Sequential Decision Making and Reinforcement Learning

(SDMRL)

Advanced topics in RL

Sarah Keren

The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgments

- David Sliver's course on RL: *https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silve*
- Slides by Malte Helmert, Carmel Domshlak, Erez Karpas and Alexander Shleyfman.

Complex Decision Making

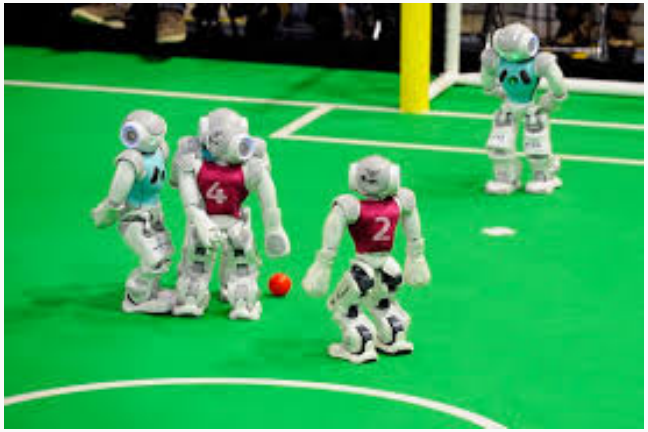
So Far

- Single agent
- Simple tasks
- No ability to learn complex tasks from experience.
- No high-level planning and reasoning.
- No account for other agents in the environment

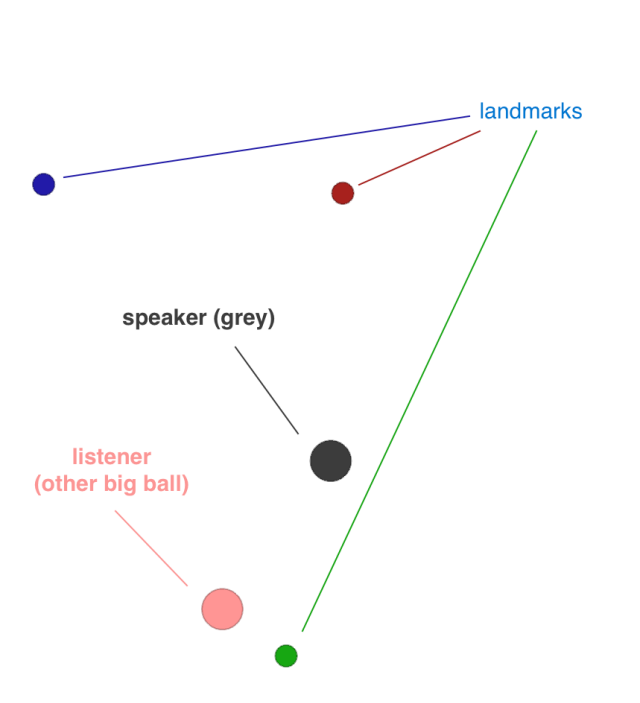
Complex Tasks

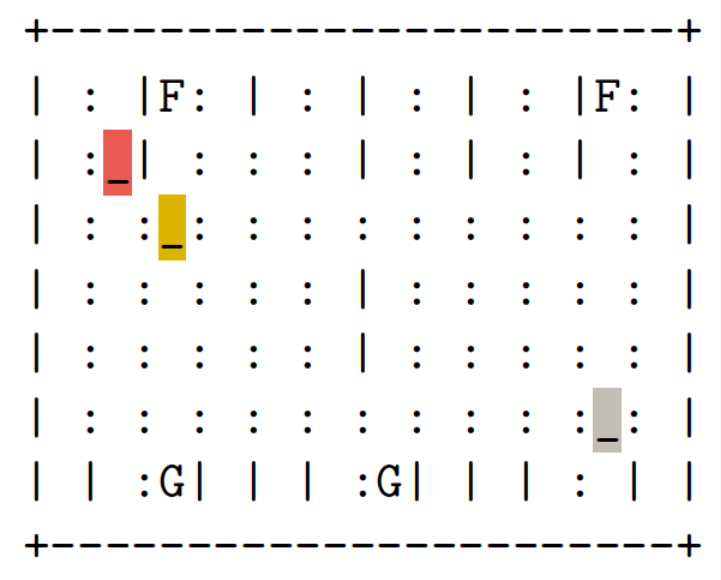
”Doing for our robots what nature did for us” - Leslie Kaelbling (MIT) *https://youtu.be/5R-xL9YmdR0*

Accounting for Other Agents



Accounting for Other Agents



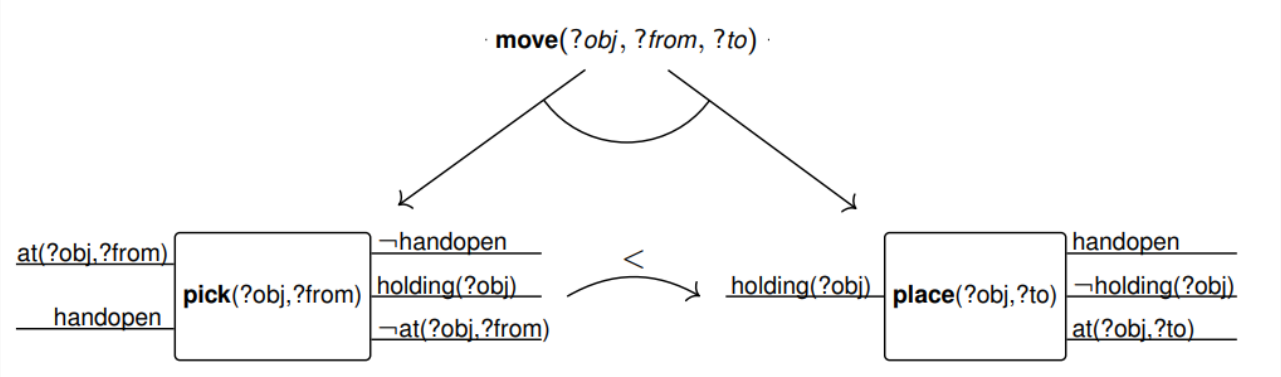


Hierarchical Planning

So Far

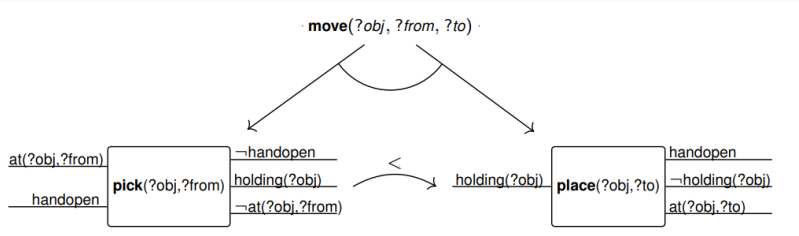
- Model based planning has only one kind of actions
- Performed offline - usually using grounded actions and via progression search
- Solutions are action sequences (or policies)

Hierarchical Planning



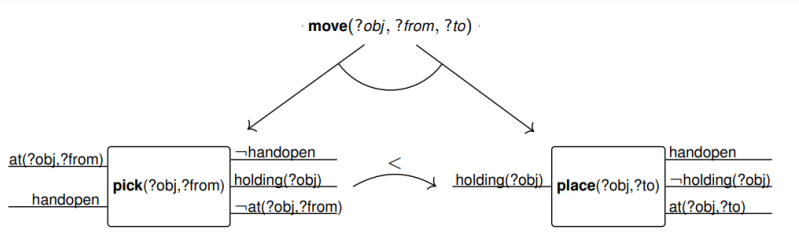
Hierarchical Planning

- The model specifies a task hierarchy:
  - compound (or complex, abstract, high-level) tasks need to be decomposed into primitive tasks.
- Benefits of a hierarchical model?



Hierarchical Planning

- The model specifies a task hierarchy:
  - compound (or complex, abstract, high-level) tasks need to be decomposed into primitive tasks.
- Benefits of a hierarchical model?
  - More flexibility with regard to modeling approach: incorporate procedural expert knowledge
  - Speed up search
  - Describe more complex behavior (i.e., pose complex restrictions on the desired solutions)
  - Allow easier user integration in the plan generation process (mixed initiative planning; MIP)
  - Communicate plans on different levels of abstraction
  - Incorporate task abstraction in plan explanations



Hierarchical Planning in Robotics: Integrated Task Planning and Motion Planning

Task planners can reason over very large sets of states by manipulating partial descriptions, while **geometric (motion) planners** operate on completely detailed specifications of world states.

A **task planner** could decide that the living room needs to be traversed, regardless of the detailed arrangement of its furniture.

Motion planners deal beautifully with geometry, but not with non-physical aspects of the domain; they can plan how to get to the phone but not decide that a phone call needs to be made.

Challenges

- Inherits challenges of both motion & classical planning
  - High-dimensional, continuous state-spaces
  - State-space exponential in number of variables
  - Long horizons
- Geometric and Continuous constraints limit high-level strategies
  - Kinematics, reachability, joint limits, collisions grasp, visibility, stability, stiffness, torque limits, ...

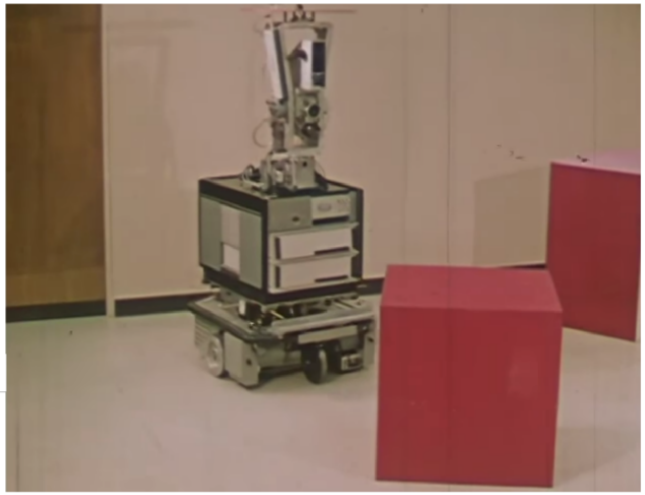
Shakey the Robot (1969)

- First autonomous mobile manipulator (via pushing)
  - Visibility graph, A* search, and STRIPS (which is a grounded form of PDDL) [Fikes 1971, Nilson 1984]
- **Decoupled** task and motion planning
  - Task planning **then** motion planning

```
type(robot robot)        type(o1 object)
name(robot shakey)       name(o1 box1)
at(robot 4.1 7.2)        at(o1 3.1 5.2)
theta(robot 90.1)        inroom(o1 r1)
                         shape(o1 wedge)
                         radius(o1 3.1)
```

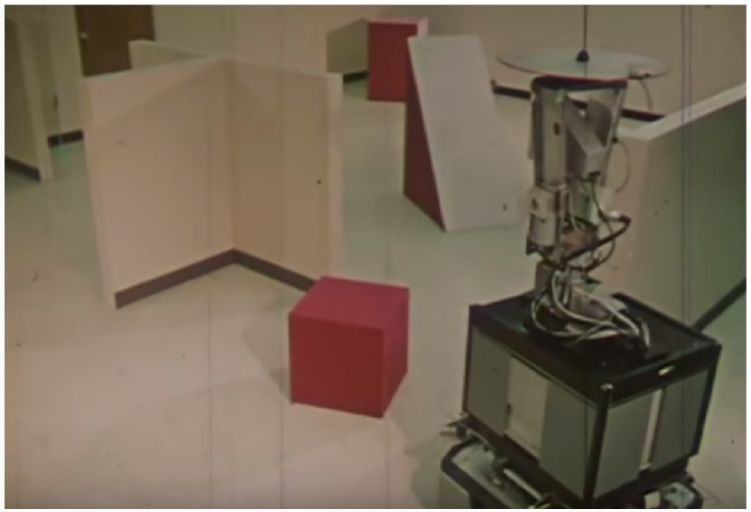
GOTHRU(d,r1,r2)

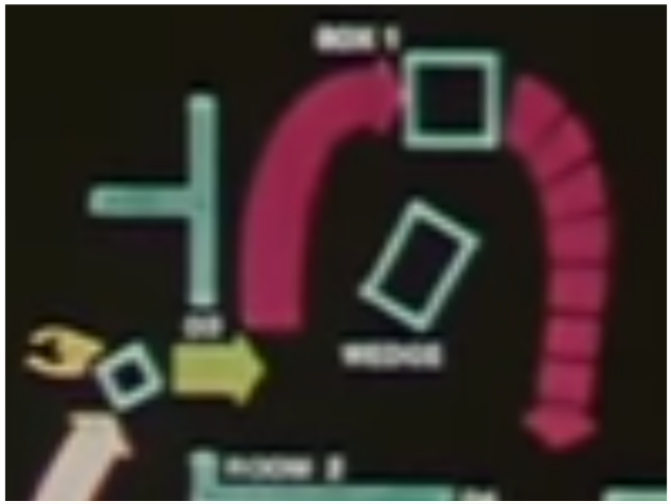
Precondition	INROOM(ROBOT,r1) ∧ CONNECTS(d,r1,r2)
Delete List	INROOM(ROBOT,$)
Add List	INROOM(ROBOT,r2)



Shakey the Robot (1969)

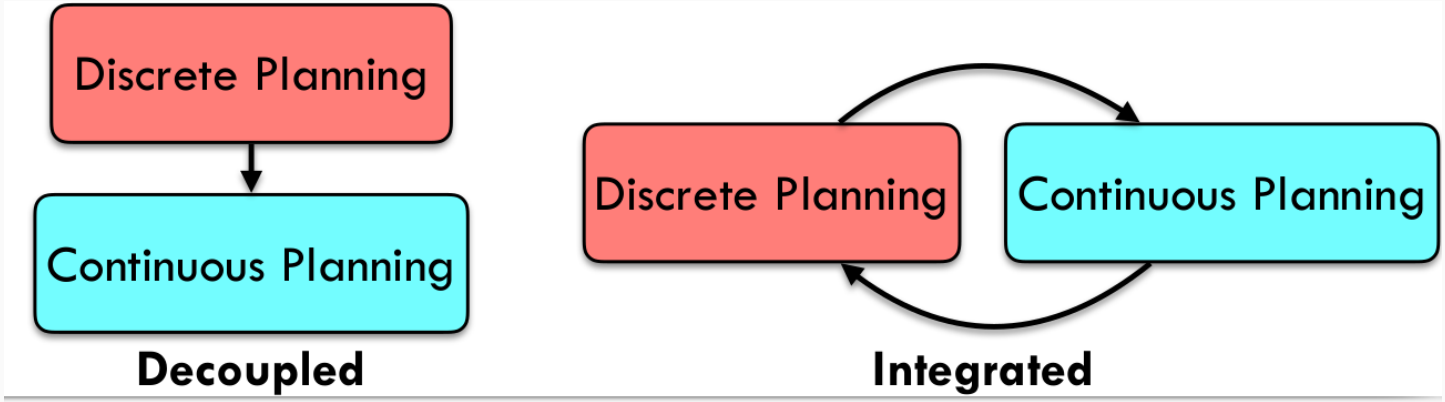
- What if a movable block prevented Shakey from safely moving into the adjacent room?
- Shakey could push it out of the way or go around it
- What's more efficient? How to push it? ...





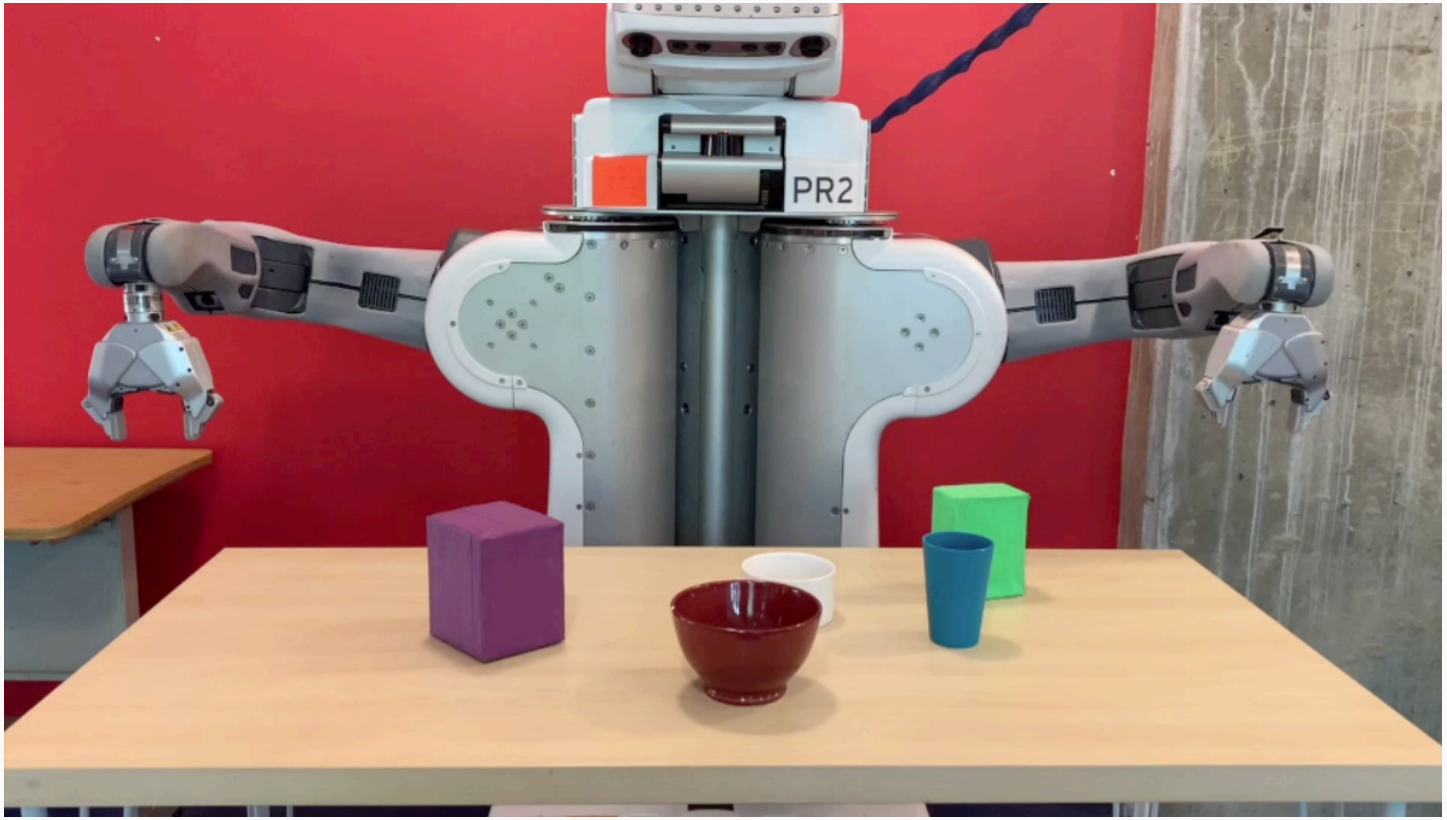
Decoupled vs. Integrated Task and Motion Planning

- **Decoupled:** discrete (task) planning then continuous (motion) planning
  - Requires a strong downward refinement assumption: every correct discrete plan can be refined into a correct continuous plan (from hierarchal planning)
- **Integrated:** simultaneous discrete & continuous planning



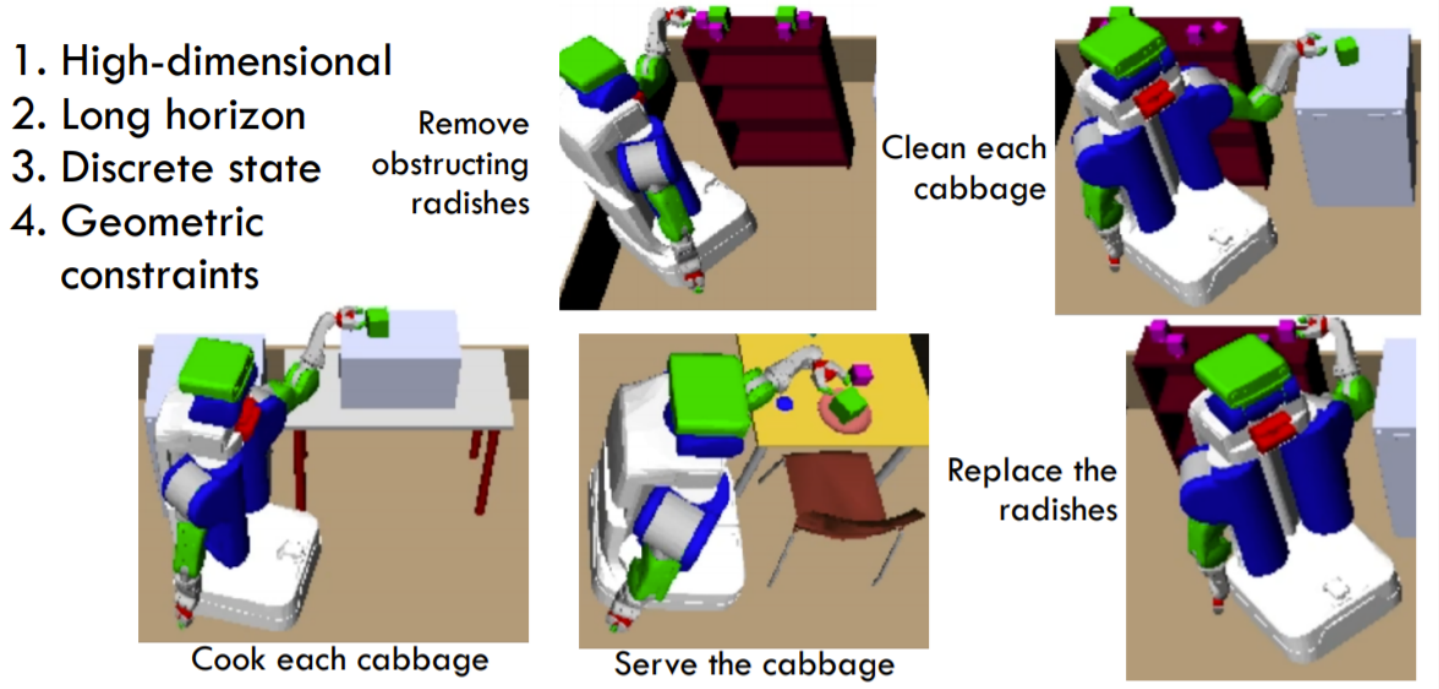
In both cases motion and task planners communicate via predicates that need to hold at the end of execution (e.g. 'Block A in gripper')

Example: Preparing a meal



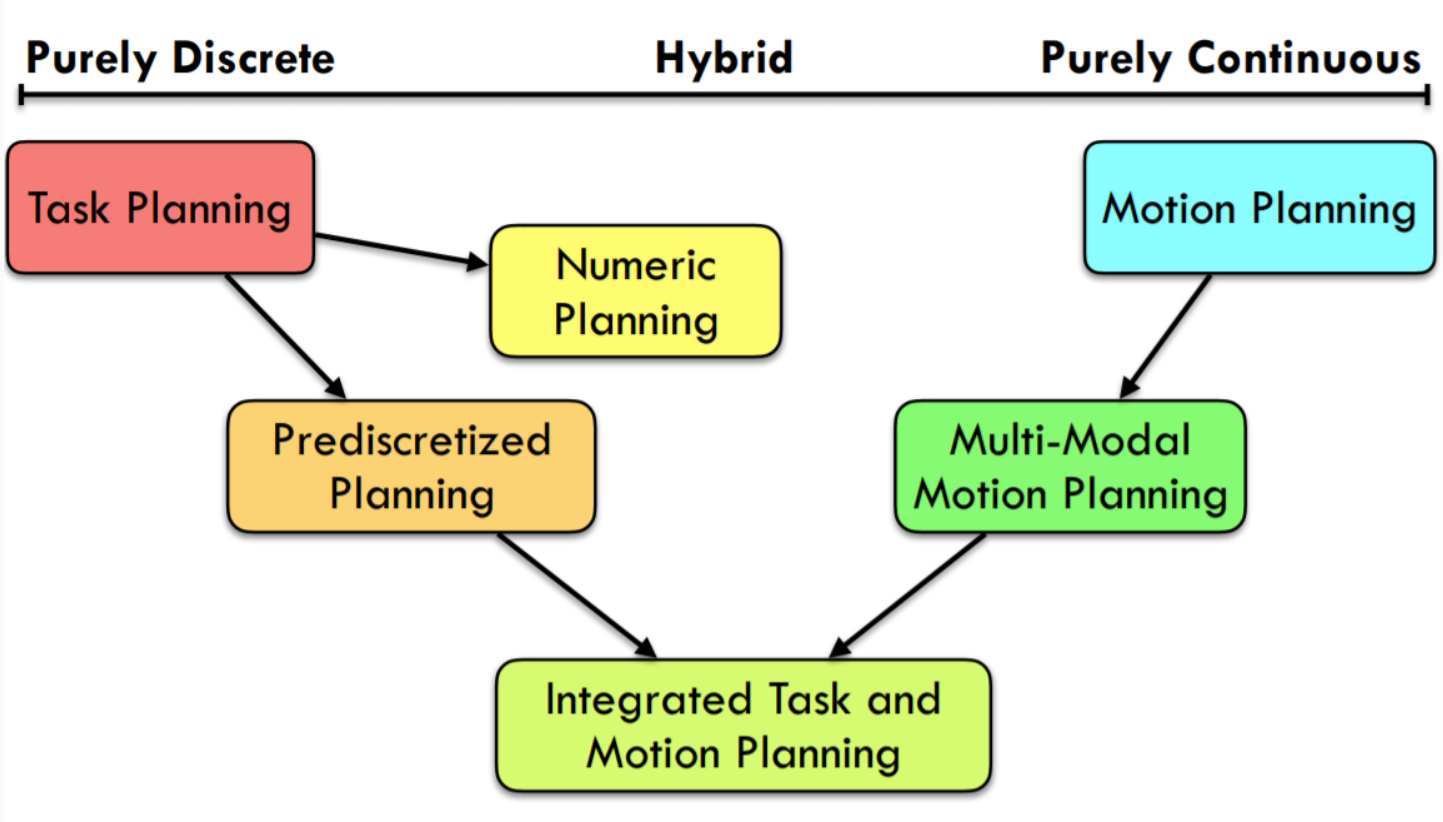
Example: https://youtu.be/JNOk1rylDpU?t=1557

Example: Preparing a meal



- Clean 3 blue cups and clean/cook 2 green cabbages
- 64 continuous and 10 discrete variables

Hybrid Planning Spectrum



Ideas ?

Hierarchical RL

Hierarchical RL

- Hierarchical reinforcement learning (HRL) decomposes a reinforcement learning problem into a hierarchy of subproblems or subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions.
- A decomposition may have multiple levels of hierarchy. Some or all of the subproblems can themselves be reinforcement learning problems.
- When a parent-task is formulated as a reinforcement learning problem it is commonly formalized as a semi-Markov decision problem because its actions are child-tasks that persist for an extended period of time.

https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_363

Benefits of Hierarchical Reinforcement Learning (HRL)

- Hierarchical decomposition reduces the computational complexity if the overall problem can be represented more compactly
- Subtasks are reusable learned (transfer learning) and can be provided independently
- Increased **robustness**.

Formal Definition of Options

- **Definition:** An option O is the following triplet: $O = \{I, \pi, \beta\}$ where:
  1. I is the set of initial states $\{s_1, ..., s_k\}$ from which the O can be operated.
  2. π is the option-policy, which dictates how to operate while O is active. Formally: $\pi : S_{tates} \times A_{ctions} \rightarrow P_{Actions}$
  3. β is an option-terminator which dictates when an O is terminated. Formally: $\beta : S_{tates} \times A_{ctions} \rightarrow \{0, 1\}$
- **Meta-Policy:** In addition to the options, there is a meta-policy $\pi_{Meta} : \mathcal{S} \times O_{ptions} \rightarrow P_{Options}$ which dictates which option to operate, when an option is terminated.

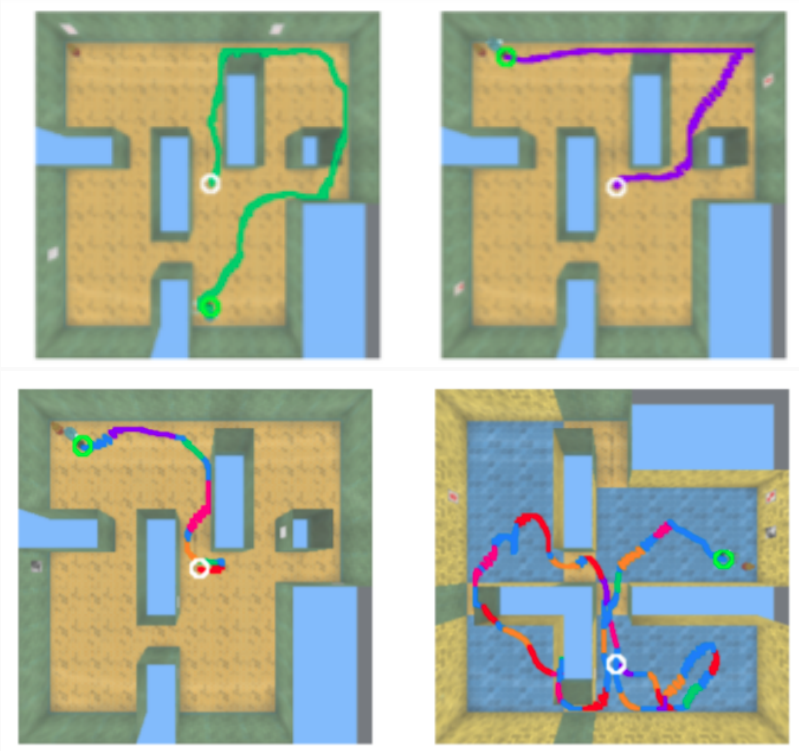
Options - Problem Definition

- Problem definition:
  - **Input:** an RL environment (the environment model is not necessarily known) for an agent to take actions in. Formally: An MDP: $E = < \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$
  - **Output:** An agent that takes actions in the environment to maximize rewards.
    Formally: An agent's policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the accumulated reward.

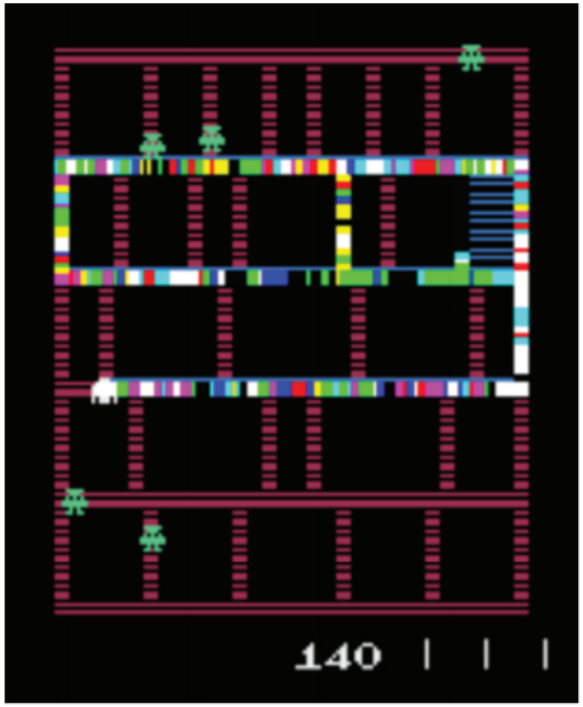
Options - Problem Definition (Continued)

- **Method:** the agent should learn a set of option policies (and a meta-policy) instead of the final goal directly.
  Formally: learn the following:
  1. A set of K option-policies $O_\pi = \pi_{o_1}, ..., \pi_{o_k}$ where each option-policy $\pi_{o_i} : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$
  2. A set of K option-terminators $B = \beta_{o_1}, ..., \beta_{o_k}$ where each option-terminators $\beta_{o_i} : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$
  3. A meta-policy $\pi_{Meta} : \mathcal{S} \times O \rightarrow \{0, 1\}$
  4. Some papers describe the additional I_{O_i} as the initial states from each option can be started from.

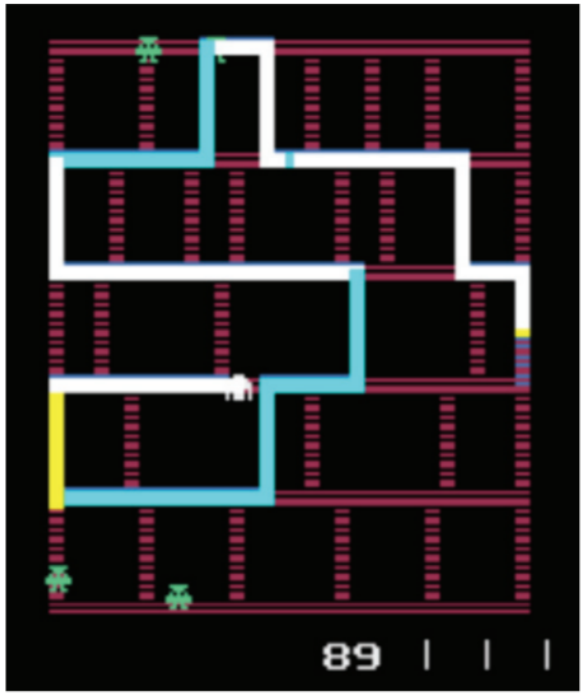
Deepmind Examples



Amidar Examples



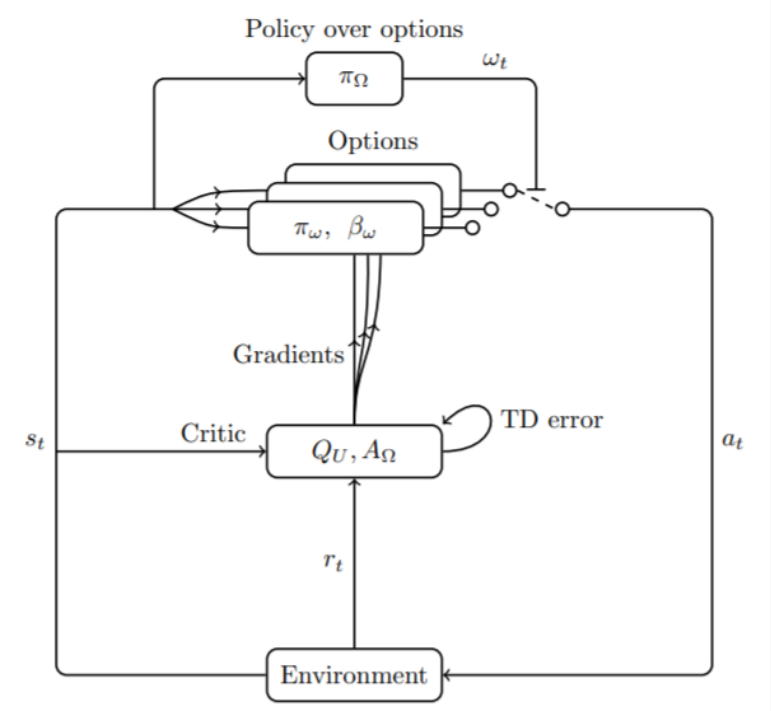
(a) Without a deliberation cost, options terminate instantly and are used in any scenario without specialization.



(b) Options are used for extended periods and in specific scenarios through a trajectory, when using a deliberation cost.

MLSH demonstration of options in a Mujoco game

Option-Critic (Bacon et al. 2016 AAAI 2017)



Option-Critic (Bacon et al. 2016 AAAI 2017)

- Main Idea:
  - (Novelty) Learns using an actor-critic framework over the options space:
    Learns a set of option-policies and termination functions, as well as a value function which depends on the current state and option, i.e. $V_{alue} : S_{tates} \times O_{ptions} \rightarrow \mathbb{R}$
  - Uses a meta-policy to determine which option to perform (originally used a simple DQN)
  - Relies on clever mathematical "trick" for optimizing β_{o_i} (terminations): consider the output of β_{o_i} for the calculation of the learned value function.
    Meaning that the correct formulation is really:
    $V_{alue} : S_{tates} \times O_{ptions} \times B_{(terminations)} \rightarrow \mathbb{R}$

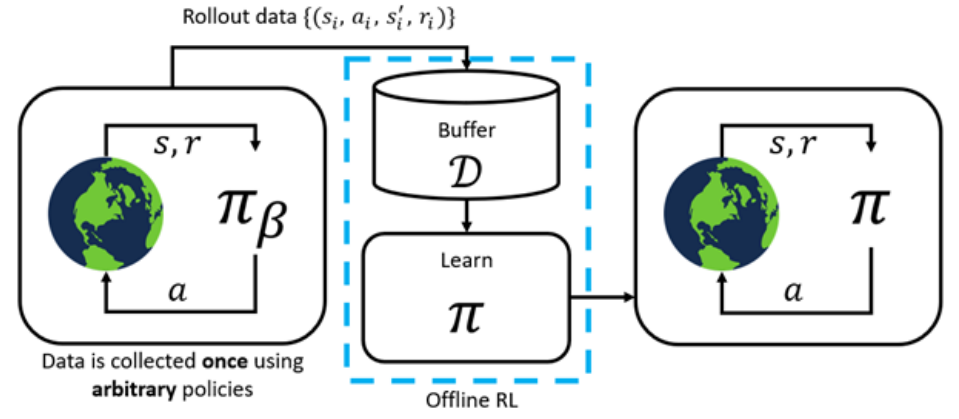
Offline RL

Offline RL

Tutorial: *https://arxiv.org/abs/2005.01643* Lecture: *https://www.youtube.com/watch?v=NV4oSWe1H9o&list=PL_iWQOsE6TfVYGEGiAOMaOzzv41Jfm_Ps&index=64*

What is Offline RL?

- Offline RL (Batch RL) learns a policy from a pre-collected dataset.
- The agent does not interact with the environment during training.
- Dataset: $\mathcal{D} = \{(s, a, r, s')\}$ collected from prior policies or human experts.
- Goal: Train a policy π that performs well when deployed in the environment.

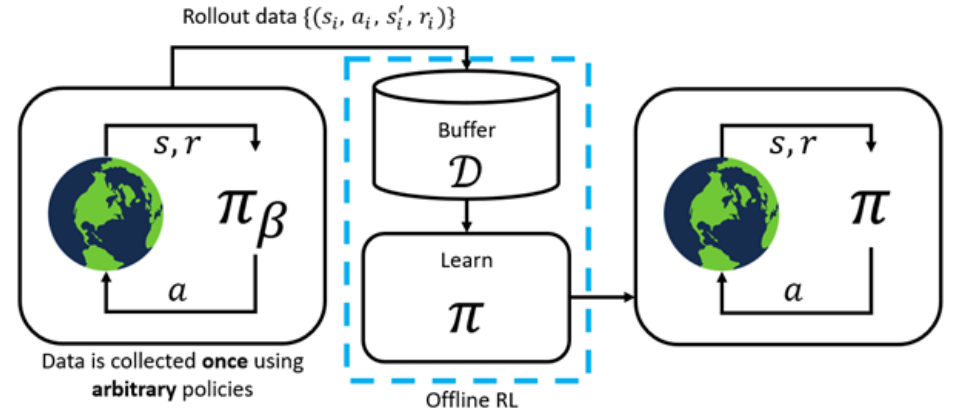


Challenges in Offline RL

- **Distributional Shift:** Policy π may propose actions outside the dataset distribution.
- **Extrapolation Error:** Value estimates for out-of-distribution actions are unreliable.
- **No Exploration:** Agent cannot collect new data to refine the policy.
- **Bias-Variance Trade-off:** Risk of overfitting to the dataset.

Key Approaches in Offline RL

1. **Behavior Regularization:** Ensure policy stays close to dataset policy π_{data}.
2. **Conservative Value Estimation:** Penalize overestimation of rewards.
3. **Uncertainty Estimation:** Avoid actions with high model uncertainty.
4. **Policy Constraints:** Restrict actions to the support of the dataset.



Applications of Offline RL

- **Healthcare:** Optimizing treatments using historical patient data.
- **Robotics:** Learning control policies from demonstrations.
- **Autonomous Vehicles:** Training driving policies using logged driving data.
- **Finance:** Learning trading strategies from historical data.
- **Recommender Systems:** Improving recommendations based on user logs.