Interpretable Reinforcement Learning

Definitions, algorithms, and more…

Scenario..

Imagine you are a doctor aiming to prescribe medication to a patient.

You go to an **AI expert** to help you prescribe the right medicine.
The consultant gives you a decision making algorithm to help you do this.

As a doctor you would want this algorithm to possess the following key property; which is **INTERPRETABILITY.**

In my research, we aim to help this doctor by delving into the field of **interpretable reinforcement learning.**

RL preliminaries

We recall the definition of a **Markov Decision Process** (MDP). An MDP is a tuple given by $(T, \gamma, (S_t)_{t=1}^T, (A_t)_{t=1}^T, (r_t)_{t=1}^T, (P_t)_{t=1}^T)$

Here we have that:

1. **T** := Time horizon for the MDP
2. γ:= Discount factor for the MDP
3. **S_t** := State space for the different timesteps
4. **A_t** := Action space for the different timesteps
5. $r_t : S_t \times A_t \longrightarrow \mathbb{R}$, the reward function for the different timesteps
6.
7. $P_t : S_{t+1} \times S_t \times A_t \longrightarrow \mathbb{R}$ the transition function for the different timesteps

RL preliminaries

A policy π at timestep t, is a random variable from $S_t \rightarrow A_t$ we can also interpret this as a map $\pi_t(s_t, a_t)$ which would be the chance that we take action a_t at state s_t or we can have that $\pi_t(s_t)$ is a random variable over the space of actions at time t.

Given a policy π we define the **return** of the policy to be given by:

$R(\pi) := \sum_{t=0}^{T} \gamma^t \mathbf{E}_\pi[r_t(s_t, a_t)]$ Where here we have that the $(s_{t+1}|s_t, a_t) \sim P_t(.|s_t, a_t)$ and we have that $a_t \sim \pi(s_t)$.

The objective of a reinforcement learning agent is to find the policy that maximizes the return: $\arg\max_\pi R(\pi)$

MDPs and Interpretability - Lit. Review

The following papers give various formulations of interpretability in Markov Decision Processes.

1. **Interpretable Machine Learning for Resource Allocation with Application to Ventilator Triage** by Julien Clement. et. al:
    a. Compute optimal decision trees using Dynamic Programming akin to Bellman Recursion.
2. **Interpretable Dynamic Treatment Regimes** by Zhang. et. al:
    a. Compute optimal tree policies directly from trajectory data by performing a Q-learning based inverse regression.
3. **Interpretable MDPs** by Petrik and Luss et. al:
    a. Mixed integer formulation of the interpretability problem for MDPs.

MDPs and Interpretability (continued)

4. **Equivalence notions and model minimization in Markov decision processes** by Givan et. al**:**
    a. Group together states to form a factored MDP; create algorithms which run on the smaller MDPs**.**

5. Genetic Programming for Reinforcement Learning by Hein et. al:
    a. Perform reinforcement learning on existing trajectory data to obtain policies which are represented by basic algebraic equations that are restricted to an adequate complexity.

6. **Iterative Bounding MDPs** by Topin et.al:
    a. Wrap the base MDP to form IBMDP. Modify our current RL algorithm to form modified algorithm which runs on the IBMDP.

Cost of Interpretability

In the **Cost Of Interpretability - Blocked Value Iteration** paper, we define interpretable policies as region-wise constant maps. i.e

Here we have $S = \sqcup_{i=1}^{k} R_i$

$$\pi(s) = \begin{cases} a_1 & \text{if } s \in R_1 \\ a_2 & \text{if } s \in R_2 \\ \vdots & \vdots \\ a_k & \text{if } s \in R_k \end{cases}$$

C_p := differences in the probability transition kernels for different actions
C_r := differences in the reward maps for the different timesteps
V_b := Upper bounds on the value function
s_k for all k \in [0,1,2,..K-1] leader set for the partition

With this in mind, we compute an upper bound on the cost of interpretability:

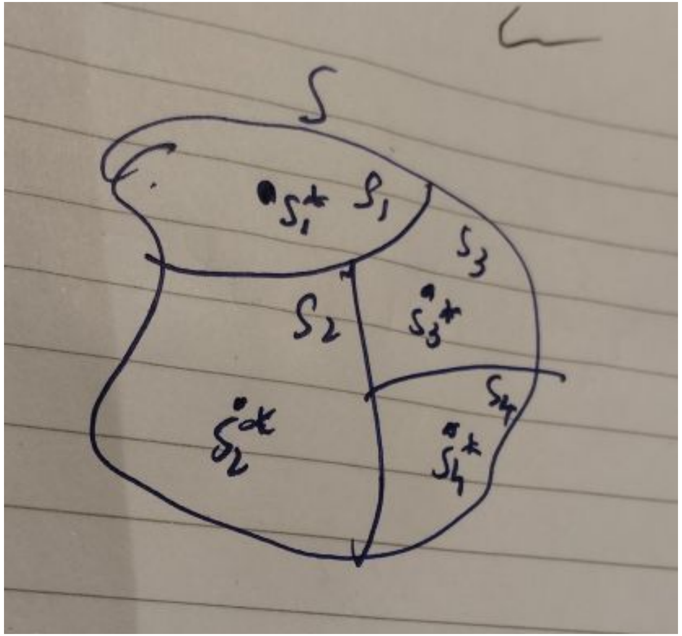
$$\mathcal{C}(\mathcal{P}, S^*) \leq \frac{1}{1-\gamma|S|}(C_r + \gamma V_b C_p) \sum_{k=1}^{K} \frac{\int_{S_k} d(\pi^*(s), \pi^*(s_k))ds}{|S|}$$

Blocked value iteration

<u>Proof sketch</u>: The proof proceeds by partitioning the state space using a blocked value iteration procedure.

This works as follows:
We split the state space into disjoint regions and pick a ‘leader’ in each disjoint subset.

We then perform value iteration in each subset;
We use the rule that we update the actions for each of the elements of the subset with the optimal chosen action for the leader.



$$V_\pi(s) = r(s, \pi^*(s_k)) + \gamma P_{\pi^*(s_k)} V_\pi(s)$$

$$V_{\pi^*}(s) = r(s, \pi^*(s)) + \gamma P_{\pi^*(s)} V_{\pi^*}(s)$$

Update on the general and update on the leader. We subtract these to derive the bound

Introduction

- **Problem statement**: How do we compute optimal interpretable policies in the presence of modeling parameters?
- We propose **VIDTR**, a model based interpretable reinforcement learning algorithm to compute interpretable policies.
- **Input**: Transition probabilities (**P_t**), rewards (**r_t**) for the different timesteps, time horizon T, and max lengths per time horizon (**L_t**) for t \in [T]
- **Output**: Interpretable policies for the different time and length steps

Contributions

- Upper bound on cost of interpretability
- VIDTR - algorithm design
- Computational engineering of the algorithm
- Grid environment results
- Theoretical bounds for the VIDTR
- Optimizations for the VIDTR:
  - Mixed Integer Optimization formulation for the subproblem
  - Q-trees
  - Evolutionary algorithm
- Future work

Methodology : Last timestep

- We perform greedy model based splits based on the Bellman Equation:
  - On the last timestep we know that the optimal policy is given by $\max_a [r_T(s,a)]$
  - Assume we choose a condition R_1 and choose action a_1 , here we optimize greedily over the first split:
  -

$$\Psi_1^{T*} := \min_{R_1, a_1} \Psi(R_1, a_1) = \frac{1}{|S|} \int_{R_1} [(\max_a (r_T(s,a)) - r_T(s, a_1))ds - \eta_T V(R_1) + \rho_T c(R_1)]$$

Here we promote splitting based on the constant \eta and we penalize the split based on the constant \rho. We choose action a_1 in region R_1; else we choose the best possible action

Assume now that $G_1 = R_1$; The optimization problem is then given by

$$\Psi_2^T := \min_{R_2, a_2} \Psi(R_2, a_2) := \frac{1}{|S|} \int_{R_2 - G_1} [\max_a [r_T(s,a)] - r_T(s, a_1)]ds - \eta_T V(R_2 - G_1) + \rho_T c(R_2)$$

For the lengthstep $l < l_t^*$ and timestep T, we have the optimization given by

$$\Psi_l^{T*} := \min_{R_l, a_l} \Psi_l^T(R_l, a_l) = \int_{R_l - G_{l-1}} [\max_a (r_T(s,a)) - r_T(s, a_l)]ds - \eta_T V(R_l - G_{l-1}) + \rho_T c(R_l)$$

Timestep t < time horizon T

$$\text{Here we do} \max_a (r_T(s,a)) \leftrightarrow \max_a (r_t(s,a) + \gamma P_t^a V_{t+1}(s))$$

Optimizing equation is given by:

$$\Psi_t(R_l, a_l) := \int_{R_l - G_{l-1}} [\max_a (r_t(s,a) + \gamma P_t^a V_{t+1}(s)) - (r_t(s,a_l) + \gamma P_t^a V_{t+1}(s))] ds - \eta_t V(R_l) + \rho_t c(R_l)$$

For $l = l_{max}$,we need to just pick the optimal action; the equation here is

$$\Psi_t^{l*} := \min_{a_l} \Psi_t(a_l) := \int_{S - G_{l_{max}-1}} [\max_{a_l} (r_t(s,a) - r_t(s,a_l)) + \gamma P_t^a V_{t+1}(s) - P_t^{a_l} V_{t+1}^I(s)]$$

Value iteration computation

$$V_t(s) = \max_a [r_t(s,a) + \gamma P_t^a V_{t+1}(s)]$$

$$V_t^I(X_{it}) = \begin{cases} r_t(X_{it}, a_1^t) + \gamma P_t^{a_1^t} V_{t+1}^I(X_{it}) \text{ if } X_{it} \in R_1^t \\ r_t(X_{it}, a_2^t) + \gamma P_t^{a_2^t} V_{t+1}^I(X_{it}) \text{ if } X_{it} \in R_2^t \\ \cdots \\ r_t(X_{it}, a_{l_t}^t) + \gamma P_t^{a_{l_t}^t} V_{t+1}^I(X_{it}) \text{ if } X_{it} \in R_{l_t}^t \end{cases}$$

Interpretable value function computation

Computational engineering

We describe computational hacks we employ to aid the calculation of the objective in the VIDTR algorithm.

1. **DisjointBoxUnion** : The evaluation of the objective in the VIDTR, involves the calculation of $\mathcal{I}[X_t^l \in R - G_t^l]$, where we check if a point belongs in the region we iterate on subtracted from the remaining space.

   For doing this, we must be able to perform algebraic manipulations on the sets picked, remaining space and so on. To do so, we see that the space is given by a **disjoint union of boxes**.

   In this class, we can do
    - Unions
    - Subtraction
    - Intersections
    - Function integration between the different elements of the set

2. Memoization

At each timestep in the evaluation of the optimal policy we recursively store the value functions:

$$V_t(s) = \max_a [r_t(s,a) + \gamma P_t^a V_{t+1}(s)]$$

Storing this functionally at each timestep is inefficient since the evaluation of V_1 would involve the evaluation of all the terms till the last timestep

Hence we store it as dictionary first with keys being values in the domain and values being chosen from the range

$$V_t(s) := \max_a [r_t(s,a) + \gamma P_t^a V_{t+1}(s)]$$
$$Dict(V_t)(s) = V_t(s) \forall s \in S$$
$$\text{Redefine } V_t(s) = Dict(V_t)(s)$$
$$\text{Here } V_t \text{ does not depend on } V_{t+1}$$

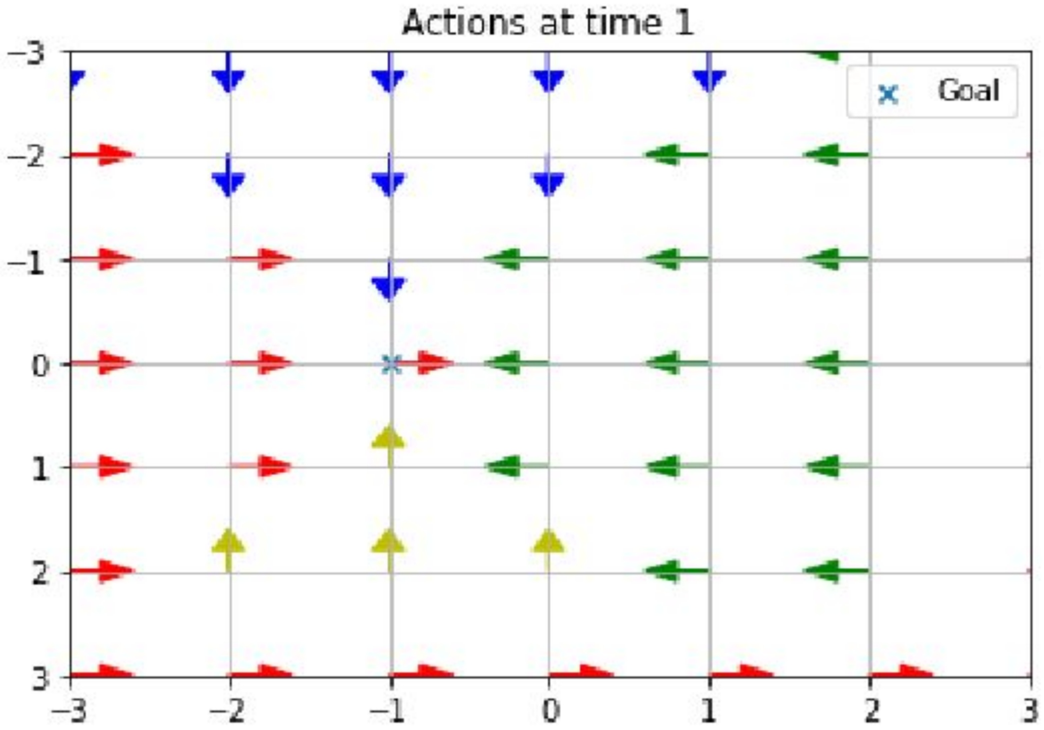
Proof of concept setup

Assume we have a setup as follows:

Here we have a goal denoted by an **X**

Objective of an agent is to find that direction which takes it to the goal in the shortest time

The optimal policies are pictured as in the right picture



Parameters - environmental and algorithmic

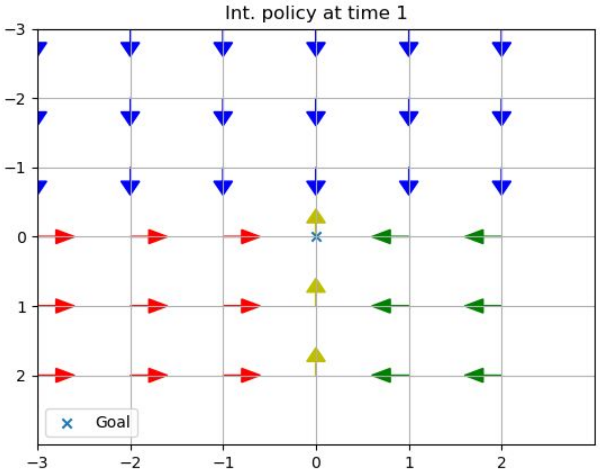
- The state space in this setting would be the 2D points on a grid
- The actions here would be ['up', 'down', 'left', 'right']
- The transition map is given by $P(s'|s,a) := \mathcal{I}[s+a=s']$, for all timesteps
- The rewards are given by $r(s,a) := \dfrac{rc}{f||s+a-g||+1}$, here
    - rc : reward coefficient
    - f : friction coefficient
- Other parameters in the algorithm involve **\eta** and **\rho**; the volume promotion and the complexity penalization constant respectively.

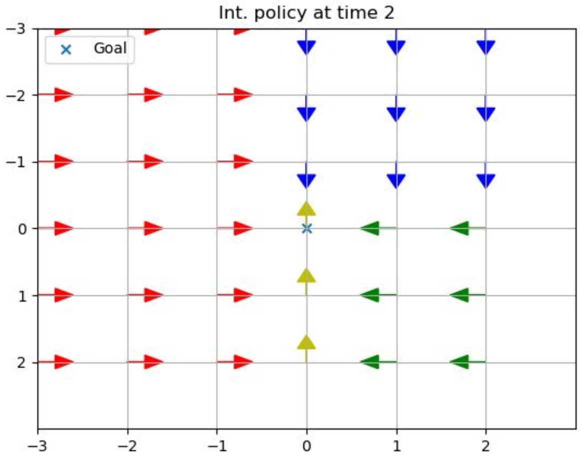
ime	Length	Centre	Lengths	Bellman_error	Constant_error	Complexity_error	Total_error	Integration_method	Conditions_strir	Action
4	0	[[-2.5 0.]]	[[1. 6.]]	0	-1.5	0.6	-0.9	integrate_static	all	[1. 0.]
4	0	[[-2.5 0.]]	[[1. 6.]]	1.002627541	-1.5	0.6	0.102627541	integrate_static	all	[-1. -0.]

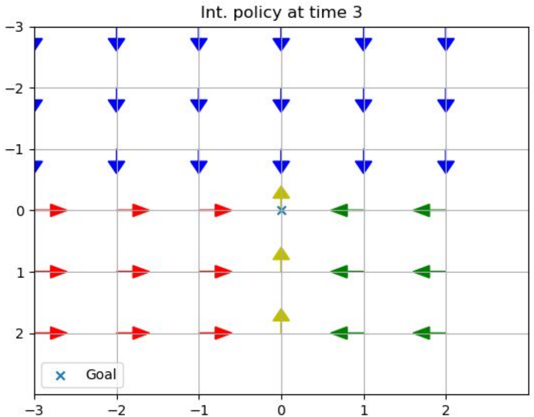
- For each time and length instance of the optimization problem at time t, length l and action a, we store an excel sheet noting the contribution of the different error terms with the various parameters.

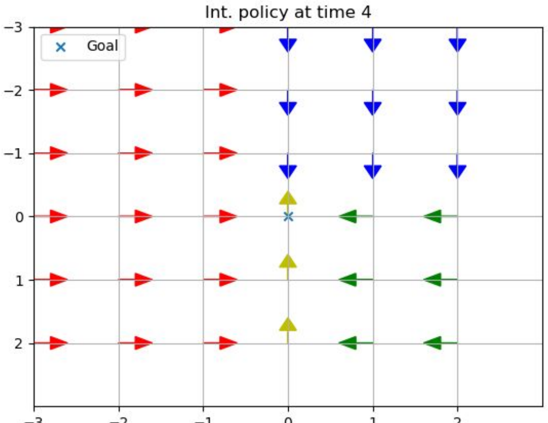
If the DBU we were interested in at a given time and length step is not picked, we retune the parameters as a way to do **hyperparameter tuning**

Int. policies at the different timesteps









Hyperparameters chosen

- **etas** = [0.05 * 9 , 0.05 * 8, 0.05 * 7, 0.05*6 , 0.05 * 5]

rhos = 0.1
reward_coeff = 4.5
friction = 2.0

VIDTR bounds assumptions

1) Lipschitz continuity in the second coordinate of the rewards

$$|r_T(s,a) - r_T(s,b)| \leq C_r d_A(a,b)$$
$$\forall s \in S, (a,b) \in \mathcal{A}^2$$

2) The action space is bounded

$$d_A(a,b) \leq D$$

3) Smoothness in the transition kernels

$$|P_t(s'|s,a) - P_t(s'|s,b)| \leq C_p d_A(a,b)$$

4) Value function bounds

$$|V_{t+1}(s)| \leq W_{t+1} \forall s \in S$$

5) **State space bounds:** $S = \Pi_{i=1}^{D}[a_i, b_i]$, then we see

$$|b_i - a_i| \leq |S|^{\frac{1}{d}}$$

Bounds for VIDTR

For timesteps t and lengthsteps l_t, the optimizing function is Ψ^*_{tl}, if $G_{t,l-1} = \cup_{j=1}^{l-1} R_{tj}$; which gives

$$\begin{aligned}\Psi^*_{tl} \leq \min_{R_l}(C_r D|R_l - G_{t,l-1}| + \gamma C_p D \frac{1-\gamma^{T-t}}{1-\gamma}\bar{r}|R_l - G_{t,l-1}|)+ \\ (C_r D + 2\gamma\bar{r})\frac{1-\gamma^{T-t}}{1-\gamma}|R_l - G_{t,l-1}|+ \\ 2\gamma^2\bar{r}\frac{1-\gamma^{T-t-3}[\gamma^3 - 1(T-t-2)(1+\gamma^2(1+\gamma))]}{(1-\gamma)^2}|R_l - G_{t,l-1}|+ \\ -\eta|R_l - G_{t,l-1}| + \rho_t c(R_l)\end{aligned}$$

We also see that the following property is true for the VIDTR:

$$V_t(s) \geq V_t^I(s) \forall s \in S, \forall t \in [T]$$

VIDTR objective reformulations

The main objective in the VIDTR optimization problem for the different time and length-steps is given by:

Optimizing equation is given by:

$$\Psi_t(R_l, a_l) := \int_{R_l - G_{l-1}} [\max_a(r_t(s,a) + \gamma P_t^a V_{t+1}(s)) - (r_t(s,a_l) + \gamma P_t^a V_{t+1}(s))] ds - \eta_t V(R_l) + \rho_t c(R_l)$$

In the next subsections, we list different ways to solve the above objective:

1) MIO formulation
2) Q-trees
3) Genetic algorithm

MIO formulation for the VIDTR

We formulate the main optimization problem at timestep t and lengthstep l as a mixed integer optimization problem.

We see that the objective can be written as a MIO with coefficients:

$$\sum_{i=1}^{N} z_i U_{iat}^{l}$$

With constraints $x_{ij} \geq a_j - M(1 - z_i)$

M >> 0 $\quad x_{ij} \leq b_j + M(1 + z_i)$

In other words: $z_i := \mathcal{I}[X_{it} \in [\vec{a}, \vec{b}]]$

$$U_{iat}^{l} := [\max_{\alpha}[r_t(X_{it}, \alpha) + \gamma P_t^{\alpha} V_{t+1}(X_{it})] - [r_t(X_{it}, a) + \gamma P_t^{a} V_{t+1}^{I}(X_{it})] - \eta_t]\mathcal{I}[X_{it} \notin G_t^{l-1}]$$

Geometric interpretation of the above problem

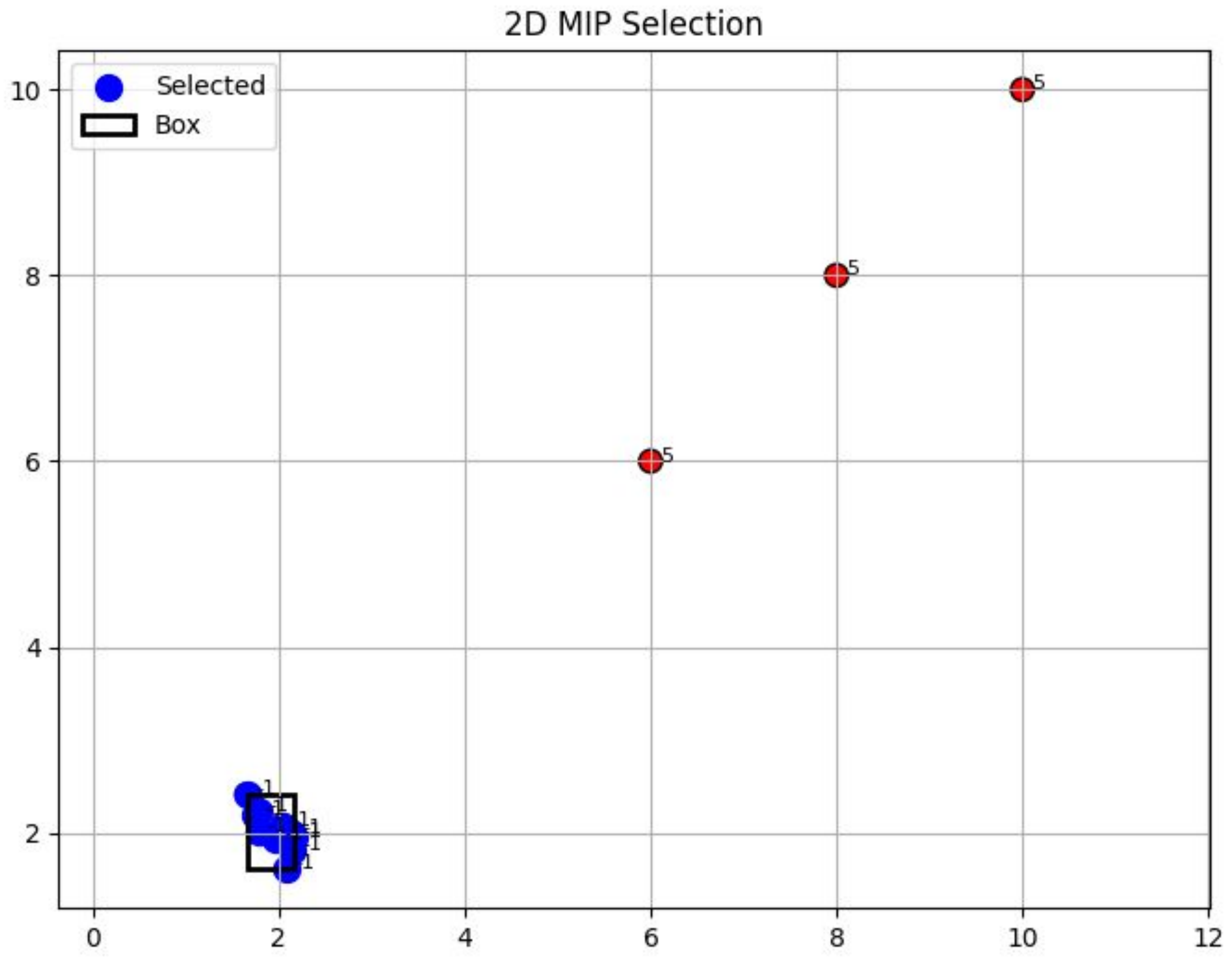
$$\text{Solve the following equation :}$$

$$((\hat{a}_1, \hat{b}_1), (\hat{a}_2, \hat{b}_2), \ldots, (\hat{a}_q, \hat{b}_q)) :=$$

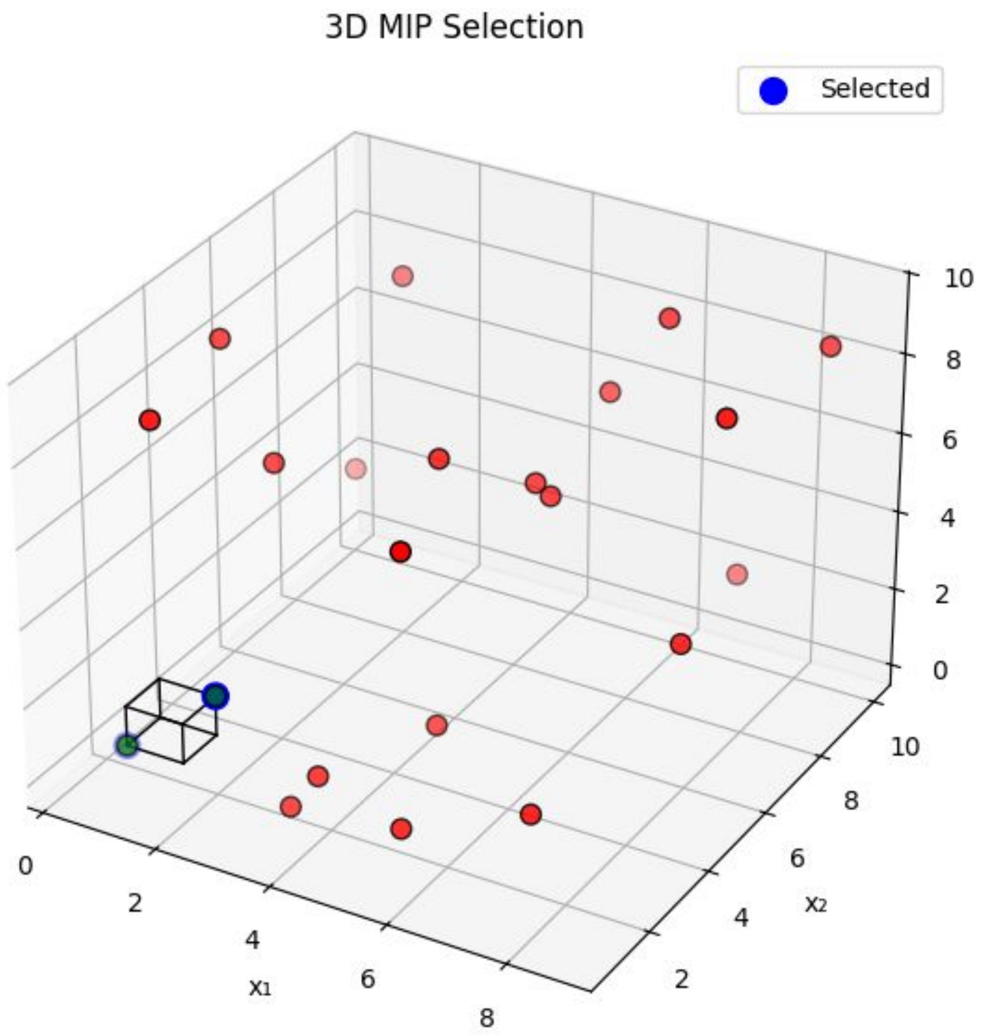
$$\arg_{a_1, a_2, \ldots, a_q} \min_{b_1, b_2, \ldots, b_q} \sum_{i=1}^{n} U_i \mathcal{I}[(X[i,1] \in [a_1, b_1]), (X[i,2] \in [a_2, b_2]), (X[i,3] \in [a_3, b_3]) \ldots (X[i,q] \in [a_q, b_q])]$$

The above MIP can geometrically be framed as: Given N points in q-space each with a value U_i associated to it; How would we choose a box in q-space such that the sum of the U-values for all the points in the box is minimized?

Here we pick the box with a collection of -1 values



3D MIP problem



Q-trees

VIDTR optimization problem VERSUS **IDTR** optimization problem

$$[(\hat{\sigma}_1, \hat{\tau}_1), (\hat{\sigma}_2, \hat{\tau}_2), ..., (\hat{\sigma}_q, \hat{\tau}_q)] :=$$

$$\arg_{(\sigma_1,\tau_1),...,(\sigma_q,\tau_q)} \min \sum_{i=1}^{n} U_{i,a} I[\sigma_1 \leq X_{ii_1} < \tau_1, \\ \sigma_2 \leq X_{ii_2} < \tau_2, \\ ..., \\ ..., \\ \sigma_q \leq X_{ii_q} < \tau_q]$$

We provide a data structure to solve the above problem termed as Q-trees

A Q-tree is a tree with Q leaves per node

IDTR Two variable optimization

$$(\hat{\tau}, \hat{\sigma}) := \arg_{\tau,\sigma} \min \sum_{i=1}^{N} U_{ia} \mathcal{I}[X_{ij} \leq \tau, X_{ij} \leq \sigma]$$

The authors of the IDTR paper come up with a tree-based formalism to solve the above

We extend the same logic to the interval setting and with Q-conditions

In the one-dimensional setting, this is achieved by Kanade's algorithm

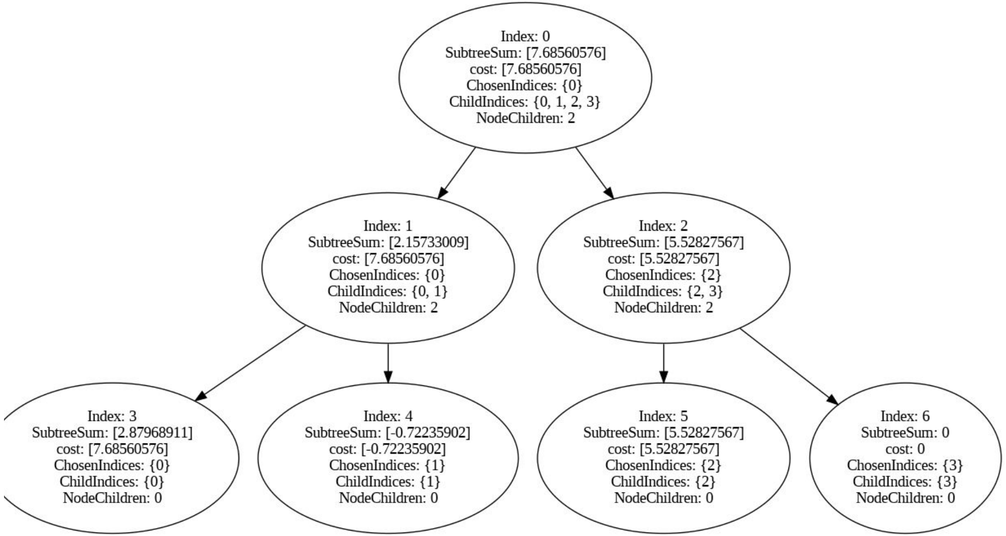
Kadane’s algorithm - One dimensional VIDTR;

Given points $[X_1, X_2, ..., X_N]$ in 1 dimensional space lying in a large interval and having functional values $[U(X_1), U(X_2), ..., U(X_N)]$ how do we determine a, b such that the sum of the U_values for all the $(X_i)_{i=1}^{N}$ points lying inside [a,b] is minimal?

Brute force: **O(N^2)** time; **Kadane’s method : O(N)** time if array is sorted; O(NlogN) otherwise

```python
def max_subarray(numbers):
    """Find the largest sum of any contiguous subarray."""
    best_sum = float('-inf')
    current_sum = 0
    for x in numbers:
        current_sum = max(x, current_sum + x)
        best_sum = max(best_sum, current_sum)
    return best_sum
```

Q-trees cont.



Right hand side; we provide an extension of the Zhang interval estimation problem to the Q-case

The Q-trees have the following operations in them:

1) Create **rank operators** r_1,...r_q where we take the rank with respect to some X[i;i_1], X[i;i_2],...,X[i;i_q]

2) **Insert** element X_i to leaf with location

r_{i_q} (r_{i_{q-1}} (... (r_{i_1}(i)...))))

3) Perform Kadane's algorithm at each node to determine the thresholding sum and the chosen indices

4) Each node has attributes:

a. **Sum** - sum of all child nodes
b. **Child indices** - Union of all child indices
c. **Thresholding sum** - Min(ai.tc, ai.ss + a(i+1).tc , ai.ss + a(i+1).ss +.a(i+2).tc , …, ai.ss + a(i+1).ss + … + aq.tc)

d. Chosen indices : The index from the above onwards; if we pick i, then the above is [i,i+1,...,j]

Genetic algorithm - overview

Genetic algorithm is an approximation scheme borrowed from the biology literature to determine the arg max of a function f over a family F. In particular,

Given a family F and a fitness function f over F, how do we find the arg max of f over F?

1. **Initialization** - Randomly choose k elements from F
2. For t in range(T):
   a. **Evaluate** the fitness of each of the k elements
   b. **Pick** two elements from the picked elements with a probability weighting proportional to its fitness
   c. Perform a **crossover** of the elements to generate new children
   d. For the last set of children picked, perform a **mutation** to re-introduce the children back into the family

Ideas borrowed from - **Evolution** in biology

GA for the subproblem

Here the population space is the space of all conditions (R):

Recall an abstract condition looks like - for indices [i_1, i_2,.. ,i_k] in [q]

$$[(a_{i_1}, b_{i_1}) \times (a_{i_2}, b_{i_2}) \times (a_{i_3}, b_{i_3}) ... \times (a_{i_k}, b_{i_k})]$$

The **fitness** evaluation map is given by

$$R \mapsto \int_{R-S} (\max_{\alpha}[r_t(s,\alpha) + \gamma P_t^{\alpha} V_{t+1}(s)] - [r_t(s,a) + \gamma P_t^{a} V_{t+1}(s)] - \eta) ds + \rho c(R)$$

The **crossover** operation is:

$$[(a_{i_1}, b_{i_1}) \times (a_{i_2}, b_{i_2}) \times (a_{i_3}, b_{i_3}) ... \times (a_{i_k}, b_{i_k})] \propto [(a_{j_1}, b_{j_1}) \times (a_{j_2}, b_{j_2}) \times (a_{j_3}, b_{j_3}) ... \times (a_{j_l}, b_{j_l})] :=$$
$$[(a_{i_1}, b_{i_1}) \times (a_{i_2}, b_{i_2}) \times ... \times (a_{j_{r_2}}, b_{j_{r_2}}) ... \times (a_{i_k}, b_{i_k})] \times [(a_{j_1}, b_{j_1}) \times (a_{j_2}, b_{j_2}) \times ... (a_{i_{r_1}}, b_{i_{r_1}}) ... \times (a_{j_l}, b_{j_l})]$$

The **mutation** here is:

$$\mathcal{M}[(a_{i_1}, b_{i_1}) \times (a_{i_2}, b_{i_2}) \times (a_{i_3}, b_{i_3}) ... \times (a_{i_k}, b_{i_k})] :=$$
$$[(a_{i_1}, b_{i_1}) \times (a_{i_2}, b_{i_2}) \times (a_{i_3}, b_{i_3}) ... \times (a_{i_r} + \epsilon_1, b_{i_r} + \epsilon_2) \times ... \times (a_{i_k}, b_{i_k})]$$

Next steps

1. Simulated and real world experiments for the VIDTR - **scaling**

2. Computational experiments and comparisons for the optimizations

3. Knowledge of **data generation** mechanism:

Given the **causal SCM** generating the MDP, how do we derive interpretable policies?

4. **Pruning** methods while we search for the optimal condition

Code for the VIDTR working on grid data and the optimizations given in: https://github.com/DeepakBadarinath?tab=repositories