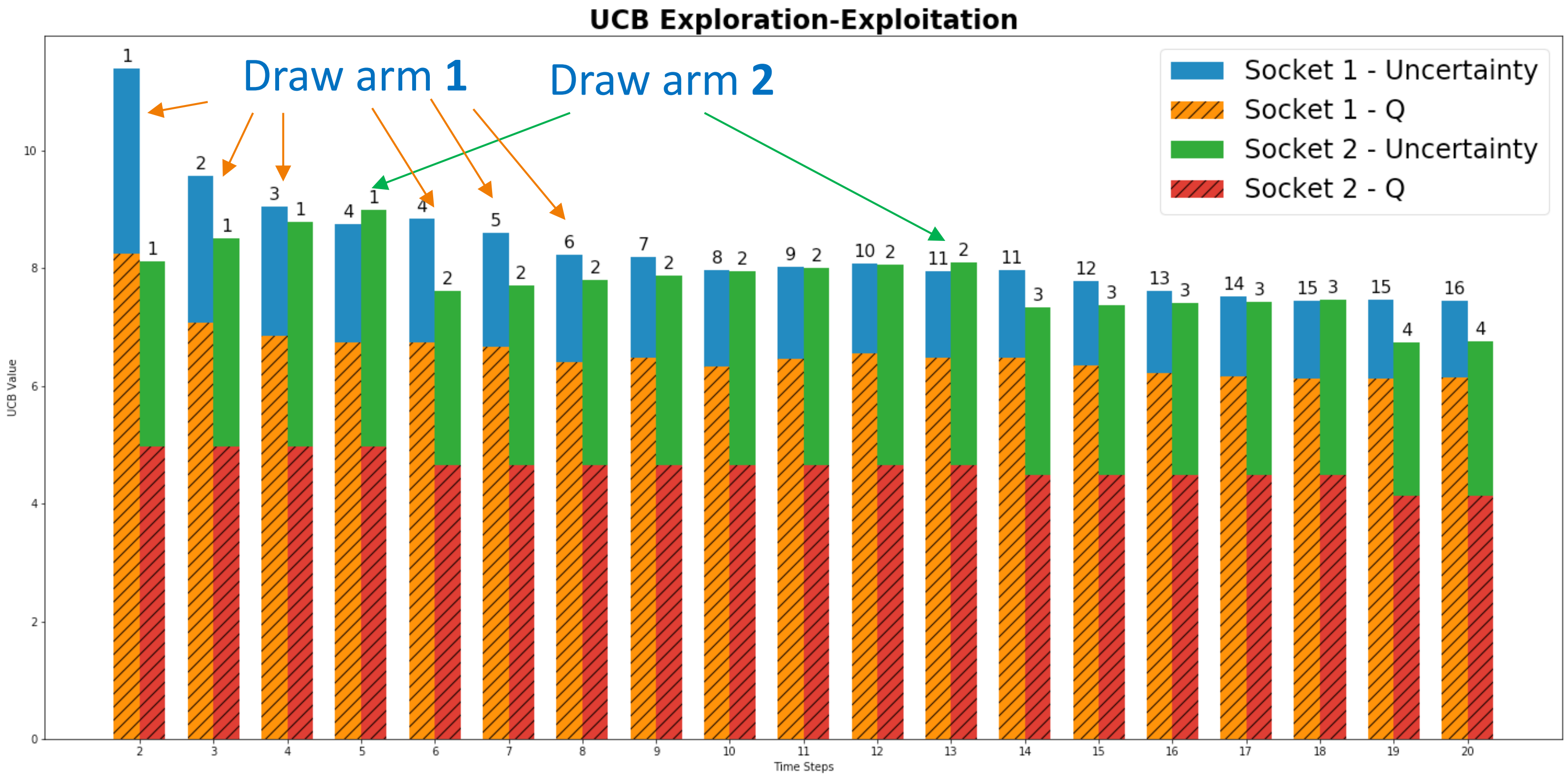
- Let us look at a very simple scenario with **two arms**

  - **Arm 1** with mean value of **6**

  - **Arm 2** with mean value of **4**

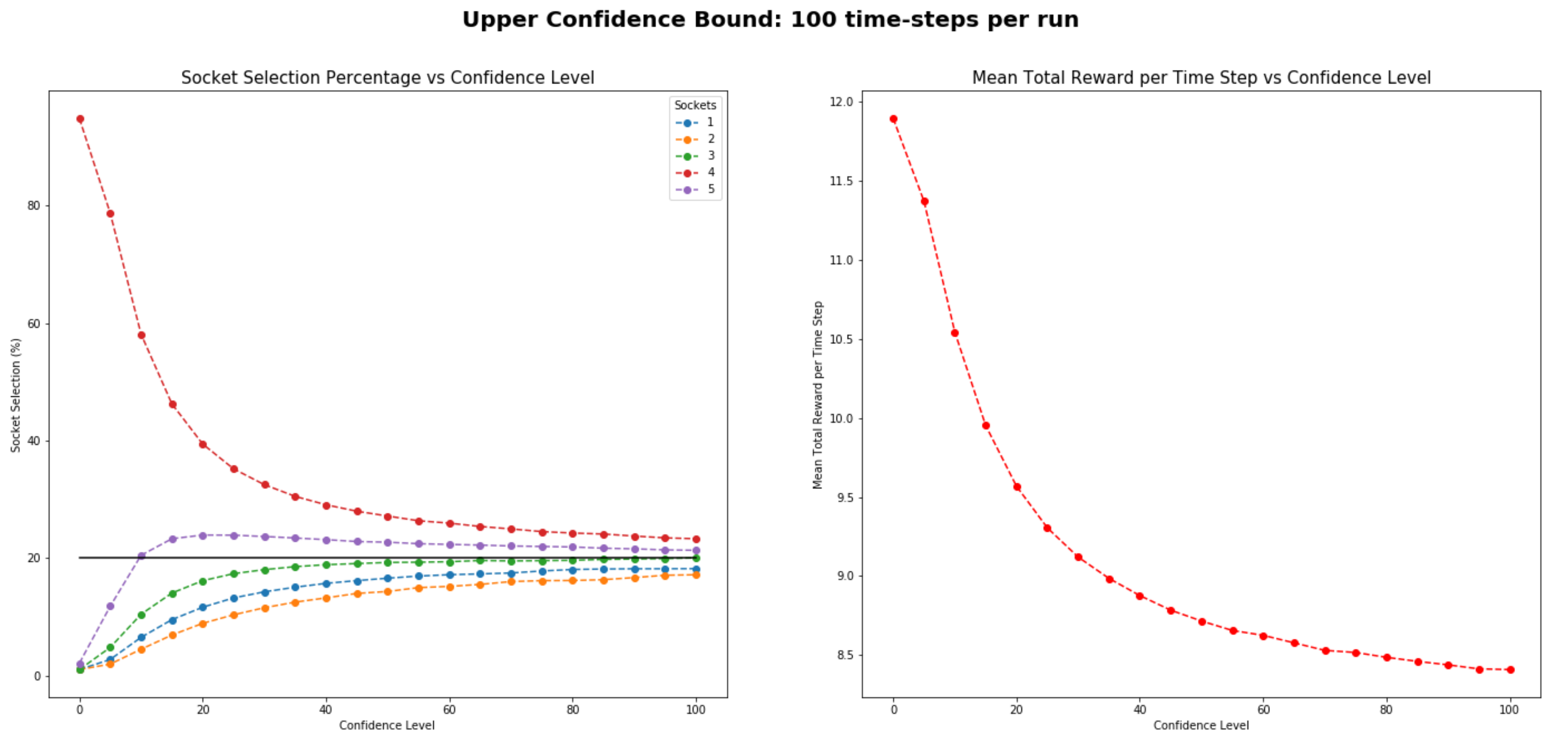
Upper Confidence Bound (UCB)

- The **solid part** of each bar represents the **exploration** part of the equation, therefore diminishing with increasing number of tests
- The **shaded part** of the bar represents the estimate of each arm's actual value, which converges to the true value



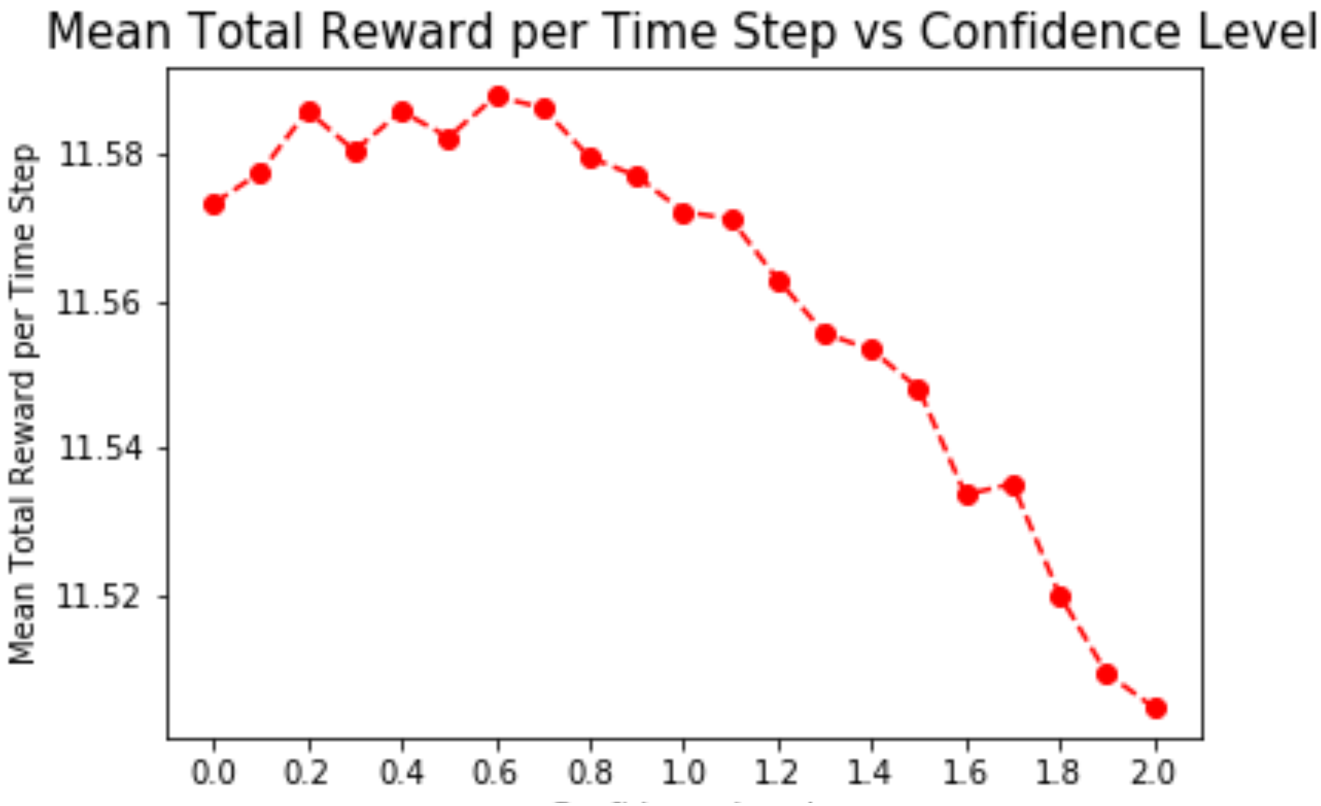
Upper Confidence Bound (UCB)

- Suppose 5 arms with mean rewards :
  - Arm 1 : **6** Arm 2 : **4** Arm : 3 : **8** Arm 4 : **12** Arm 5 : **10**



Here, the problem is simple and almost no exploration is needed

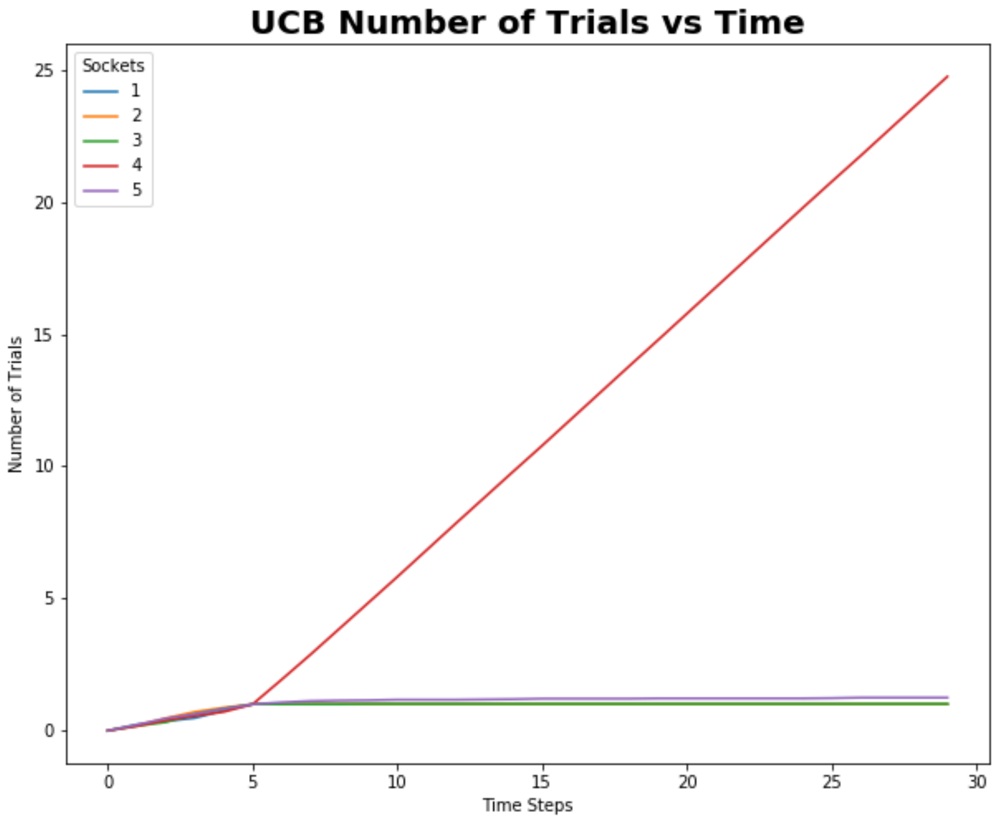
Upper Confidence Bound (UCB)



Looking more carefully, a **small value of exploration** (e.g. $C \sim 0.7$) yields the best result (here). The theory would favor $C = \sqrt{2}$

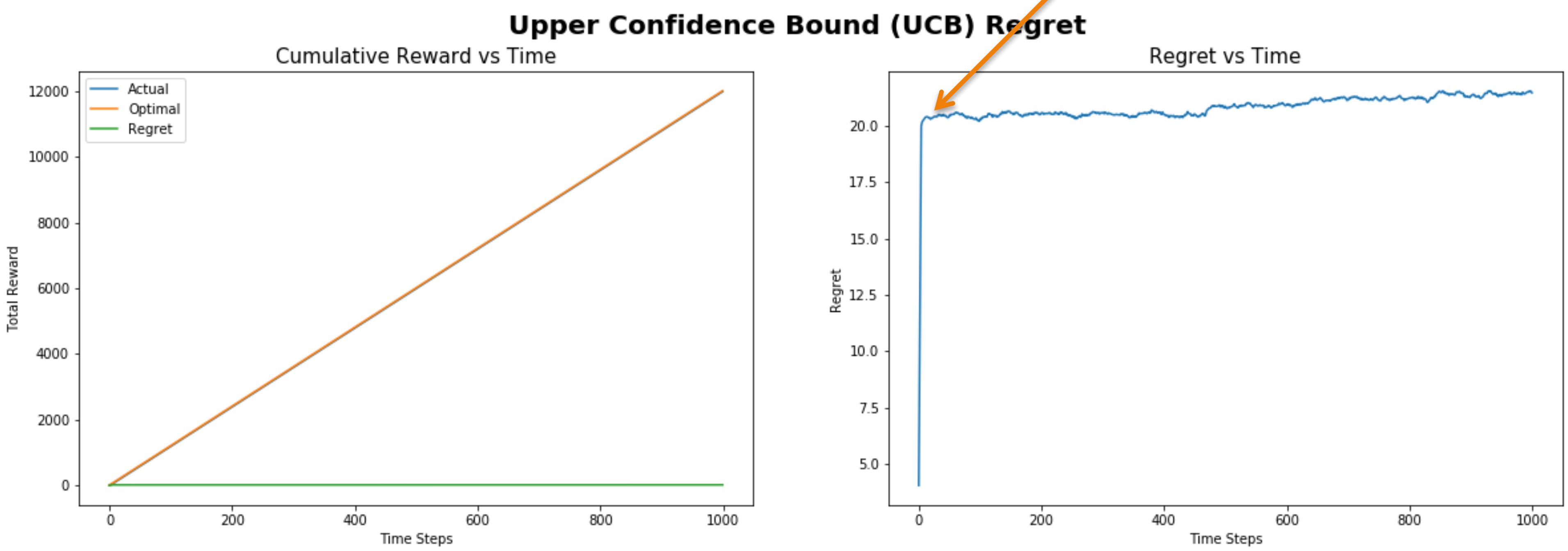
Upper Confidence Bound (UCB)

- Au début, les 5 bras sont explorés, puis ensuite c'est pratiquement toujours le bras 4 qui est tiré



Upper Confidence Bound (UCB)

- Presque tout le **regret** vient de l'exploration des 5 bras au début, puis il varie approximativement en log(n)



Theoretical properties of UCB1

The main question: rate of convergence to optimal arm

- Typical goal: achieve a regret of $\mathcal{O}(\log n)$ for n trials
- For many kinds of problems, it is **not possible to do better**
- **UCB1 is a simple algorithm** that achieves this asymptotic bound for many input distributions
- Huge amount of literature on different bandit problems and their properties

UCB vs. ε-greedy

ε-greedy	UCB
• Choix **aléatoire** parfois • Favorise les meilleurs coups apparents en **probabilité** • Se **généralise facilement** pour grands espace d'états et pour des environnements non stationnaires (apprentissage par renforcement)	• Choix **déterministe** • Favorise les meilleurs coups en apparence par la **formule** • Ne se **généralise** pas facilement - environnement **non stationnaire** - **très grand** espace d'états

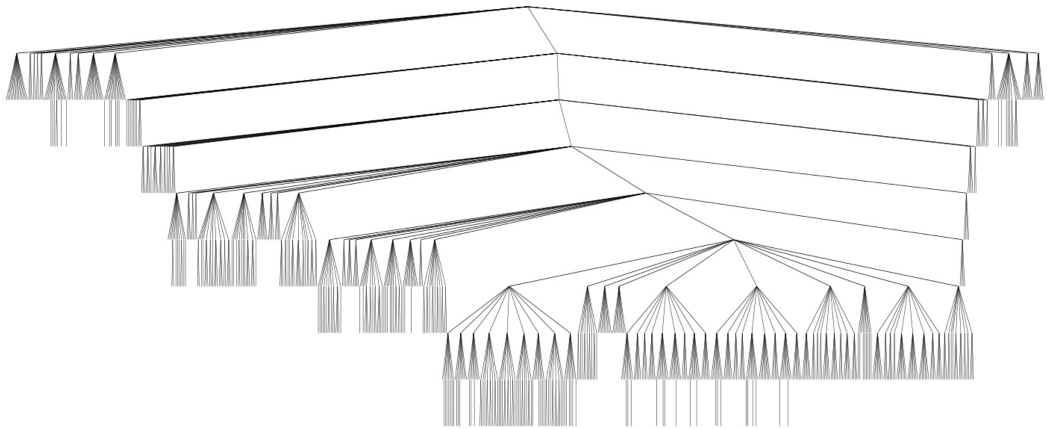
…

Monte Carlo Tree Search

- Au lieu de se concentrer sur **un seul « bandit »**

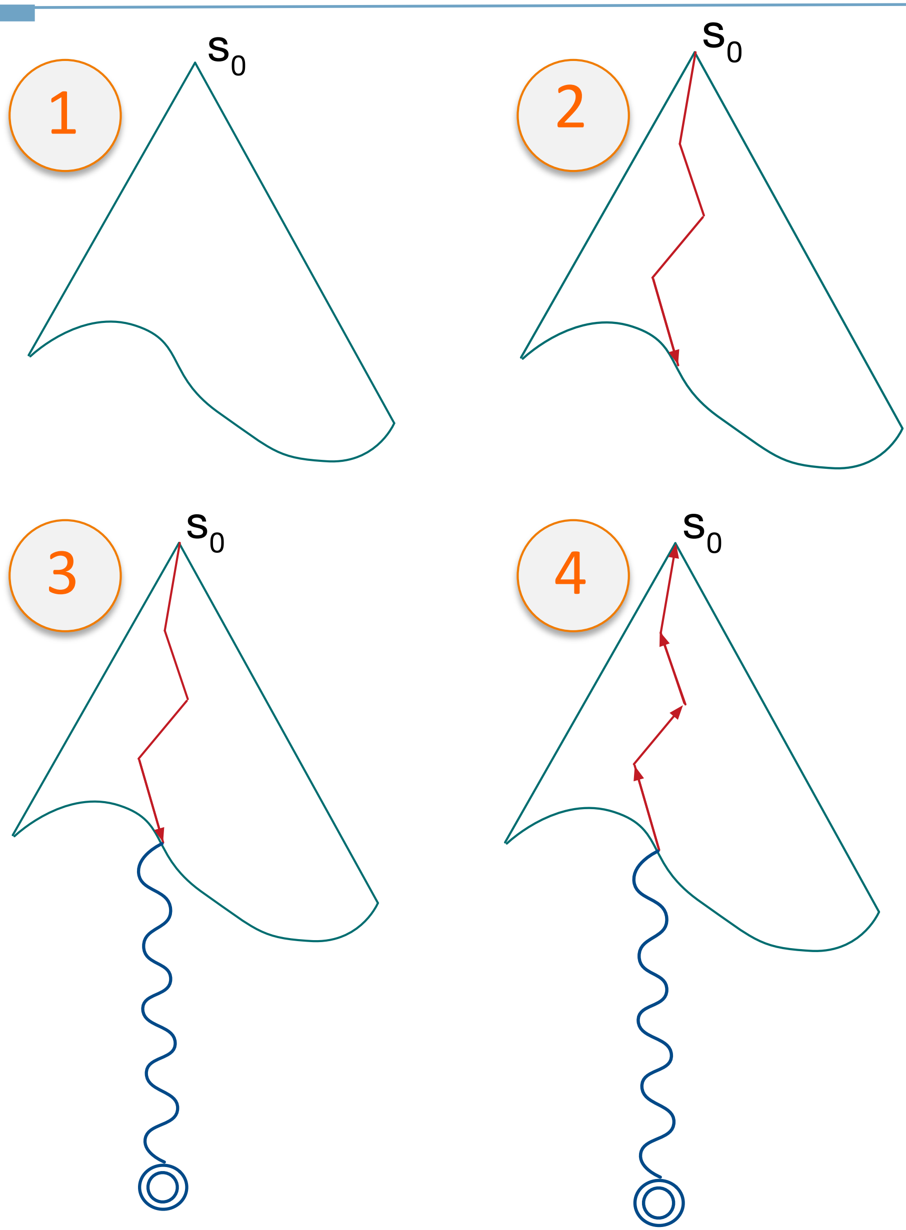
- MCTS réalise **l'exploration d'arbres MinMax**
  - Chaque **successeur** est considéré comme un bandit
  - Une **séquence de bandits** est explorée à chaque épisode

- Contrairement à l'algorithme MinMax
  - MCTS réalise une exploration à **profondeur** et **largeur** variables

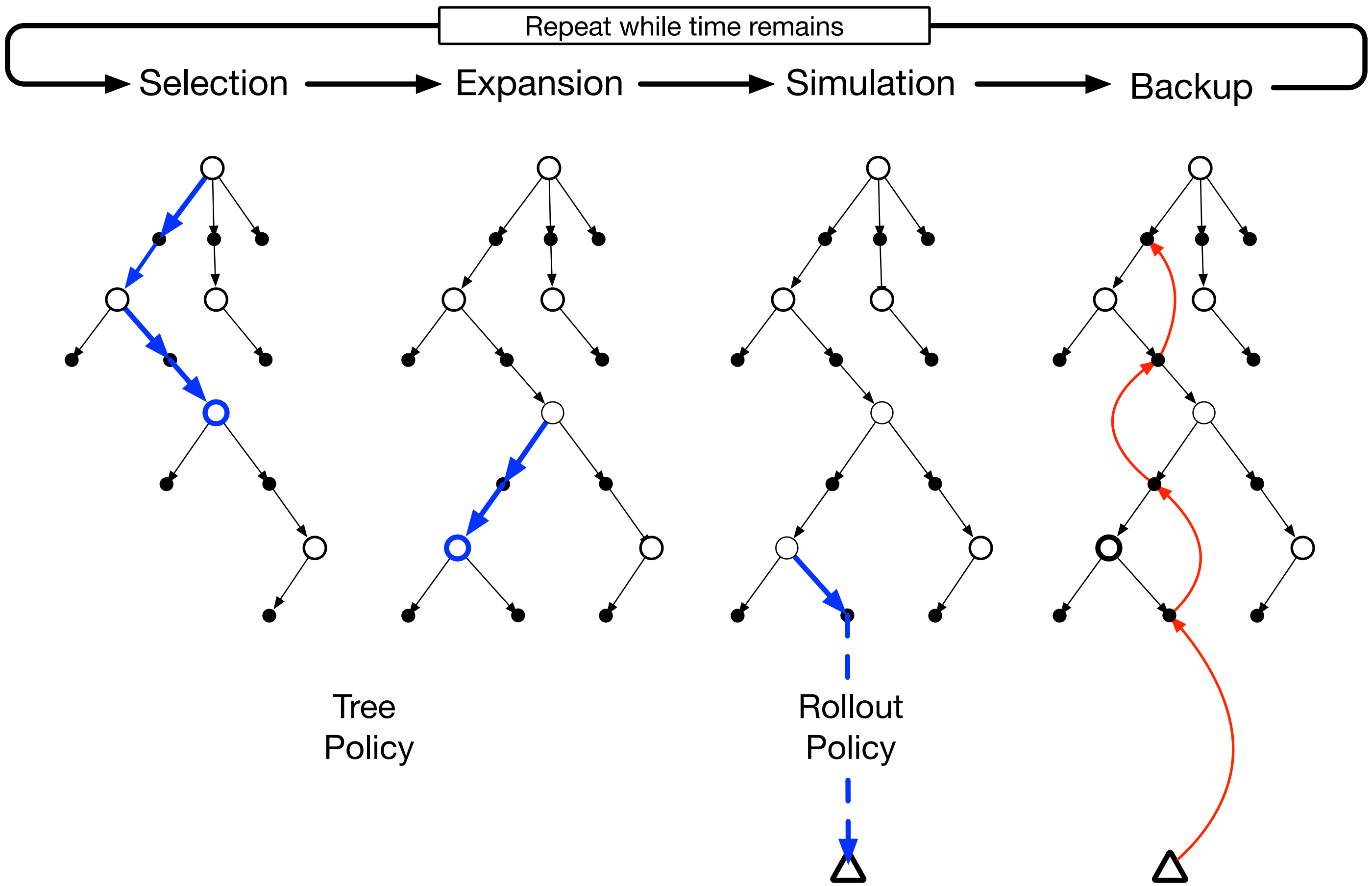


MCTS

- A generic way of exploring a game tree
- Four steps
  1. Selection
     - Of a leaf to expand
  2. Expansion
  3. Simulation (roll out)
  4. Back-propagation
     - **Updating** the value of each ancestor node of the expanded leaf



Upper Confidence Tree = MCTS + UCB



From Sutton & Barto (2018). « **Reinforcement learning. An introduction** » (2nd edition)

Upper Confidence Tree = MCTS + UCB

- How to perform the **selection**?

- Answer: use **UCB**

- Upper Confidence Tree:

  use a look-ahead tree with **selection guided by** UCB and

  exploration/**evaluation** performed by Monte-Carlo sampling

Upper Confidence Tree = MCTS + UCB

1. Tree traversal
2. Node expansion
3. Rollout (random simulations)
4. Backpropagation

Plan

Upper Confidence Tree = MCTS + UCB

1. Selection NOT using UCB (here)

Max

Max **wins** 12 out of 21 plays

Min

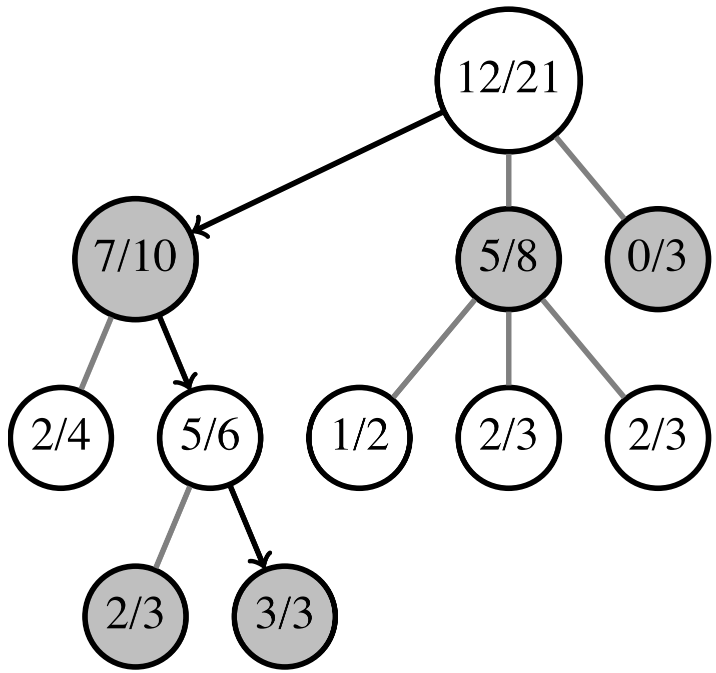
Max **wins** 7 out of 10 plays, ...

Max

Max **wins** 5 out of 6 plays

Min

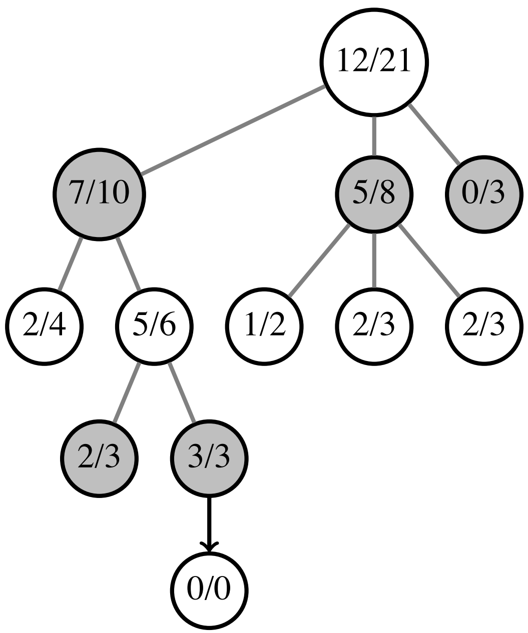
Max **wins** 3 out of 3 plays, ...

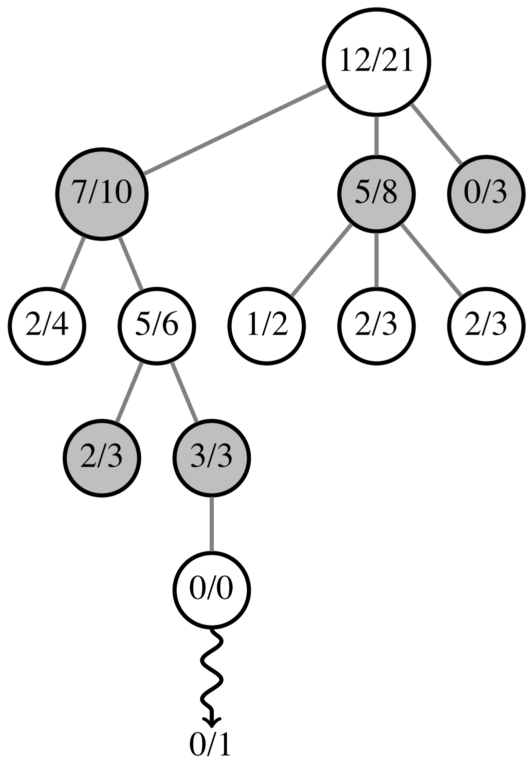


From [Wikipedia « Monte-Carlo Tree Search ».]

Upper Confidence Tree = MCTS + UCB

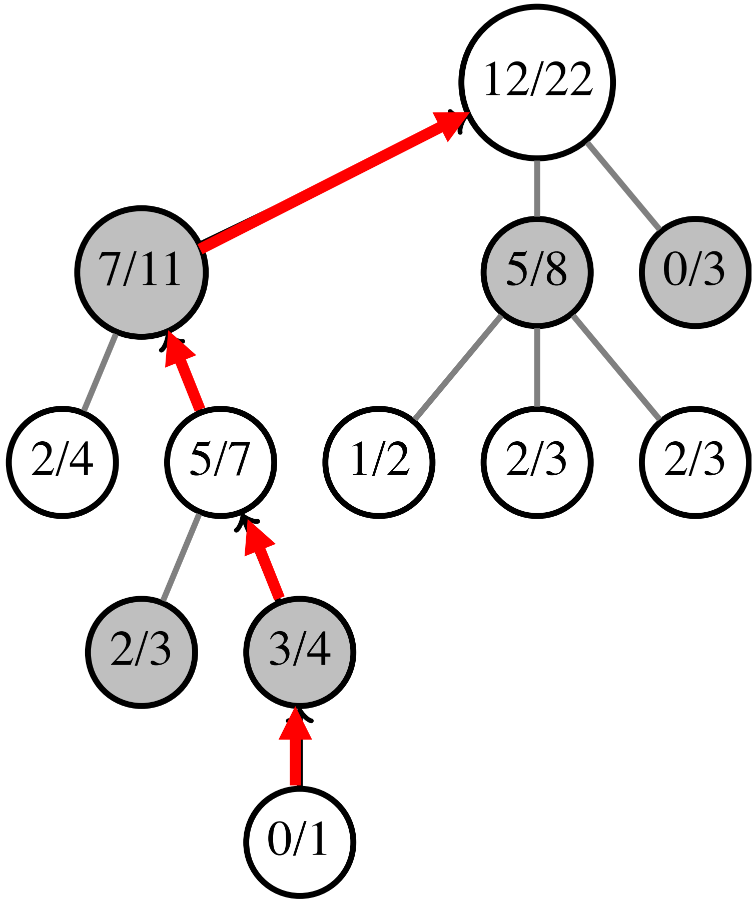
2. Node **expansion** / rollout





Upper Confidence Tree = MCTS + UCB

4. Backpropagation



Upper Confidence Tree = MCTS + UCB

- Note that the **node selection could have been different** according to the value of *C* in:

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

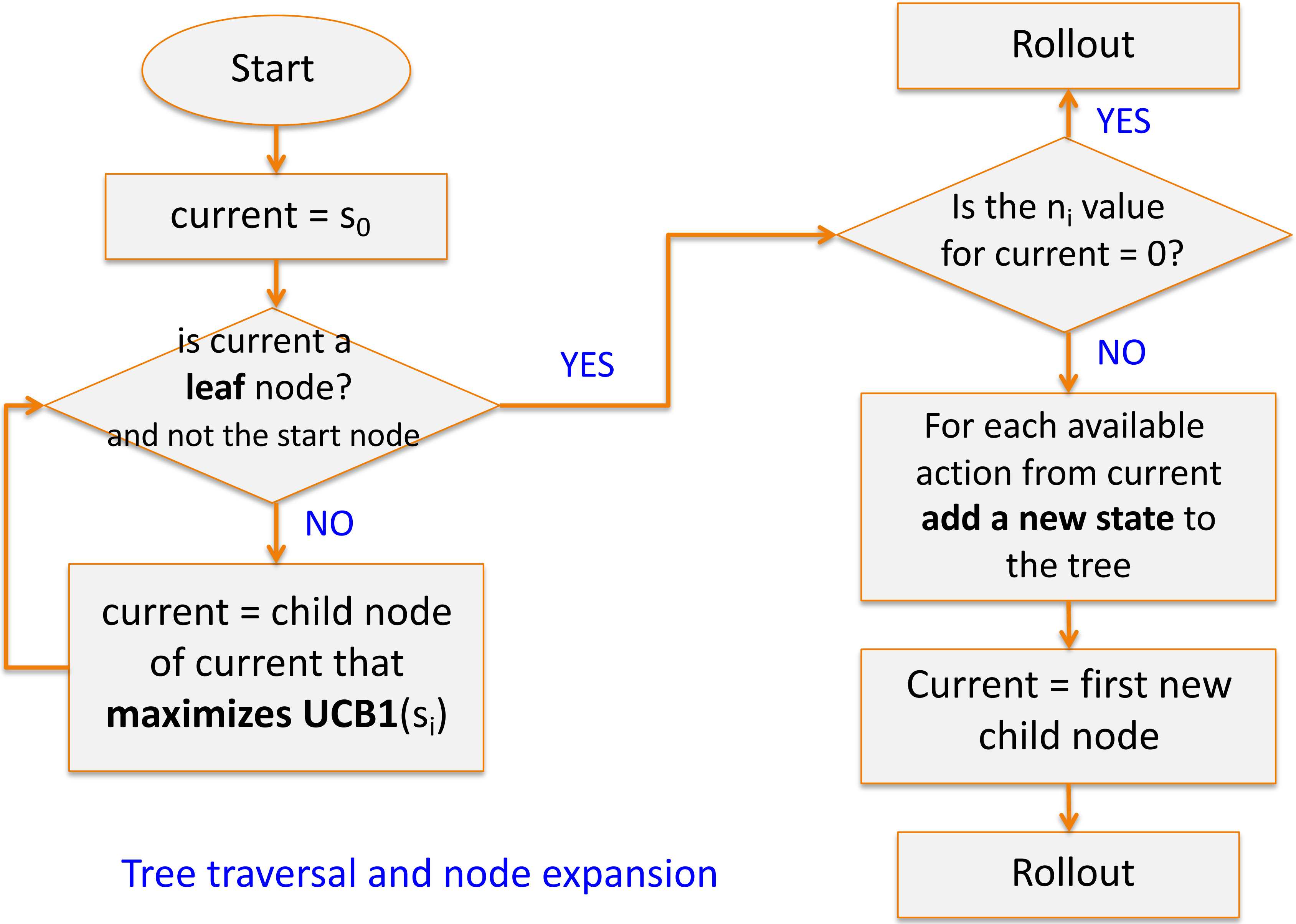
value estimate

tunable parameter

total number of trials

num trials for arm i

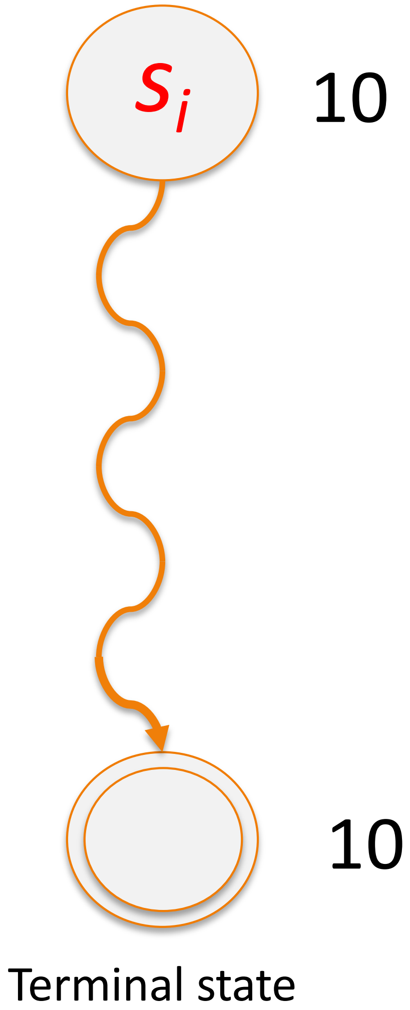
UCT : the **algorithm**



Tree traversal and node expansion

Rollout(s_i)

- **Loop until** s_i is a **terminal state**
  1. a_i = random(available_actions(s_i))
  2. s_i := simulate(a_i, s_i)
- Return value(s_i)



Here, an example where the return is not win or loose, but a number

Worked out example:

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i

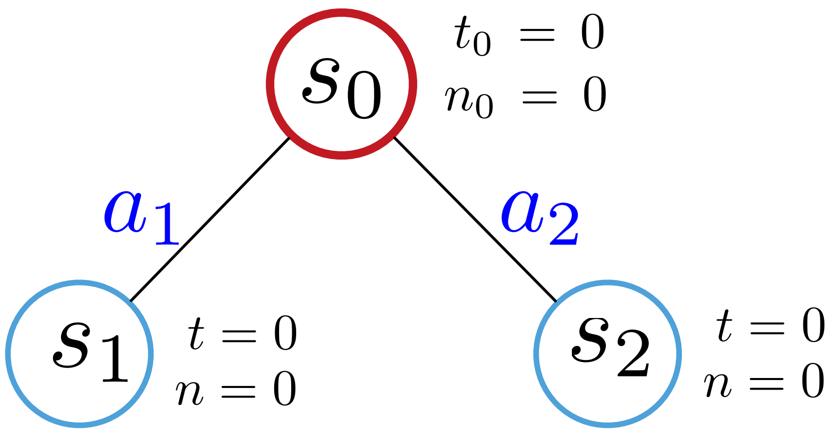
$$s_0 \quad \begin{aligned} t_0 &= 0 \\ n_0 &= 0 \end{aligned}$$

Worked out example: 1st iteration

$$UCB1(s_i) \;=\; \overline{v_i} \;+\; 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



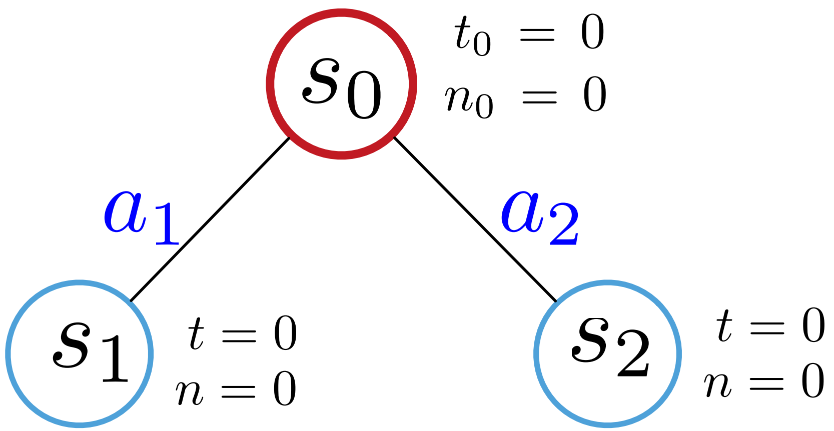
Choice of action

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



Choice of an action

$UCB1(s_1) = \infty$ $\quad UCB1(s_2) = \infty$

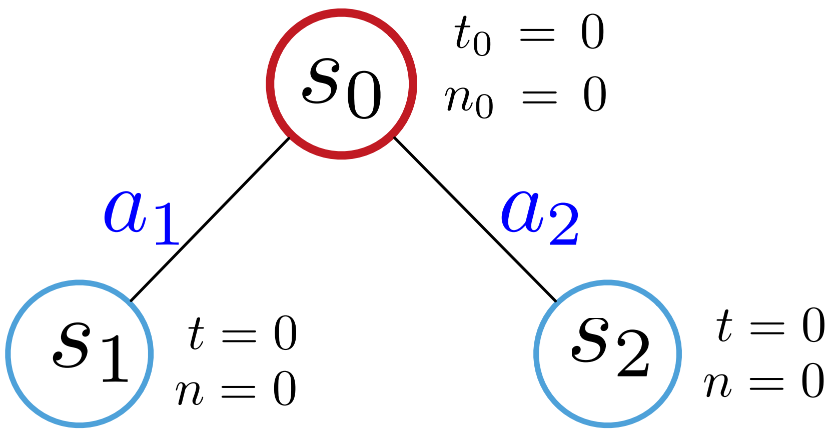
Choice of action a_1

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



Choice of action

$$UCB1(s_1) = \infty \qquad UCB1(s_2) = \infty$$

Choice of action a_1 not visited yet

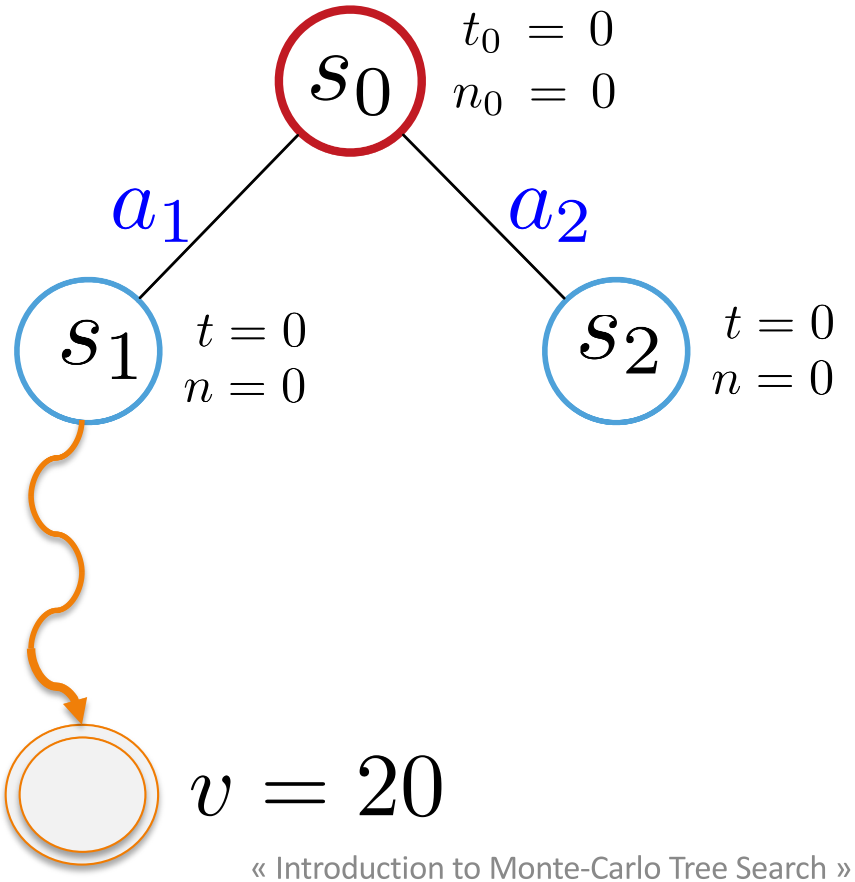
Rollout

Worked out example: 1st iteration

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



Choice of action

Rollout

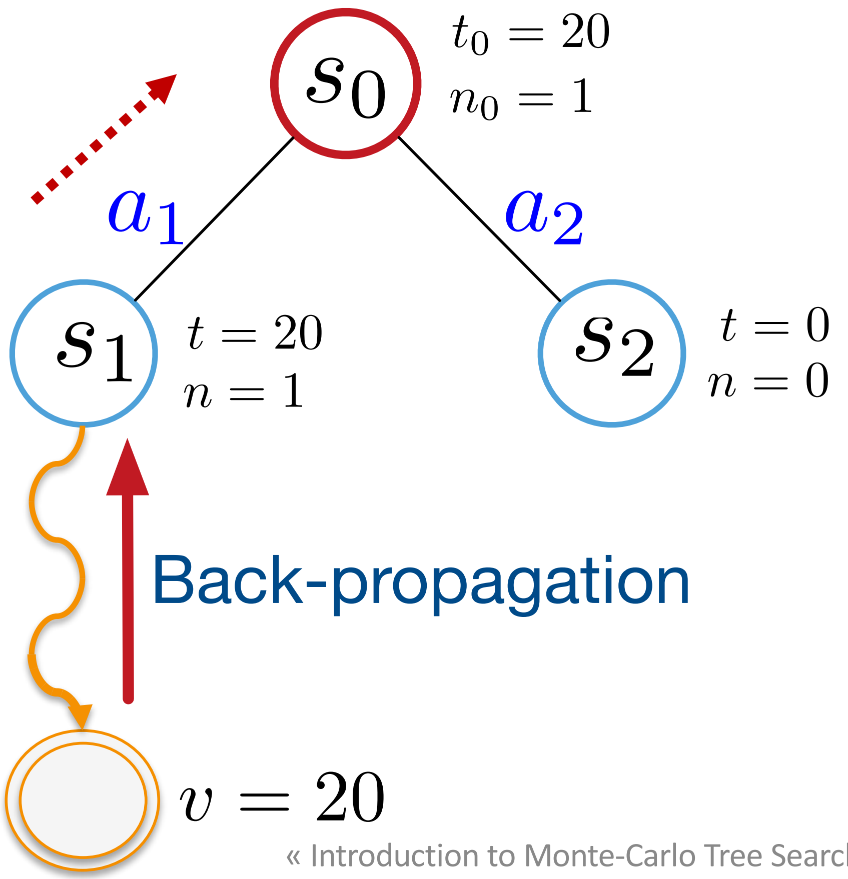
Worked out example: 1st iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



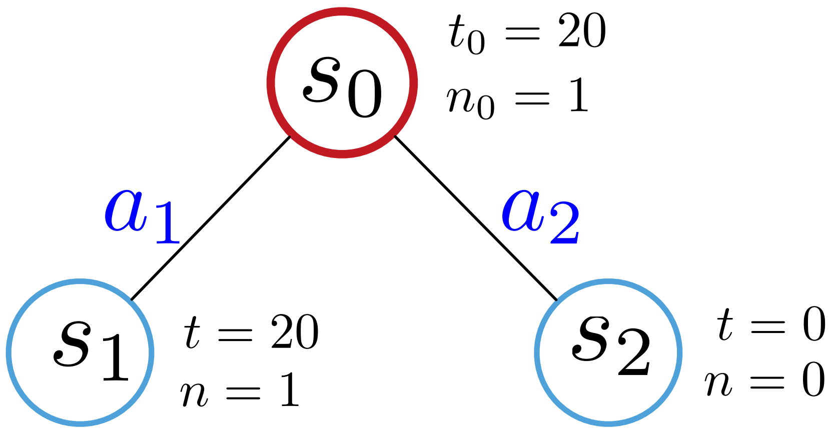
Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



Choice of an action

$$UCB1(s_1) = 20 + 2\sqrt{\frac{\ln 1}{1}} = 20 \qquad UCB1(s_2) = \infty$$

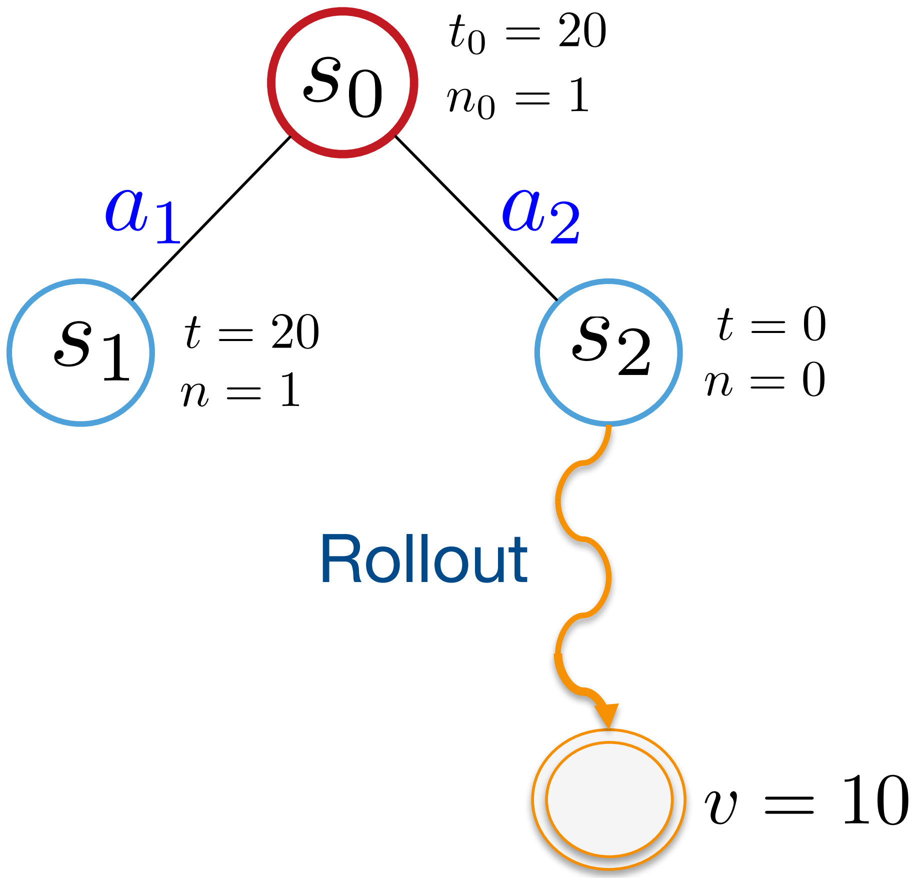
Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

Number of trials of the parent node

Number of trials of the node n_i



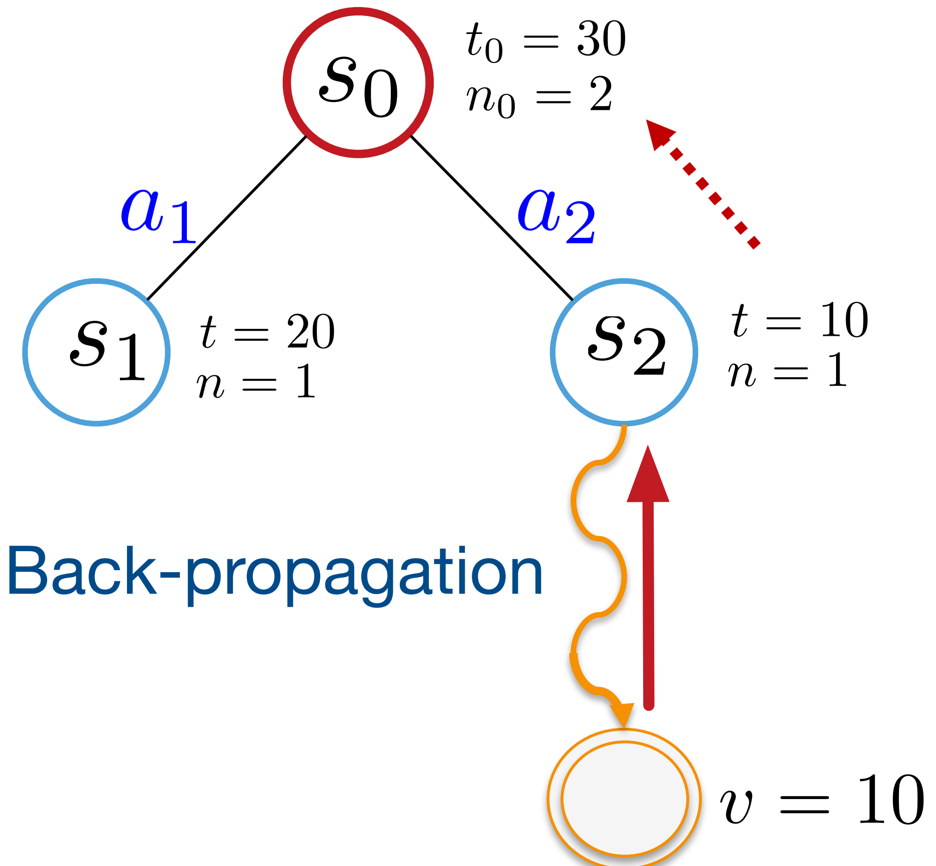
Worked out example: 2nd iteration

Example

$$UCB1(s_i) = \overline{v_i} + 2\sqrt{\frac{\ln N}{n_i}}$$

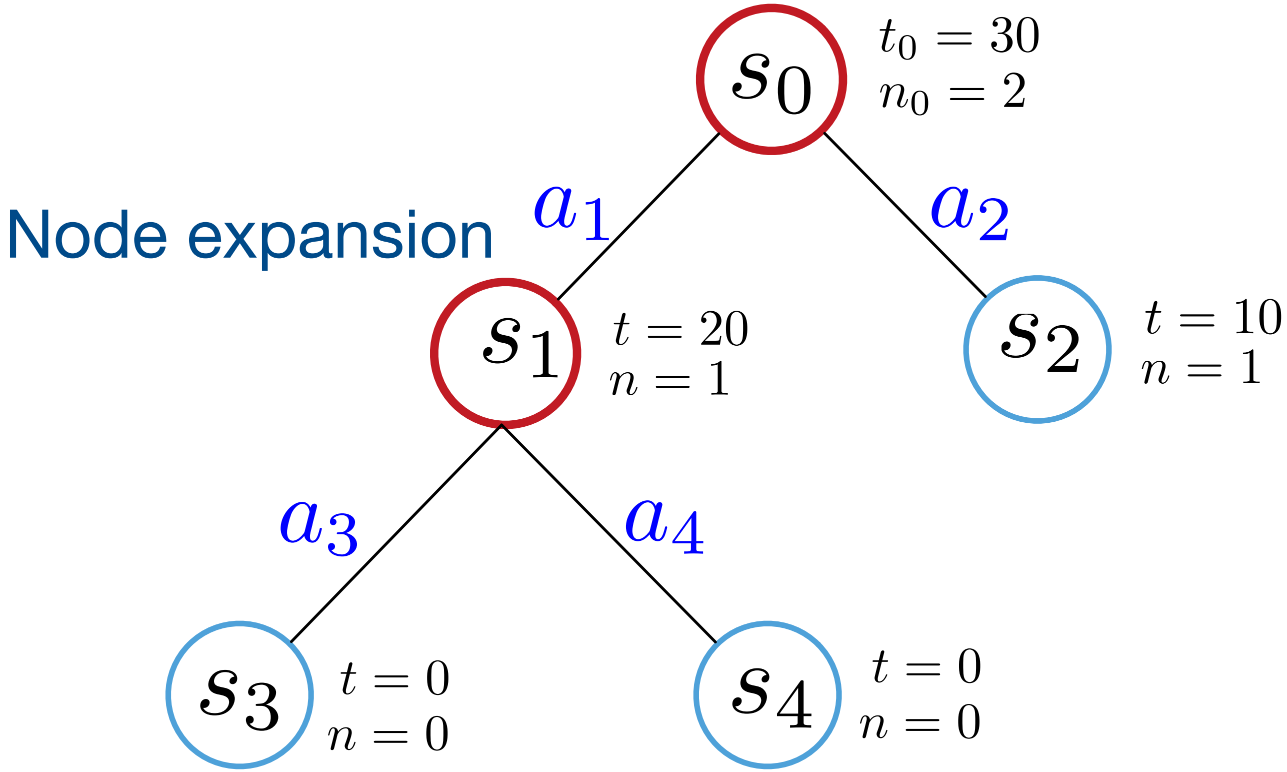
Number of trials of the parent node

Number of trials of the node n_i



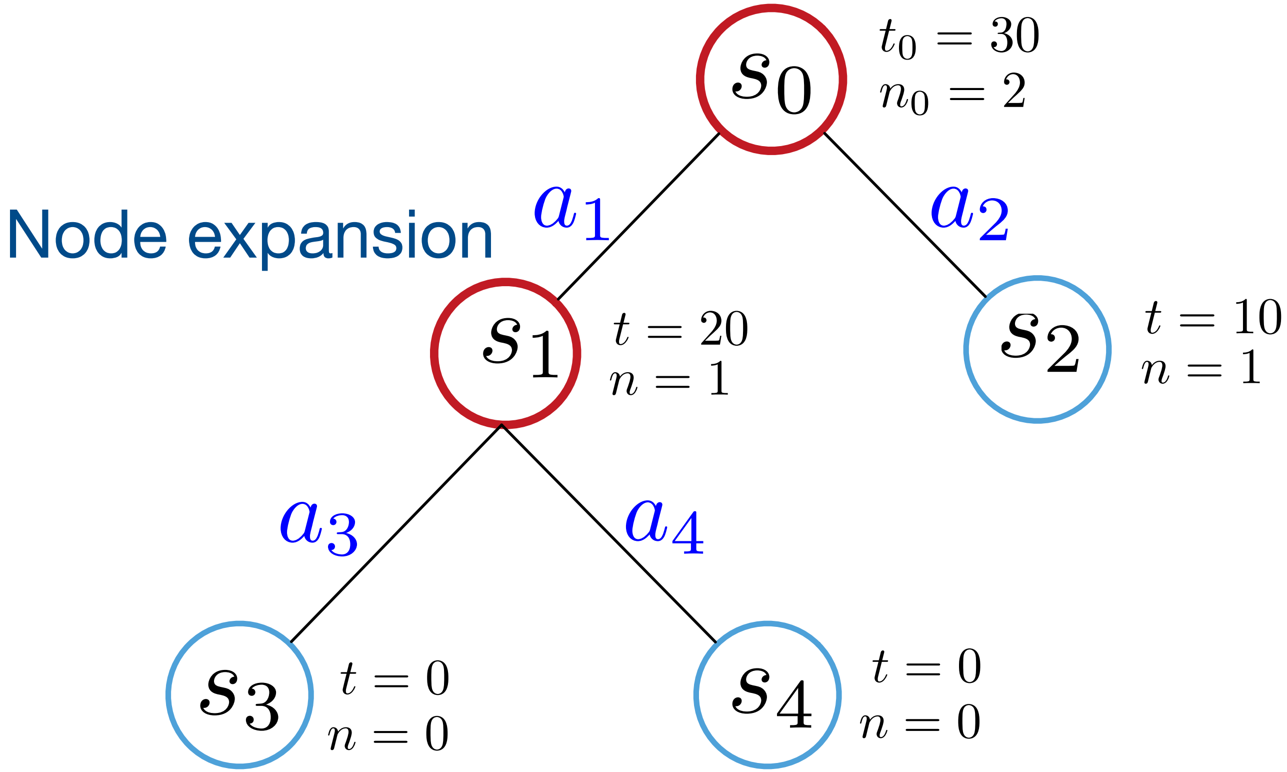
Worked out example: 3rd iteration

Example



Worked out example: **3rd iteration**

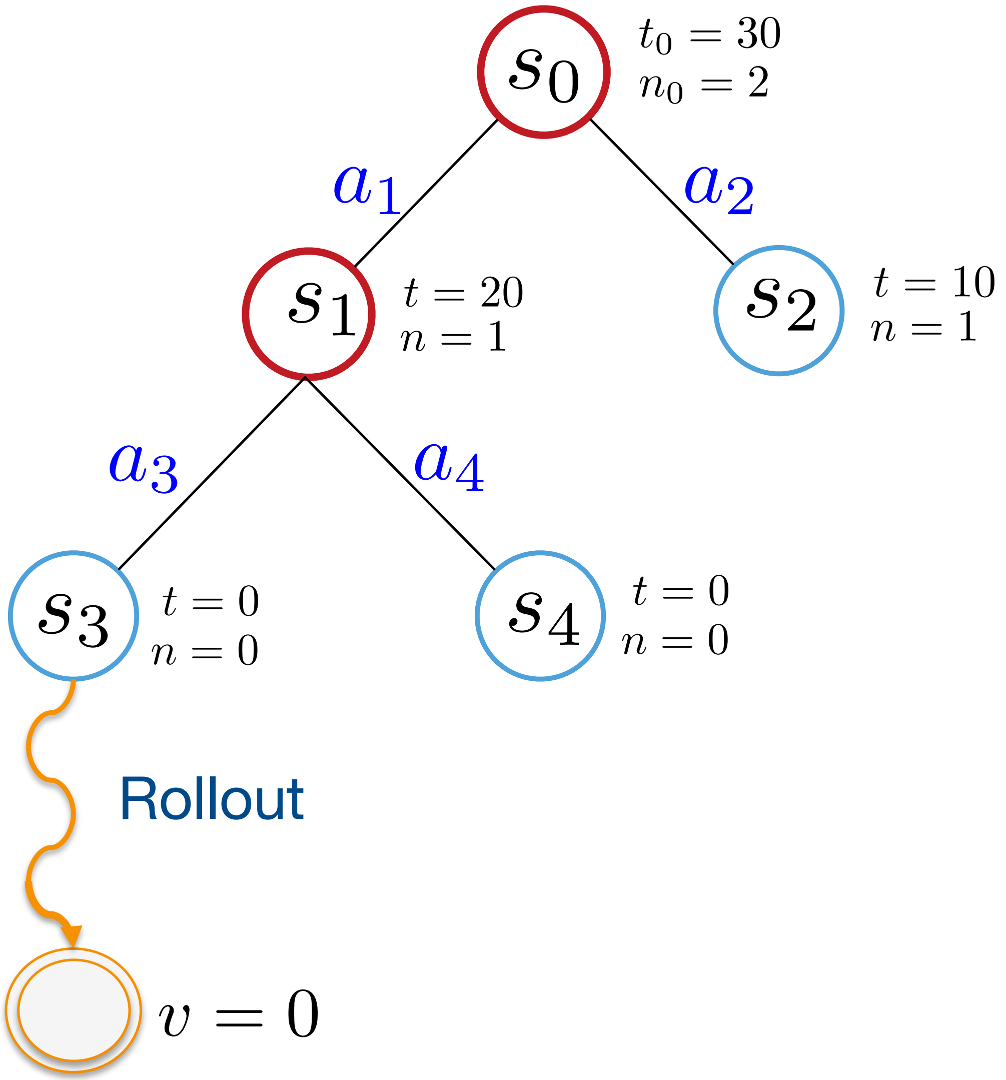
Example



$$UCB1(s_3) = \infty \qquad UCB1(s_4) = \infty$$

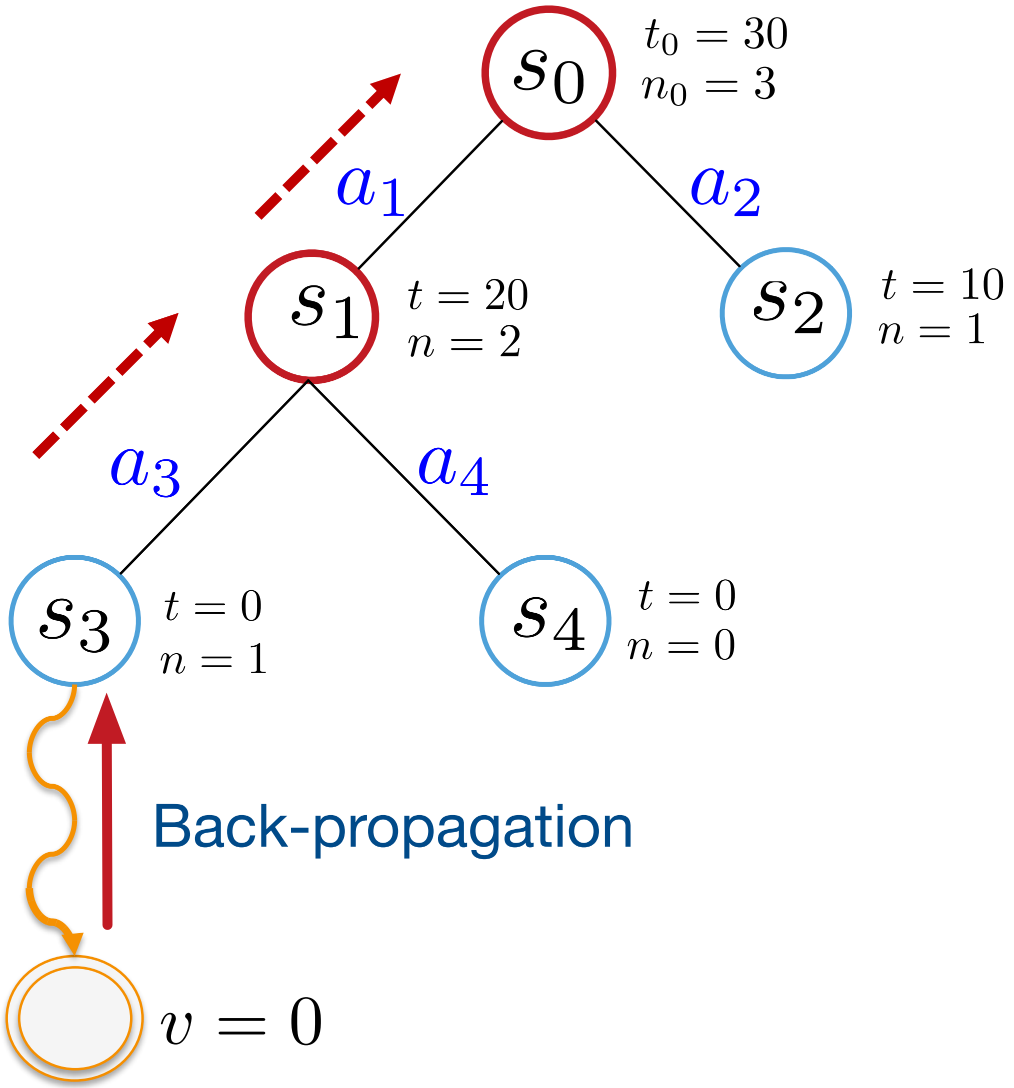
Worked out example: 3rd iteration

Example

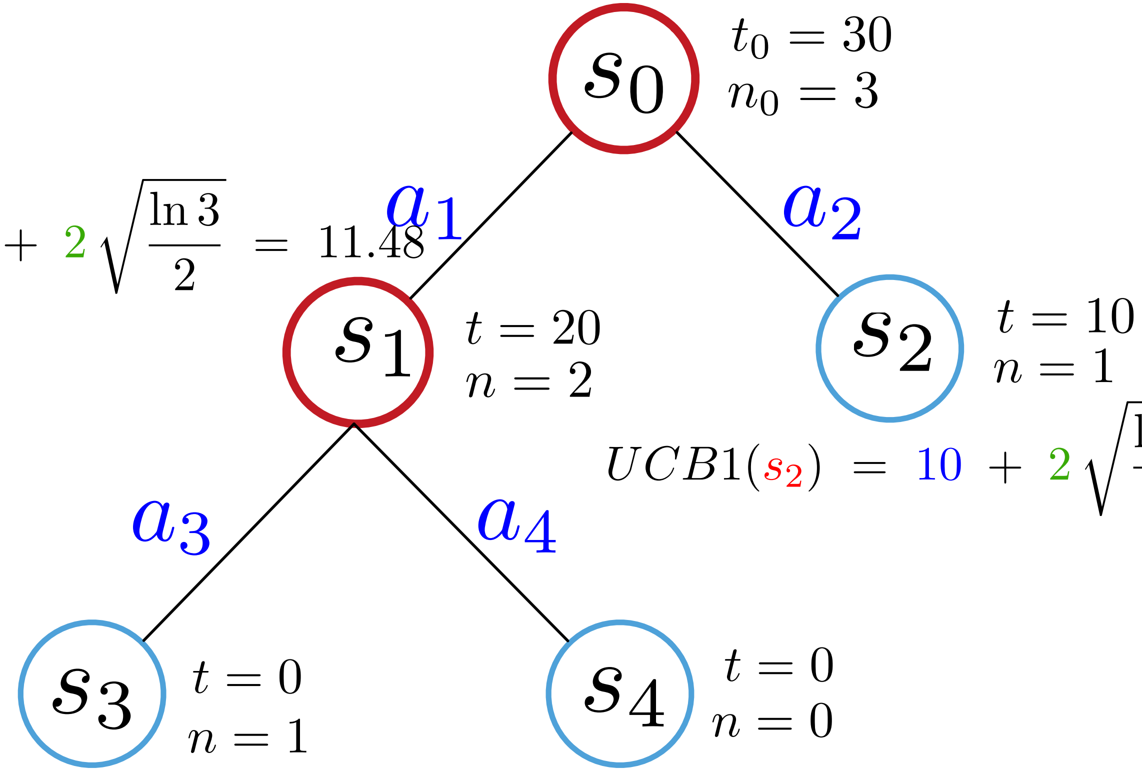


Worked out example: 3rd iteration

Example



Example

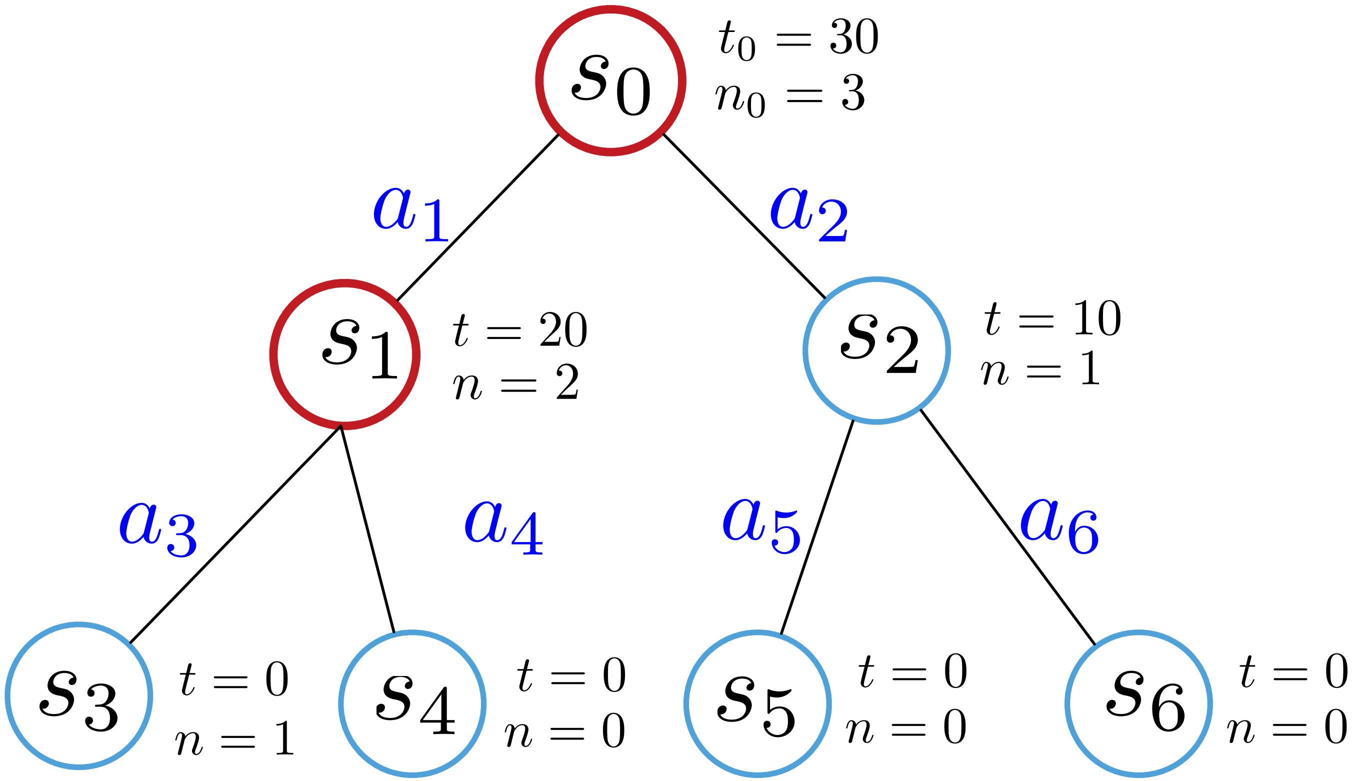


$$UCB1(s_1) \;=\; 10 \;+\; 2\sqrt{\frac{\ln 3}{2}} \;=\; 11.48$$

$$UCB1(s_2) \;=\; 10 \;+\; 2\sqrt{\frac{\ln 3}{1}} \;=\; 12.10$$

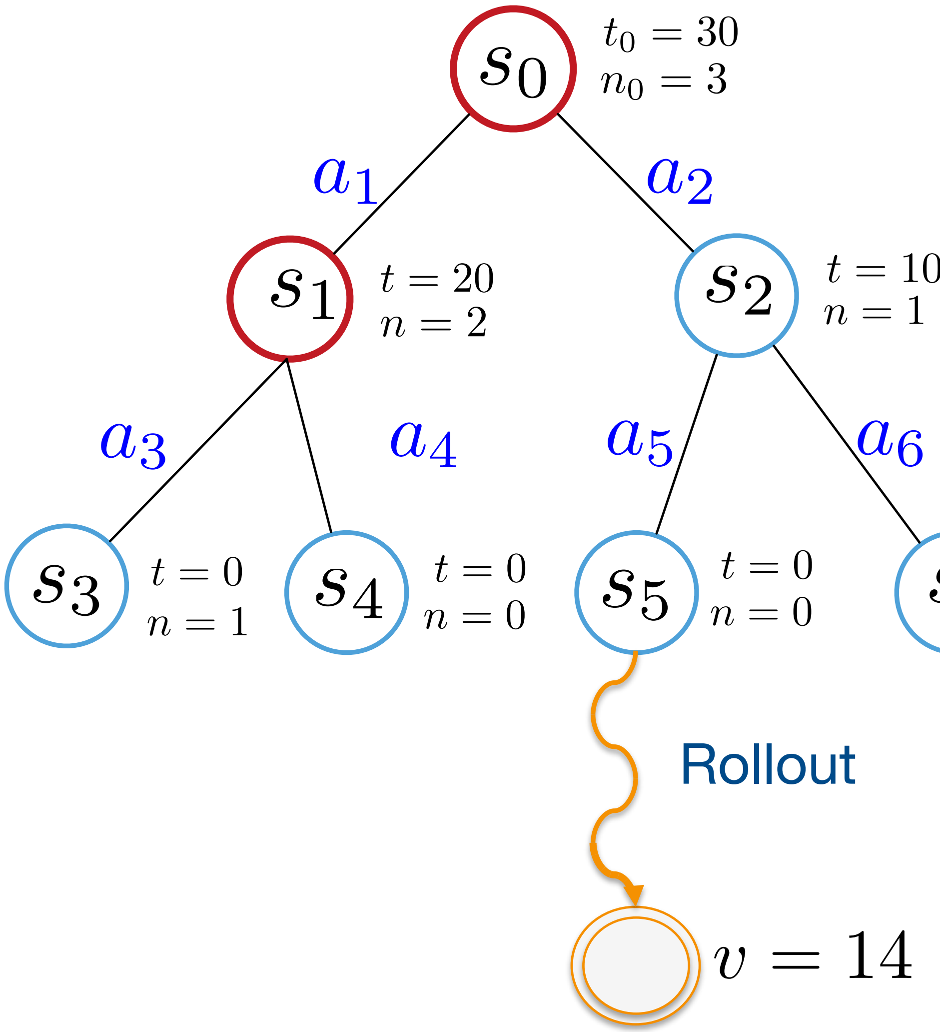
Worked out example: **4th iteration**

Example

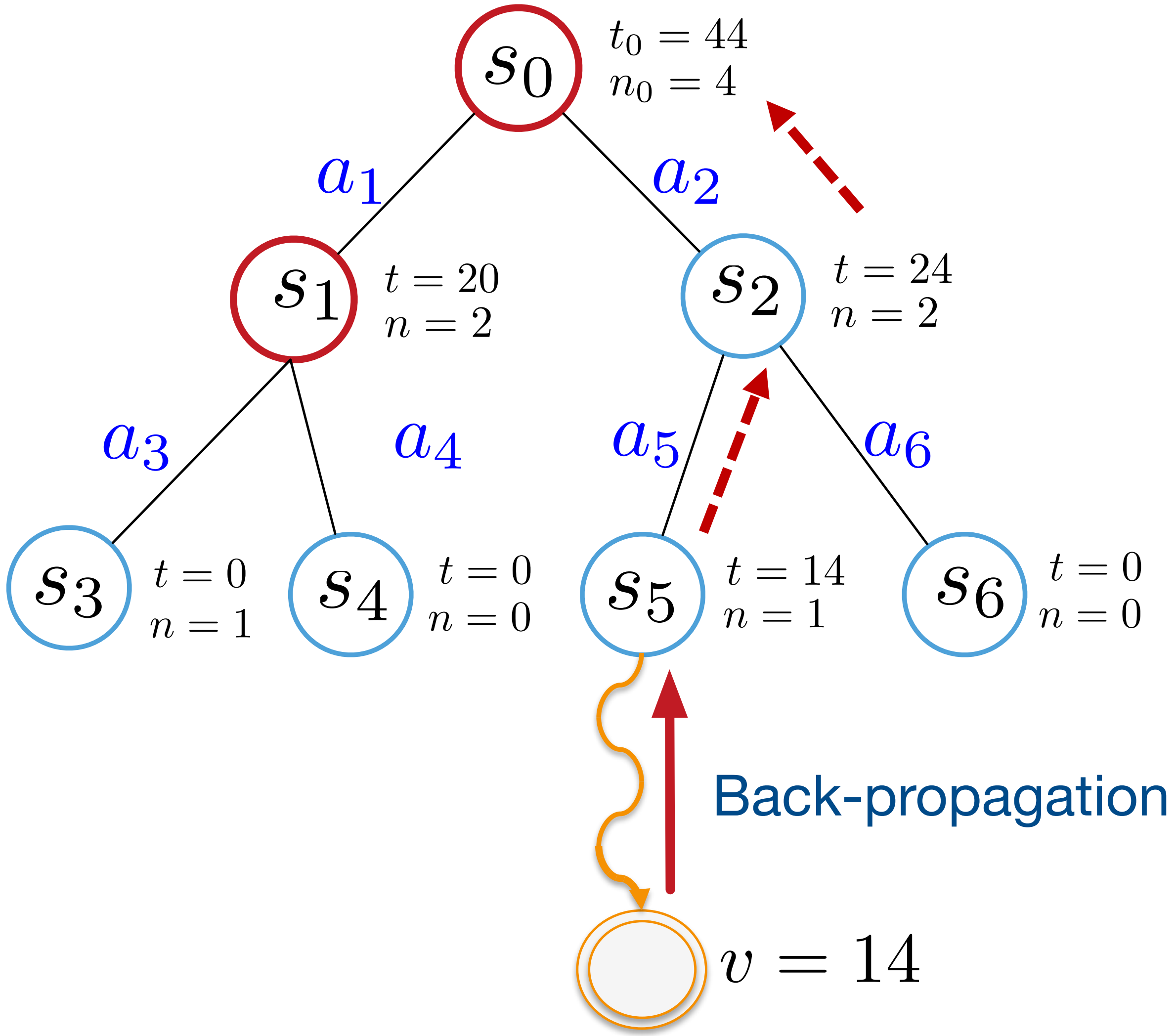


Worked out example: 4th iteration

Example



Worked out example: 4th iteration



What would be the best action to take from s_0?

Conclusions sur UCT

On continue le processus jusqu'à épuisement des ressources calcul allouées

+ Preuve de **convergence** vers MinMax,
  mais lent dans la version de base montrée ici
+ Pas besoin de **fonction d'évaluation**
+ Très bon quand le **facteur de branchement** est **important**.
  Contrôle bien le compromis exploration vs. Exploitation
+ Algorithme « **anytime** »

Conclusions (2)

- UCT (= MCTS + UCB) is powerful in order to **chose among alternatives**
  - By exploring "intelligently" the tree of possible consequences of potential decisions

- Works when it is possible to explore possible scenarios **by simulation**

Requires a **good model of the world**

Conclusions (3)

- Compétitions “**General Game Playing**” : toutes gagnées par des algorithmes utilisant MCTS depuis 2007
- **AlphaGo**, **AlphaGo Zero** et **Alpha Zero** utilisent une variante de MCTS (PUCT)
- Peut-être combiné avec de l’**apprentissage par renforcement profond** (Deep RL)

Predicting the structure of large protein complexes using AlphaFold and Monte Carlo tree search

Patrick Bryant[1,2]*, Gabriele Pozzati[1,2], Wensi Zhu[1,2], Aditi Shenoy[1,2], Petras Kundrotas[1,3] and Arne Elofsson[1,2]

[1]Science for Life Laboratory, 172 21 Solna, Sweden
[2]Department of Biochemistry and Biophysics, Stockholm University, 106 91 Stockholm, Sweden
[3]Center for Computational Biology, The University of Kansas, Lawrence, KS 66047, USA

*Corresponding author, email: patrick.bryant@scilifelab.se

Abstract

AlphaFold can predict the structure of single- and multiple-chain proteins with very high accuracy. However, the accuracy decreases with the number of chains, and the available GPU memory limits the size of protein complexes which can be predicted. Here we show that one can predict the structure of large complexes starting from predictions of subcomponents. We assemble 91 out of 175 complexes with 10-30 chains from predicted subcomponents using Monte Carlo tree search, with a median TM-score of 0.51. There are 30 highly accurate complexes (TM-score ≥0.8, 33% of complete assemblies). We create a scoring function, mpDockQ, that can distinguish if assemblies are complete and predict their accuracy. We find that complexes containing symmetry are accurately assembled, while asymmetrical complexes remain challenging. The method is freely available and accesible as a Colab notebook https://colab.research.google.com/github/patrickbryant1/MoLPC/blob/master/MoLPC.ipynb.

Keywords
Protein structure prediction
AlphaFold
Complex assembly
Monte Carlo tree search

Energy Conversion and Management 252 (2022) 115047

Contents lists available at ScienceDirect

Energy Conversion and Management

journal homepage: www.elsevier.com/locate/enconman

Wind farm layout optimization using adaptive evolutionary algorithm with Monte Carlo Tree Search reinforcement learning

Fangyun Bai [a], Xinglong Ju [b], Shouyi Wang [c], Wenyong Zhou [a], Feng Liu [d,*]

[a] Department of Management Science and Engineering, Tongji University, Shanghai, China
[b] Price College of Business, University of Oklahoma, Norman, OK, 73019, USA
[c] Department of Industrial, Manufacturing, & Systems Engineering, The University of Texas at Arlington, Arlington, TX 76019, USA
[d] School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030, USA

A R T I C L E I N F O

Keywords:
Evolutionary computations
Adaptive genetic algorithm
Reinforcement learning
Monte-Carlo Tree Search
Wind farm layout optimization

A B S T R A C T

Recent years have witnessed an enormous growth of wind farm capacity worldwide. Due to the wake effect, the velocity of incoming wind is reduced for the wind turbines in the downwind directions, thus causing discounted power generation in a wind farm. Previously, a self-informed adaptivity mechanism in evolutionary algorithms was introduced by the authors, which is inspired by the individuals' self-adaptive capability to fit the environment in the natural world, where relocating the worst wind turbine with a surrogate model informed mechanism was found to be effective in improving the power conversion efficiency. In this paper, the exploitation capability in the adaptive genetic algorithm is further improved by casting the relocation of multiple wind turbines into a single-player reinforcement learning problem, which is further addressed by Monte-Carlo Tree Search embedded within the evolutionary algorithm. In contrast to the moderate improvements of the authors' previous algorithms, significant improvement is achieved due to the enhanced algorithmic exploitation. The new algorithm is also applied to solve the optimal layout problem for a recently approved wind farm in New Jersey, and showed better performance against the benchmark algorithms.

1. Introduction

Climate change and global warming have been a major concern for sustainable social and economic development around the world. It is estimated that the portion of renewable energy should be at least 67% among all resources of energies in 2050 compared to 20% in 2018 [1], in order to meet the target of limiting the global temperature within 1.5 °C above the preindustrial level according to Intergovernmental Panel on Climate Change (IPCC) [2] on Climate Change [2]. Wind energy has become an indispensable alternative to fossil fuels given its advantage of being sustainable, economically competitive, and abundant [3], which has shown steady growth of capacity and power generation over the past decades. In 2020 alone, the US has grown the capacity of wind energy by 23 Gigawatts (GW), the largest in history. Optimal design of wind farms has been thoroughly investigated from different perspectives, such as cite selection [4], wind turbine design [5], electrical cable placement [6], wake effect modeling [7], wind speed forecasting [8], and wind power prediction [9].

One challenge for maximizing the power output is to find an optimal layout of the wind turbines to reduce the wake effect [10]. Wake effect refers to the situation when the input wind speed for the wind turbines in the downwind directions are discounted after the wind turbines in the upwind directions absorb the kinetic energy from the wind [11]. In addition to the energy output decrease caused by the wake effect, the wake effect can also cause fatigue loads due to the increased turbulence of wind flow, which can cause mechanical failure and shorten the life expectancy of wind turbines [12]. Every percentage of improvement in efficiency can mean significant profit income, thus requires a meticulous effort of investigation. The wind farm layout optimization problem (WFLOP) is a highly complicated problem as even 30 wind turbines could lead to a high 10^{44} potential solutions given discrete and uniform turbine types [13] and suffer from "curse of dimensionality" for increase numbers of wind turbines [14]. With the recent trend of constructing wind farms with larger capacities, the WFLOP is even more challenging to solve. The nonconvex and NP-hard nature in WFLOP poses challenges for exact solution methods such as linear programming, mixed integer programming. However, there are some attempts using mixed integer programming [15,16]. Many nature-inspired, population-based meta-heuristic algorithms have been proposed to solve the WFLOP, such as

* Corresponding author.
E-mail addresses: 1510353@tongji.edu.cn (F. Bai), xinglong.ju@ou.edu (X. Ju), shouyiw@uta.edu (S. Wang), liu22@stevens.edu (F. Liu).

https://doi.org/10.1016/j.enconman.2021.115047
Received 19 April 2021; Received in revised form 11 October 2021; Accepted 19 November 2021
Available online 2 December 2021

Available online at www.sciencedirect.com
ScienceDirect
Advances in Space Research 70 (2022) 792–811
ADVANCES IN SPACE RESEARCH (a COSPAR publication)
www.elsevier.com/locate/asr

Sensor tasking in the cislunar regime using Monte Carlo Tree Search

Samuel Fedeler [a,*], Marcus Holzinger [a], William Whitacre [b]

[a] University of Colorado at Boulder, Boulder, CO, USA
[b] The Charles Stark Draper Laboratory, Inc., Cambridge, MA, USA

Received 14 October 2021; received in revised form 25 March 2022; accepted 2 May 2022
Available online 10 May 2022

Abstract

Maintaining tracks on space objects with limited sets of observers is a critical problem, made more urgent with exponential growth in the population of near-Earth satellites. An optimally convergent decision making methodology is proposed for sensor tasking, using the Monte Carlo Tree Search methodology. This methodology is underpinned by the partially observable Markov decision process framework; it utilizes polynomial exploration of the action space, and double progressive widening to avoid curses of history. The developed tasking techniques are applied to a large-scale application considering the tracking problem in the emerging cislunar regime. Uncertainty studies are performed for a set of 500 objects in a variety of candidate periodic and highly elliptical orbits, with realistic sensor models incorporating physical parameters and explicit probability of detection. These simulations are utilized as a means to evaluate observer quality, considering candidate space-based sensors following L1 Lyapunov and L2 Northern Halo orbits. Results demonstrate the importance of space-based observers for maintaining estimates on objects in cislunar space and give insight into the criticality of relative motion between observers and targets when optical measurements are utilized.

Keywords: Sensor Tasking; Monte Carlo Tree Search; Cislunar SSA; Optical Sensor Systems; Orbit Determination

1. Introduction

Choosing tasking policies for a set of sensors maintaining custody of space objects in various orbit regimes has long been a relevant problem in Space Domain Awareness (SDA). As a result of accelerating growth in space object (SO) populations, it is imperative that limited observational assets are utilized efficiently. Collision concerns have increased in recent years, especially in well-populated environments such as low-Earth orbit; as such, ensuring collision avoidance requires careful tracking of in-orbit satellites and debris. The problem at hand quickly becomes combinatoric as the object catalog considered expands, and multiple competing objectives are often desired to leverage uncued detection of objects in addition to catalog maintenance. As such, the sensor tasking problem is largely broken into tractable subproblems, in which the objective is to capture a single aspect of the overarching goal.

Also of interest when considering the sensor tasking problem is application to the cislunar regime of space. Relatively little literature has been produced on the subject, and the region is expected to be a growing frontier for space exploration in coming years (Holzinger et al., 2021; Bobskill, 2012). As volumes of space further from Earth are considered, dynamic complexities are introduced, and it is no longer sufficient to neglect perturbations from the Moon and the Sun. Trajectories in the cislunar regime are not necessarily stable, and many initial conditions are chaotic even when the circular restricted three-body simplification is applied for analysis. Periodic orbits exist in the circular and elliptic-restricted three-body problems (Folta

* Corresponding author.
E-mail addresses: samuel.fedeler@colorado.edu (S. Fedeler), marcus.holzinger@colorado.edu (M. Holzinger), wwhitacre@draper.com (W. Whitacre).

https://doi.org/10.1016/j.asr.2022.05.003

Symbolic Physics Learner: Discovering governing equations via Monte Carlo tree search

Fangzheng Sun[1], **Yang Liu**[2], **Jian-Xun Wang**[3], and **Hao Sun**[4,5,*]

[1]Department of Civil and Environmental Engineering, Northeastern University, Boston, MA 02115, USA
[2]School of Engineering Sciences, University of the Chinese Academy of Sciences, Beijing, 101408, China
[3]Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN, USA
[4]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, 100872, China
[5]Beijing Key Laboratory of Big Data Management and Analysis Methods, Beijing, 100872, China

Abstract

Nonlinear dynamics is ubiquitous in nature and commonly seen in various science and engineering disciplines. Distilling analytical expressions that govern nonlinear dynamics from limited data remains vital but challenging. To tackle this fundamental issue, we propose a novel Symbolic Physics Learner (SPL) machine to discover the mathematical structure of nonlinear dynamics. The key concept is to interpret mathematical operations and system state variables by computational rules and symbols, establish symbolic reasoning of mathematical formulas via expression trees, and employ a Monte Carlo tree search (MCTS) agent to explore optimal expression trees based on measurement data. The MCTS agent obtains an optimistic selection policy through the traversal of expression trees, featuring the one that maps to the arithmetic expression of underlying physics. Salient features of the proposed framework include search flexibility and enforcement of parsimony for discovered equations. The efficacy and superiority of the PSL machine are demonstrated by numerical examples, compared with state-of-the-art baselines.

1 Introduction

We usually learn the behavior of a nonlinear dynamical system through its nonlinear governing differential equations. These equations can be formulated as

$$\dot{\mathbf{y}}(t) = d\mathbf{y}/dt = \mathcal{F}(\mathbf{y}(t)) \tag{1}$$

where $\mathbf{y}(t) = \{y_1(t), y_2(t), ..., y_n(t)\} \in \mathbb{R}^{1\times n}$ denotes the system state at time t, $\mathcal{F}(\cdot)$ a nonlinear function set defining the state motions and n the system dimension. The explicit form of $\mathcal{F}(\cdot)$ for some nonlinear dynamics remains underexplored. For example, in a mounted double pendulum system, the mathematical description of the underlying physics might be unclear due to unknown viscous and frictional damping forms. These uncertainties yield critical demands for the discovery of nonlinear dynamics given observational data. Nevertheless, distilling the analytical form of the governing equations from limited and noisy measurement data, commonly seen in practice, is an intractable challenge.

Ever since the early work on the data-driven discovery of nonlinear dynamics [1, 2], many scientists have stepped into this field of study. In the recent decade, the escalating advances in machine learning, data science, and computing power enabled several milestone efforts of unearthing the governing equations for nonlinear dynamical systems. Notably, a breakthrough model named SINDy based on

*Corresponding author

Preprint.

Plan

AlphaGo Zero

- Utilise MCTS pour **générer des exemples d'apprentissage** de qualité pour l'entraînement du réseau de neurones profond.
- Qui est lui-même utilisé pour générer de nouvelles parties d'Alpha Zero contre Alpha Zero.

AlphaGo Zero

- **October 2015:**

  AlphaGo wins 5-0 against a Go professional **Fan Hui**

- **March 2016:**

  AlphaGo wins 4-1 against **Lee Sedol**, winner of 18 world titles

- **January 2017:**

  An improved online version of AlphaGo, called **Master**, achieved 60 straight wins against top international players

- **May 2017:**

  Ke Jie, considered as the best human Go player, loses 3-0 against **AlphaGo** (**Master**)

- **Late 2017:**

  AlphaGo Zero is revealed and wins 100-0 against **AlphaGo**

  Self-taught using no human games

AlphaGo Zero

- **4.9 millions training games**
  vs. 30 millions training games (using human history of games) for AlphaGo
- **3 days of training**
  vs. Several months for AlphaGo
- **A single machine with 4 TPUs** (Tensor Processing Units)
  vs. Multiple machines with 48 TPUs
- Input: **the raw board description**
  vs. manually engineered descriptors of the board

AlphaGo Zero

- uses a **deep neural network** f_θ with parameters θ.
  - This neural network takes as an **input** the raw board representation s of the position and its history (7 past positions for black and 7 for white), and **outputs** both move probabilities and a value: $(p, v) = f_\theta(s)$.
  - The **vector of move probabilities** p represents the probability of selecting each move a (including pass), $p_a = Pr(a | s)$.
  - The **value** v is a scalar evaluation, estimating the probability of the current player winning from position s.
  - This **neural network combines** the roles of both **policy network** and **value network** into a single architecture.

AlphaGo Zero

- Playing
  - Using UCT = MCTS + UCB
  - But no rollouts
  - Until end of game

- MCTS
  - Choses each move using $Q(s,a) + U(s,a)$ (UCB)
  - When a leaf node s' is encountered: evaluate $(P(s', \cdot), V(s')) = f_\theta(s')$
    Instead of using a rollout

AlphaGo Zero

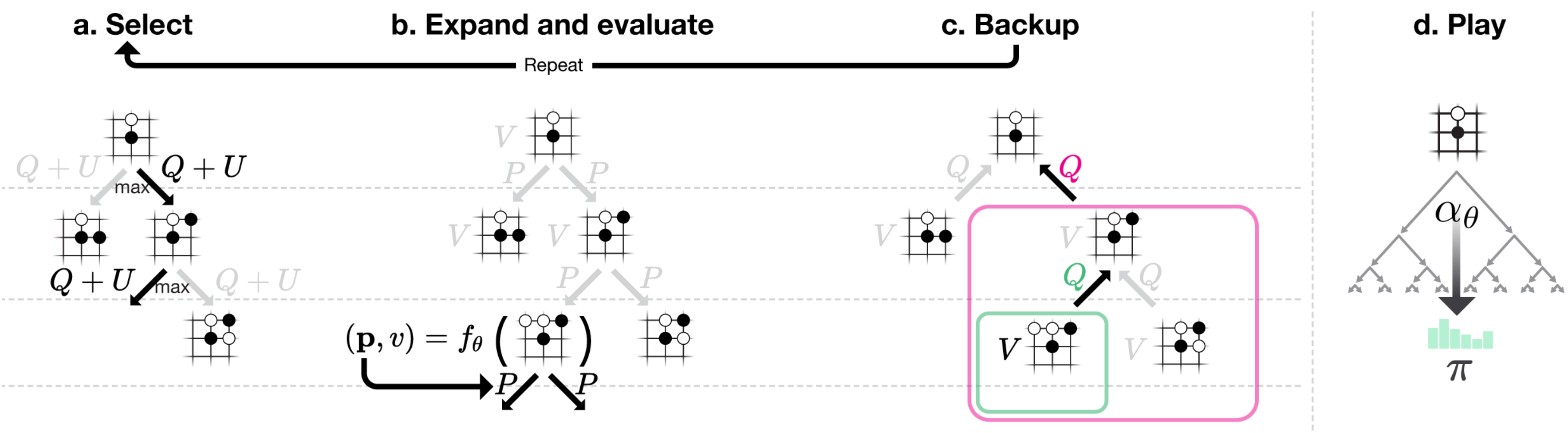
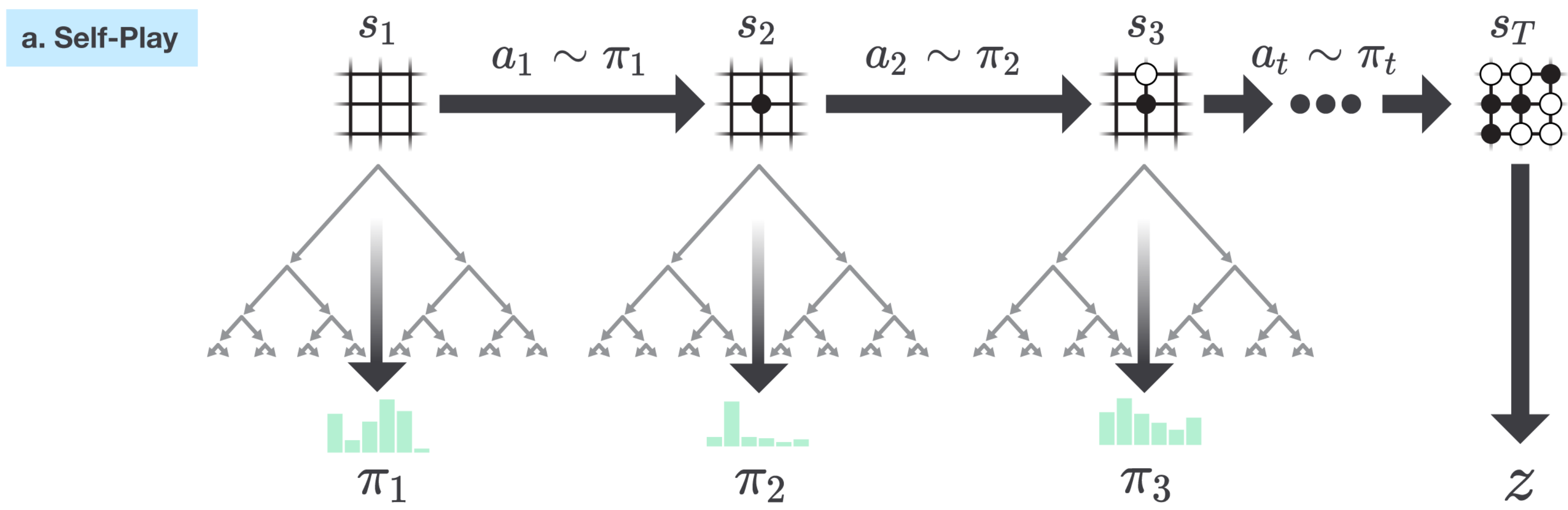


Figure 2: **Monte-Carlo tree search in *AlphaGo Zero*.** **a** Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b** The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c** Action-values Q are updated to track the mean of all evaluations V in the subtree below that action. **d** Once the search is complete, search probabilities $\boldsymbol{\pi}$ are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

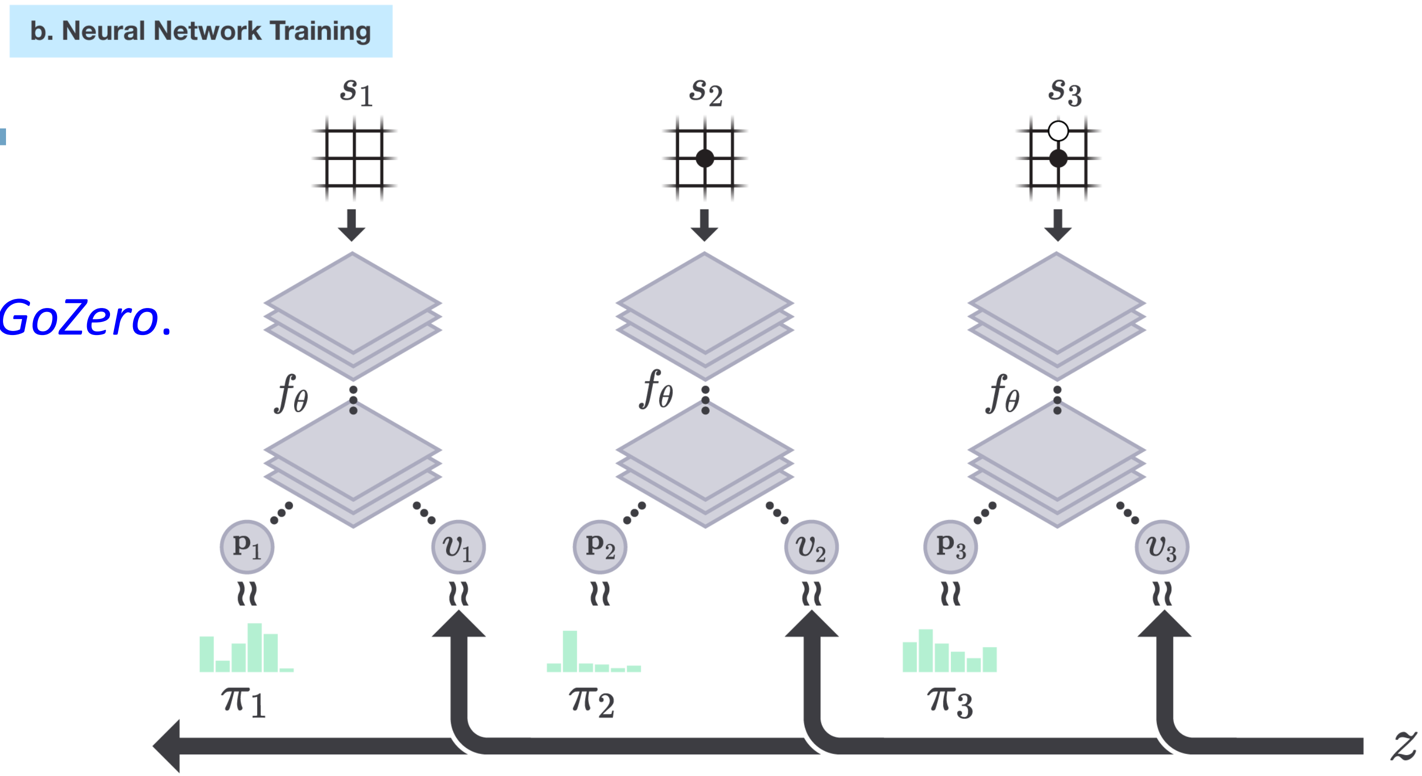
[Silver, David, et al. "***Mastering the game of go without human knowledge***. » Nature 550.7676 (2017): 354-359.]

AlphaGo Zero



Self-play in *AlphaGoZero*.

- The program plays a game s_1, ..., s_T against itself.
- In each position s_t, a Monte Carlo Tree Search (MCTS) is executed using the latest neural network f_θ.
- Moves are selected according to the search probabilities a_q computed by the MCTS, $\alpha \sim \pi_\theta$.
- The terminal position s_T is scored according to the rules of the game to compute the game winner z.



Learning in *AlphaGoZero*.

The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\mathbf{p_t}$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated so as to maximise the similarity of the policy vector $\mathbf{p_t}$ to the search probabilities $\boldsymbol{\pi_t}$, and to minimise the error between the predicted winner v_t and the game winner z (see Equation 1). The new parameters are used in the next iteration of self-play **a**.

AlphaGo Zero

- AlphaGo Zero plays like humans in the **openings** and in the **end games** which seems to show that this is the way to play best

- But its **middle-game plays** are often **truly mysterious**

References

- Vincent Barra, Antoine Cornuéjols & Laurent Miclet: "*Apprentissage artificiel. Concepts et algorithms. De Bayes et Hume au Deep Learning*". Eyrolles, 2021.
- Aske Plaat: "*Deep Reinforcement Learning*". Springer, 2022
- Max Pumperla & Kevin Ferguson: "*Deep Learning and the Game of Go*". Manning, 2019
- Stuart Russell & Peter Norvig: "*Artificial Intelligence. A modern approach*". Global Edition, 2022
- Richard Sutton & Andrew Barto: "*Reinforcement Learning. An Introduction*". MIT Press, 2018 (la bible par les pionniers du RL)
- https://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/ (pour ceux qui aiment les maths)
- https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f (très pédagogique, avec des bouts de code python)