

Topics in Reinforcement Learning:
AlphaZero, ChatGPT, Neuro-Dynamic Programming,
Model Predictive Control, Discrete Optimization, Applications
Arizona State University
Course CSE 691, Spring 2025

Links to Textbook, Videolectures, and Slides at
<http://web.mit.edu/dimitrib/www/RLbook.html>

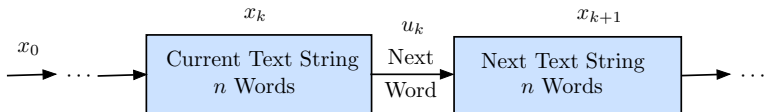
Prof. Dimitri P. Bertsekas (dimitrib@mit.edu)
and
Dr Yuchao Li (yuchaoli@asu.edu)

Lecture 8

Rollout Algorithms for Most Likely Sequence Generation in n -Grams, Transformers,
HMMs, and Markov Chains

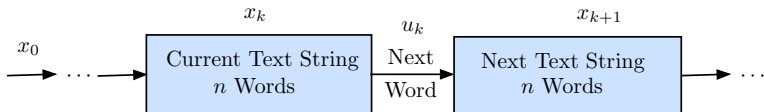
- 1 Most Likely Sequence Generation in n -Grams and HMMs
- 2 Exact and Approximate DP Algorithms for Likely Sequence Generation
- 3 Rollout Algorithms and Performance Improvement
- 4 Computational Experiments with Markov Chains
- 5 Computational Experiments with a GPT

Recall the n -Gram Model of Next Word Generation



- One word added to the front and one word deleted from the back
- This is a deterministic control system: Text window is the state x_k . Word selection is the control u_k
- Suppose we have freedom to select the next word according to a policy of our choice
- The n -gram provides transition probabilities $p(x_{k+1} | x_k)$ to which we have access
- Think of texting/next word suggestions; we can follow the suggested words or choose our own
- Our problem: Find a next word selection policy that produces a highly likely sequence $\{x_1, x_2, \dots, x_N\}$ starting from a given initial state/prompt x_0

View as a DP Problem



- **The optimal policy:** Starting at x_0 , select the most likely sequence $\{x_1, x_2, \dots, x_N\}$, according to the n -gram's suggestions.
- This is the one that **maximizes**

$$\text{Prob}(x_1, x_2, \dots, x_N \mid x_0)$$

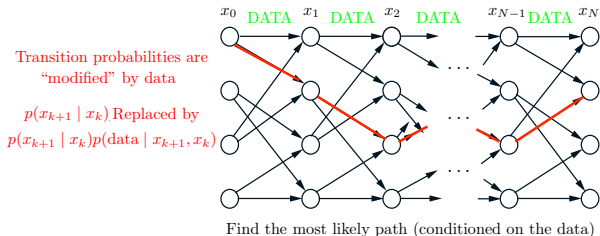
or equivalently **maximizes**

$$p(x_1 \mid x_0) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_2) \cdots p(x_N \mid x_{N-1})$$

[assuming a Markov property, i.e., $P(x_{k+1} \mid x_0, x_1, \dots, x_k) = P(x_{k+1} \mid x_k)$, and using the multiplication rule of conditional probability].

- One of its advantages is that **it plans into the future**.
- **We will use DP:** (max product of rewards \equiv max of sum of the reward logarithms)
- But exact DP requires **intractable computation**. We will focus on approximate DP/RL methods

A Closely Related Problem of Huge Practical Interest: Max-Likelihood Inference in Hidden Markov Models

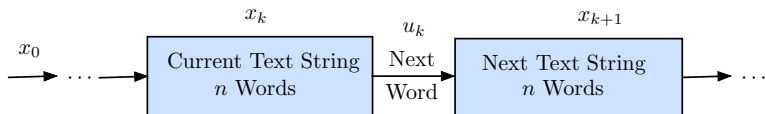


Many applications: Speech recognition, language translation, computational linguistics, coding and error correction, bioinformatics, etc

Language translation example: Given English sentence (sequence of English words, i.e., the **data**), translate it into French sentence (a sequence of transitions between candidate French words, i.e. the **states**)

- Often solved by a specialized form of DP, the **Viterbi algorithm** (1960s).
- For large state spaces **exact solution is intractable** and suboptimal shortest path-type algorithms have been used.
- **Our DP-based rollout algorithms fully apply** (just replace the transition probabilities with data-dependent/time-dependent “weights”)

We Will Consider Suboptimal Policies

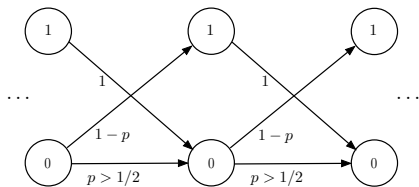


- **The greedy selection policy:** Select at each x_k the next word x_{k+1} that maximizes the next word transition probability $p(x_{k+1} | x_k)$.
- **The rollout selection policy** that uses the greedy as base policy: At x_k , it selects x_{k+1} that **maximizes the greedy Q-factor $Q(x_k, x_{k+1})$:**

$$Q(x_k, x_{k+1}) = \text{Prob}(x_{k+1}, \text{Greedy sequence starting from } x_{k+1} | x_k)$$

- **Variants of rollout:** Multistep lookahead, truncated, simplified, and their combinations.
- **Double rollout:** Rollout using as base policy the rollout-based-on-greedy (and its variants).
- Recall that **this is a deterministic DP problem** (since the n -gram system is deterministic)
- As a result, we can **contemplate powerful/sophisticated variants of rollout** involving multistep lookahead, pruning, etc (see Lecture 6, and Section 2.4 of the textbook).

An Illustrative Example: A 1-Gram with Vocabulary = {0, 1}



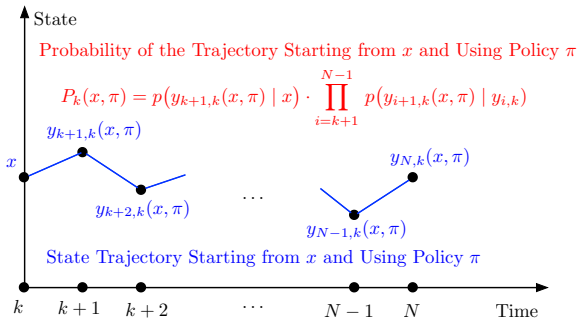
- Starting at $x_0 = 0$
- **The greedy selection policy is:** $\{x_0, x_1, x_2, \dots\} = \{0, 0, 0, 0, \dots\}$
- **The most likely selection policy is:** If $p^2 < 1 - p$ it selects $\{0, 1, 0, 1, \dots\}$ [because $p^N < (1 - p)^{N/2}$]; otherwise it selects $\{0, 0, 0, \dots\}$
- **The rollout selection policy with one-step lookahead**, starting from $x_0 = 0$, compares two Q-factors corresponding to the two next states $x_1 = 0$ and $x_1 = 1$.
- If $p^2 < 1 - p$ it selects $x_1 = 1$; otherwise it selects $x_1 = 0$. Thus **it generates the same sequence as the most likely selection policy.**

Generally, an n -gram defines a Markov chain with transition probabilities

$$p(x_{k+1} \mid x_k)$$

But the state space is enormous! (An n -gram with vocabulary consisting of q words involves q^n states.)

DP Formulation for Markov Chains: Next State Selection Policies



Given a Markov chain with transition probabilities $p(x_{k+1} | x_k)$

- A **selection policy** π is a sequence of functions $\{\mu_0, \dots, \mu_{N-1}\}$, which given the current state x_k , determines the next state x_{k+1} as $x_{k+1} = \mu_k(x_k)$.
- Given π and a starting state x at time k , the future states are denoted $y_{i,k}(x, \pi) =$ **state at time $i > k$ starting at state x and using π**
- The probability of its occurrence (the **reward-to-go function**) is

$$P_k(x, \pi) = p(y_{k+1,k}(x, \pi) | x) \cdot \prod_{i=k+1}^{N-1} p(y_{i+1,k}(x, \pi) | y_{i,k})$$

Most Likely and Greedy Selection Policies

- The **most likely selection policy**, denoted by $\pi^* = \{\mu_0^*, \dots, \mu_{N-1}^*\}$, maximizes over all policies π the probabilities $P_k(x, \pi)$ for every initial state x and time k :

$$P_k^*(x) = P_k(x, \pi^*) = \max_{\pi} P_k(x, \pi).$$

- The (multiplicative form of the) DP algorithm to obtain π^* and its probabilities $P_k^*(x)$:

- ▶ First compute the probabilities $P_k^*(x)$ backwards, for all x , according to

$$P_k^*(x) = \max_y \rho(y | x) P_{k+1}^*(y), \quad k = N-1, \dots, 0,$$

starting with $P_N^*(x) \equiv 1$.

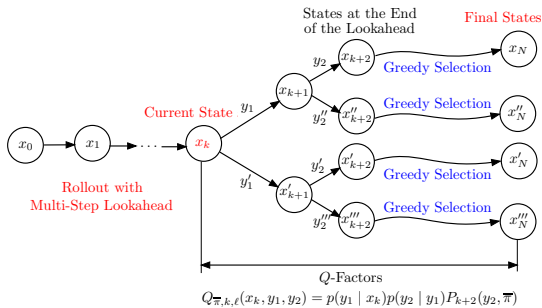
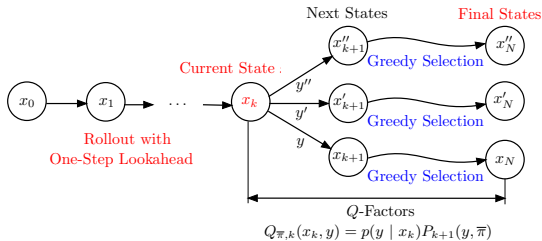
- ▶ Then generate forwards the selections x_1^*, \dots, x_N^* of π^* , according to

$$x_{k+1}^* = \mu_k^*(x_k^*) \in \arg \max_y \rho(y | x_k^*) P_{k+1}^*(y),$$

starting with $x_0^* = x_0$.

- It is equivalent to the usual DP for multistage additive costs, after we take logarithms of the multiplicative expressions defining the probabilities $P_k(x, \pi)$.
- The **greedy policy** $\bar{\pi} = \{\bar{\mu}_0, \bar{\mu}_1, \dots, \bar{\mu}_{N-1}\}$ produces the next state by maximization of the corresponding transition probability over all y : $\bar{\mu}_k(x) = \arg \max_y \rho(y | x_k)$ (ties are broken according to a fixed rule for sequential consistency).

One-Step and Multistep Rollout Selection Policies



There are also truncated and simplified variants, etc

The Performance Improvement Property

Induction proof of performance improvement for rollout with one-step lookahead. (Sequential consistency holds here; the base heuristic is a policy)

- Want to show that for the greedy policy $\bar{\pi}$ and the rollout policy $\tilde{\pi}$, we have

$$P_k(x, \bar{\pi}) \leq P_k(x, \tilde{\pi}), \quad \text{for all } x \text{ and } k$$

- For $k = N$ this holds, since we have $P_N(x, \bar{\pi}) = P_N(x, \tilde{\pi}) \equiv 1$.
- Assuming that

$$P_{k+1}(x, \bar{\pi}) \leq P_{k+1}(x, \tilde{\pi}), \quad \text{for all } x,$$

we will show that

$$P_k(x, \bar{\pi}) \leq P_k(x, \tilde{\pi}), \quad \text{for all } x.$$

- We have

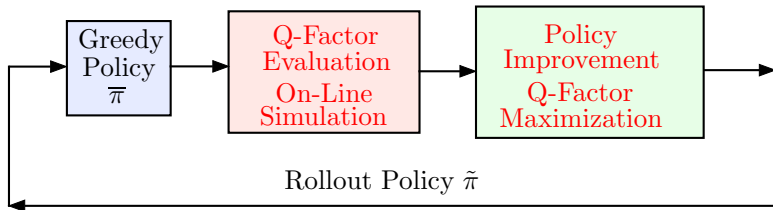
$$P_k(x, \tilde{\pi}) = p(\tilde{\mu}_k(x) | x) P_{k+1}(\tilde{\mu}_k(x), \tilde{\pi}) \quad (\text{by definition}) \quad (1)$$

$$\geq p(\tilde{\mu}_k(x) | x) P_{k+1}(\tilde{\mu}_k(x), \bar{\pi}) \quad (\text{by the induction hypothesis}) \quad (2)$$

$$\geq p(\bar{\mu}_k(x) | x) P_{k+1}(\bar{\mu}_k(x), \bar{\pi}) \quad (\text{rollout maximizes greedy Q-factor}) \quad (3)$$

$$= P_k(x, \bar{\pi}) \quad (\text{by definition}) \quad (4)$$

Multiple Policy Iterations - Double Rollout Algorithm



- **Single rollout** is rollout with greedy as base policy
- **Double rollout** is rollout with single rollout as base policy
- **Triple rollout** is rollout with double rollout as base policy
- **k -order rollout** is rollout with $(k - 1)$ -order rollout as base policy
- These rollout algorithms can be **multistep lookahead, truncated, simplified, etc**
- For double rollout, at any encountered state x_k , we run the single rollout from every next state y and select x_{k+1} to maximize the Q-factor $Q_{\tilde{\pi},k}(x_k, y)$ over y
- **Double rollout requires $O(q \cdot N)$ applications of single rollout** (q is the number of possible next states after simplification)

With a large enough number of policy iterations, the most likely sequence is obtained

A Fifteen-Minute Break

Catch our breath and think about issues relating to the first half of the lecture.
Ask questions when you return.

Computational Experiments with Markov Chains

States	1	2	...	100
1	$p(1 1)$	$p(2 1)$...	$p(100 1)$
2	$p(1 2)$	$p(2 2)$...	$p(100 2)$
⋮	⋮	⋮	⋮	⋮
100	$p(1 100)$	$p(2 100)$...	$p(100 100)$

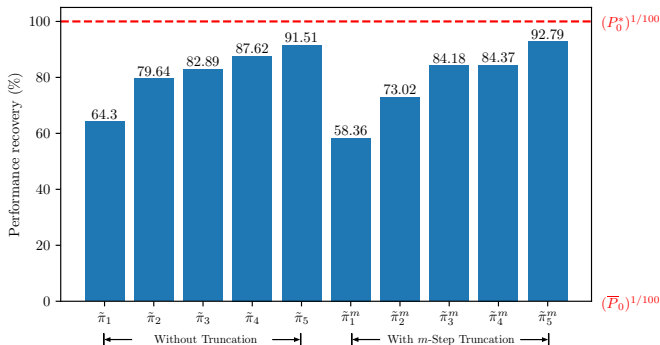
States 1, 2, ..., 100
xth row: transition probabilities
from x to all other states

- We consider $N = 100$ most likely sequence for a set C of Markov chains involving 100 states. For each state x , there are 5 states y such that $p(y|x) > 0$
- The transition probabilities are given in a lookup table
- The most likely sequence can be computed via DP
- We measure the performance of rollout by the percentage recovery of optimality loss of the greedy policy, given by

$$\frac{(\tilde{P}_0)^{1/N} - (\bar{P}_0)^{1/N}}{(P_0^*)^{1/N} - (\bar{P}_0)^{1/N}} \times 100 (\%)$$

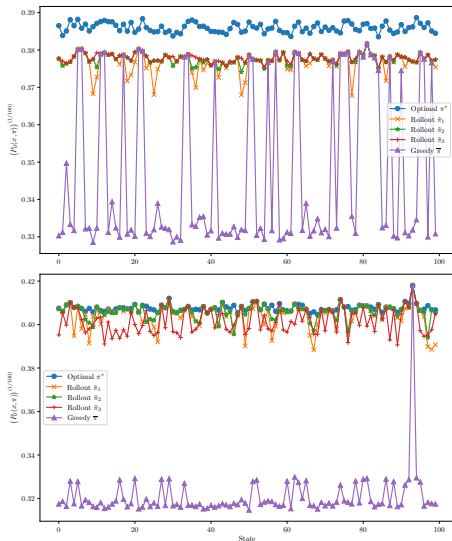
where $(\bar{P}_0)^{1/N}$, $(\tilde{P}_0)^{1/N}$, and $(P_0^*)^{1/N}$ are the average transition probabilities under $\bar{\pi}$, $\tilde{\pi}$, and π^* , respectively.

Percentage Recovery by Rollout



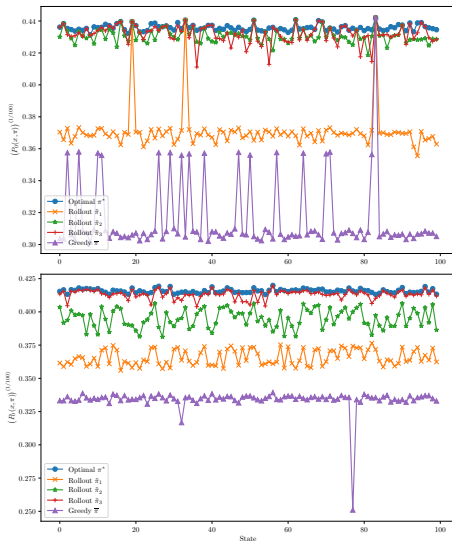
- We consider **rollout with one-step and ℓ -step lookahead** ($\ell = 2$ to 5), denoted by $\tilde{\pi}_\ell$, and their **m -step truncated counterparts with $m = 10$** , denoted by $\tilde{\pi}_\ell^m$
- General observations from the experiments:
 - ▶ Rollout leads to **substantial improvements** over the greedy policy in all test cases
 - ▶ Longer lookahead **results in improvement on average**
 - ▶ The performance seems **unaffected by the 90% truncation of the rollout horizon**

Typical Patterns of Rollout (1)



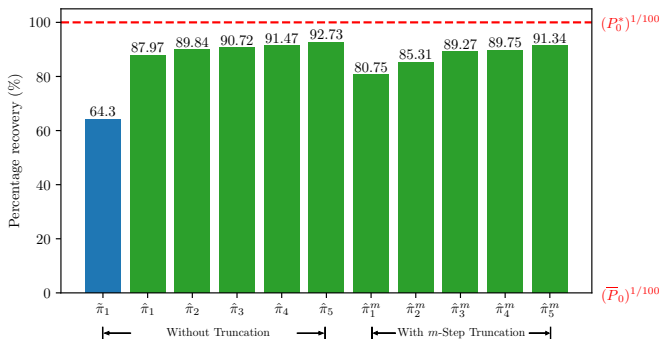
For a given Markov chain, $(P_0(x, \pi))^{1/N}$ with π being π^* , $\tilde{\pi}_\ell$ with $\ell = 1, 2, 3$, or $\bar{\pi}$

Typical Patterns of Rollout (2)



For a given Markov chain, $(P_0(x, \pi))^{1/N}$ with π being π^* , $\tilde{\pi}_\ell$ with $\ell = 1, 2, 3$, or $\tilde{\pi}$

Percentage Recovery by Double Rollout



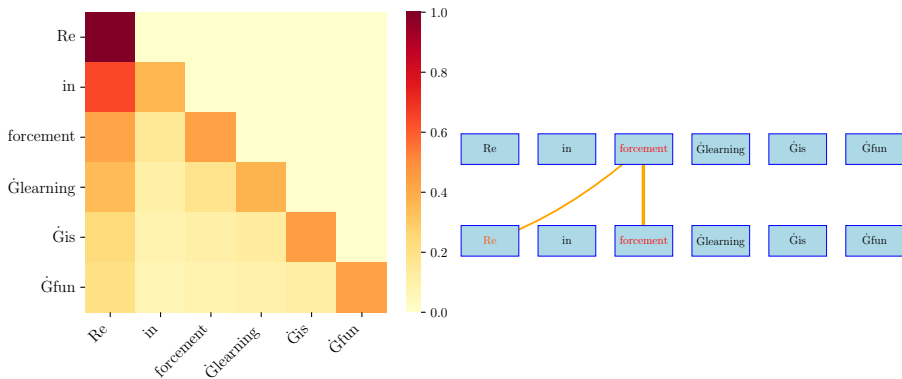
- We consider **double rollout with one-step and ℓ -step lookahead** ($\ell = 2$ to 5), denoted by $\hat{\pi}_\ell$, and their **m -step truncated counterparts with $m = 10$** , denoted by $\hat{\pi}_\ell^m$, and their results are shown in green bars
- General observations from the experiments:
 - ▶ Double rollout algorithm and its variants lead to significant performance improvement over **both the greedy policy and (single) rollout with one-step lookahead**
 - ▶ The truncated versions of double rollout **remain effective, despite large computational savings**

Generative Pre-Trained Transformer - GPT



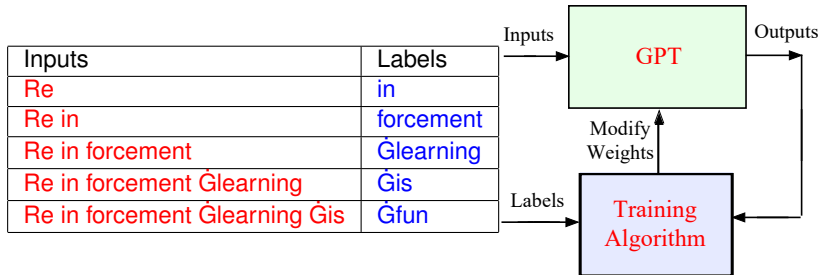
- **Transformer** architectures improve in important ways on the earlier neural networks by using the **attention mechanism**
- Such neural networks are often **Pre-Trained** with **a lot of general purpose data** before they are further trained (fine-tuned) with special purpose data
- Owing to both the architecture and pre-training, the resulting neural networks are **Generative**: being able to create new content and to perform open-ended tasks
- Tremendous amount of applications: natural language processing, computer vision, image/music generation (see the figure above) ...
- We focus on **text generation** and view the GPT as an **n -gram**.

Attention Mechanism in a GPT



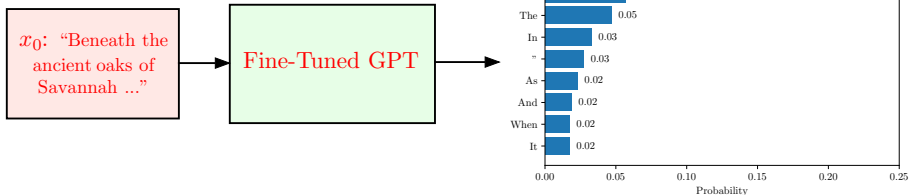
- Consider the text string 'Reinforcement learning is fun' as an input
- It is viewed as 6 'words' (called **tokens**) by the GPT
- For each feasible ordered 'word' pair, an **attention score** is generated, measuring the **affinity** between them: how strongly they are related
- The '**attention**' of the neural network is given to relations with **high scores**

Training Data and Algorithms for GPT



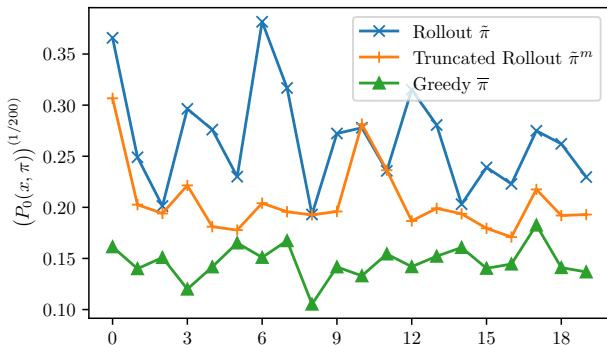
- The text string 'Reinforcement learning is fun' corresponds to 5 input-label pairs used for training
- Using text strings for training does **not** require explicit data-labeling.
- The automatic generation of labeled data from unlabeled data for training **enables the pre-training with large amount of general purpose data**
- The labeled data generation and training process are collectively known as **self-supervised learning**
- The training algorithms used for GPT are scaled versions of **stochastic gradient descent**, more on this in the next lecture

Most Likely Word Sequence from a GPT



- We generated most likely sequences, using a fine-tuned GPT, which defines an ***n*-gram and its associated Markov chain**. We used $N = 200$ and $n = 1024$.
- The transition probabilities are **generated by the transformer**
- The number of different n -grams is 50258^{1024} , **enormous! Intractable via DP**
- The large vocabulary size leads to **excessive Q-factor computations**
- We applied **simplified** rollout and its **truncated** counterpart
- Rollout can take advantage of the **parallel processing power** of graphical processing units (GPU)

Performance of Simplified Rollout



- We applied two simplification techniques:
 - ▶ Computing only 10 Q-factors corresponding to top ten most likely next words: **simplified rollout with one-step lookahead**
 - ▶ In addition, truncating the simulation after 10 steps: **m-step truncated rollout**
- General observations from the experiments:
 - ▶ Simplified rollout has substantial improvement **over the greedy policy**, with modest computation increase
 - ▶ The truncated counterpart **still improves upon the greedy policy** in all our test cases

Rollout and approximation in value space for stochastic and continuous spaces problems. Monte-Carlo Tree Search. Model predictive control.