Large Scale Optimization via Monte Carlo Tree Search

Larkin Liu [1]

larkin.liu@tum.de

[1]Technical University of Munich

Paris October 5, 2023

Overview

Larkin Liu

Larkin Liu (born 1992) is a Chinese-Canadian research scientist. He studied first at the University of Toronto, obtaining his Master's degree in Industrial Engineering. Larkin has worked extensively as a Data Scientist in companies across both Germany and Canada. Currently, he is a Doctoral Student at the Technical University of Munich in Computer Science, specializing in research in machine learning and operations research.

Introduction

Motivation

Topic of this talk: **Large Scale Optimization via Monte Carlo Tree Search**.

Objectives

- Share research findings and encourage dialogue.
- Identify areas of collaboration, via shared objectives.
- Get candid feedback and criticism, please go ahead.

Objectives

- Introduces the problem of high dimensional sequential decision making.
- Proposes an MDP formulation for maritime bunkering, and proposes a stochastic programming solution based on scenario tree generation.
- Proposes an application of Monte Carlo Tree Search, to address the curse-of-dimensionality associated with large scale MDP's.

Sequential Decision Making

- Decision occurs with **state transitions** and **rewards** (and/or consequences) based on each state.
- Sequential decision making can be **stochastic** or **deterministic**. Applies to both policy and/or state transitions.
- Optimization over a **finite** time horizon or **infinite** time horizon.
- **Learning** of model parameters vs **optimization** of a model.

Markov Property

- A state should summarize past sensations so as to retain all essential information.
- The probability of transitioning to a state, and its reward, is dependent only on the previous state.
- Previous history can be discarded.

Markov Property

$$\mathbf{P}(R_{t+1}, S_{t+1} | S_0, A_0, R_1 ... R_t, A_t, S_t) = \mathbf{P}(R_{t+1}, S_{t+1} | R_t, A_t, S_t) \tag{1}$$

Markov Decision Process

Q function

$Q(S_t, a_t)$ provides a measure of the discounted reward provided action a is taken in state S_t

$$Q(S_t, a_t) = R(S_t, a_t) + \gamma \sum_{S_{t+1} \in S} P(S_{t+1} | S_t, a_t) V(S_{t+1})$$

$$\pi^*(S_t) = \underset{a \in A}{\operatorname{argmax}}\, Q(S_t, a) \tag{3}$$

Markov Decision Process

Key Challenges for real-world MDP's

- Parameters of the underlying process $MDP\langle S, A, \mathbb{T}, R\rangle$ are unknown.
- Imperfect conditions and/or unobservable information.
- High dimensionality of state and action space.

Value Based Planning

The policy of an agent can be driven by the value of a state

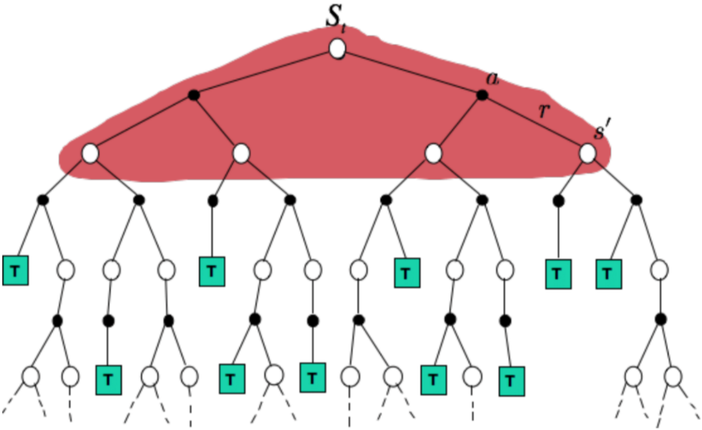
Value Definitions by Policy

$$G_t = R_{t+1} + ... + R_{t+2} + ... + R_T \tag{4}$$
$$V_\pi(S_t) = E_\pi[G_t | S_t] \tag{5}$$
$$V_\pi(S_t) = E_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})] \tag{6}$$

Value Definitions by Maximization

$$V(S_t) = \max_{a \in A}[R_{t+1} + \gamma V(S_{t+1}, a)] \tag{7}$$
$$V(S_t) = \max_{a \in A} Q(S_t, A_t) \tag{8}$$

Dynamic Programming Visualization



Visualization of Dynamic Programming

Policy vs Value Iteration, and DP Limitations

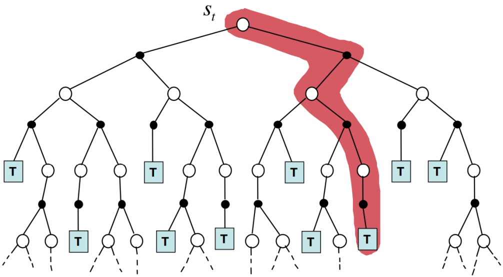
Limitations of DP - Intractable for large state action spaces.

- High number of states.
- High branching factor.

Complexity

- Value iteration: Each iteration $O(|S|^2|A|)$.
- Policy iteration: Each iteration $O(|S|^3 + |S|^2|A|)$.
- DP Methods are suitable for problems under 10^6 states.

Bias Variance Tradeoff - Monte Carlo Estimation



1 Iteration of Monte Carlo Update.

- TD Learning uses the one-step ahead Value V(S') to estimate the true G function.
- A full MC update may be biased, but have less variance.

UCB-1

Algorithm UCB1 Strategy

1: $Q = \emptyset$
2: **for** $t = 0 \to T$ **do**
3: **for** $k = 1 \to K$ **do**
4: Compute $\widehat{\mu_t^a}$
5: **end for**
6: Play $a_t = \text{argmax}_a\, m_t^a$
7: $Q \leftarrow Q(a)$
8: **end for**

We seek to maximize m_t^k where,

$$m_t^a = \mu_t^a + \sqrt{\frac{2 \log t}{n(a)}} \tag{9}$$

UCB1 Regret Bound (Auer, 2002)

$$\mathbb{E}[R_T(\pi)] \geq 8 \sum_{a:\mu_t^a < \mu_t^*} \left(\frac{\log n(a)}{\mu_t^a - \mu_t^*} \right) + \left(1 + \frac{\pi^2}{3}\right) \left(\sum_{a=1}^{A} \mu_t^a - \mu_t^* \right) \tag{10}$$

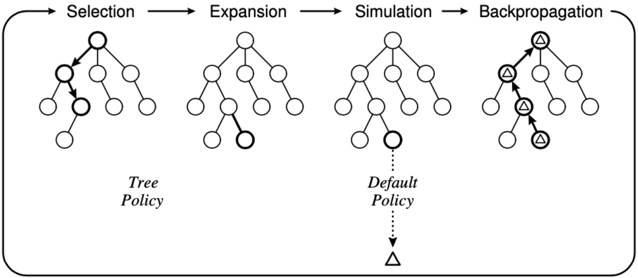
Extending Multi-Armed Bandit to Markov Decision Processes

Challenges:

- Incomplete model, so need to estimate values of states and actions
- Need to balance exploration vs exploitation
- MDPs are stochastic in nature
- Large branching factors (width) and many steps until reward (depth)

Proposed solution: Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search



UCT guides the tree search from to the next possible state S' via UCB1 MAB strategy. Where n is the number of visits at the parent state at S and n' is the number of visits for S'. $E(S')$ is the expected reward.

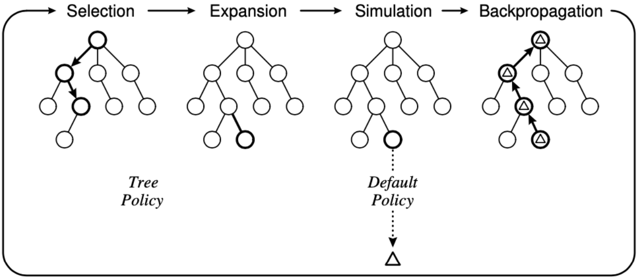
UCT Selection Strategy

$$UCT(S') = E(S') + \sqrt{\frac{2 \ln n}{n'}} \tag{11}$$

MCTS Algorithm - Layman's Version

- **Selection**
  - Select an unvisited node.
  - Select an action according to UCB1.
- **Expansion**
  - Perform exploratory action at a frontier.
  - Obtain one new node.
- **Simulation (rollout)**
  - Simulate randomly, without indicators such as UCB, to obtain an unbiased approximation of the payoff.
- **Backpropagation**
  - Terminal node has been reached.
  - Propagated discounted reward up to the root node.

MCTS Application to MDP's



- (Chang et al. 2010) demonstrates MCTS is a adaptation of the MAB strategy to for MDP's.
- (Bertsimas et al. 2014) showed that MP and MCTS perform similarly. Where MCTS performance in indifferent to the MDP formulation.

MCTS Application to MDP's (Cont.)

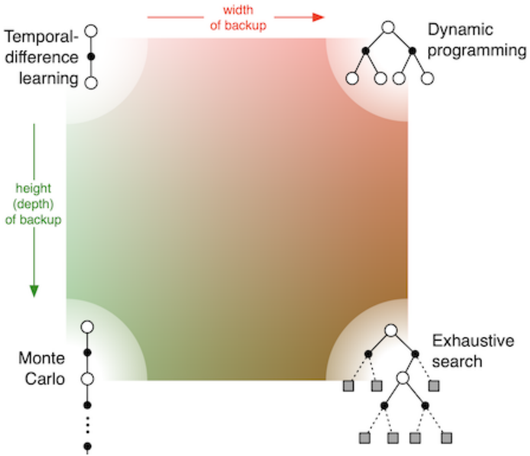
Challenges of MCTS

- Falling in local minima/maxima Value Function traps.
- No guarantees on value function convergence under imperfect scenarios.
- Guesswork involved with determining exploration heuristics.

Advantages of MCTS

- Does not require model parameters $MDP\langle S, A, \mathbb{T}, R\rangle$.
- Stochastically explores search space and can handle large depths and widths.
- $E(S')$ can be determined flexibly, allowing room for heuristics and hybridization with Mathematical Programming (Baier 2013) Baier and Winands 2013.

Different variants of MCTS



- There's an entire spectrum of search methods to choose from!

Optimization strategies for MCTS

- Hybrid with Dynamic Programming Feldman and Domshlak 2014
- Heuristics, from human knowledge, or Deep Learning.
- Value/policy function approximators (potentially from Deep Learning).
- Parallelism

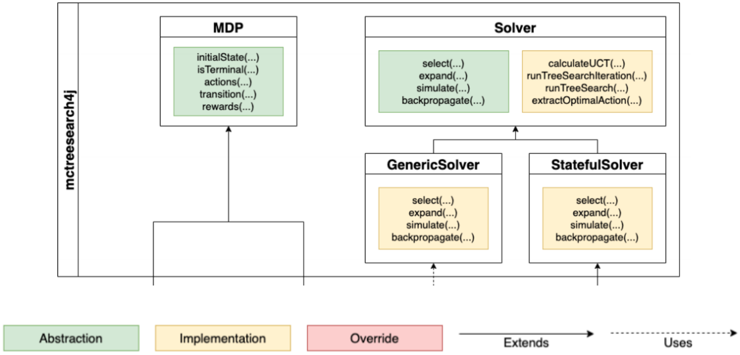
Dynamic Programming and MCTS

- Dynamic Programming (DP) is an exact solution to the MDP.
- DP is backward induction vs. MCTS forward approximation via sampling (Approx DP.)
- With stochastic DP, used learned MDP parameters to produce a weighted sum expected reward.

References:

- Feldman, Zohar, and Carmel Domshlak. "Monte-Carlo tree search: To MC or to DP?." ECAI. 2014.

Core mctreesearch4j library



- The core library provides both implementations and abstractions for MCTS.
- Solver class abstractions are predefined whereas MDP abstractions require definition.

Defining the MDP via Abstract Class

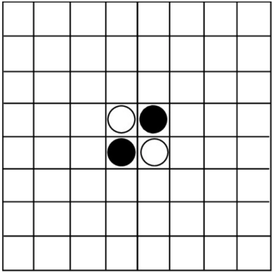
- An MDP is defined via an Abstract Class.
- The State Action Space can be defined via Generic Types.

```
abstract class MDP<StateType, ActionType> {
    abstract fun transition(StateType, ActionType) : StateType
    /* Definee the State (StateType) Action (ActionType) transition */

    abstract fun reward(StateType, ActionType?, StateType) : Double
    /* Returns a reward (Double) given state transitions parameters */

    abstract fun initialState() : StateType
    /* Return the initial state of the root (StateType) */

    abstract fun isTerminal(StateType) : Boolean
    /* Return boolean indicating if the state is terminal. */

    abstract fun actions(StateType) : Collection<ActionType>
    /* Return an Iterable of legal actions given a current state. */
}
```

Demo Time

Demo Time

Reversi Heuristic



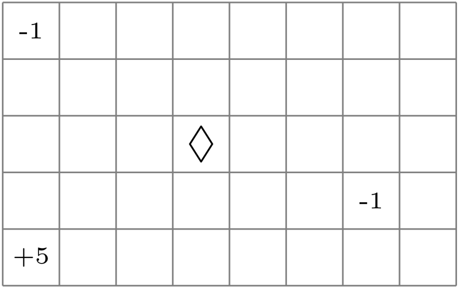
100	-10	11	6	6	11	-10	100
-10	-20	1	2	2	1	-20	-10
10	1	5	4	4	5	1	10
6	2	4	2	2	4	2	6
6	2	4	2	2	4	2	6
10	1	5	4	4	5	1	10
-10	-20	1	2	2	1	-20	-10
100	-10	11	6	6	11	-10	100

- A simple heuristic was implemented using domain knowledge to give value to states, and alter the MCTS search mechanism.

Experimental Results

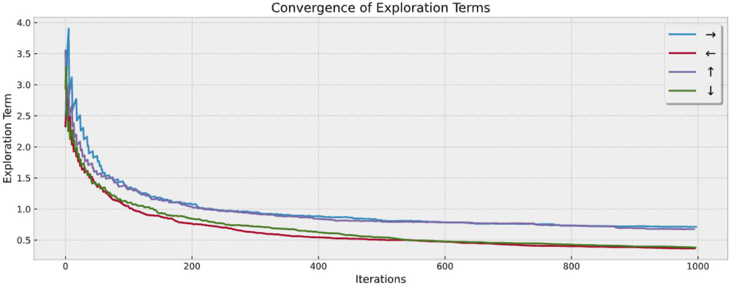
Experimental Results

GridWorld



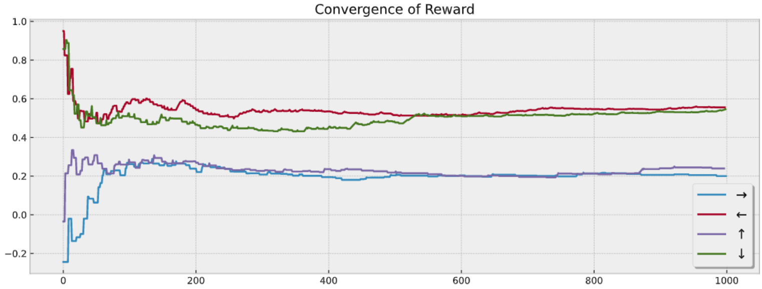
- In Gridworld, actions are not always deterministic, but the agent can go in any direction given an action. The state transitions are governed by discrete probabilities

GridWorld Results



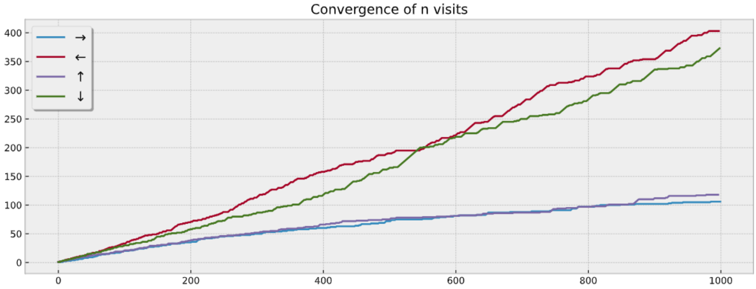
- Convergence of exploration terms.

GridWorld Results



- Convergence of reward.

GridWorld Results



- Convergence of visits.

Wrap-up

What did we learn today? What's next?

- **Modular and Extensible Design** The design of *mctreesearch4j* enables the whole or partial reuse or redefinition of all key components of MCTS.
- **Lightweight implementation** The relatively lightweight implementation of MCTS, allows it to run on any device (ie. Mobile Applications etc.)
- **Research Platform** Extending from the design of *mctreesearch4j*, it can be used as an experimentation platform for future research in MCTS-base algorithms (hybrid or modification).

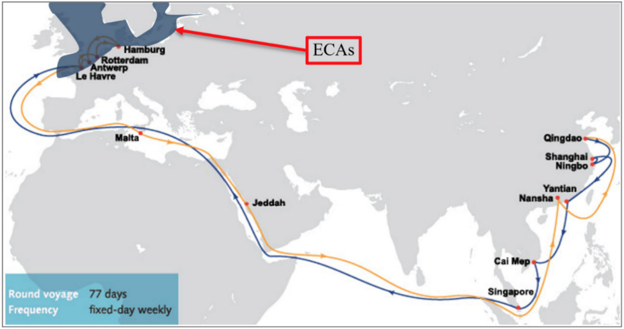
Case Study: Maritime Logistics

Optimizing Fuel Consumption for a Maritime Liner

Ship Bunkering at a Port-of-Call

The Bunkering Problem (Maritime Refuelling)

Consider a Liner must travel a fixed schedule for n ports. $n = 0, 1, 2, 3..N$. The distance between ports n_1 and n_2 is $d(n, n')$ given by a distance matrix. The route schedule is fixed D.



Example of a fixed liner route (Asia Europe LL5).

The Bunkering Problem (Maritime Refuelling)

Consider a Liner must travel a fixed schedule for n ports. $n = 0, 1, 2, 3...N$. The distance between ports n_1 and n_2 is $d(n, n')$ given by a fixed distance matrix D. The route schedule is fixed. A liner must determine how much fuel to refuel (bunker) at each port-of-call. The objective is to complete the trip, with the least fuel consumption.

Simplifying Assumptions:

- Fuel prices are subject to global stochastic variation.
- Fuel consumption is linear and deterministic.
- Distance, to and from each port, is fixed and deterministic.
- Sailing speed is fixed, there is no time penalty for late/early arrivals.
- No possibility of service disruptions.

The Bunkering Problem - MDP Definition

Proposed State Definition

$$
\begin{aligned}
S_n &= (X_{n,1}, P_{n,k}, n') && (12)\\
A_n &= \Delta_n = X_{n,2} - X_{n,1} && (13)\\
P(S_{n'}|A_n, S_n) &= (X_{n,2} - f(n, n'), \mathbf{1}[P_n = P_{n,k}], n') && (14)\\
R(S_n, A_n) &= \Delta_n P_n + Y_n B_n, && (15)\\
Y_n &= \mathbf{1}[X_{n,2} - X_{n,1} > 0] && (16)\\
P_{n,k} &\sim Multi(K) && (17)
\end{aligned}
$$

Objective

$$
C_\pi = \mathbb{E}\left[\sum_{n \in \mathbf{N}} P_n(X_{n,2} - X_{n,1}) + B_n Y_n\right] \qquad (18)
$$

Notation and Variable Definitions

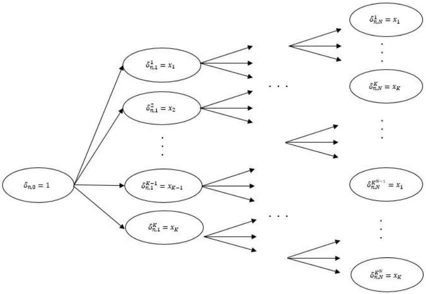
Notation	
N	Number of port-of-calls.
$X_{n,1}$	Fuel level at port n when arriving at port.
$X_{n,2}$	Fuel level when departing port n.
$f(n, n+1)$	Fuel consumption function from port n until next port $n+1$.
P_n	Price of fuel at port n.
$Y_n \in (0, 1)$	Indicator for bunkering decision.
B_n	Fixed bunkering cost.

Formulate as Stochastic Programming

Solution can also be obtained via stochastic programming.

$$
\begin{aligned}
\min_{X} \quad & C^* = \sum_{n \in \mathbf{N}} P_n(X_{n,2} - X_{n,1}) + B_n Y_n && \text{(19a)} \\
\text{subject to} \quad & X_{n+1,1} = X_{n,2} - f(d(n, n+1)), && \text{(19b)} \\
& X_{n+1,2} \geq X_{n,2} - X_{n+1,1} && \text{(19c)} \\
& f(d(n, n+1)) \geq 0 && \text{(19d)} \\
& Y_n \in (0, 1) && \text{(19e)} \\
& Y_n = \mathbf{1}[X_{n,2} - X_{n,1} > 0] && \text{(19f)}
\end{aligned}
$$

Scenario tree generation



Scenario tree. (i) The nodes of a scenario tree represent the fuel price percentage changes, (ii) The values x_i are events of the multinomial distribution $Multi(K)$ that occur with equal probabilities, (iii) The root node assumes no price change, i.e., $x = 1$, (iv) There is a total of $S = K^N$ scenarios (tree leafs) where N is the number of ports and K is the number of events, (v) Scenarios are shared between all ports.

Let the cost minimization begin!

- Stochastic Programming (SP) in this example, provides the theoretical expected cost minimum.
- However, if the scenario distribution is more general, ie Mixture of Gaussians, SP is limited to discredited scenarios.
- SP cannot be used if model parameters unknown, we rely on MCTS for learning of parameters.

Stochastic programming optimization vs. MCTS.

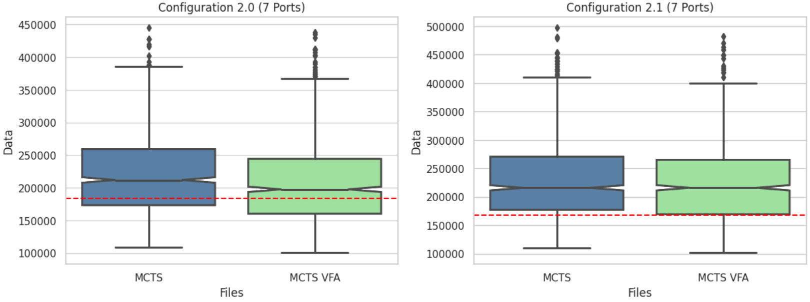
Current framework available in Python and Kotlin.

Value Function Approximation

Algorithm Monte Carlo Tree Search (MCTS) with Value Function Approximator

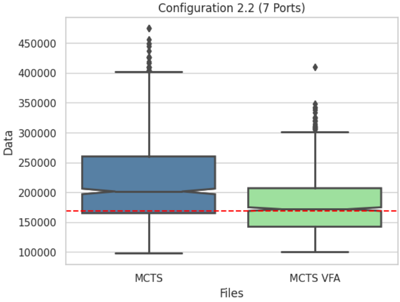
```
Input: Initialize state (chance node) s_0.
Output: Best action a*
while Max iterations not exceeded. do
    s_selected → Selection(v_0)
    a_expanded ← Expansion(v_selected) "Decision node (action)"
    if α > Uniform[0, 1] then
        Q(s, a) ← Simulation(a_expanded)
    else
        Q(s, a) ← 𝒥(s, a)
    end if
    Backpropagation(s_expanded, Q(s, a))
end while
a* ← BestAction(s_0)
return a*
```

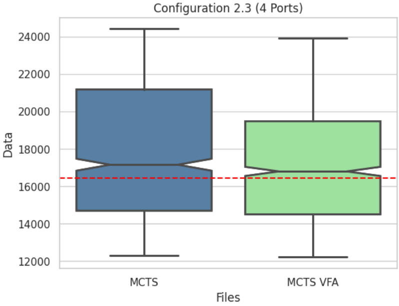
VFA Results



MCTS Enhancement using value function approximator where $\mathcal{J}(s, a)$ is a coarse grained stochastic programming solution on expected cost.

VFA Results - cont.





MCTS Enhancement using value function approximator where $\mathcal{J}(s, a)$ is a coarse grained stochastic programming solution on expected cost.

Julia Integration with Python

Julia Integration with Python

Callback with JuliaPy

- Implemented a callback function to support calling Julia subroutines inside of Python, via PyJulia.
- Callbacks can be used to trigger Julia code inside of Python.
- Python and Julia share similar data structures and philosophies (i.e. high level, multi-paradigm, interoperability).

Demo time: Calling the cascade_func() in both Python and Julia.

```
from julia import Julia
from common.properties import *
import time

jl = Julia(compiled_modules=False)
jl.include("julia/julia_callbacks.jl")

jl_result = jl.cascade_func(arg1, arg2)
result = cascade_func(arg1, arg2)
```

Integration with JuMP

- Call the Julia callback function using Julia.function_name (args) to create the JuMP optimization model with the callback constraints.
- Solve the optimization problem and retrieve results.

```
import julia
from julia import Julia
import JuMP

julia.install()
julia.include("stochastic_programming.jl")

JuMP.optimize!(julia_callback)
optimal_value = JuMP.objective_value(julia_callback)

julia_callback = Julia.stochastic_programming_callback(args)

JuMP.optimize!(julia_callback)
optimal_value = JuMP.objective_value(julia_callback)
```

Implementation of mctreesearch4j in Julia

- It is on the roadmap that we develop a new version of mctreesearch4j/mcts4py, in Julia.
- mcts4julia follows the same principles of modular component design, and state action abstraction.
- Essentially cross-lingual implementations.

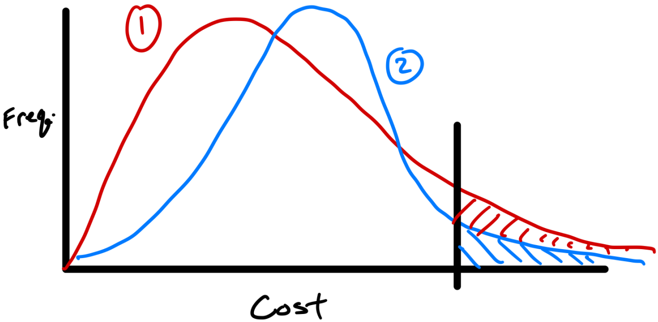
Future Theoretical Directions

Future Theoretical Directions

Additional Theoretical Concepts

- **Progressive widening**: One can more efficiently explore continuous action spaces via discrete action approximation.
- **The Learning Problem:** In reality the transition of prices are unknown, and this poses a learning problem.
- **Optimal Stopping Problem:** Assuming even if we have an optimzation.
- **Robust Optimization:** Value at risk and worst case scenario, versus expected cost.

Cost Robustness



Illustrating cost robustness.

Acknowledgements

- Jun Tao Luo (Carnegie Mellon University)
- Matej Jusup (ETH Zürich)

References I

Baier, H. and M. H. M. Winands (2013). "Monte-Carlo Tree Search and minimax hybrids". In: pp. 1–8. DOI: 10.1109/CIG.2013.6633630.

Bertsimas, Dimitris et al. (2014). *A Comparison of Monte Carlo Tree Search and Mathematical Optimization for Large Scale Dynamic Resource Allocation*. arXiv: 1405.5498 [math.OC].

Chang, Hyeong Soo et al. (2010). "Adaptive Adversarial Multi-Armed Bandit Approach to Two-Person Zero-Sum Markov Games". In: vol. 55. 2, pp. 463–468. DOI: 10.1109/TAC.2009.2036333. URL: https://doi.org/10.1109/TAC.2009.2036333.

Feldman, Zohar and Carmel Domshlak (2014). "Monte-Carlo Tree Search: To MC or to DP?" In: *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. Vol. 263. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 321–326. DOI: 10.3233/978-1-61499-419-0-321. URL: https://doi.org/10.3233/978-1-61499-419-0-321.

Copyright Notice