Reinforcement Learning: Models and Hierarchy

Benjamin Rosman

benjros@gmail.com

School of Computer Science and Applied Mathematics
University of the Witwatersrand
South Africa

African Masters of Machine Intelligence (Kigali, Rwanda)
January 13-17 2020

Further reading…

- Reinforcement Learning: An Introduction (Sutton and Barto)
  - Chapter 8
- CS 294-122 (Sergey Levine)
  - Lecture 9
- COMPM050/COMPGI13 (David Silver)
  - Lecture 8
- CompSci 590.2 Hierarchical Robot Learning and Planning (George Konidaris)
- Taylor, M.E. and Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(Jul), pp.1633-1685.

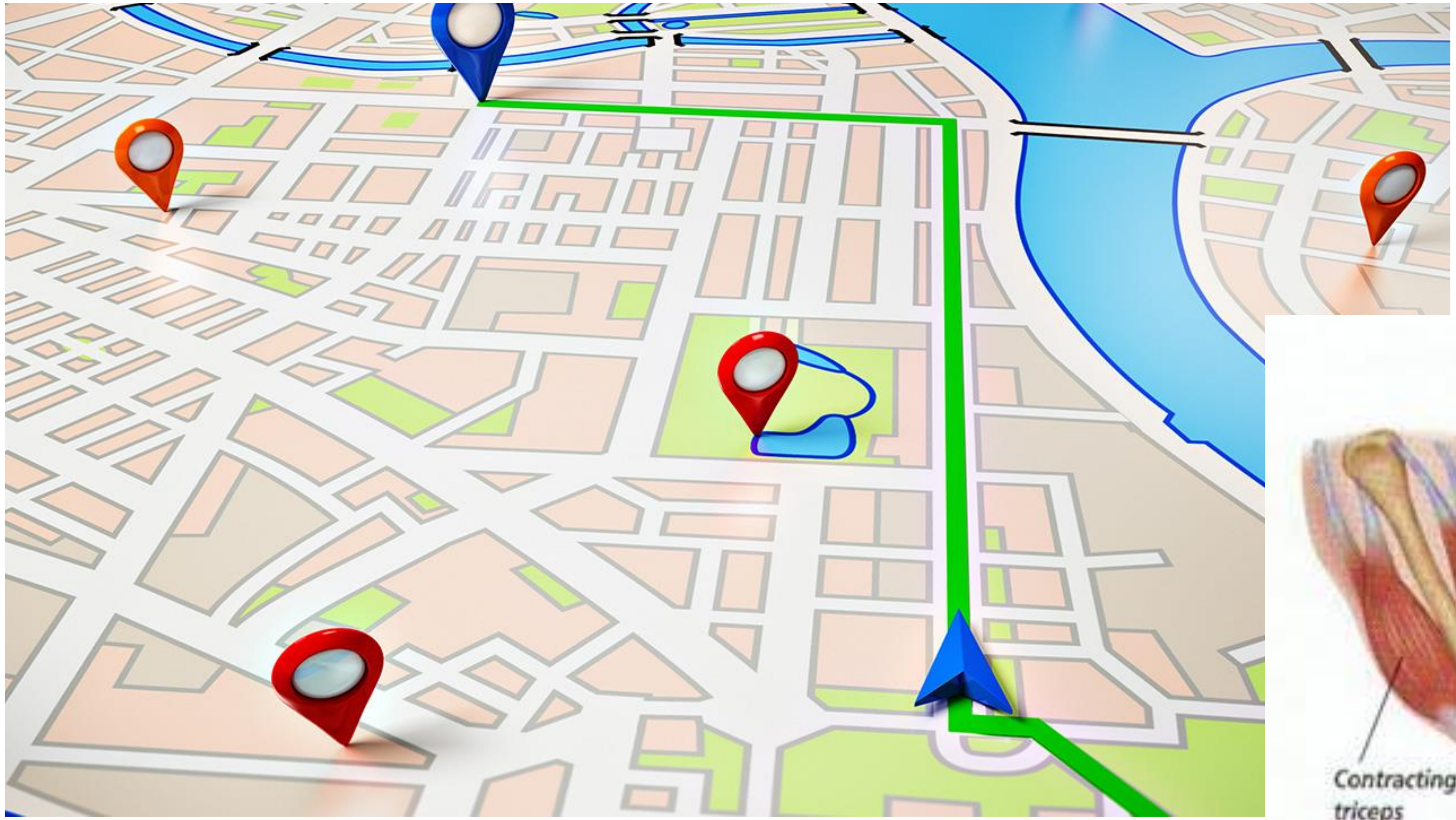
Why is RL hard?

Sparse and delayed rewards

- Sparsity:
  - Most actions give no reward feedback

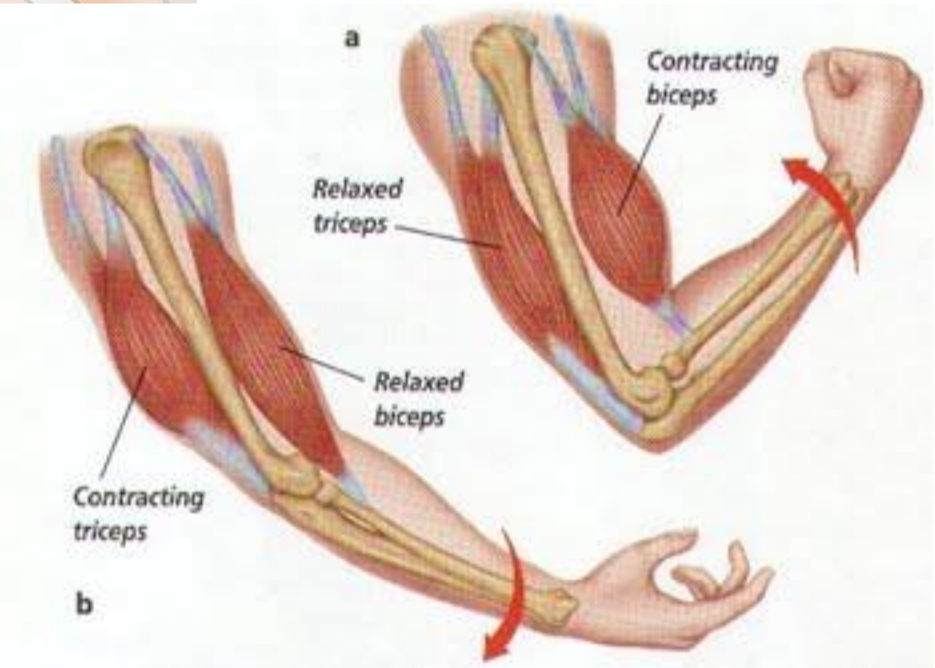
- Delayed:
  - Rewards may come after executing whole trajectories

Long action sequences



What is the depth and breadth of the navigation problem?

What about at the level of muscle activations?



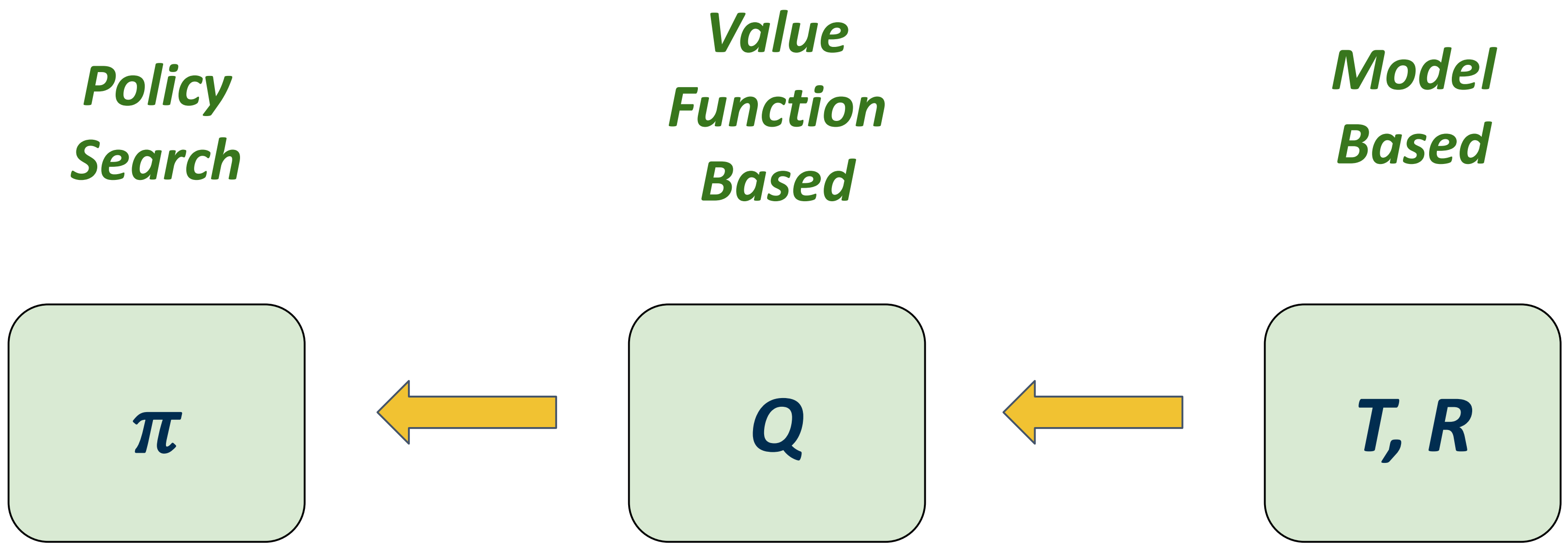
Addressing these issues?

- General ideas:
  - Make predictions of what may happen
  - Exploit structure of the problem
    - Representations of states and transitions
    - Action spaces
    - Rewards
  - Reuse knowledge

Addressing these issues?

- General ideas:
  - **Make predictions of what may happen**
  - Exploit structure of the problem
    - Representations of states and transitions
    - Action spaces
    - Rewards
  - Reuse knowledge

RL approaches

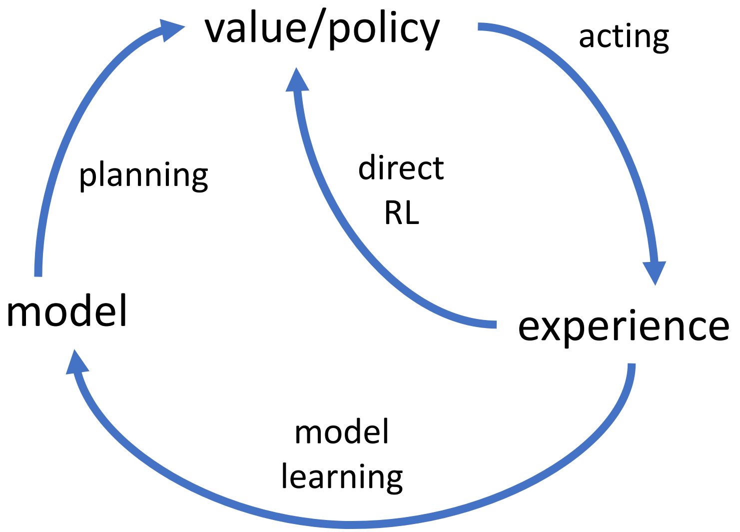


From Values to Environment Models

- Model based reinforcement learning

- Learn a model (T and R) from experience
  - $s' \sim \hat{T}(s'|s,a)$
  - $r \sim \hat{R}(r|s,a)$
  - Simulator: ask questions about the domain
  - Supervised learning problem

- Models let you:
  - Predict next state and reward
  - Reason about uncertainty
  - Be more efficient in how you use data



Models

- Models can be:
  - Distribution models
    - Produces the distribution $p(s', r|s, a)$
  - Sample models
    - Produces a sample s', r given a current s, a

- What are the advantages/disadvantages of these kinds of models?
- How might you use these two kinds of models?

$$Q(s,a) \leftarrow \sum_{s',r} \boxed{\hat{p}(s',r|s,a)} \left[r + \gamma \max_{a'} Q(s',a') \right]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\boxed{R + \gamma \max_{a'} Q(S',a')} - Q(s,a) \right]$$

Model Based RL

Learn a Transition and Reward Model

On receiving experience (s_t, a_t, r_t, s_{t+1})

$$R(s_t, a_t) \Leftarrow R(s_t, a_t) + \alpha(r - R(s_t, a_t))$$

$$T(s_t, a_t, \boxed{s_{t+1}}) \Leftarrow T(s_t, a_t, s_{t+1}) + \alpha(\boxed{1} - T(s_t, a_t, s_{t+1}))$$

$$T(s_t, a_t, \boxed{\hat{s}}) \Leftarrow T(s_t, a_t, \hat{s}) + \alpha(\boxed{0} - T(s_t, a_t, \hat{s}))$$

These can be thought of as regression problems

$$Q(s,a) = R(s,a,s') + \gamma \sum T(s,a,s')V(s')$$

Dyna-Q Algorithm

What can be parallelised here?

For each step t in episode:

- Choose a in s from Q
- Take a, observe r, s
- Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Q-learning

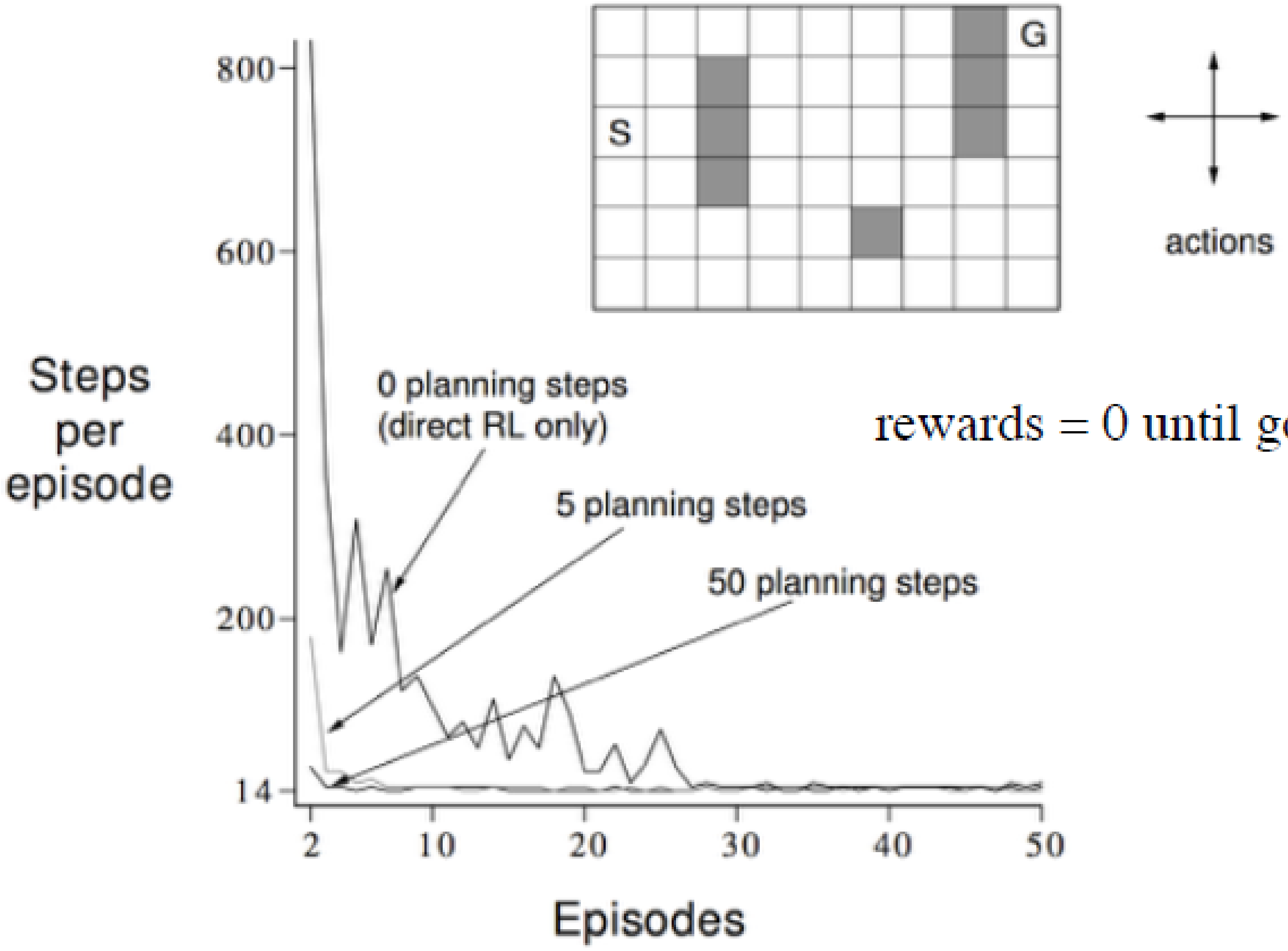
- Given (s,a,r,s'):
  - Update T and R

model learning

- Repeat n times:
  - Sample previously observed s
  - Sample previously taken a (in s)
  - Get r and s' from model
  - Update Q: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

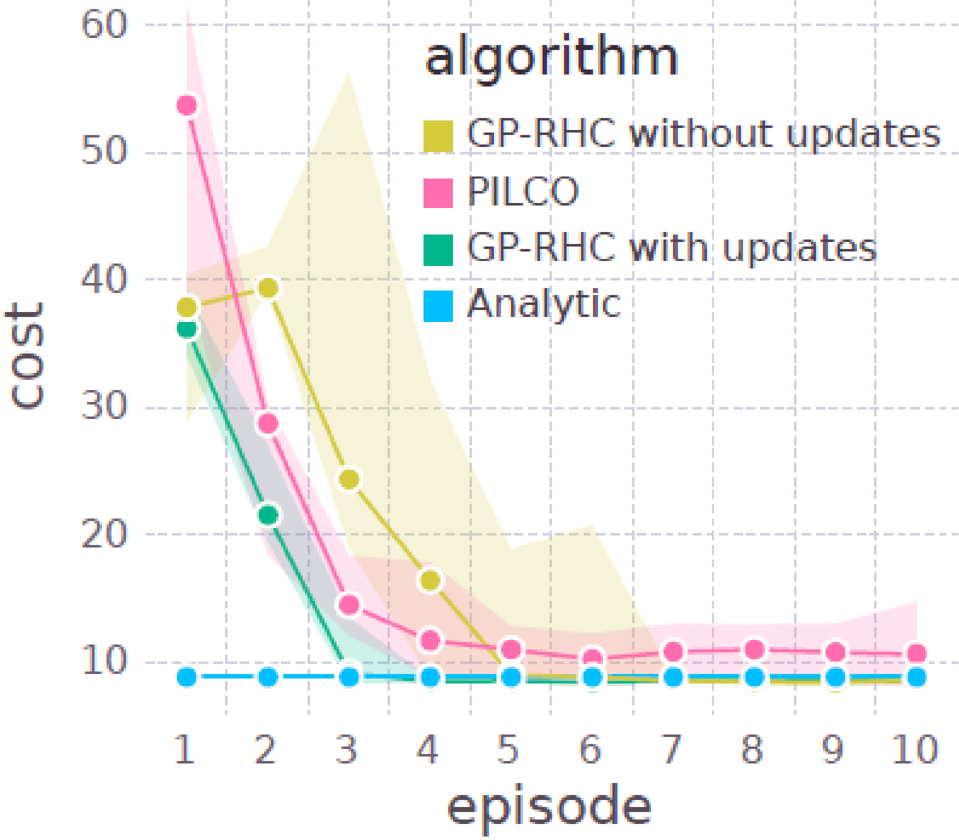
sample model to update Q: planning

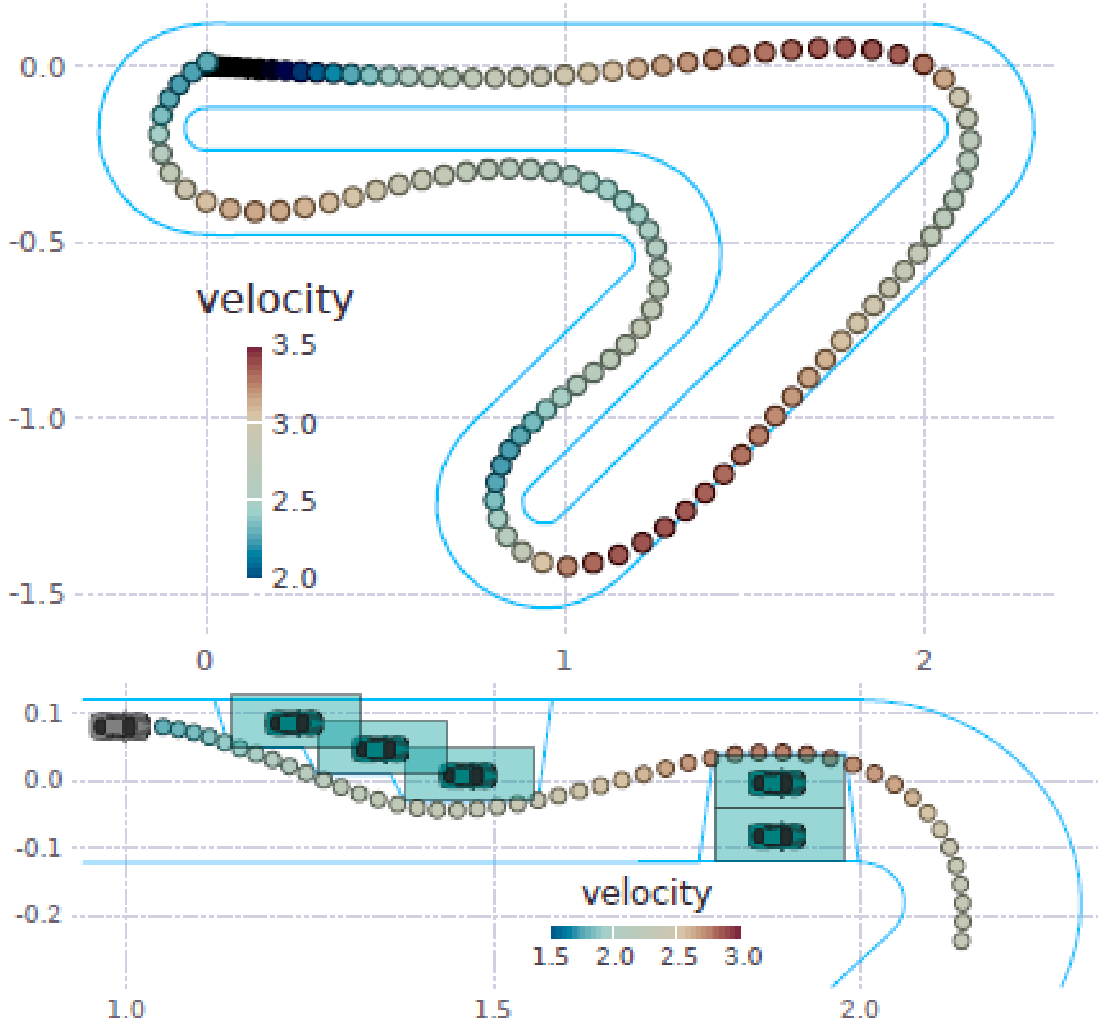
Dyna-Q example



rewards = 0 until goal, when =1

Learning models with Gaussian Processes



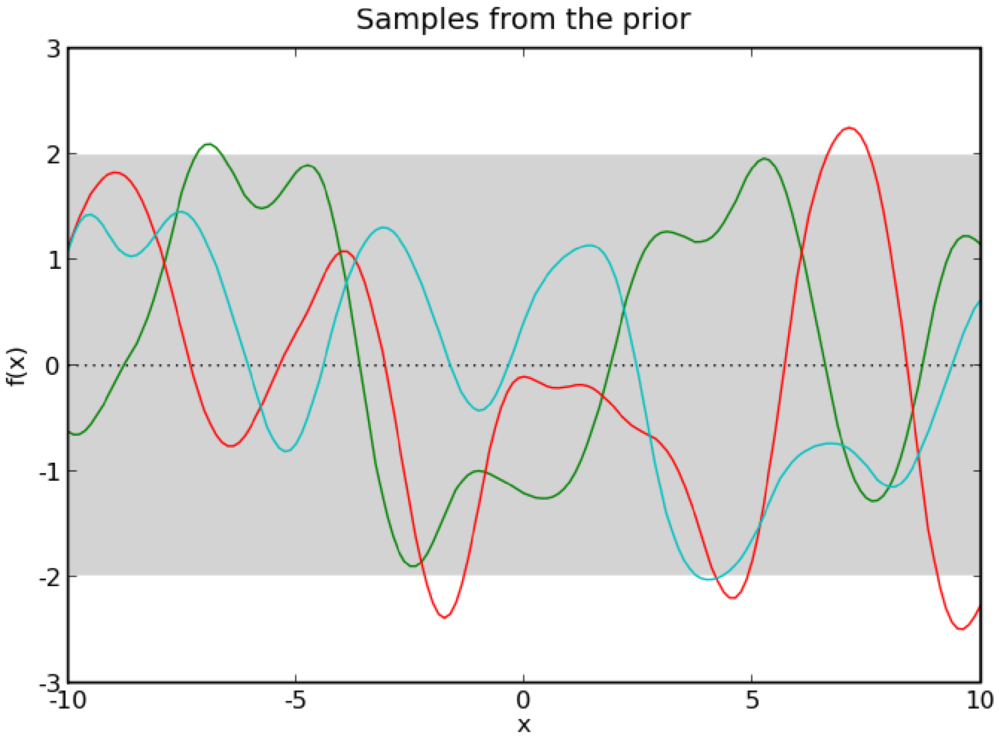


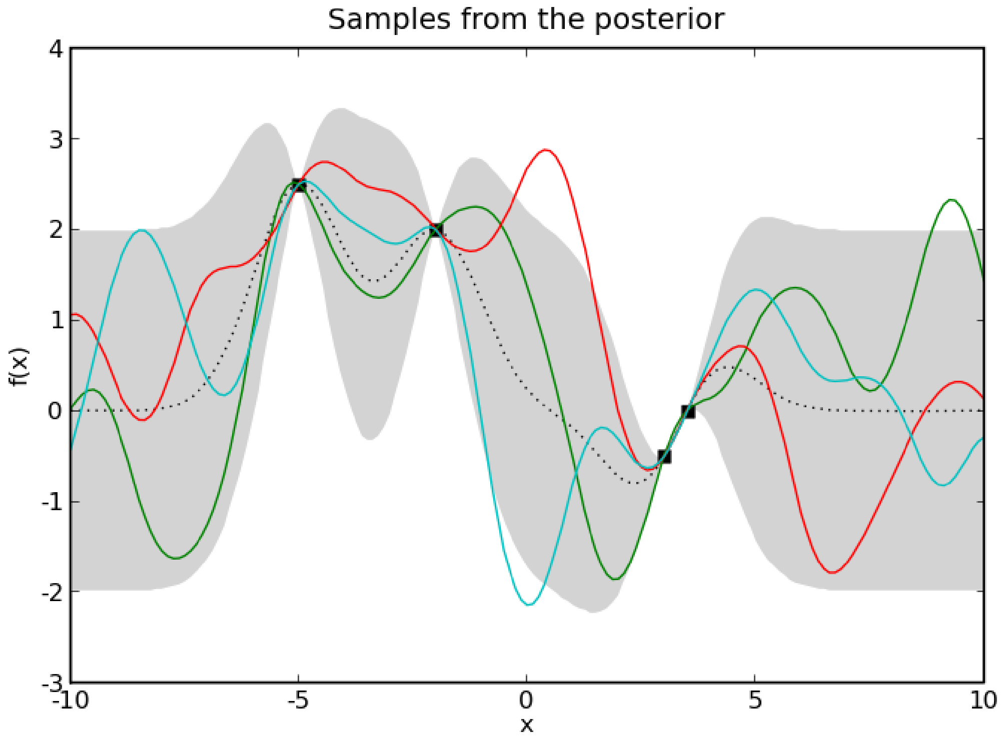
Benjamin Rosman

(van Niekerk, Damianou, Rosman, UAI 2017)

Gaussian processes

- Gaussian distribution over function space
- Regression with uncertainty





Learning models with GPs

Data: number of features D, initial training data
foreach *episode* **do**
  train GPs on accumulated data (minimize (11));
  linearise dynamics and constraints about $\mathbf{x}_0$;
  solve (5) until convergence;
  while *not terminal* **do**
    shift previous trajectory;
    for $i = 1$ *to max iterations* **do**
      linearise about current trajectory;
      solve (5) to get step direction;
      update trajectory (4);
    end
    apply control to the system;
    update dynamics model (12);
  end
end

Algorithm 1: The complete GP-RHC algorithm

Minimise

$$J(\mathbf{x}_0) = h(\mathbf{x}(t_0+T)) + \int_{t_0}^{t_0+T} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))dt$$

Subject to:

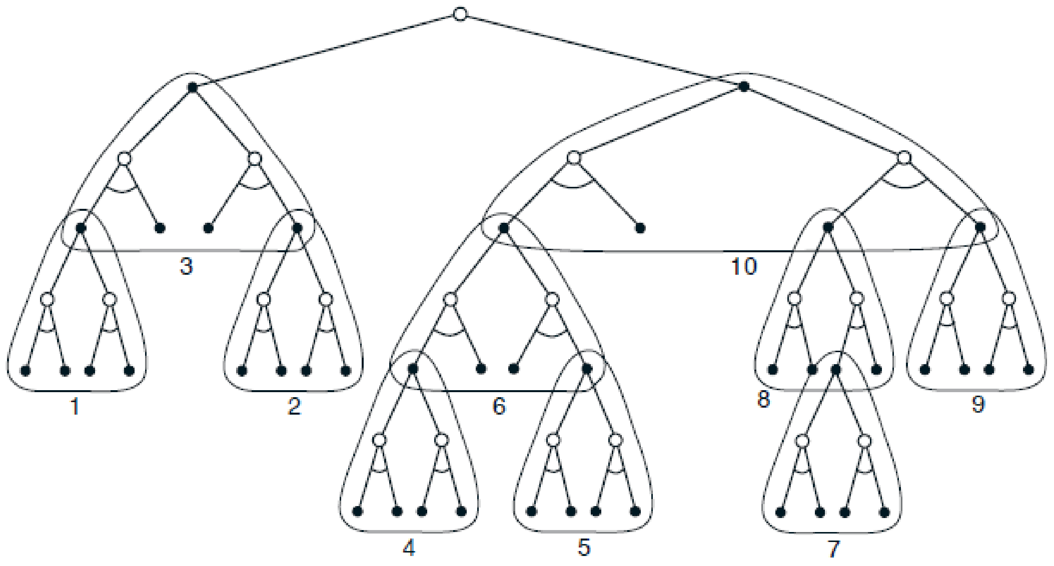
$$\mathbf{x}(t_0) = \hat{\mathbf{x}}_0,$$
$$\boxed{\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)),} \sim GP$$
$$\underline{\mathbf{u}} \leq \mathbf{u}(t) \leq \overline{\mathbf{u}},$$
$$\underline{\mathbf{x}} \leq \mathbf{x}(t) \leq \overline{\mathbf{x}},$$
$$\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \leq \mathbf{0},$$
$$\text{for all } t \text{ in } [t_0, t_0 + T]$$

(van Niekerk, Damianou, Rosman, UAI 2017)

Planning

- Computational process to use a model to create or improve a policy
  - E.g. Sampling the model to update the Q-function in Dyna Q
- Form of search over the state-action space
  - Using the model to simulate what may happen
- How else can planning be done?
  - Tree search
  - …

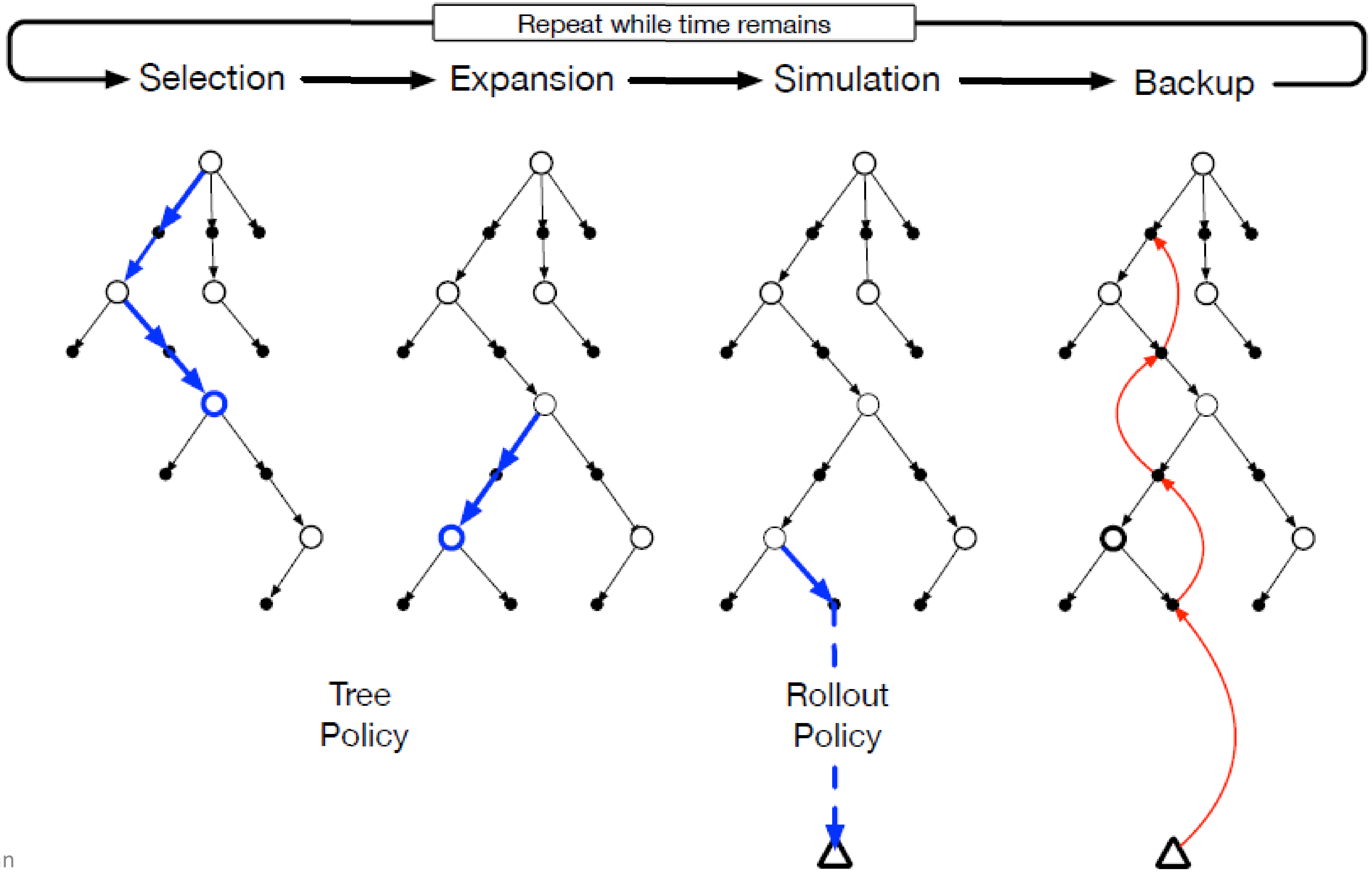
Why can't we just build a tree of the entire domain?



How to efficiently search the tree?

- Monte-Carlo Tree Search (MCTS)
  - Build a tree incrementally
    - Get around limitation of large state space
  - Run simulations (from a model/simulator)
    - Get around limitation of perfect knowledge
  - Keep estimates of the values of actions
    - Approximate value functions
  - Continue planning (simulating) for as long as we have time
    - Anytime algorithm

Monte-Carlo Tree Search (MCTS)



Monte-Carlo Tree Search (MCTS)

- Selection
  - Starting from the root of the tree, select actions until an expandable leaf node is reached (tree policy)
  - Choose nodes to maximise $UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$

    Number of times parent selected

    Number of times this node selected

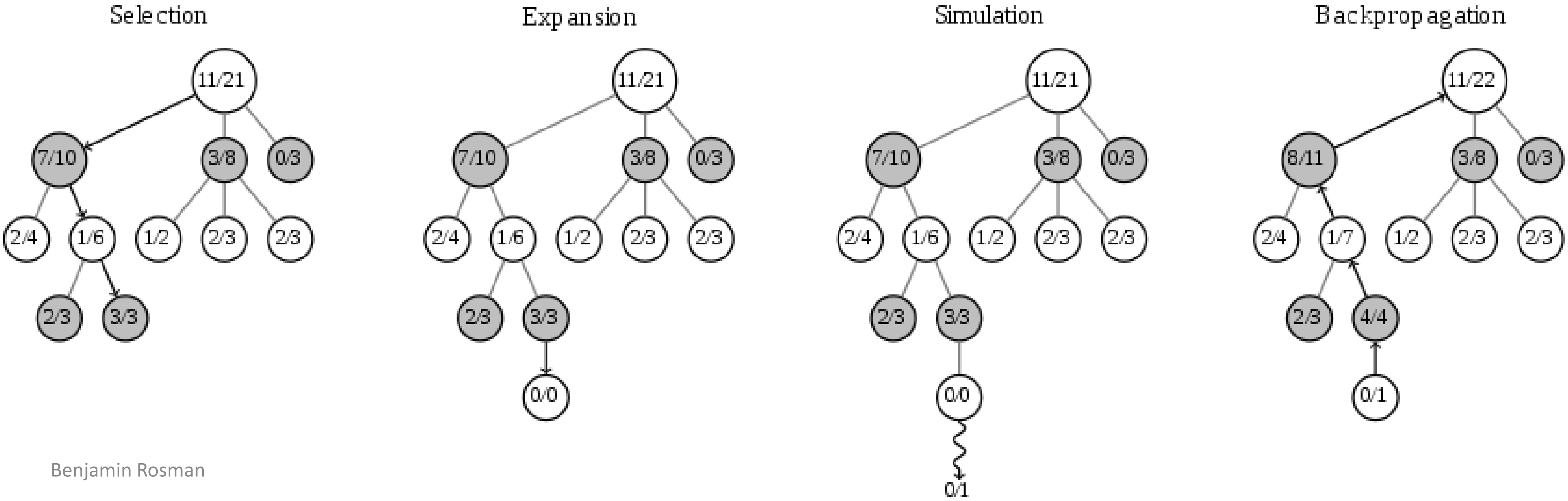
    Estimated value of action j

    (cf UCB in Bandits)
  - Exploration/exploitation
- Expansion
  - Add a new child node to that leaf node, corresponding to a new action
- Simulation
  - Run a simulation from that new node until termination with an outcome (default policy) – usually random
- Backup
  - Propagate that outcome up through the tree

MCTS illustration

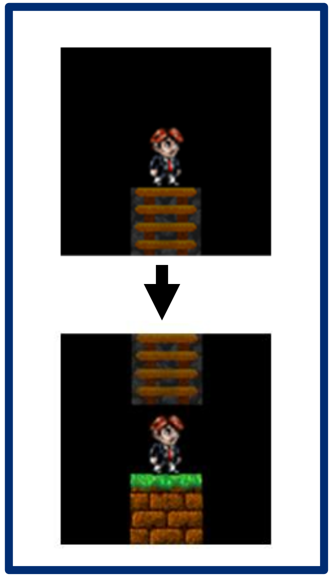
In this example:

- For each node: total wins/total playouts
- Alternating white and black player



Models – generalising knowledge

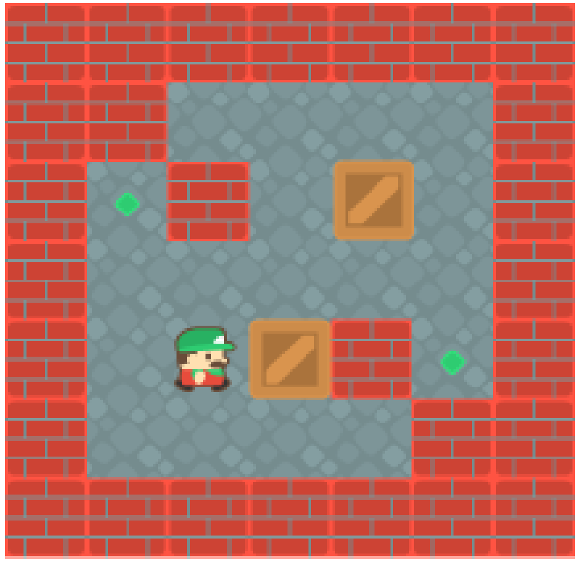
- Don’t need to learn T and R monolithically
- Learn local models
  - Tells you something about specific regions of space
- Plan in an abstracted space with portable knowledge



(James, Rosman and Konidaris, 2018)

Scaling up

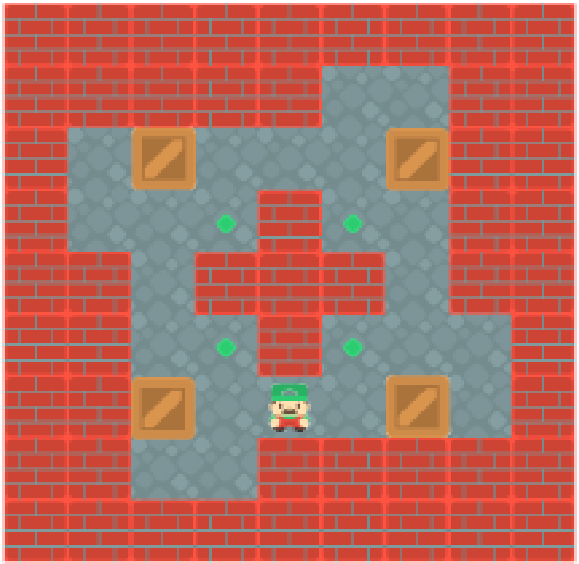
- Learn local rules in small problems (e.g. with object oriented representations – OOMDPs)
    - Transfer to larger ones



$\sim 8k$ states

$touch_{North}(Person, Wall)$

$Person.y \leftarrow Person.y + 1$ $\quad\quad \phi$



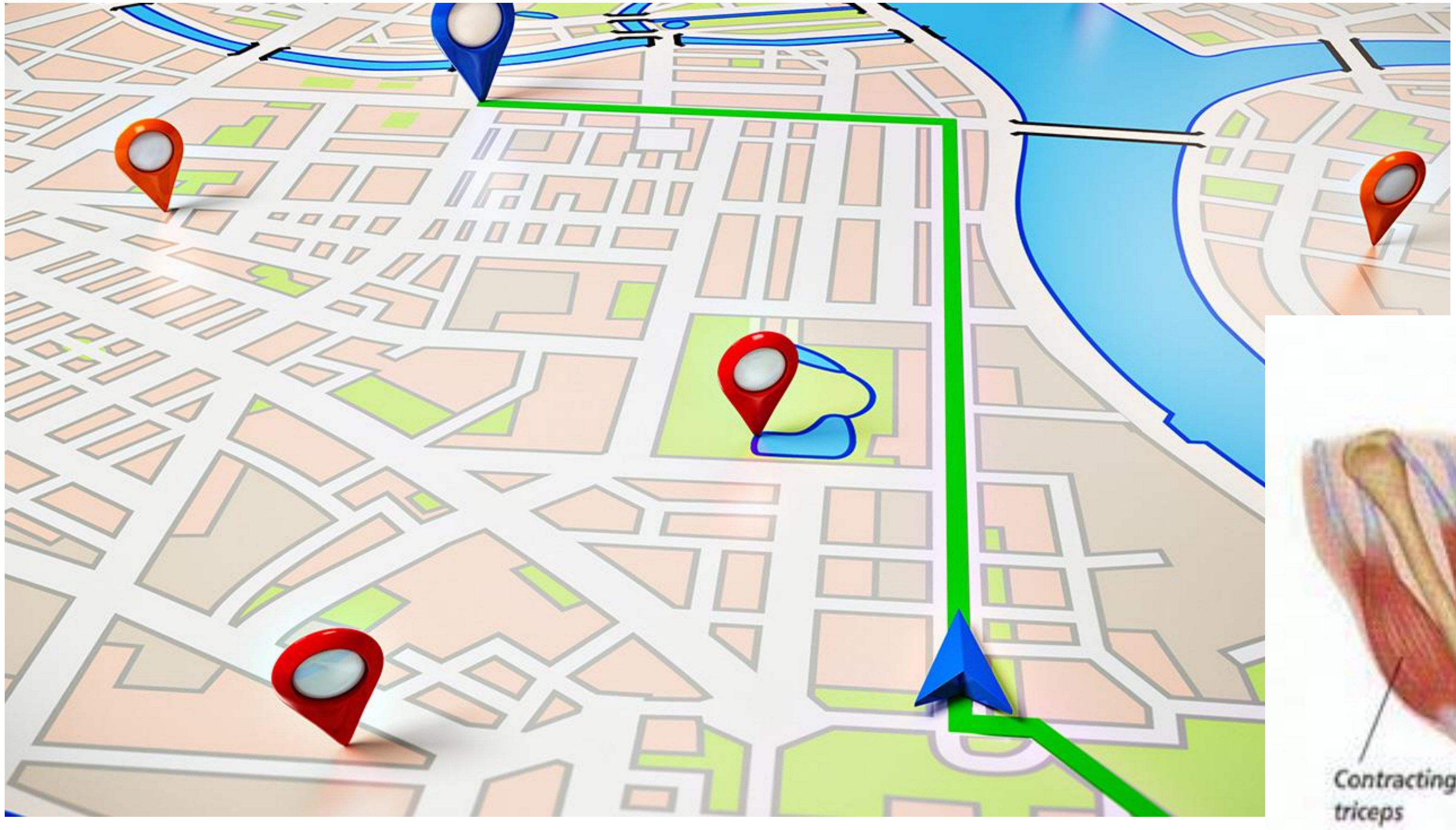
$\sim 1M$ states

(Marom and Rosman, NeurIPS 2018)

Addressing these issues

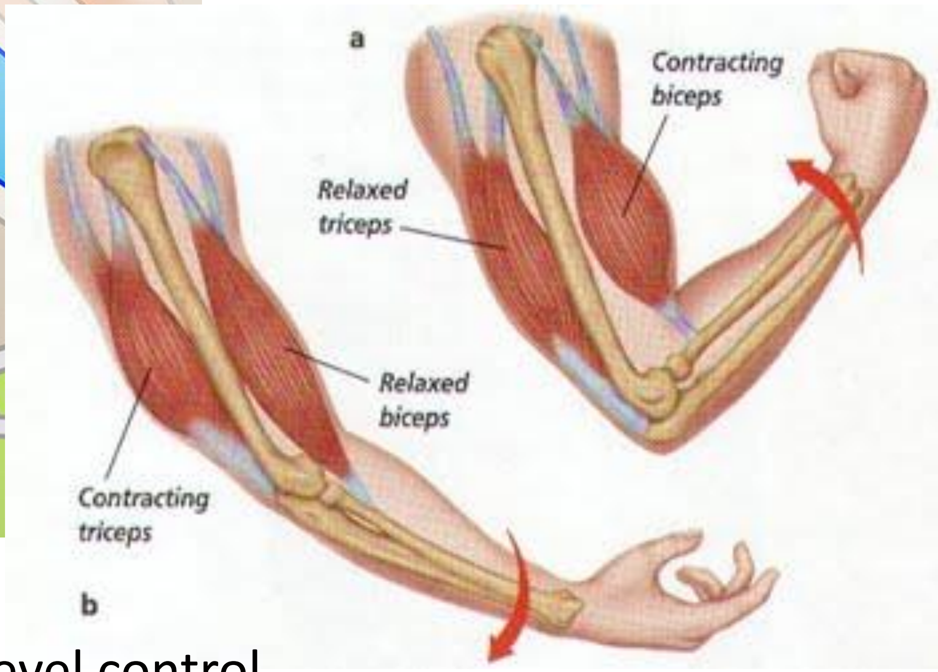
- General ideas:
  - Make predictions of what may happen
  - Exploit structure of the problem
    - Representations of states and transitions
    - **Action spaces**
    - Rewards
  - Reuse knowledge

How do humans do it?



High-level planning

Different levels of abstraction



Low-level control

Hierarchies of skills

- Structure hierarchical control around *skills*
  - Components of behaviour
  - Performs continuous, low-level control
  - Can treat as discrete action

- Behaviour is modular and compositional

Hierarchical RL

- RL typically solves a *single* problem *monolithically*
- Hierarchical RL:
  - Create and use higher-level macro-actions
  - Problem now contains subproblems
  - Each subproblem is also an RL problem
- Several major frameworks look at this problem
  - **Options Framework**: theoretical basis for skill acquisition, learning and planning using higher-level actions (options)

The options framework

- Basic idea:
  - Define a *temporally extended action* as a *policy*

- A (Markov) option *o* is a policy unit:

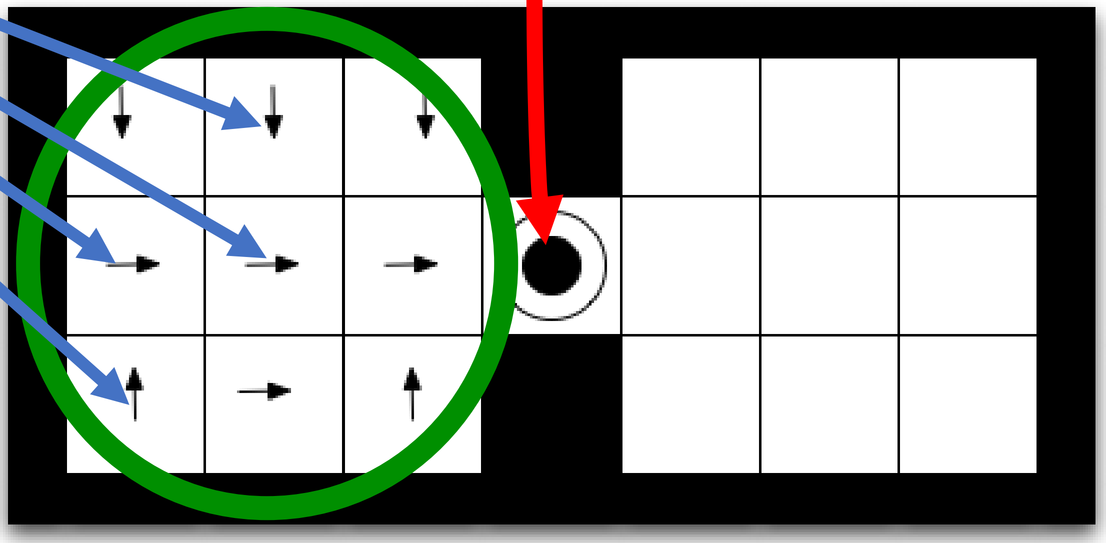
  - Initiation set $I_o : S \to \{0, 1\}$

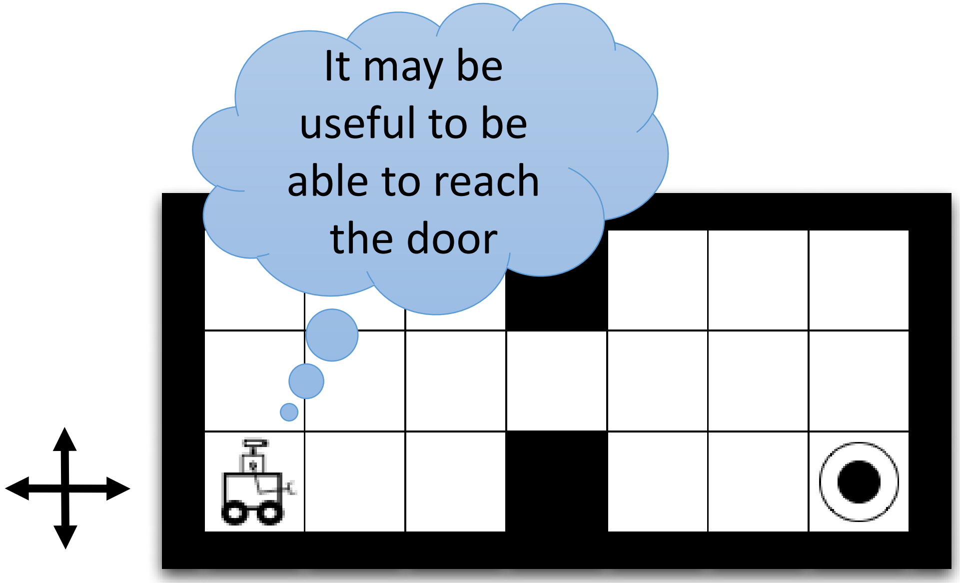
  - A termination probability $\beta_o : S \to [0, 1]$

  - A policy $\pi_o : S \times A \to [0, 1]$

Intuitively

- An option *o* is a policy unit:
  - Initiation set
  - Termination condition
  - Option policy





Non-Markov options

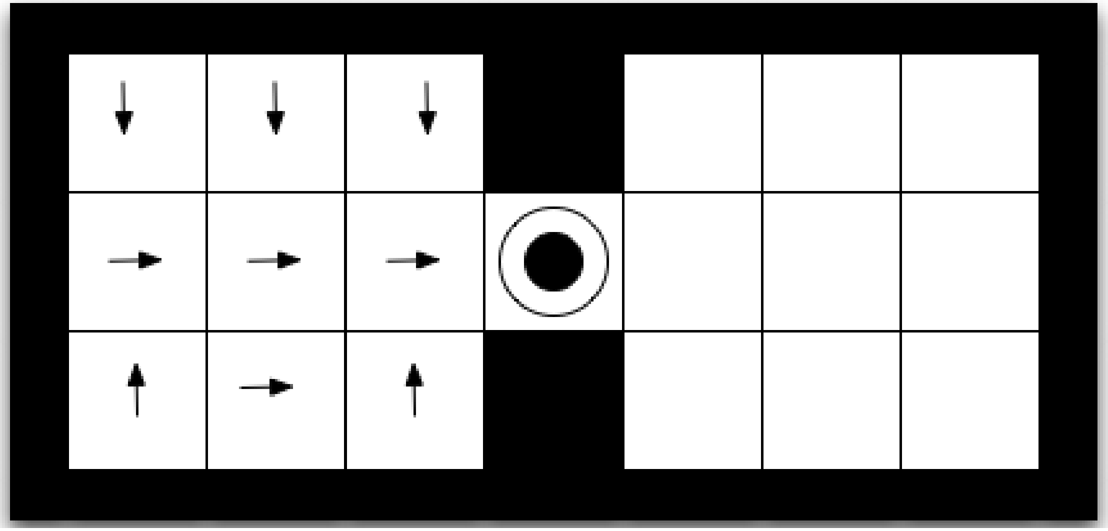
- Non-Markov policy:
  - Not *solely* functions of state
  - Also function of execution history
- Examples of non-Markov options:
  - Run for at most n steps
  - Repeat something n times
  - Any internal state
- Not often used, but can be very useful

Actions are options

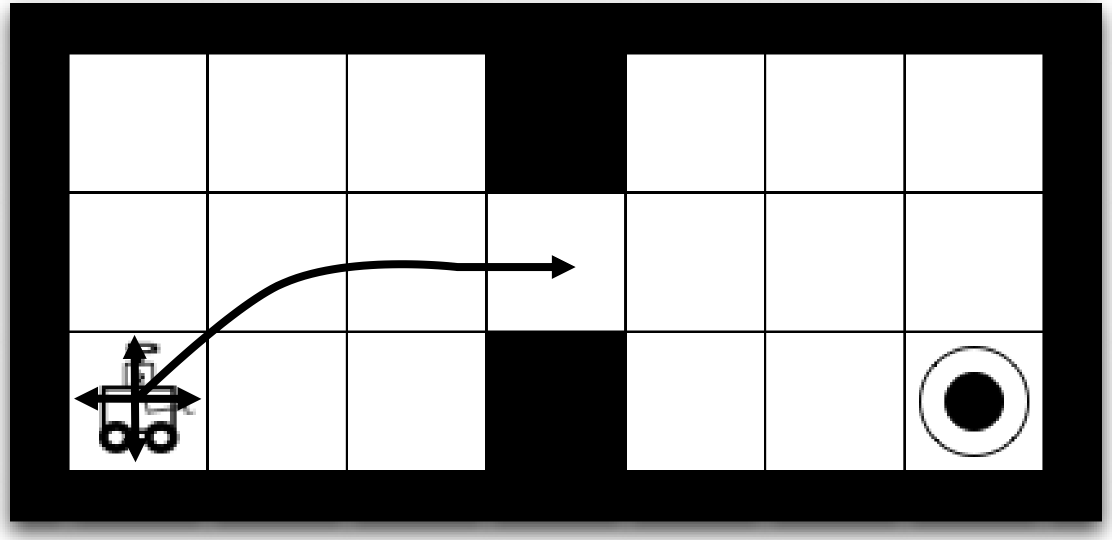
- A primitive action *a* can be represented by an option:
  - $I_a(s) = 1, \forall s \in S$
  - $\beta_a(s) = 1, \forall s \in S$
  - $\pi_a(s, b) = \begin{cases} 1 & a = b \\ 0 & otherwise \end{cases}$
- A primitive action can be executed anywhere, lasts exactly one time step, and always chooses action a

Options as actions

Option



Problem



Questions

- Given an MDP: (S, A, R, T, γ)
- Replace A with a set of options O (some may be primitive actions)

  - How do we characterise the resulting problem?
  - How do we plan using options?
  - How do we learn using options?
  - How do we characterise the resulting policies?
  - How do we learn the options?

SMDPs

- The resulting problem is a *Semi-Markov Decision Process (SMDP)*
- This consists of:
  - S Set of states
  - O Set of options
  - $P(s', t|o, s)$ Transition model
  - $R(s', s, t)$ Reward function
  - γ Discount factor (per step)

- In this case:
  - All times are integers
  - "Semi" here means transitions can last $t > 1$ timesteps
  - Transition and reward functions involve time taken for option to execute

The Bellman Equation for SMDPs

- Return to the Bellman equation:

$$\boxed{Q^\pi(s,o)} = \boxed{\mathbb{E}_{t,s'}[R(s',s,t)]} + \boxed{\mathbb{E}_{t,s'}[\gamma^t \pi(s',o') \boxed{Q^\pi(s',o')}]}$$

value of o in s

immediate reward

expected future value

value of next o' in next s'

- where:

$$\mathbb{E}_{t,s'}[R(s',s,t)] = \sum_{t,s'} P(s',t|o,s)R(s',s,t)$$

Note we are averaging over time: how long does the option run for?

$$\mathbb{E}_{t,s'}[\gamma^t \pi(s',o')Q^\pi(s',o')] = \sum_{t,s'} P(s',t|o,s)\gamma^t \pi(s',o')Q^\pi(s',o')$$

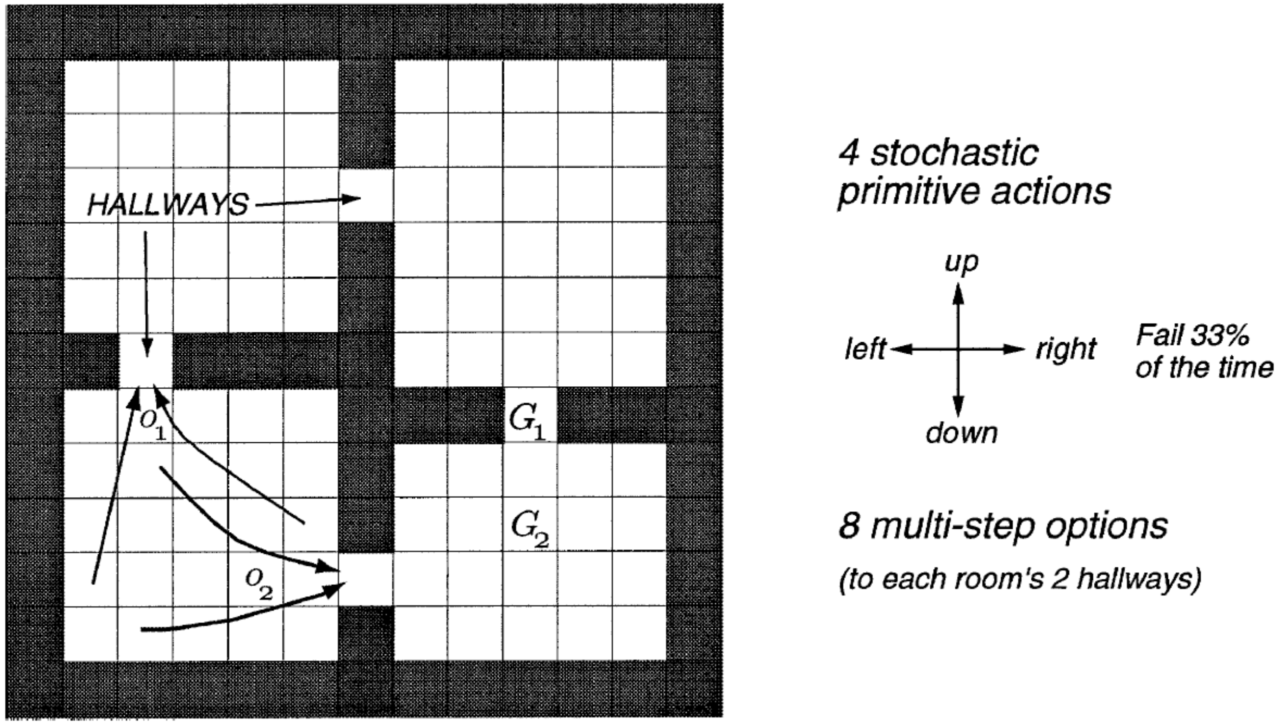
Learning and planning

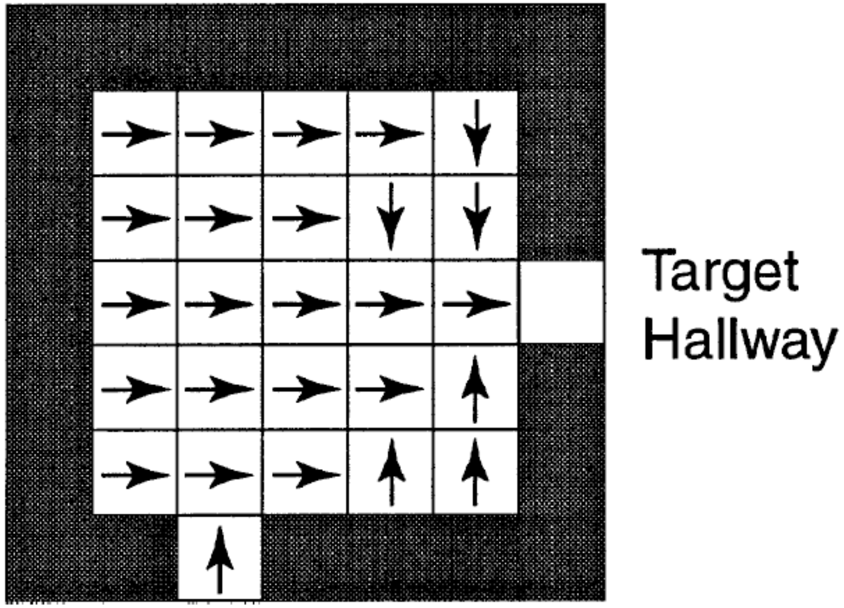
$$Q^\pi(s,o) = \mathbb{E}_{t,s'}[R(s',s,t)] + \mathbb{E}_{t,s'}[\gamma^t \pi(s',o')Q^\pi(s',o')]$$

- For learning:
  - Stochastic samples
  - Use SMDP Bellman equation

- For planning:
  - Synchronous Value Iteration
  - Value Iteration using the SMDP Bellman Equation

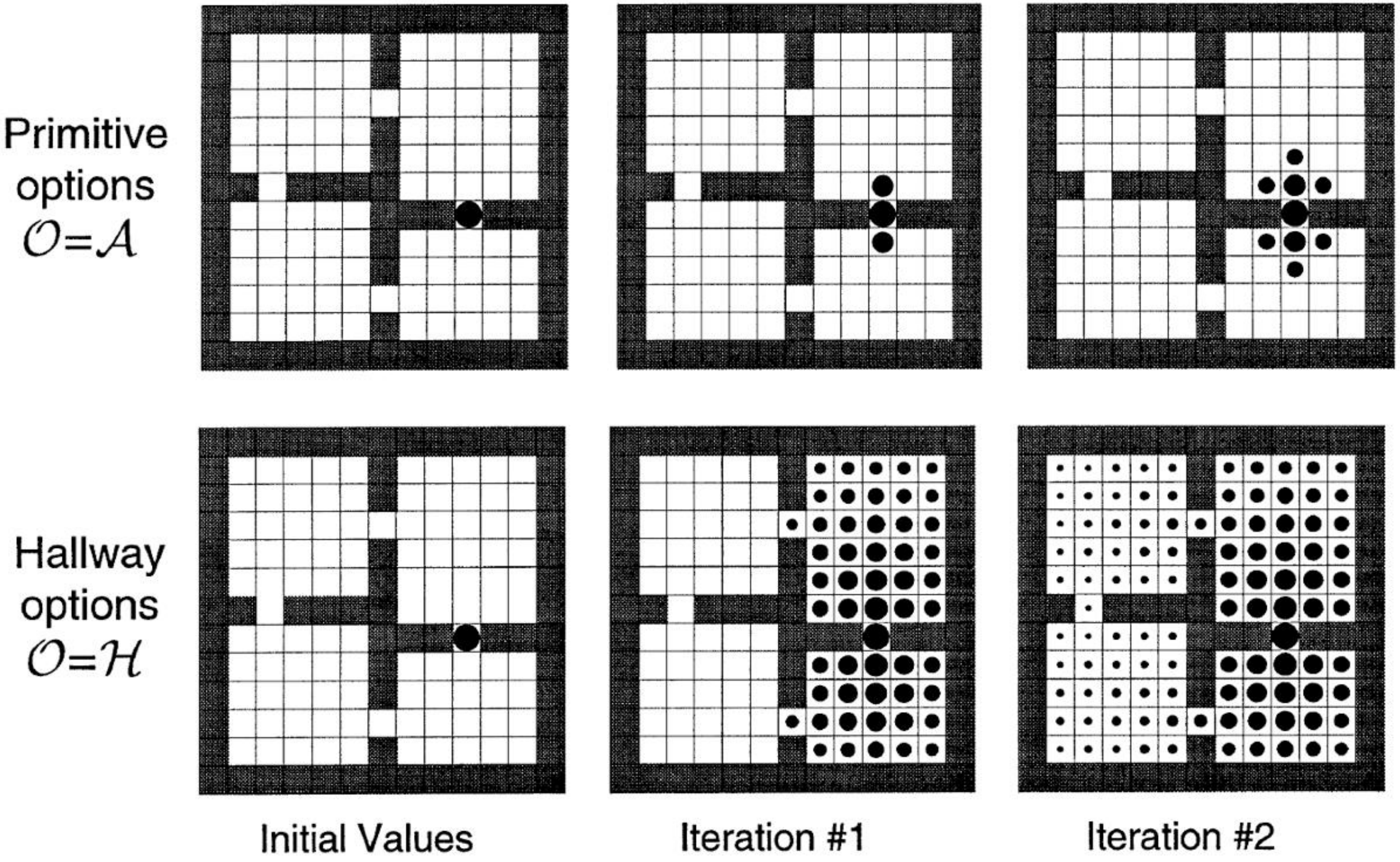
Example





(Sutton, Precup and Singh, AIJ 1999)

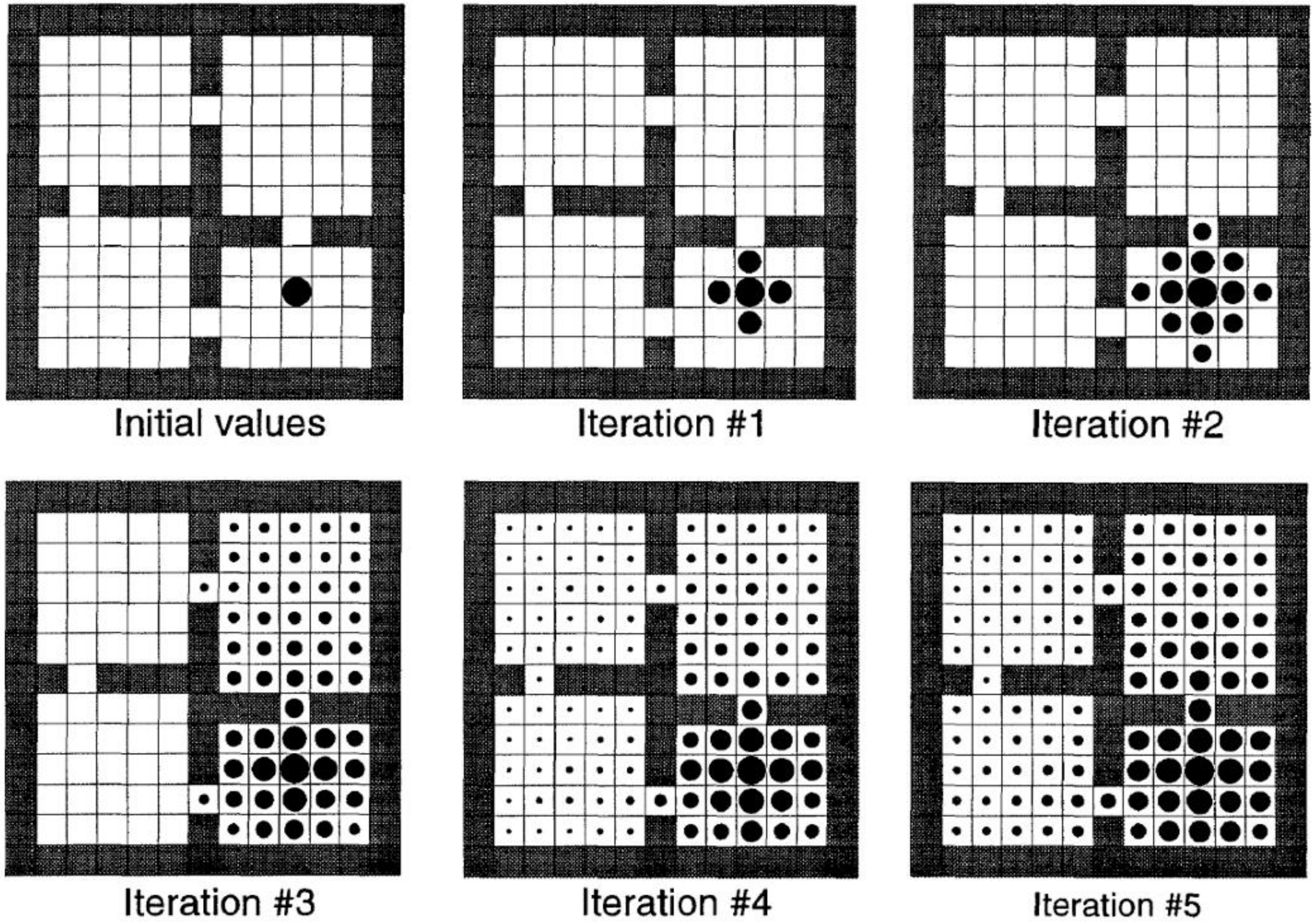
Example



(Sutton, Precup and Singh, AIJ 1999)

Example

Primitive and hallway options $\mathcal{O}=\mathcal{A}\cup\mathcal{H}$



(Sutton, Precup and Singh, AIJ 1999)

A note on policies

- A policy over an MDP with primitive actions is a *Markov policy*:

$$\pi : S \times A \to [0, 1]$$

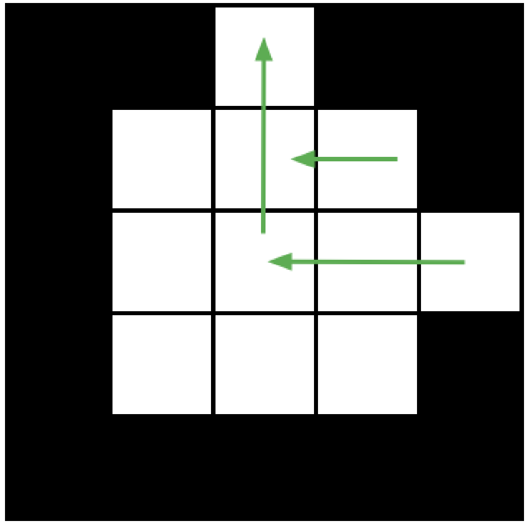
- A policy over an MDP with options could also be Markov:

$$\pi : S \times O \to [0, 1]$$

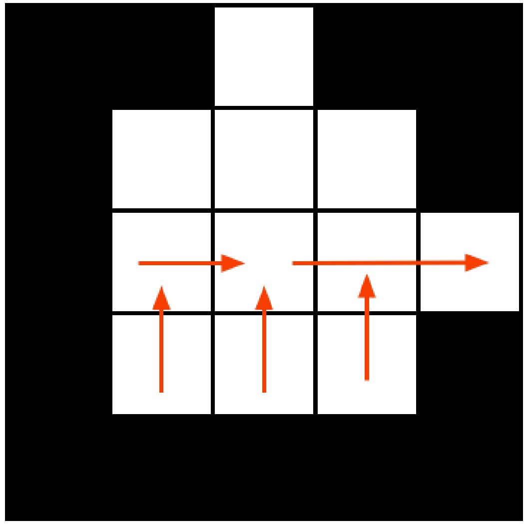
- This could imply a policy in the original MDP that is not, because the **probability of taking an action at a state depends on the option currently running**.

Example

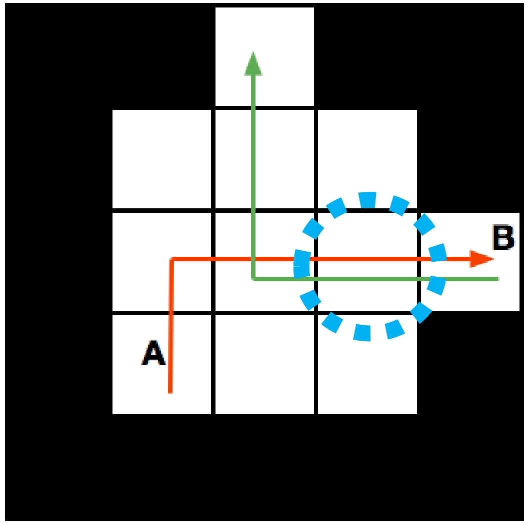
- Consider where two options overlap in s but disagree on a



Option A



Option B



Policy

Semi-Markov policies

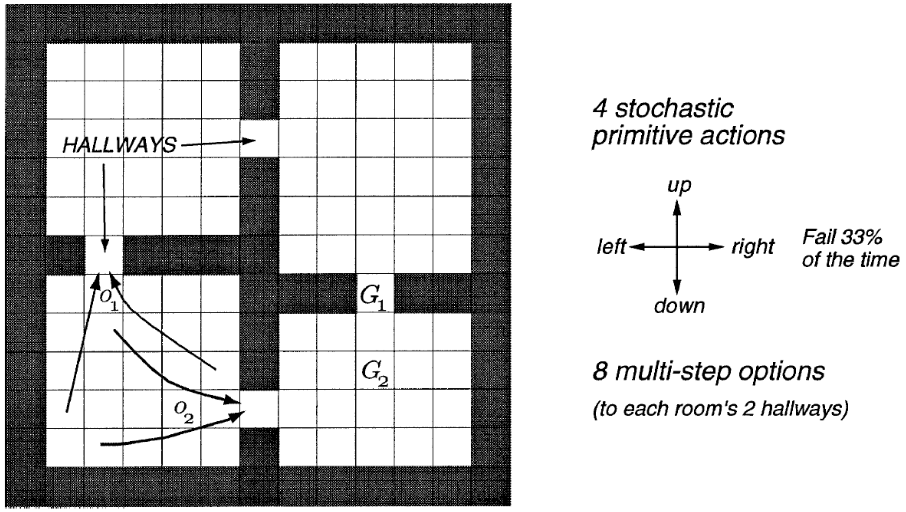
- A Markov policy for an SMDP may result in a *semi-Markov* policy for the underlying MDP
- (Even if the options are Markov options!)
- *Here, semi-Markov means that the probability of taking a primitive action at each step depends on more than the current state*

Summary

- Original problem: MDP
- MDP + Options = SMDP
- Options framework allows us to both *express a low-level policy*, and *plan and learn using the higher-level SMDP*
- Additionally, the ability to:
  - Create new options
  - Update option policies
  - Learn with options
  - Interrupt them ...

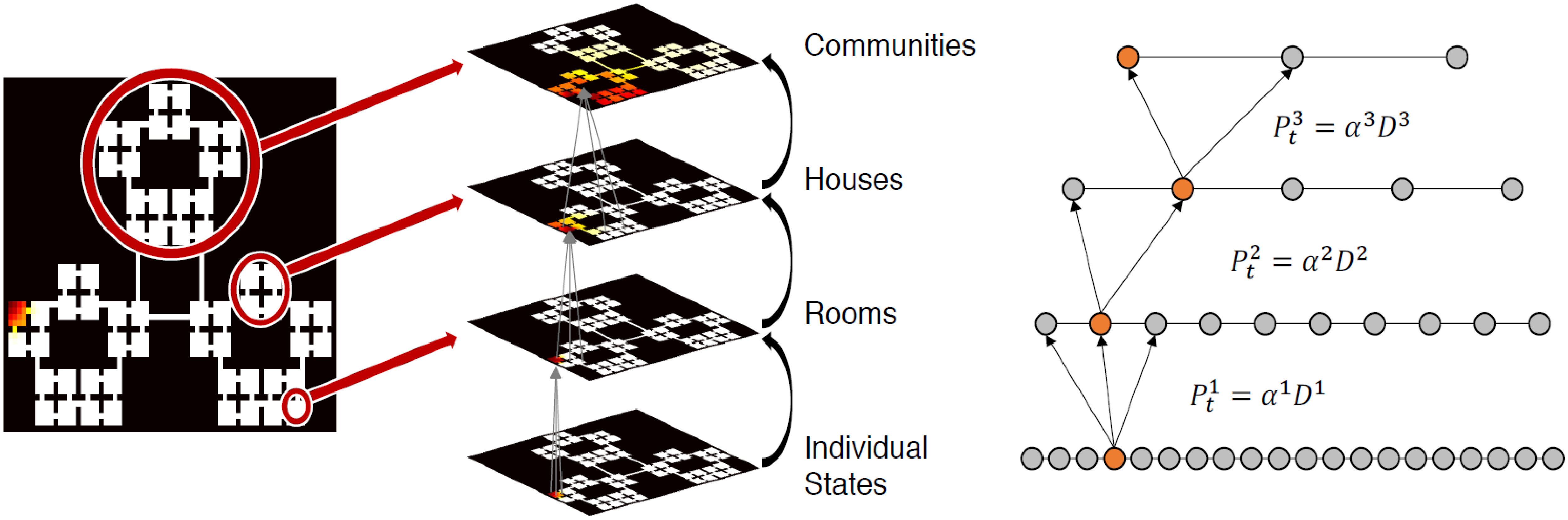
What are skills for?

- Adding an option changes the connectivity of the MDP
- This affects:
  - Learning and planning
  - Exploration
  - State-visit distribution
  - Branching factor



(Sutton, Precup and Singh, AIJ 1999)

Hierarchies of skills



(Earle, Saxe and Rosman, ICLR 2018)

Addressing these issues?

- General ideas:
  - Make predictions of what may happen
  - Exploit structure of the problem
    - Representations of states and transitions
    - Action spaces
    - Rewards
  - **Reuse knowledge**

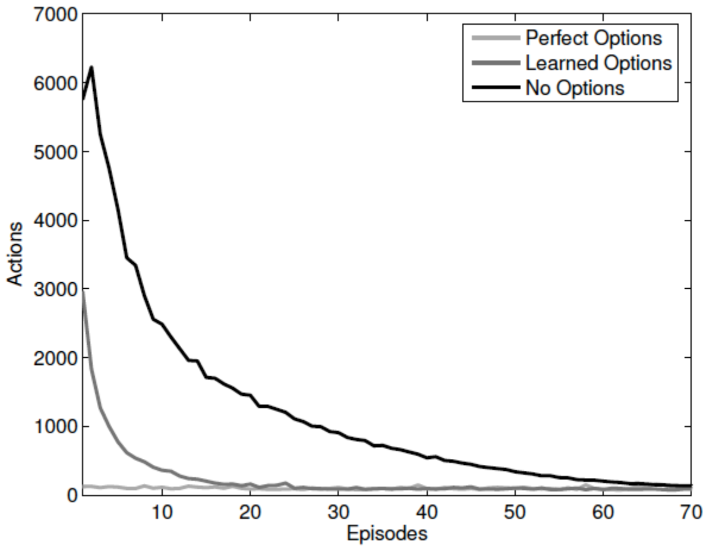
Transfer

- Use experience gained while solving one problem to improve performance in another
  - Map from one (or more) *source* task to one (or more) *target* task
  - Assume tasks drawn from some distribution
- Skill transfer:
  - Use options as mechanism for transfer
  - Transfer *components* of solution
  - Can drastically improve performance
  - Bootstrapping performance

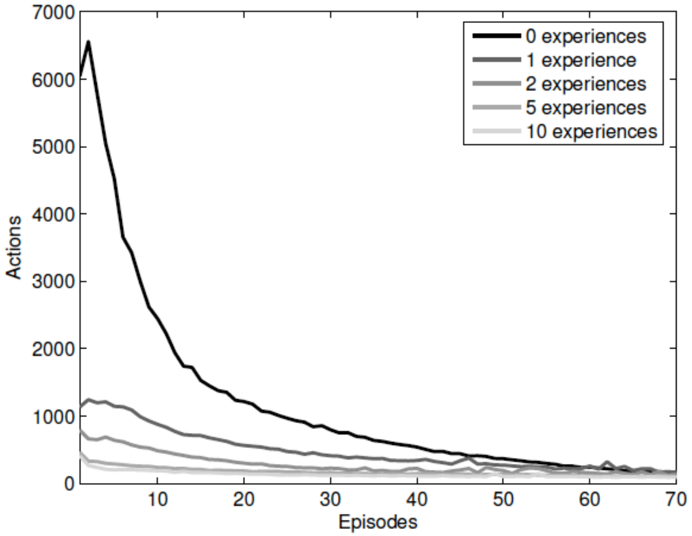
- General principle: subtasks recur across problems

Example

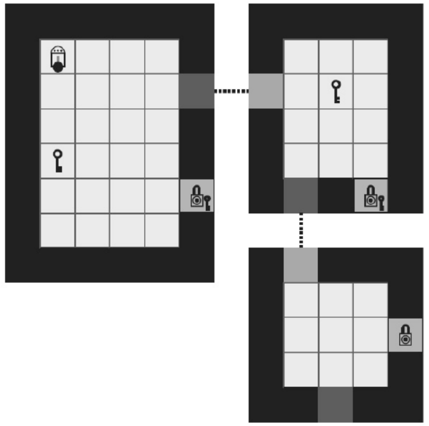
- Tasks drawn from parametrized family
  - Common features present
  - Options defined using only common features



(a) Learning curves for agents with problem-space options.



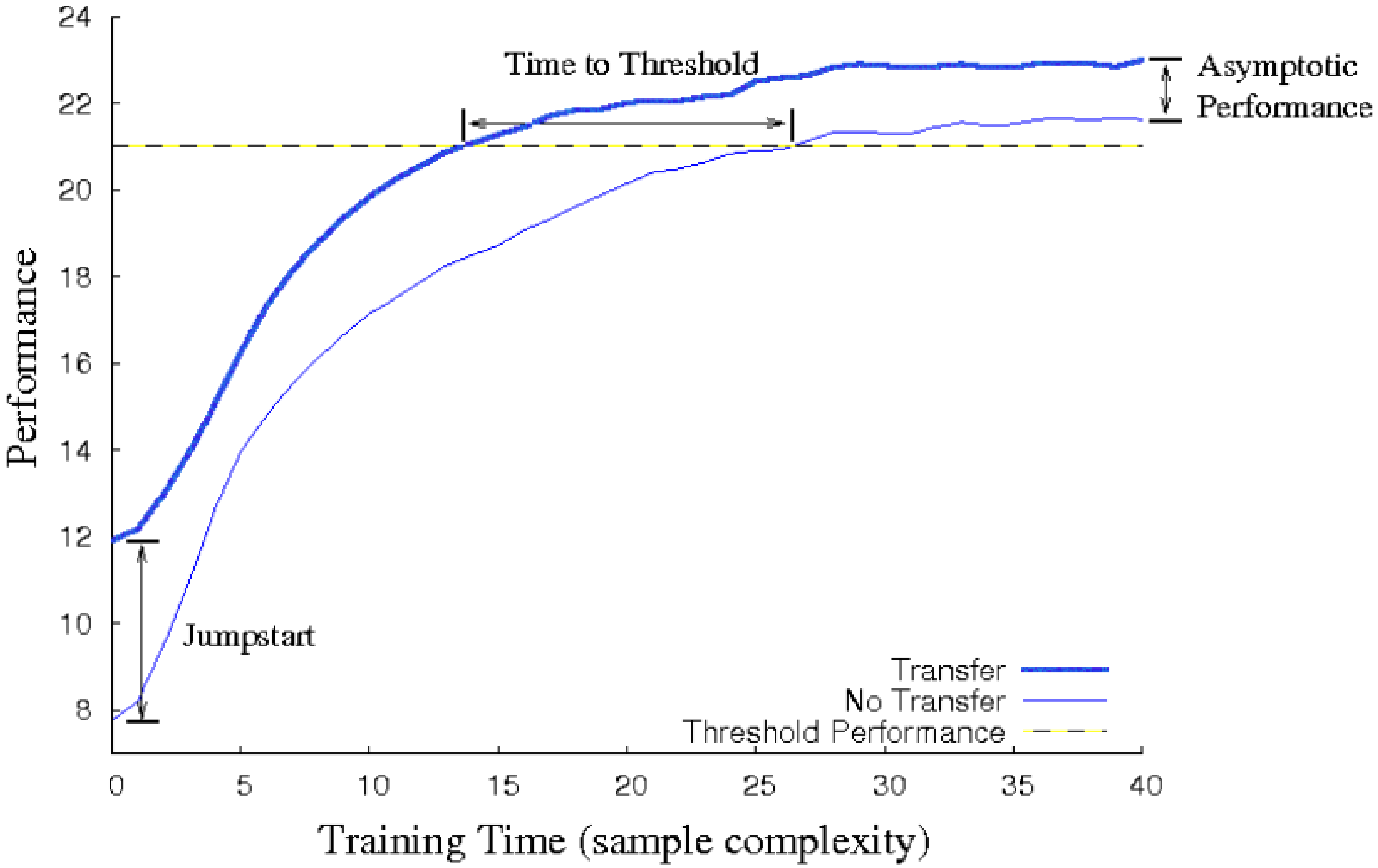
(b) Learning curves for agents with agent-space options, with varying numbers of training experiences.



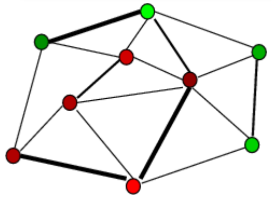
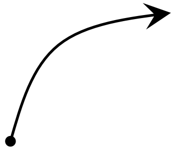
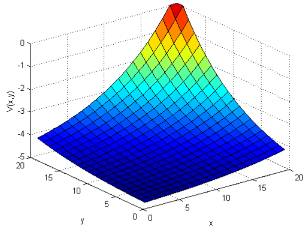
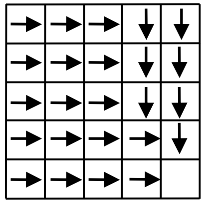
(Konidaris and Barto, IJCAI 2007)

Why do we need transfer?

- Improve performance over "regular" learning
  - Sample complexity
  - Jumpstart
  - Learning speed
  - Time to threshold
  - Asymptotic performance



Transferred knowledge

Structural Transfer		*Experience Transfer*		
Task Representation	Solution Representation	Samples	Value Function	Policy
				

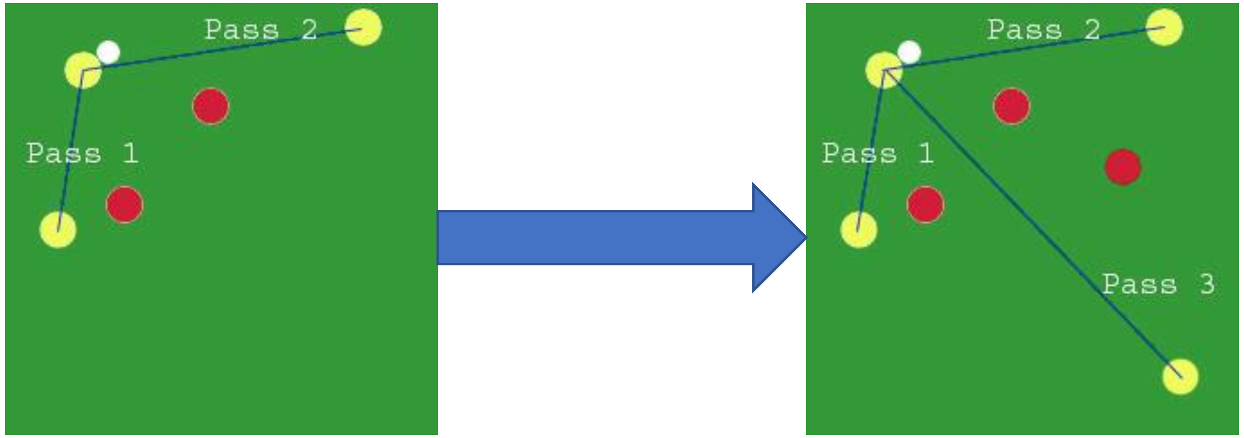
- Task representation
  - –Action space (e.g., options, task decomposition)
  - –Reward function
- Solution representation
  - –Basis functions

- Samples
  - –Collected through direct exploration
- Value function / policy
  - –Solution initialisation

Why structure and transfer?

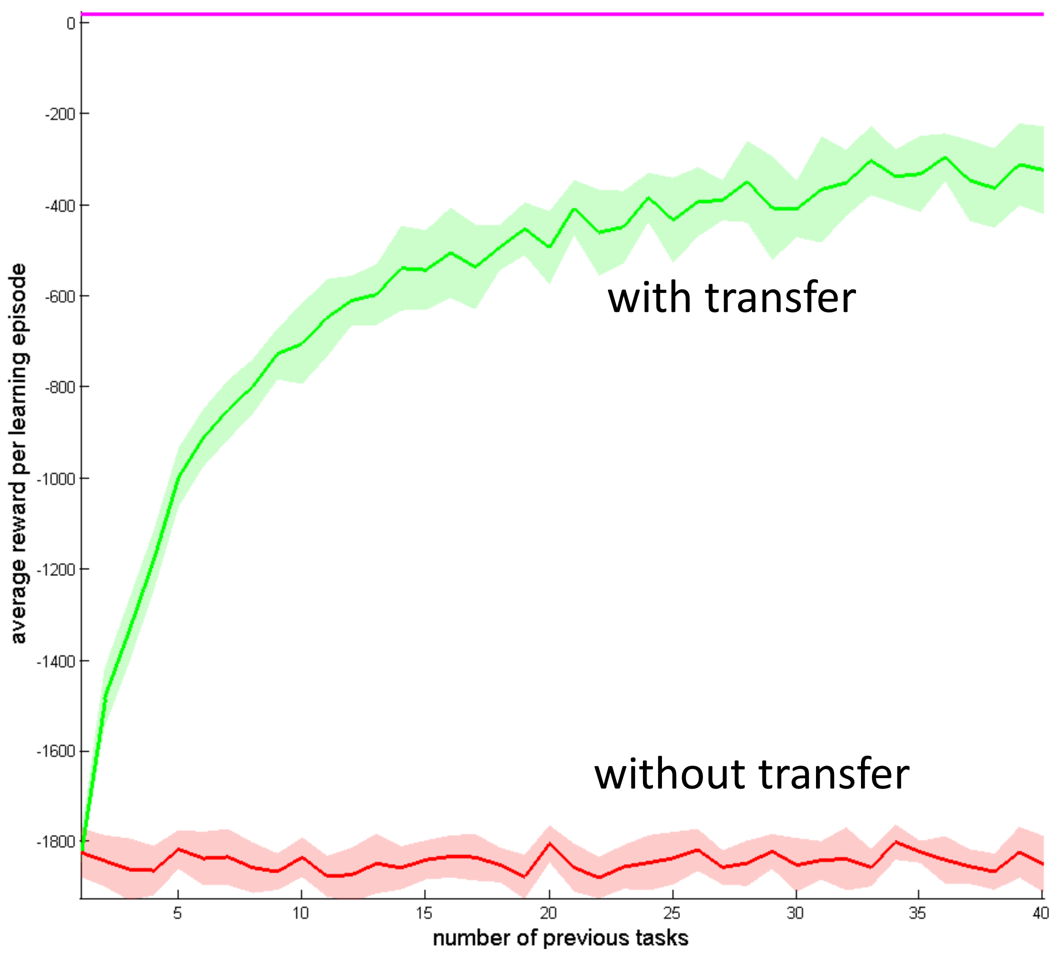
- Multitask learning
  - Multipurpose (robot) systems

- Solving large scale problems
  - Decompose into a curriculum



Why structure and transfer?

- Long term benefits
- Life-long learning



(Rosman and Ramamoorthy, TAMD 2015)

Conclusion

- Difficulties of delayed rewards, long action sequences, …
- Model-based RL
  - Dyna-Q
  - MCTS
- Action abstraction
  - Options framework
- Transfer and multitask learning