CSE 691 Spring 2020 Recitation on Rollout

Sushmita Bhattacharya

Arizona State University

sbhatt55@asu.edu

April 15, 2020

Overview

1. Multiprocessor parallelization: Pipeline Problem
2. Multiagent parallelization: Spider and Fly Problem

Palatalization in Rollout and Approximate PI

- Q-factor parallelization: At the current state x, one-step lookahead/rollout does a separate Q-factor calculation for each control $u \in U(x)$. These calculations are decoupled and can be executed in parallel.
- Monte Carlo parallelization: Each of the Q-factor calculations involves a Monte Carlo simulation when the problem is stochastic. Monte Carlo simulation can be parallelized.
- Multiprocessor parallelization: Use a state space partition, and execute separate (but coupled) value and policy approximations on each subset in parallel (Pipeline problem).
- Multiagent parallelization: When the control has m components, $u = (u_1, \ldots, u_m)$ the lookahead minimization at x involves the computation of as many as n^m Q-factors, where n is the max number of possible values of u_i. This method reduces the computation dramatically (to nm). (Spider and fly problem).

Palatalization in Rollout and Approximate PI

- Q-factor parallelization: At the current state x, one-step lookahead/rollout does a separate Q-factor calculation for each control $u \in U(x)$. These calculations are decoupled and can be executed in parallel.
- Monte Carlo parallelization: Each of the Q-factor calculations involves a Monte Carlo simulation when the problem is stochastic. Monte Carlo simulation can be parallelized.
- Multiprocessor parallelization: Use a state space partition, and execute separate (but coupled) value and policy approximations on each subset in parallel (Pipeline problem).
- Multiagent parallelization: When the control has m components, $u = (u_1, \ldots, u_m)$ the lookahead minimization at x involves the computation of as many as n^m Q-factors, where n is the max number of possible values of u_i. This method reduces the computation dramatically (to nm). (Spider and fly problem).

Palatalization in Rollout and Approximate PI

- Q-factor parallelization: At the current state x, one-step lookahead/rollout does a separate Q-factor calculation for each control $u \in U(x)$. These calculations are decoupled and can be executed in parallel.
- Monte Carlo parallelization: Each of the Q-factor calculations involves a Monte Carlo simulation when the problem is stochastic. Monte Carlo simulation can be parallelized.
- Multiprocessor parallelization: Use a state space partition, and execute separate (but coupled) value and policy approximations on each subset in parallel (Pipeline problem).
- Multiagent parallelization: When the control has m components, $u = (u_1, \ldots, u_m)$ the lookahead minimization at x involves the computation of as many as n^m Q-factors, where n is the max number of possible values of u_i. This method reduces the computation dramatically (to nm). (Spider and fly problem).

Palatalization in Rollout and Approximate PI

- Q-factor parallelization: At the current state x, one-step lookahead/rollout does a separate Q-factor calculation for each control $u \in U(x)$. These calculations are decoupled and can be executed in parallel.
- Monte Carlo parallelization: Each of the Q-factor calculations involves a Monte Carlo simulation when the problem is stochastic. Monte Carlo simulation can be parallelized.
- Multiprocessor parallelization: Use a state space partition, and execute separate (but coupled) value and policy approximations on each subset in parallel (Pipeline problem).
- Multiagent parallelization: When the control has m components, $u = (u_1, \ldots, u_m)$ the lookahead minimization at x involves the computation of as many as n^m Q-factors, where n is the max number of possible values of u_i. This method reduces the computation dramatically (to nm). (Spider and fly problem).

Overview

Pipeline Repair Problem

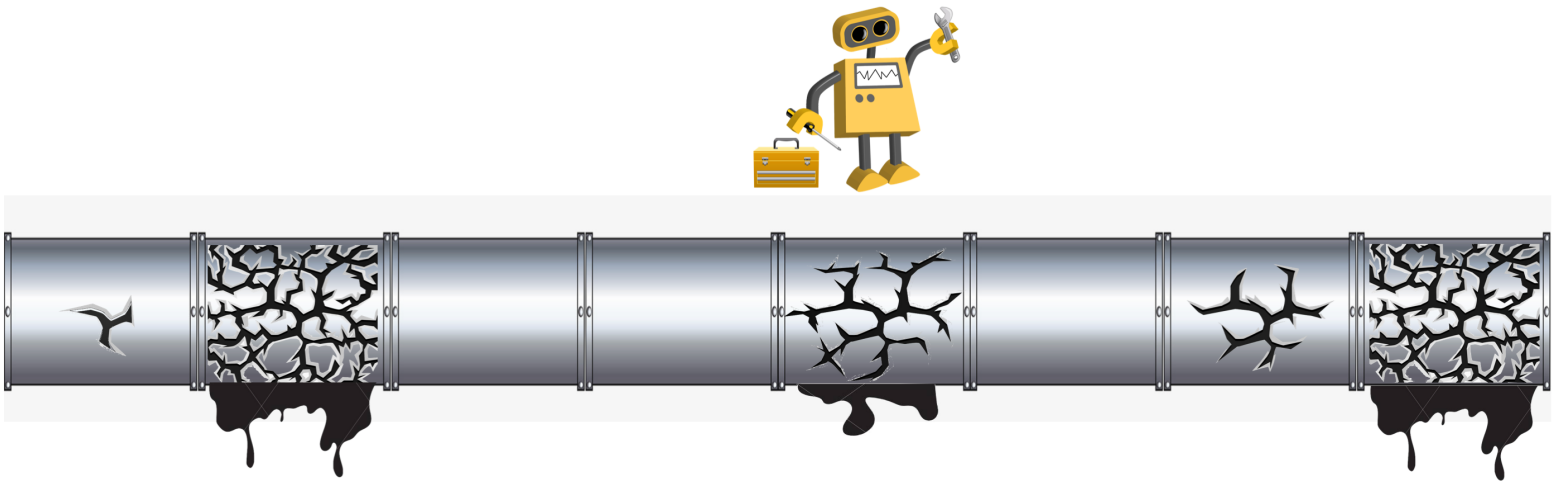


Figure: Segment of pipeline with different level of ruptures.

The problem deals with

- Partially observable space. Size of the state space ($\geq 2^{26}$ for 6x6 pipeline grid).
- Progressively worse damage levels (MDP).
- Finite control space.

Pipeline Repair Problem

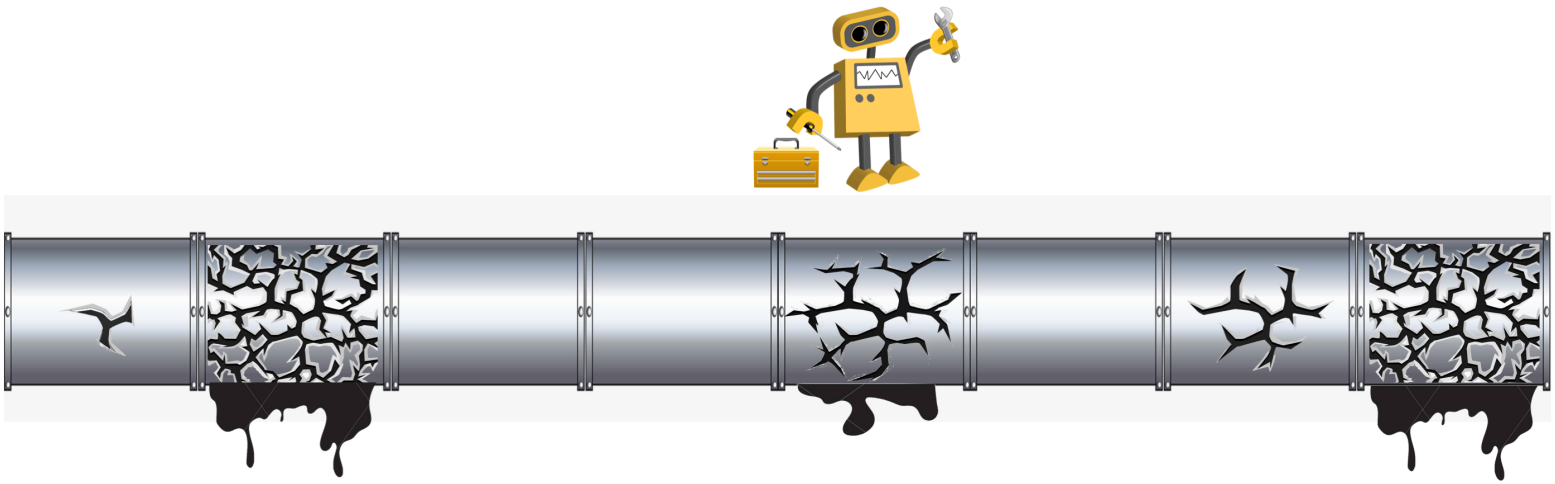


Figure: Segment of pipeline with different level of ruptures.

The problem deals with

- Partially observable space. Size of the state space ($\geq 2^{26}$ for 6x6 pipeline grid).
- Progressively worse damage levels (MDP).
- Finite control space.

Pipeline Repair Problem

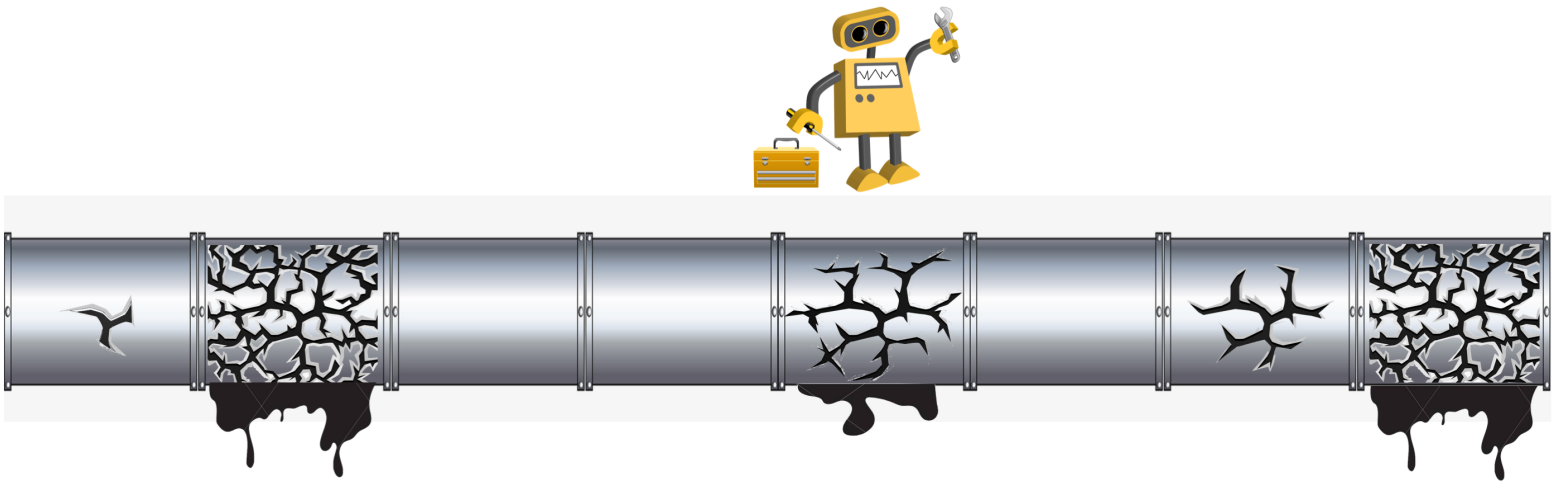


Figure: Segment of pipeline with different level of ruptures.

The problem deals with

- Partially observable space. Size of the state space ($\geq 2^{26}$ for 6x6 pipeline grid).
- Progressively worse damage levels (MDP).
- Finite control space.

Pipeline Repair Problem

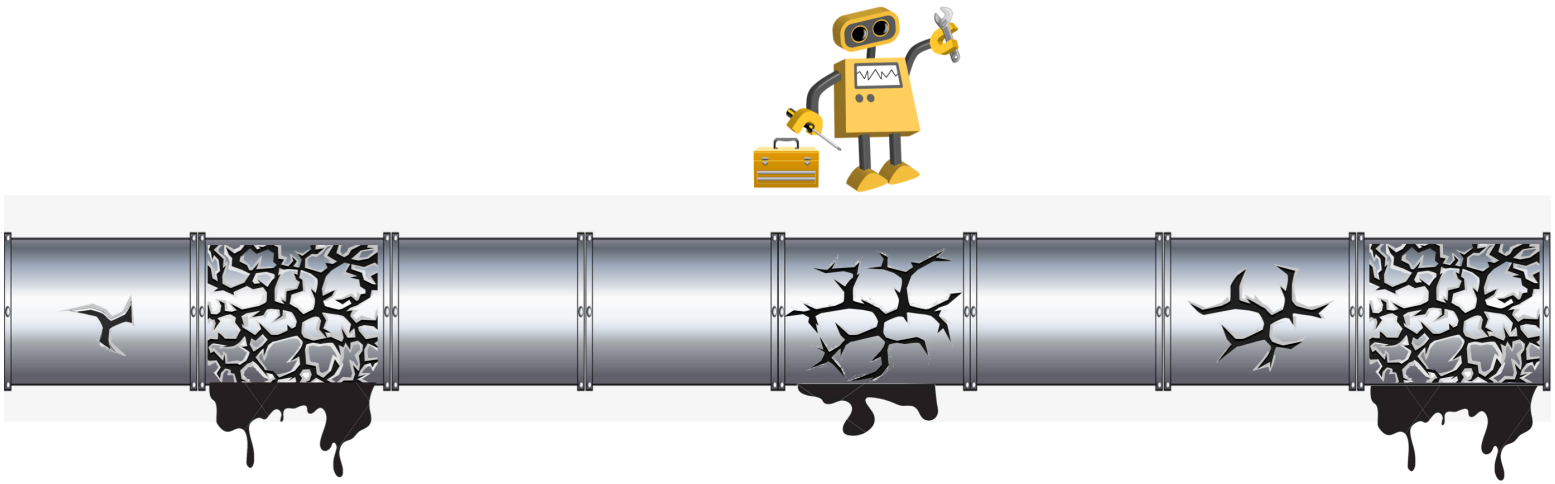


Figure: Segment of pipeline with different level of ruptures.

The problem deals with

- Partially observable space. Size of the state space ($\geq 2^{26}$ for 6x6 pipeline grid).
- Progressively worse damage levels (MDP).
- Finite control space.

Truncated Rollout Scheme

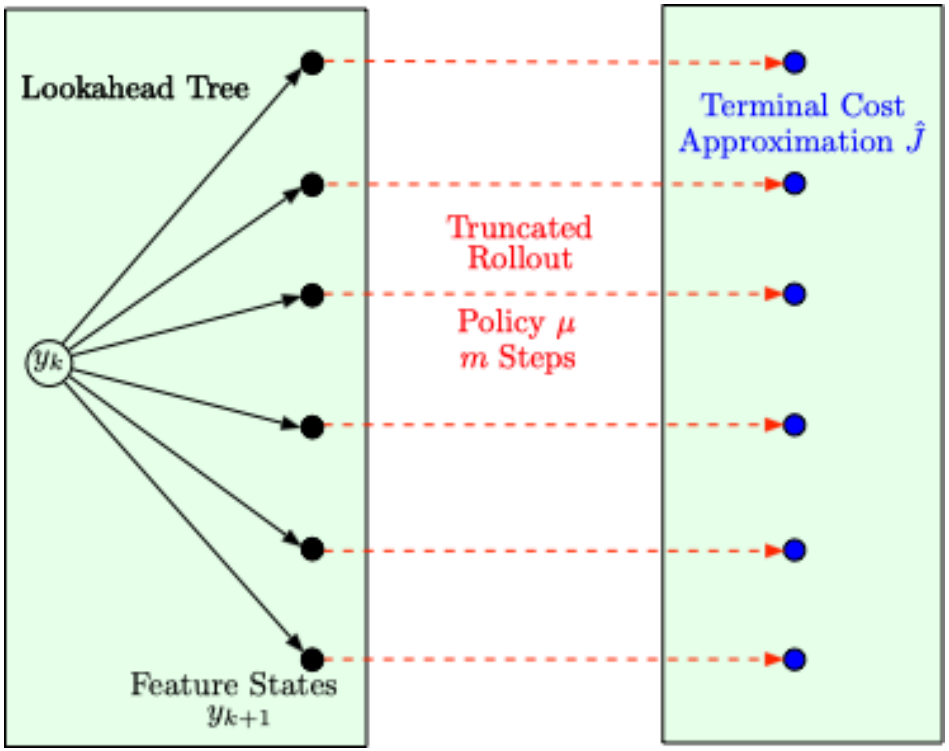


Figure: One-step lookahead is followed by a few step application of the base policy μ, and terminal cost approximation $\hat{J}$. [1]

Approximate PI

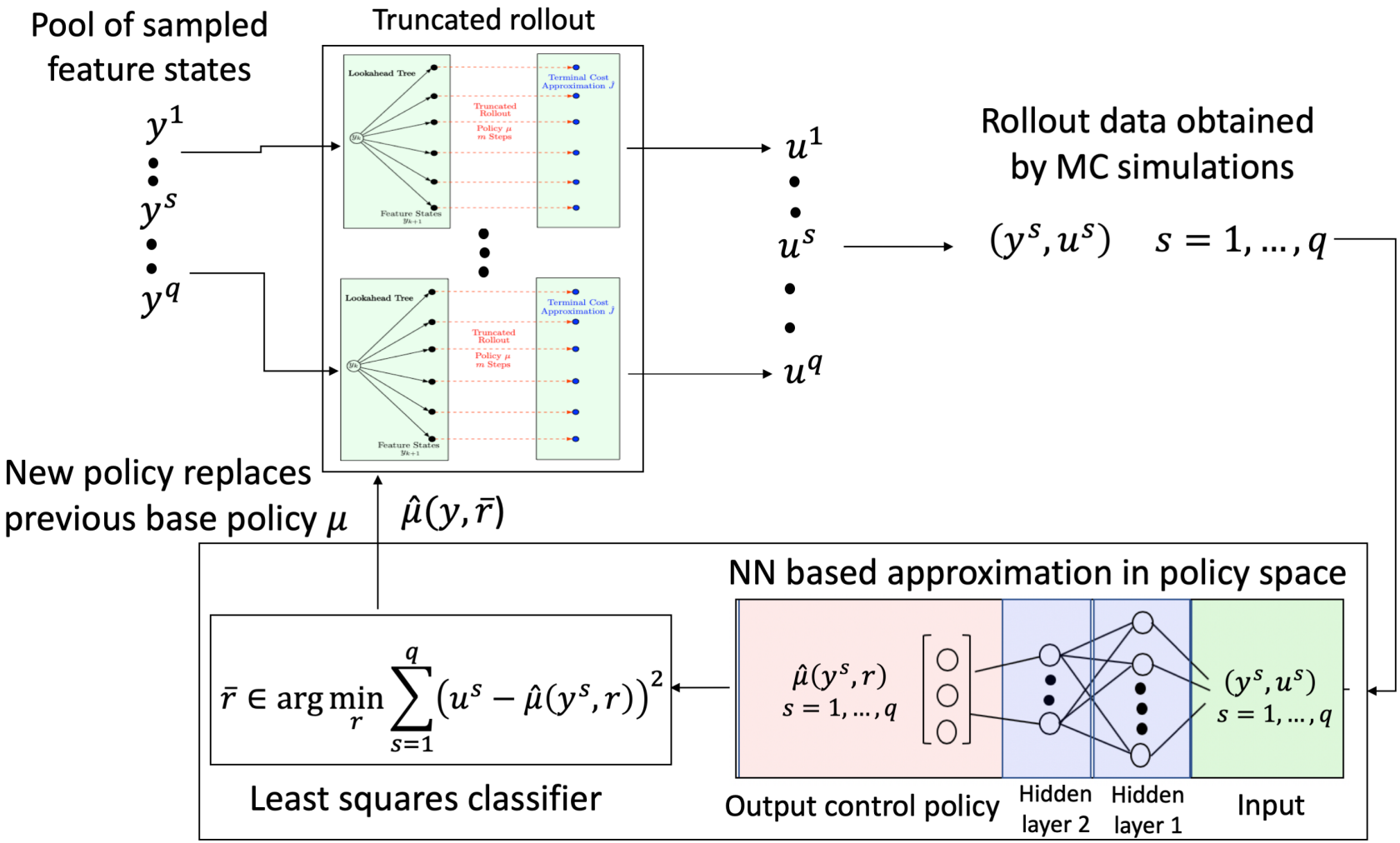


Figure: Approximate PI scheme based on rollout, approximation in policy space and optionally approximation in value space (if truncated rollout is used). [1]

Partitioning of State Feature Space

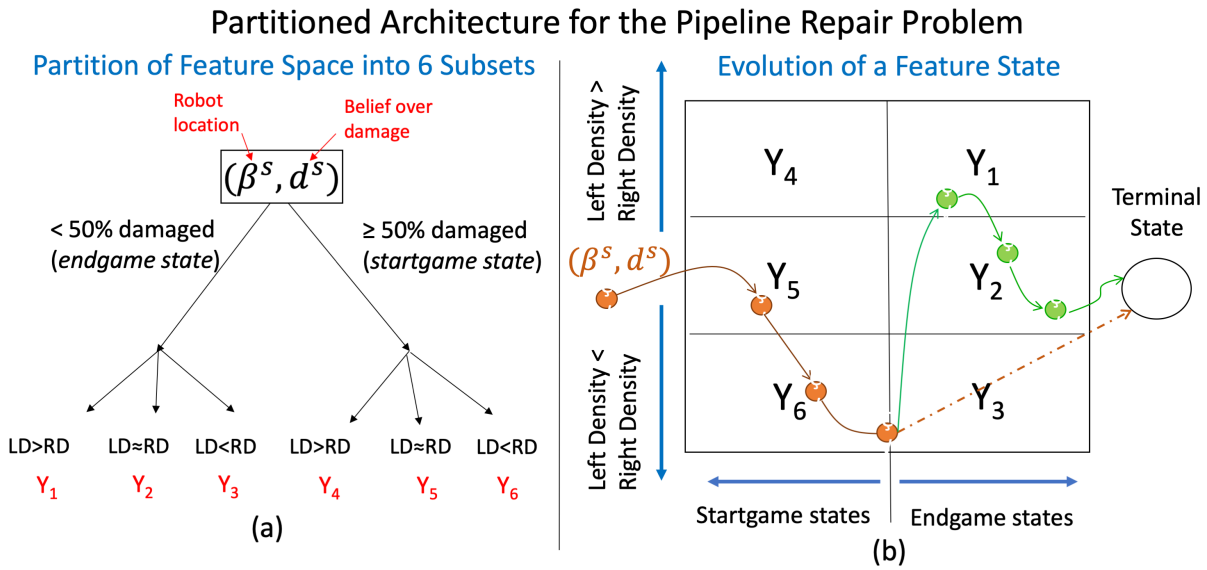


Figure: Partition of the feature space into subsets for the pipeline problem. [1]

A single neural net requires a large number of training samples. Whereas,

- Partitioning improves **computation time**. Ideal for distributed setting, namely **multi-processor parallelism** in distributed RL.
- Partitioning provides better state feature space **exploration**. One way to deal with classic exploration vs exploitation dilemma.

Partitioning of State Feature Space

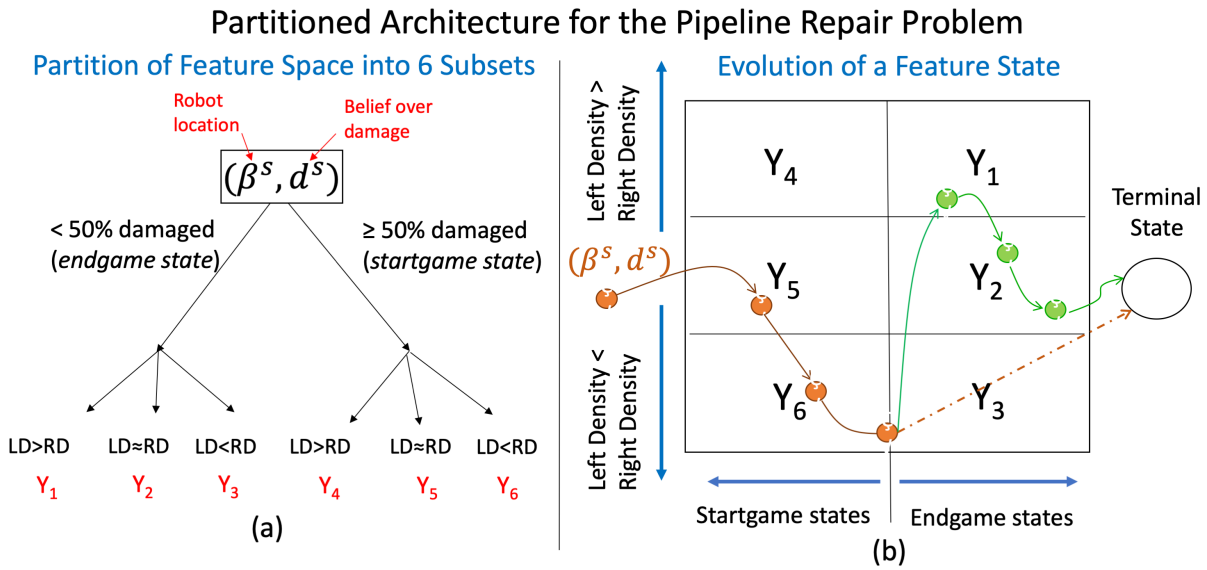


Figure: Partition of the feature space into subsets for the pipeline problem. [1]

A single neural net requires a large number of training samples. Whereas,

- Partitioning improves **computation time**. Ideal for distributed setting, namely **multi-processor parallelism** in distributed RL.
- Partitioning provides better state feature space **exploration**. One way to deal with classic exploration vs exploitation dilemma.

Partitioning of State Feature Space

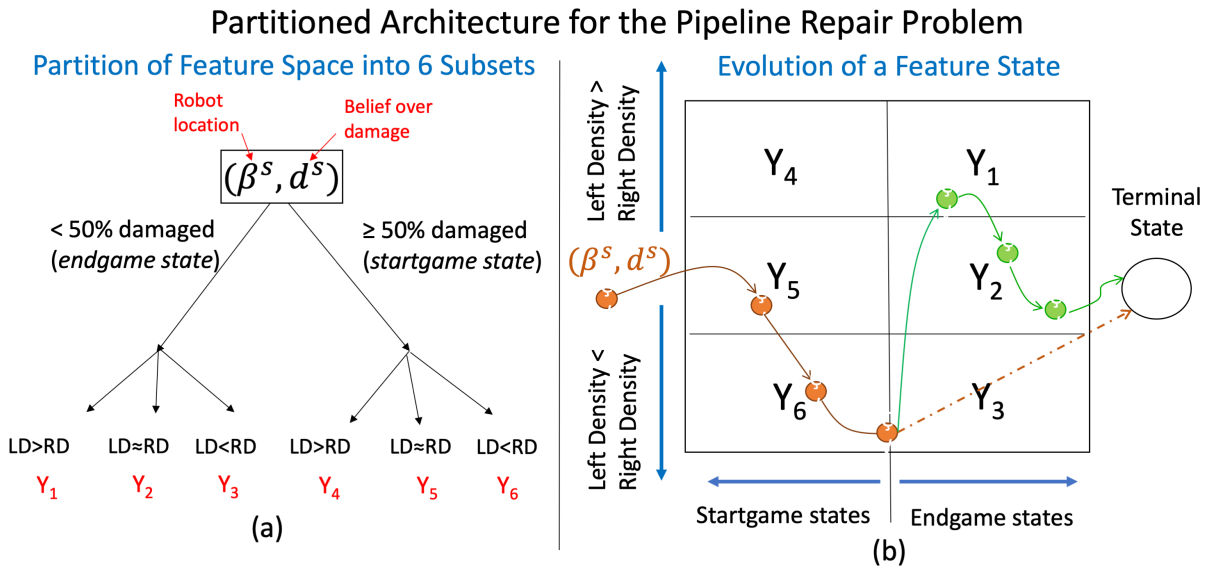


Figure: Partition of the feature space into subsets for the pipeline problem. [1]

A single neural net requires a large number of training samples. Whereas,

- Partitioning improves **computation time**. Ideal for distributed setting, namely **multi-processor parallelism** in distributed RL.
- Partitioning provides better state feature space **exploration**. One way to deal with classic exploration vs exploitation dilemma.

Partitioning of State Feature Space

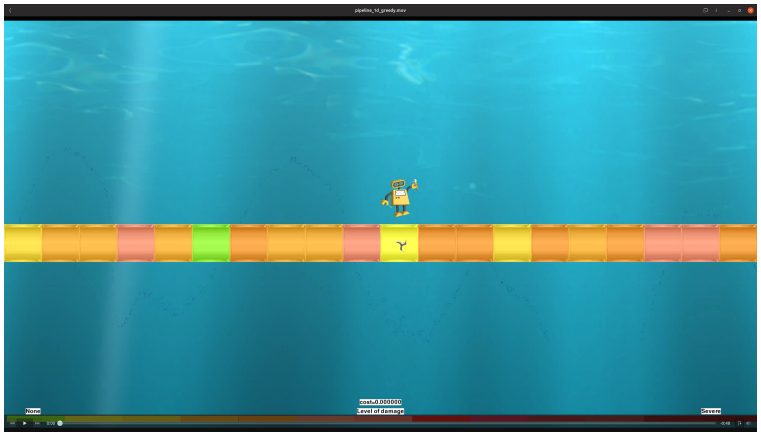


Figure: Start Game, almost equal damage in both sides

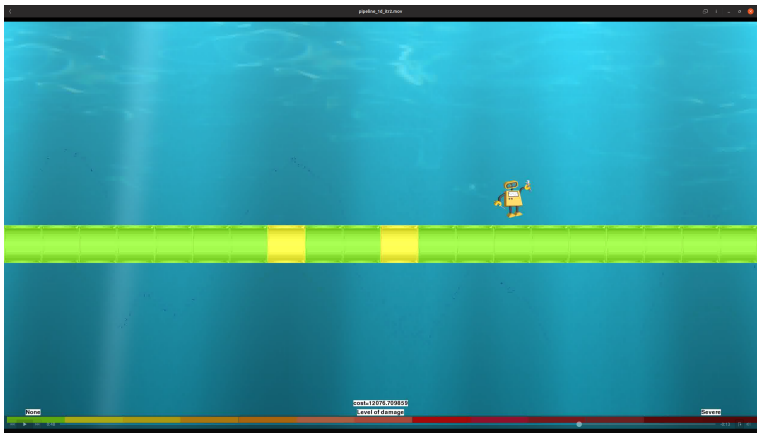


Figure: End Game, left density>right density

Comparison of Different Architectures

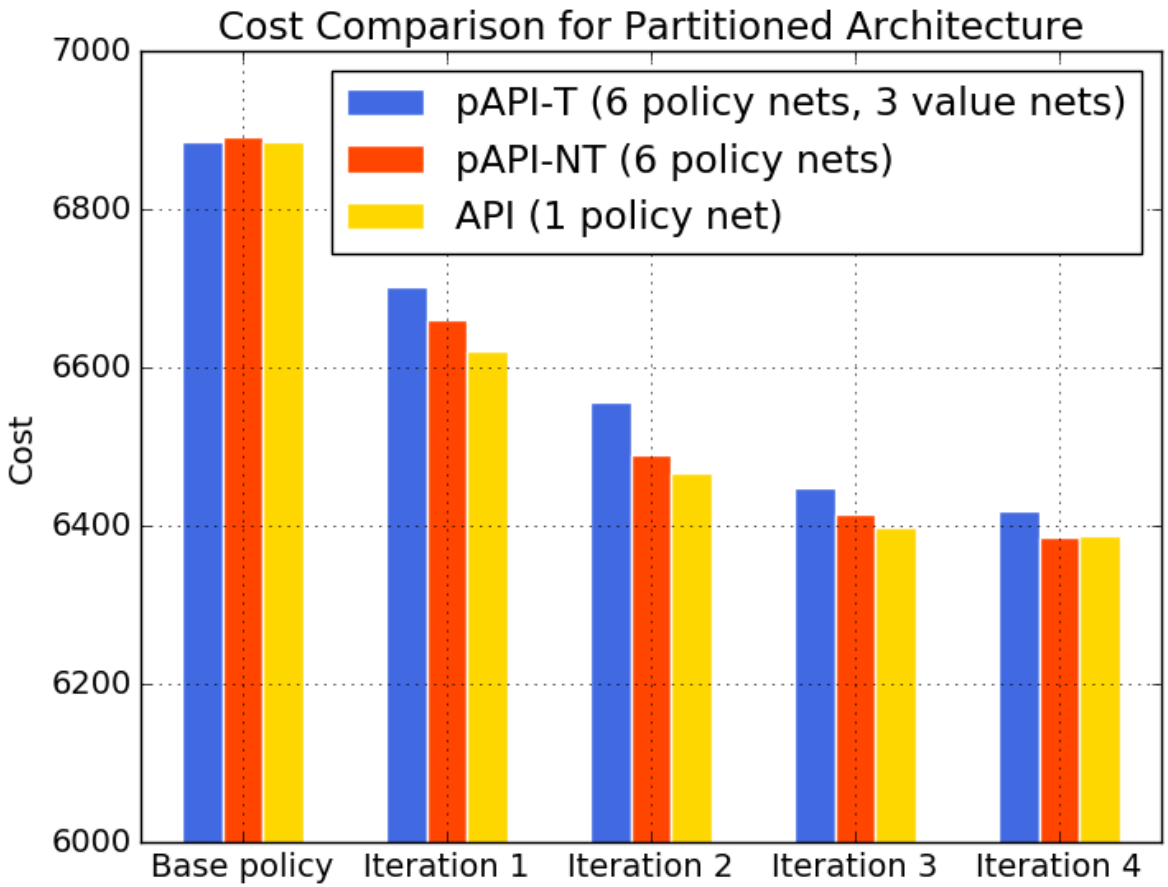
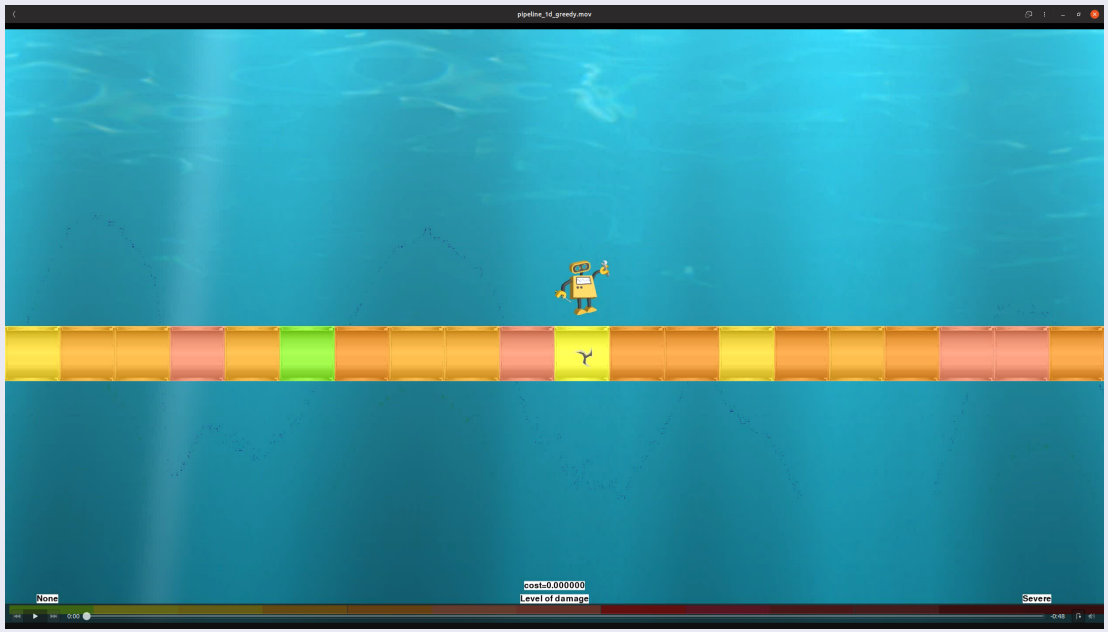


Figure: Performance comparison of API, where pAPI-T and pAPI-NT use partitioning, with and without truncation respectively. [1]

Pipeline Visualization - linear

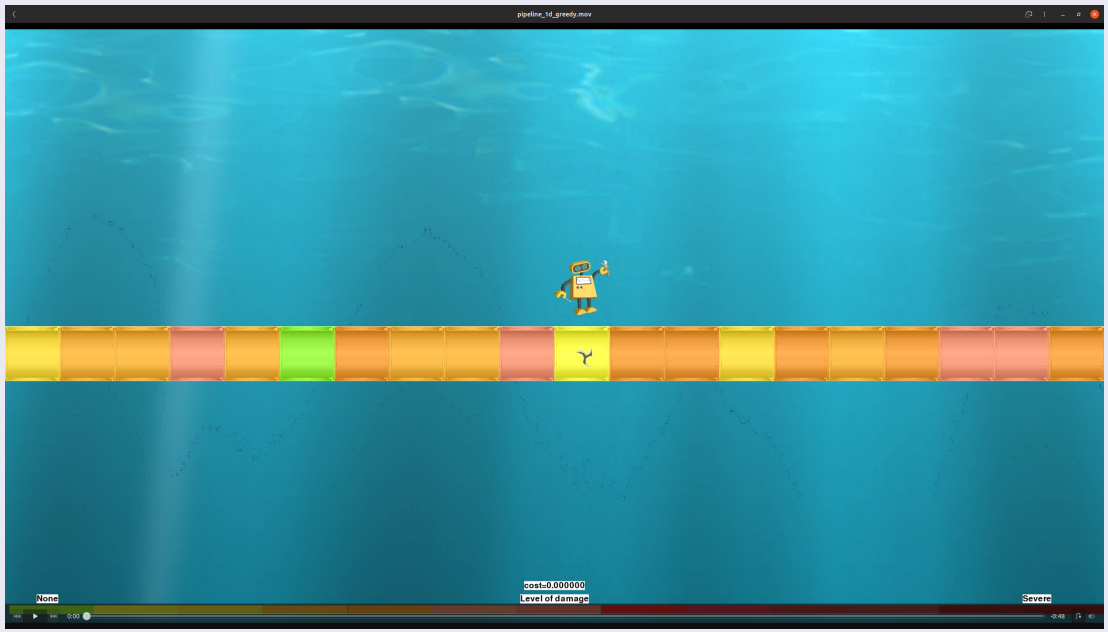
Greedy policy



cost = 13926

Pipeline Visualization - linear

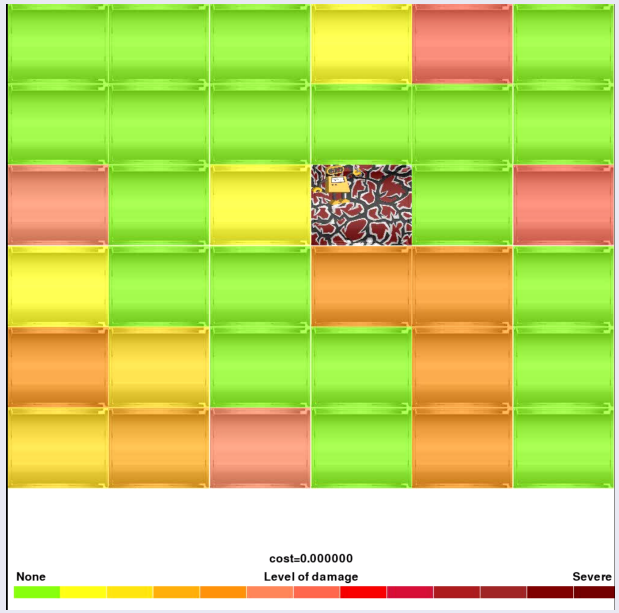
After second policy iteration



cost $= 12175$

Pipeline Visualization - 2D

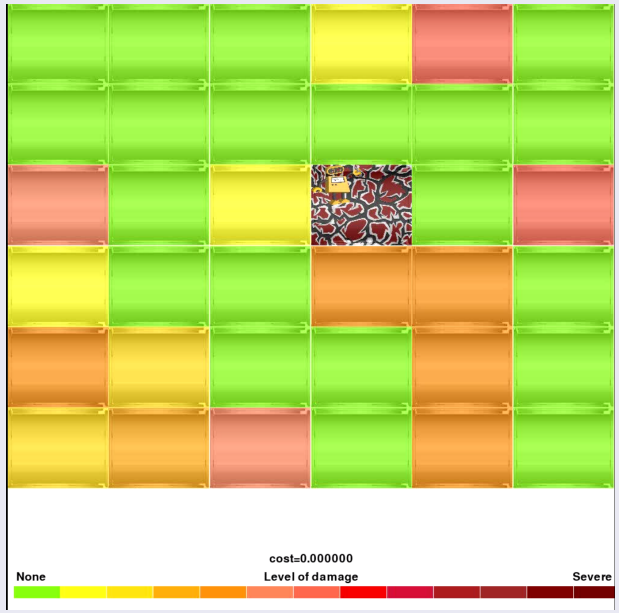
Greedy policy



cost = 7851

Pipeline Visualization - 2D

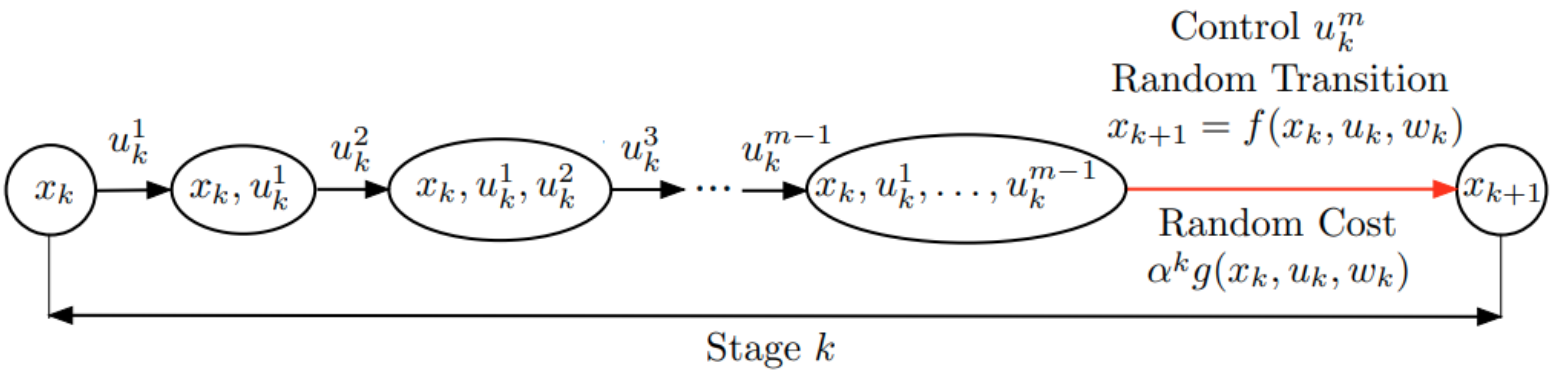
After second policy iteration



cost = 5799

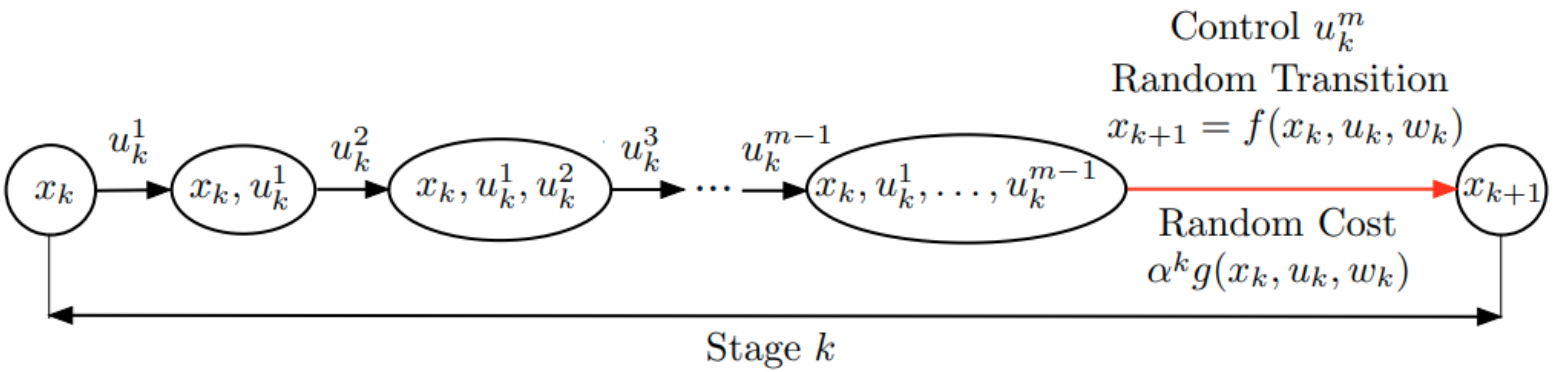
Overview

Trading off Control and State Complexity [2]



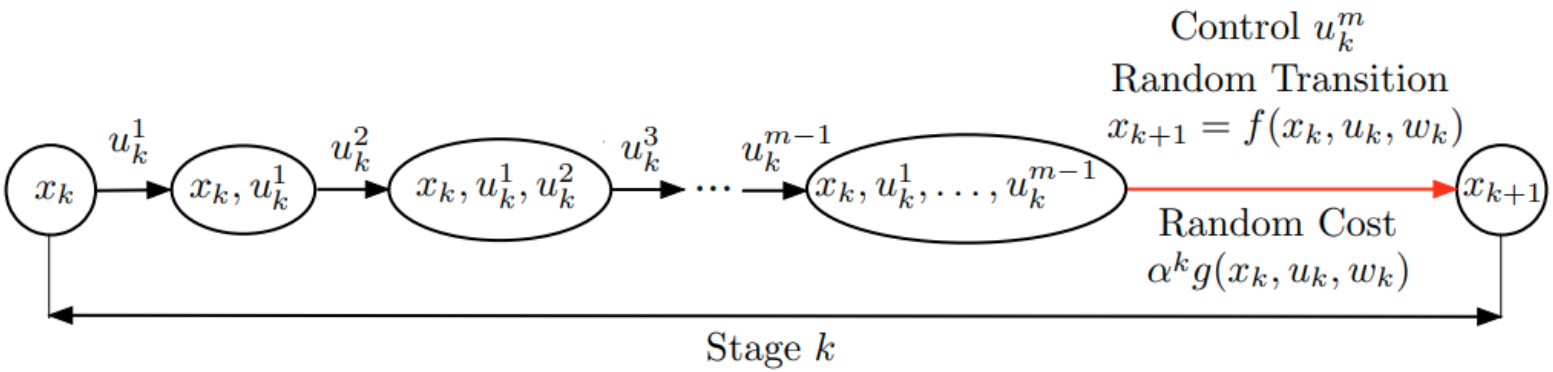
- The control space is simplified at the expense of $m-1$ additional layers of states, and corresponding $m-1$ cost functions $J^1(x_k, u_k^1), J^2(x_k, u_k^1, u_k^2), \ldots, J^{m-1}(x_k, u_k^1, \ldots, u_k^{m-1})$.
- Multiagent (one-component-at-a-time) rollout is just standard rollout for the reformulated problem.
- The cost improvement property is maintained. Complexity reduction: The one-step lookahead branching factor is reduced from n^m to nm, where n is the number of possible choices for each component.

Trading off Control and State Complexity [2]



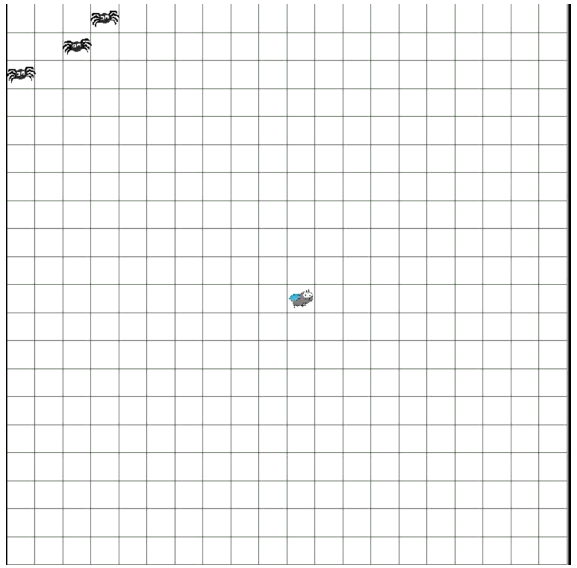
- The control space is simplified at the expense of $m-1$ additional layers of states, and corresponding $m-1$ cost functions $J^1(x_k, u_k^1), J^2(x_k, u_k^1, u_k^2), \ldots, J^{m-1}(x_k, u_k^1, \ldots, u_k^{m-1})$.
- Multiagent (one-component-at-a-time) rollout is just standard rollout for the reformulated problem.
- The cost improvement property is maintained. Complexity reduction: The one-step lookahead branching factor is reduced from n^m to nm, where n is the number of possible choices for each component.

Trading off Control and State Complexity [2]



- The control space is simplified at the expense of $m-1$ additional layers of states, and corresponding $m-1$ cost functions $J^1(x_k, u_k^1), J^2(x_k, u_k^1, u_k^2), \ldots, J^{m-1}(x_k, u_k^1, \ldots, u_k^{m-1})$.
- Multiagent (one-component-at-a-time) rollout is just standard rollout for the reformulated problem.
- The cost improvement property is maintained. Complexity reduction: The one-step lookahead branching factor is reduced from n^m to nm, where n is the number of possible choices for each component.

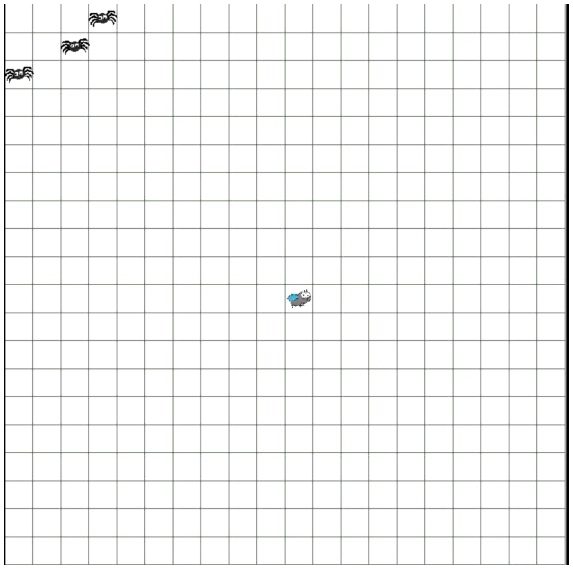
Spider and Fly Problem



The problem deals with

- Objective is to catch the fly in minimum time.
- One-step lookahead and rollout minimizes over $\approx 9^3 = 729$ Q-factors.
- One-at-a-time rollout maintains the cost improvement property: One Spiders move one-at-a-time with knowledge of other spiders' and fly's positions. The control is broken down into a sequence of 3 spider moves ($3.9 = 27$ Q-factors).

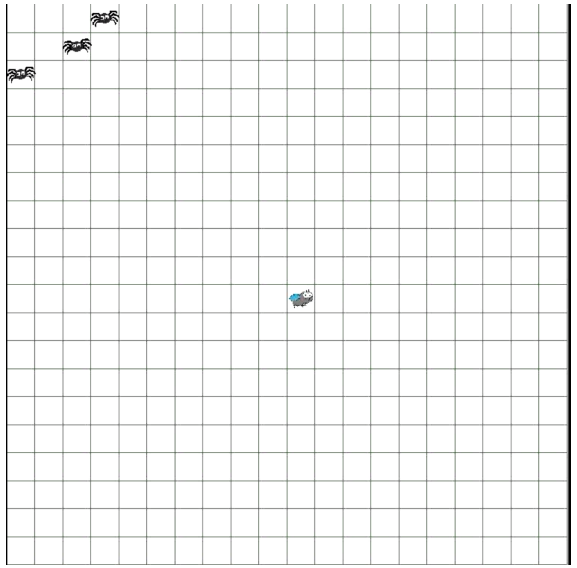
Spider and Fly Problem



The problem deals with

- Objective is to catch the fly in minimum time.
- One-step lookahead and rollout minimizes over $\approx 9^3 = 729$ Q-factors.
- One-at-a-time rollout maintains the cost improvement property: One Spiders move one-at-a-time with knowledge of other spiders' and fly's positions. The control is broken down into a sequence of 3 spider moves ($3.9 = 27$ Q-factors).

Spider and Fly Problem

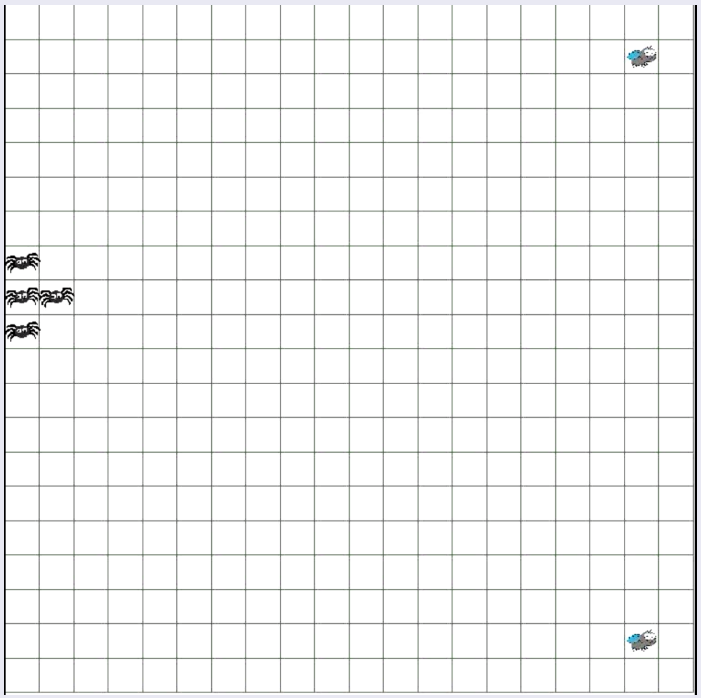


The problem deals with

- Objective is to catch the fly in minimum time.
- One-step lookahead and rollout minimizes over $\approx 9^3 = 729$ Q-factors.
- One-at-a-time rollout maintains the cost improvement property: One Spiders move one-at-a-time with knowledge of other spiders' and fly's positions. The control is broken down into a sequence of 3 spider moves ($3.9 = 27$ Q-factors).

Spider-and-fly - Splitting

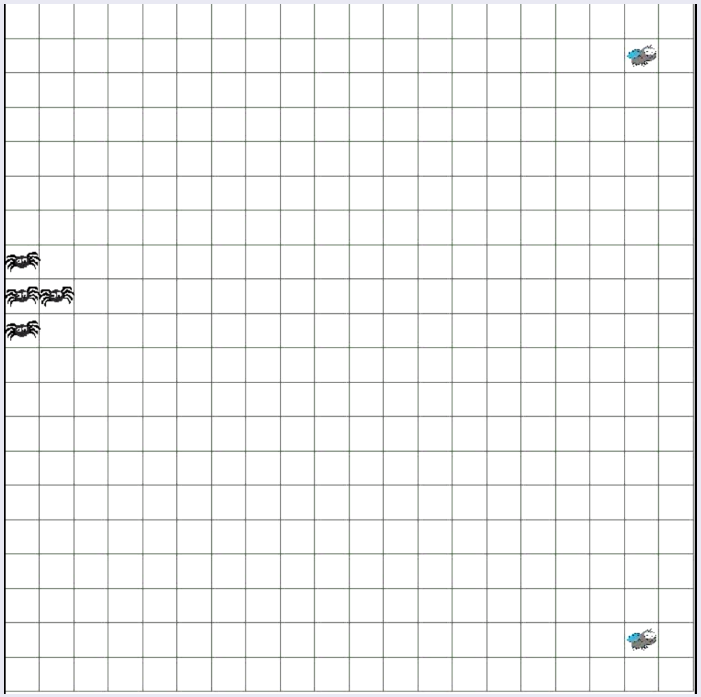
Greedy policy



cost = 61

Spider-and-fly - Splitting

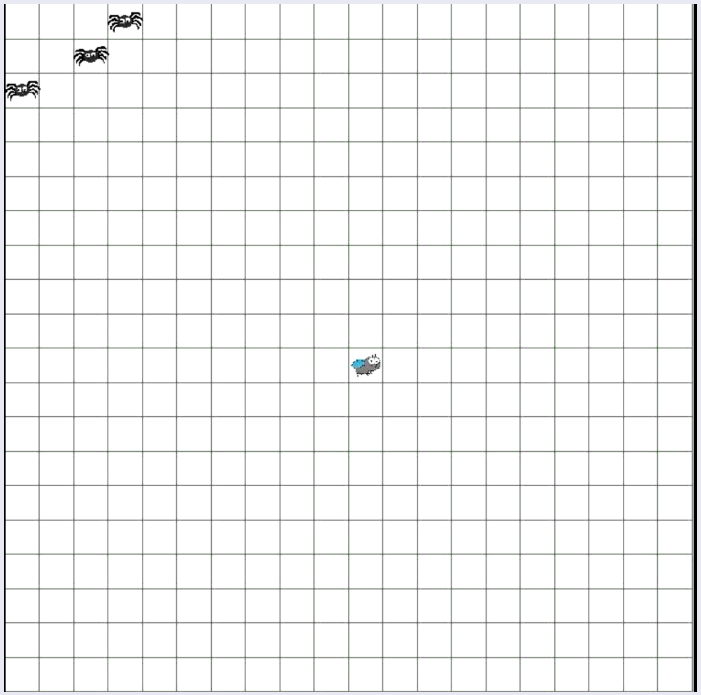
One-at-a-time



cost = 39

Spider-and-fly - Encircle

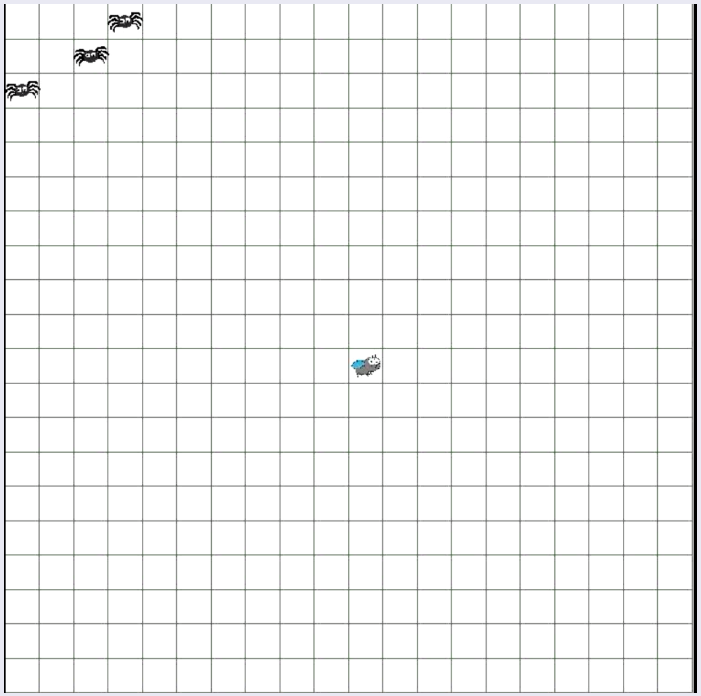
Greedy policy



cost = 21

Spider-and-fly - Encircle

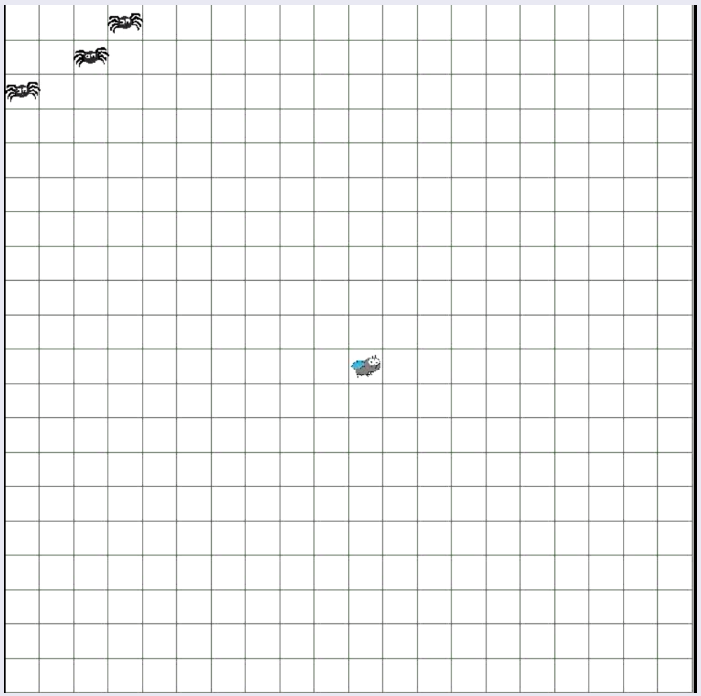
All-at-a-time



cost = 9

Spider-and-fly - Encircle

One-at-a-time



cost = 11 See, greedy does not spilt if together but rollout does

References

[1] Sushmita Bhattacharya, Sahil Badyal, Thomas Wheeler, Stephanie Gil, Dimitri Bertsekas

Reinforcement Learning for POMDP: Partitioned Rollout and Policy Iteration with Application to Autonomous Sequential Repair Problems
https://arxiv.org/abs/2002.04175

[2] Dimitri Bertsekas

Distributed and Multiagent Reinforcement Learning
http://helper.ipam.ucla.edu/publications/lco2020/lco2020_15905.pdf

Thank you