Learning in a Small World

B. Ravindran

Reconfigurable and Intelligent Systems Engineering Group

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Overview

Reinforcement Learning

- Stochastic optimization to solve sequential decision problems
- Hierarchical Decomposition to speed up solution
  - Reuse solutions
  - Reduce number of decisions needed
- What are the appropriate "sub-problems" to learn?
  - Past work uses ideas from social network analysis, e.g. betweenness

Small World Networks

- Interested in distributed navigation property
- "Long range links" inversely proportional to some power of span
  - Depends on order of lattice

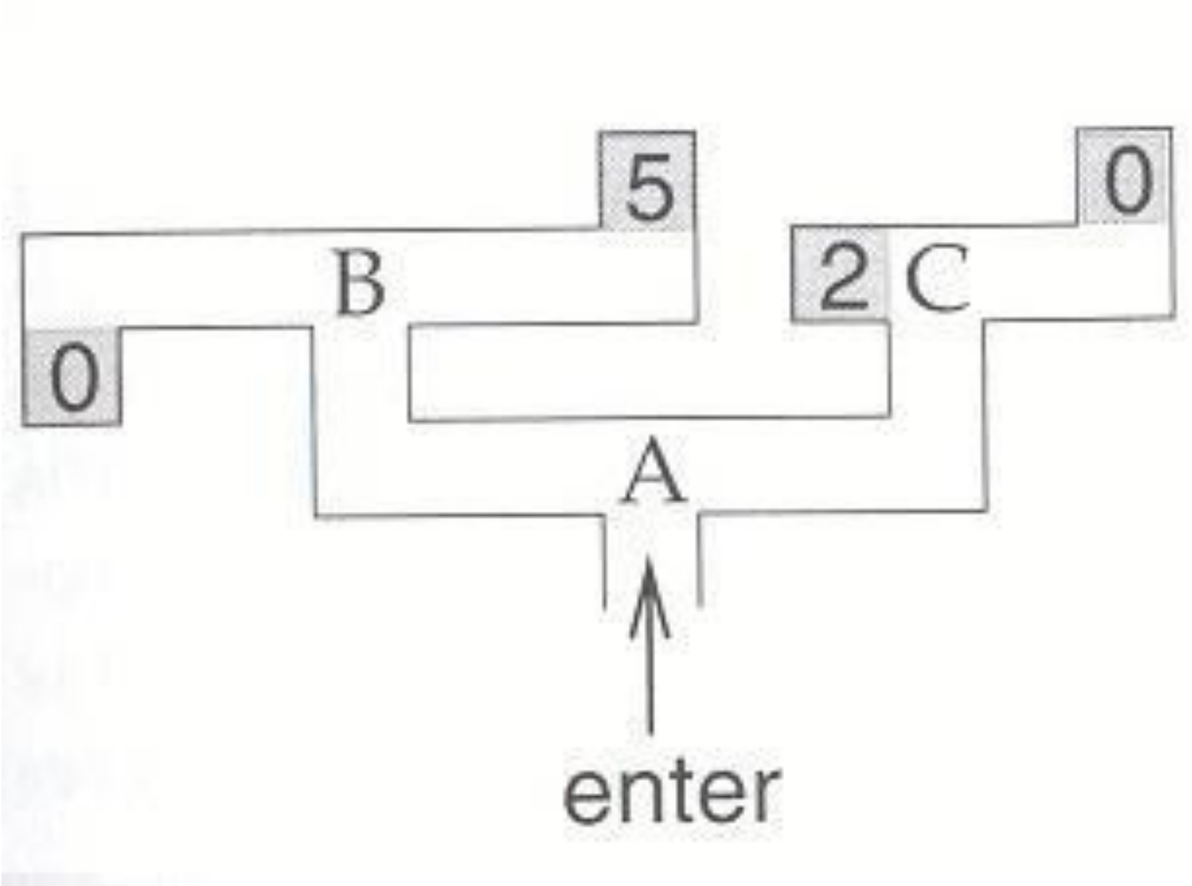
This Work

- Solutions to sub-problems are equivalent to long-range links
- Create small-world decision problems
- Efficient Learning

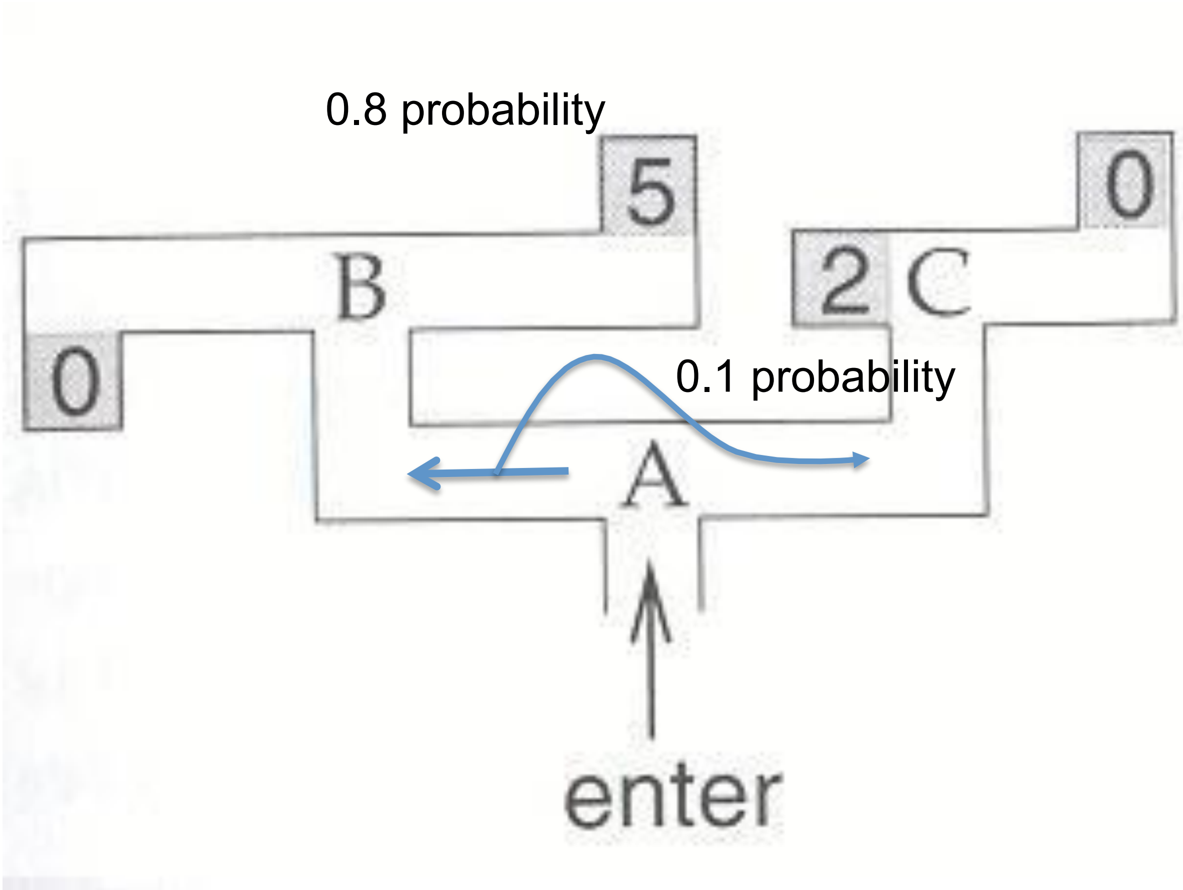
What is Reinforcement Learning?

- Learning about stimuli and actions based on rewards and punishments alone.
- No detailed supervision available
- Trial-and-error learning
- Delayed rewards
- Sequence of actions required to obtain reward
- Associative learning required
  - Need to associate actions to states
- Learn about policies not just actions
- Typically in a stochastic world

Running a Maze Dayan and Abbott



Running a Stochastic Maze Dayan and Abbott



The Agent Learns a Policy

Policy at step t, π_t :

a mapping from states to action probabilities

$$\pi_t(s,a) = \text{ probability that } a_t = a \text{ when } s_t = s$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

The Markov Property

- "the state" at step *t*, means whatever information is available to the agent at step *t* about its environment.
- The state can include immediate "sensations," highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the **Markov Property**:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} = \Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**:

$$P_{ss'}^{a} = \Pr\left\{ s_{t+1} = s' \mid s_t = s, a_t = a \right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

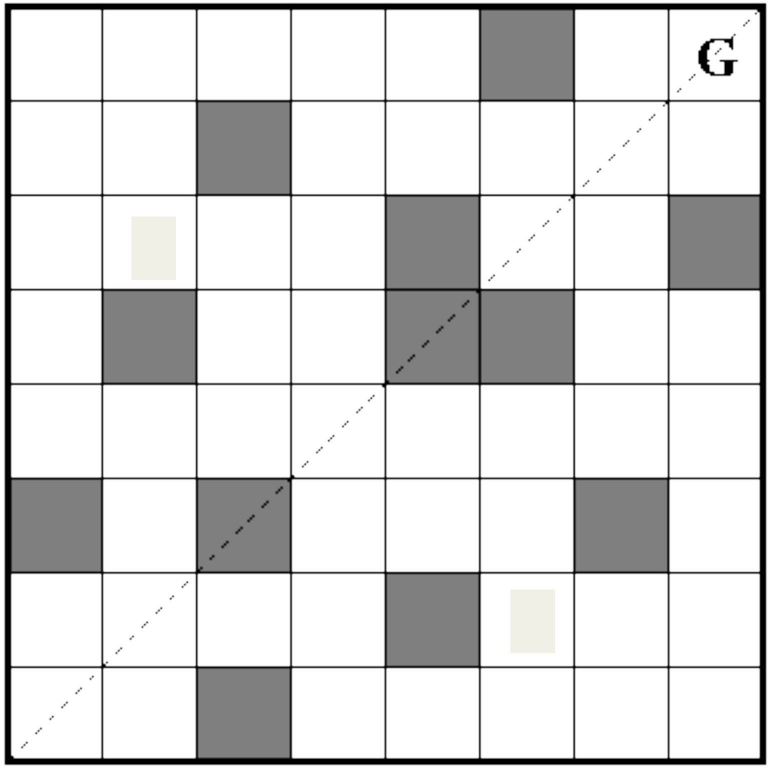
  - **reward expectations**:

$$R_{ss'}^{a} = E\left\{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \right\} \quad \text{for all } s, s' \in S, a \in A(s).$$

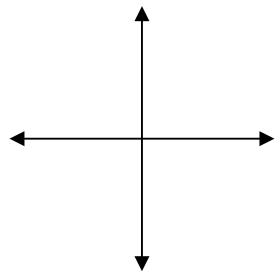
Markov Decision Processes

- MDP, *M,* is the tuple: $M = \langle S, A, \Psi, P, R \rangle$
  - *S* : set of states.
  - *A* : set of actions.
  - $\Psi \subseteq S \times A$: set of admissible state-action pairs.
  - $P : \Psi \times S \rightarrow [0,1]$: probability of transition.
  - $R : \Psi \rightarrow \Re$: expected reward.
- Policy $\pi : S \rightarrow A$ (can be stochastic)
- Maximize total expected reward

Example



$$M = \langle S, A, \Psi, P, R \rangle$$



Value Functions

- The **value of a state** (action) is the expected long-term reward starting from that state (action); depends on the agent's policy:

State - value function for policy π :

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

Action - value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s,\, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s,\, a_t = a\right\}$$

From Reinforcement Learning: An Introduction.
Sutton and Barto

Optimal Value Functions

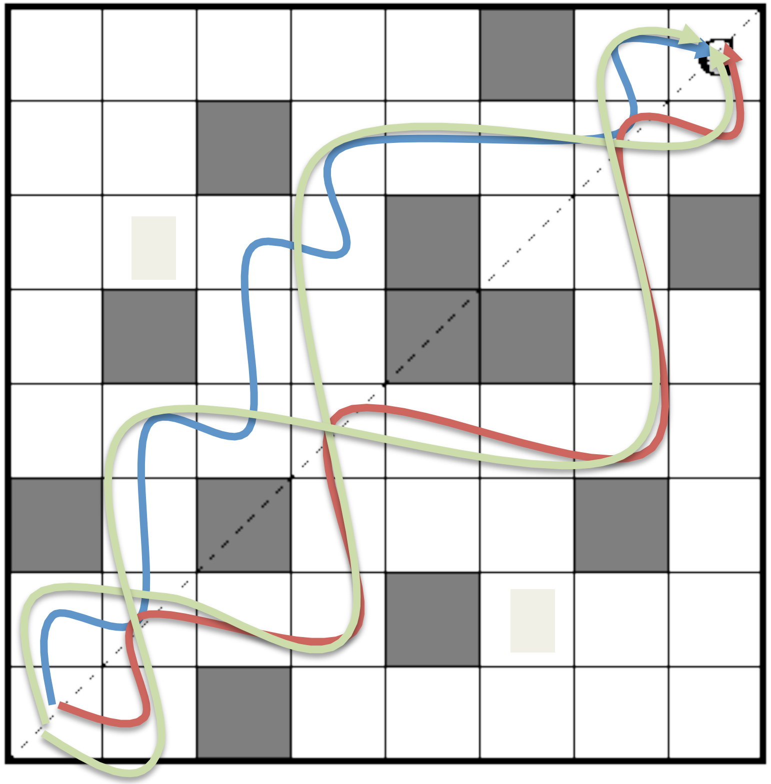
- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

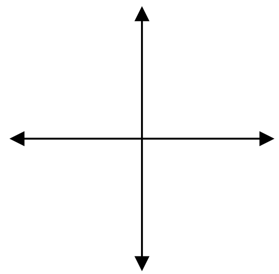
- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an optimal policy. We denote them all π^*.
- Optimal policies share the same optimal state-value function:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all} \quad s \in S$$

Example



$$M = \langle S, A, \Psi, P, R \rangle$$



Bellman Optimality Equation for *V**

$$V^*(s) = \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\}$$
$$= \max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^*(s') \right]$$

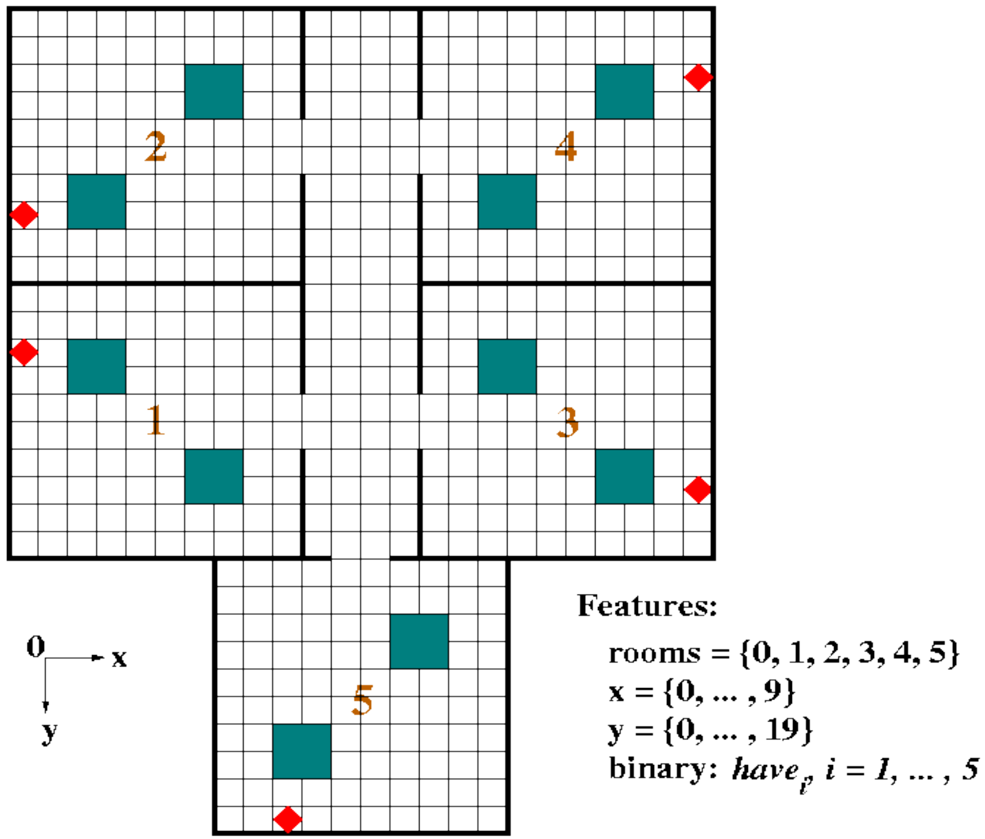
V^* is the unique solution of this system of nonlinear equations.

Similar results applicable to Q^*

Solution Methods

- Temporal Difference Methods
  - TD(λ)
  - Q-learning
  - SARSA
  - Actor-Critic
- Policy Search
  - Policy Gradient Methods
  - Evolutionary algorithms
- Stochastic Dynamic Programming

Sub-goals based decomposition



- Task is to collect all objects in the world
- 5 sub-tasks – one for each room.

Hierarchical Reinforcement Learning

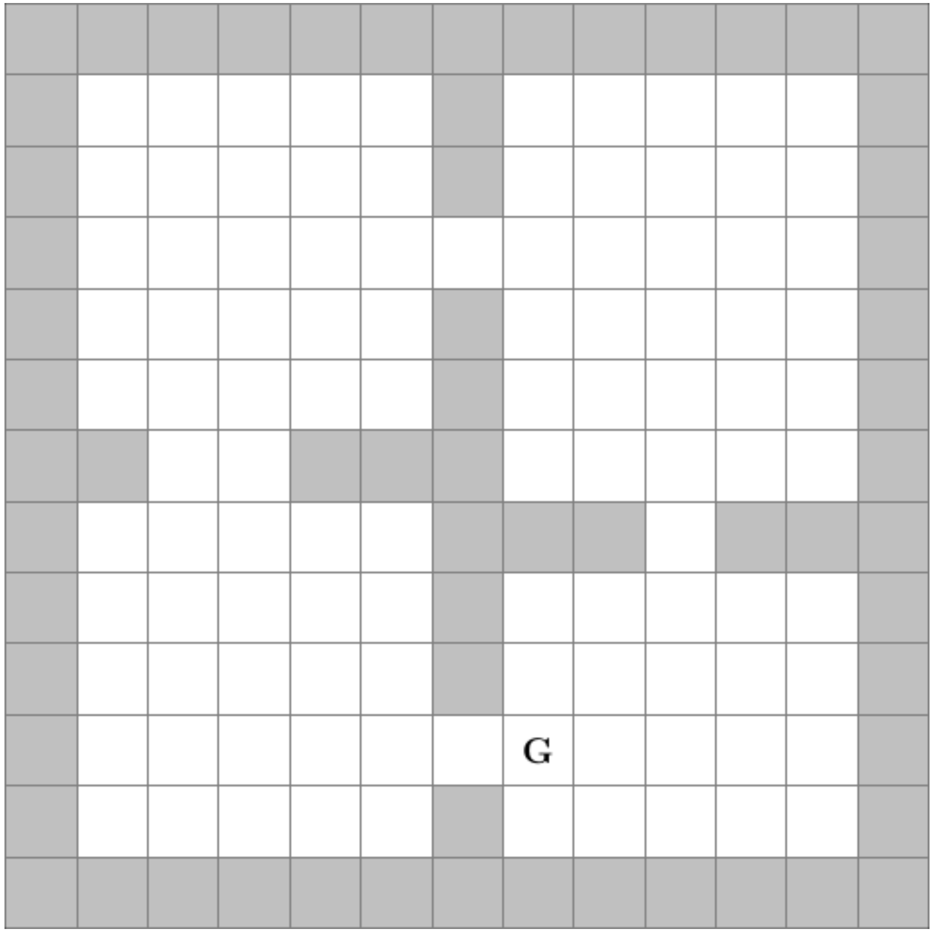
Options (Sutton, Precup, & Singh, 1999):

An option is a triple $o = < I, \pi_o, \beta >$

- $I \subseteq S$ is the set of states in which o may be started
- $\pi_o : \Psi \rightarrow [0,1]$ is the (stochastic) policy followed during o
- $\beta : S \rightarrow [0,1]$ is the probability of terminating in each state

Generalising over Tasks

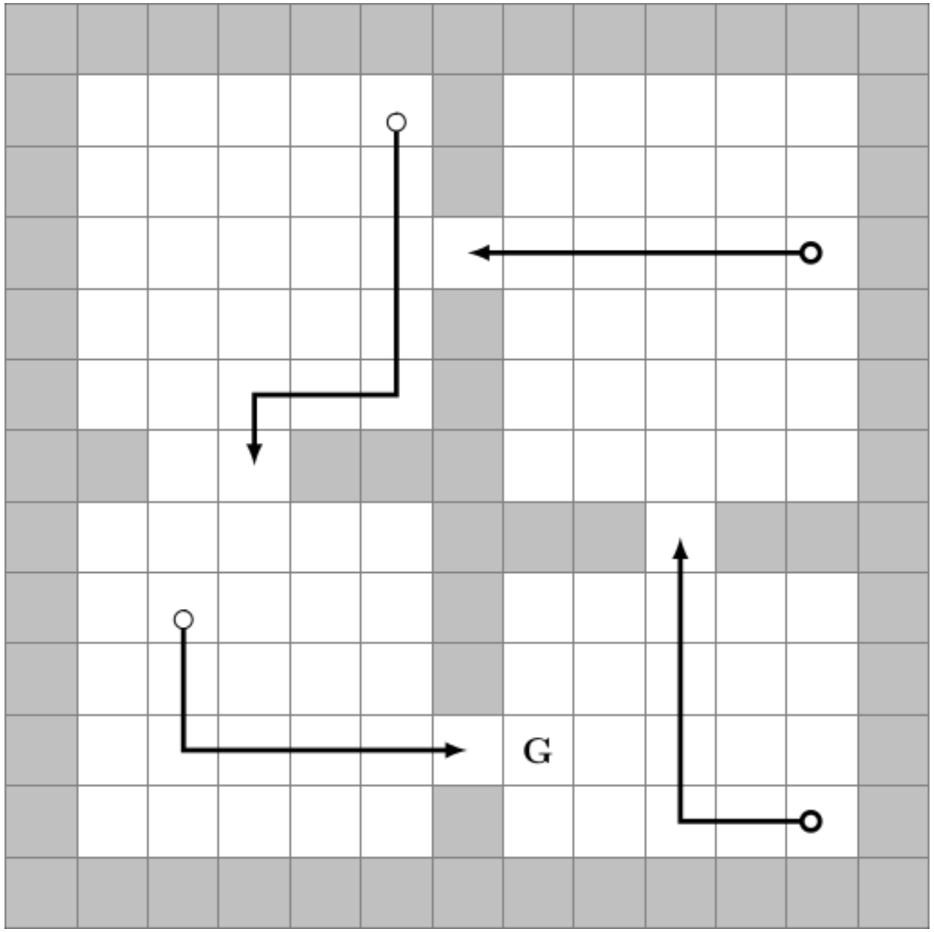
- Each task has a different reward structure in the state space



Generalising over Tasks

- Each task has a different reward structure in the state space

- Options provide a model for subtasks.



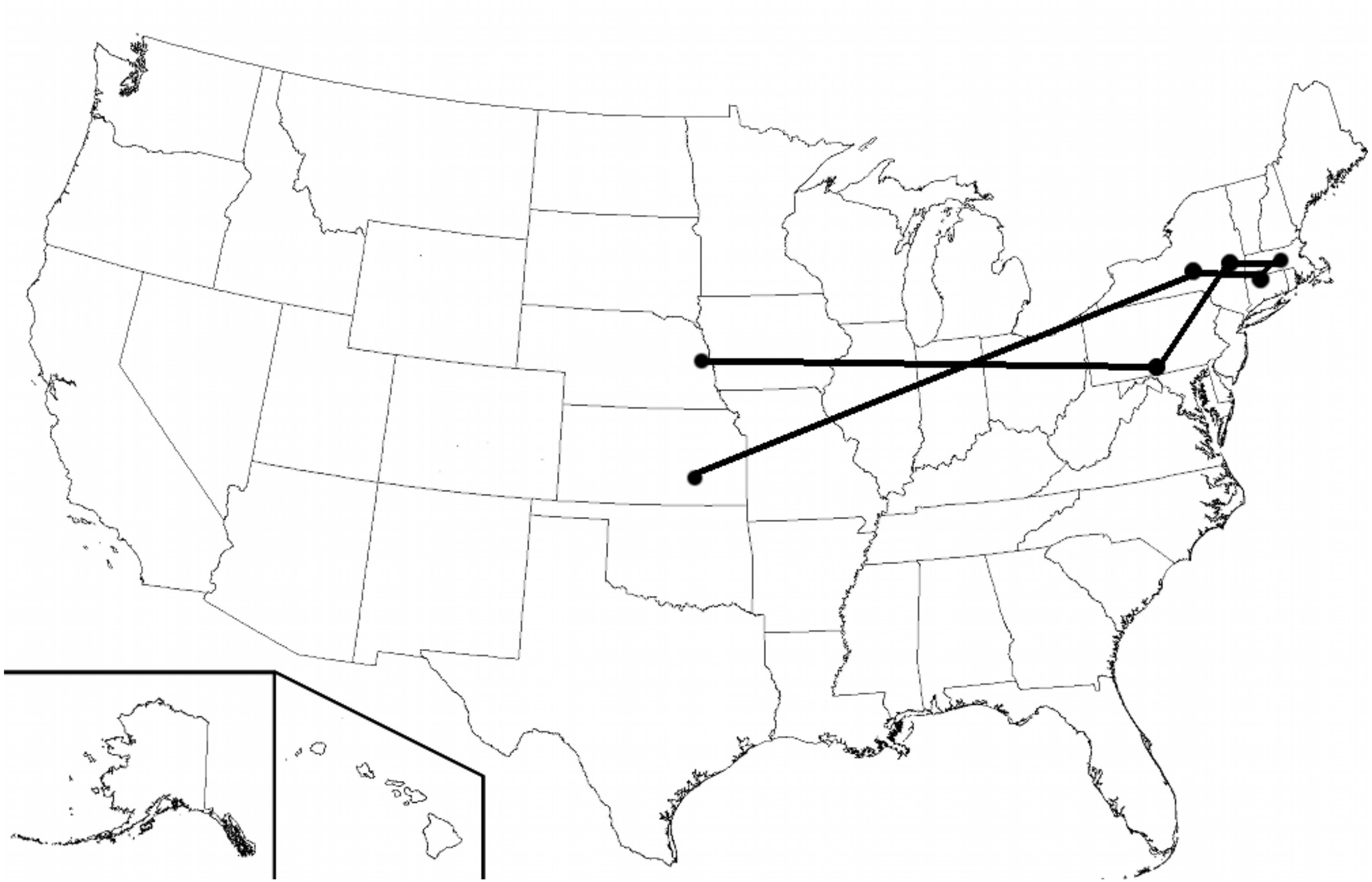
Discovering skills

- Bottlenecks [McGovern & Barto, Stolle & Precup, etc.]
- Graph partitions [Mannor et al., Mathew et al.]
- Betweenness [Simsek & Barto]
- Frequency of changes [Jonsson & Barto, Hengst]
- Bisimulation metrics [Castro & Precup]
- Our Hypothesis: Choose options so that navigating greedily using the Q function would be more effective
  – Turn the MDP into a "small" world

Small Worlds

- A network model where every node is reachable in $O(\log(n))$ hops from a node.
- More importantly, the $O(\log(n))$ path can be discovered using a decentralised algorithm.
- Stanley Milgram's Experiment

Small Worlds

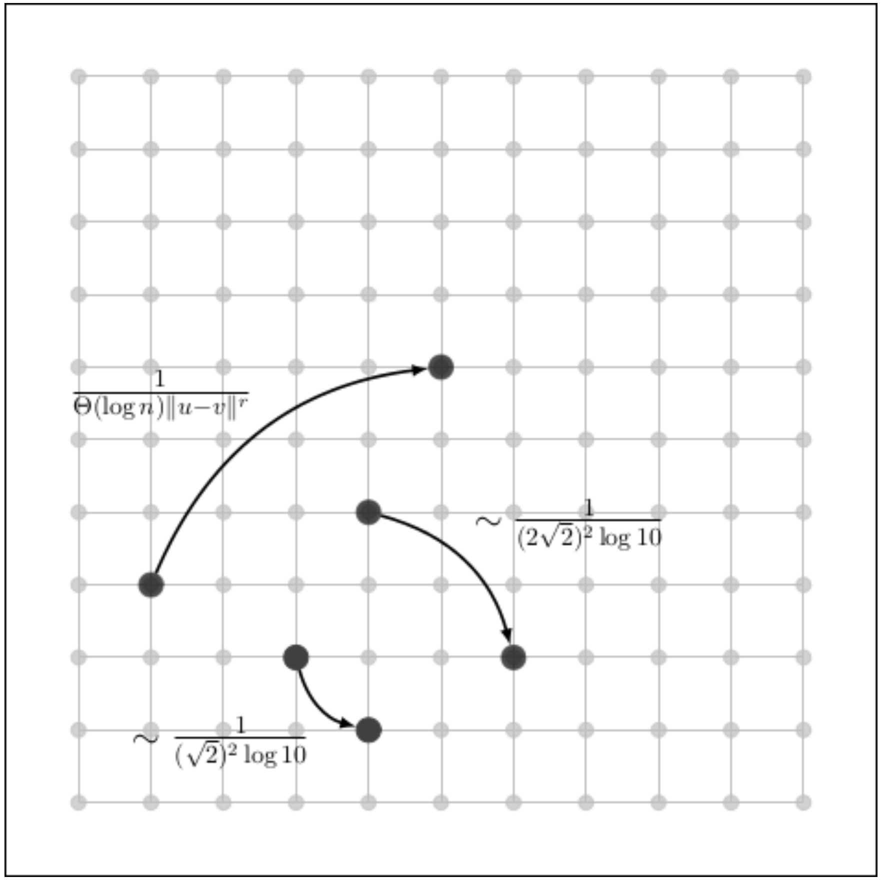


Small Worlds

- A network model where every node is reachable in O(log(n)) hops from a node.
- More importantly, the O(log(n)) path can be discovered using a decentralised algorithm.
- Stanley Milgram's Experiment
- Kleinberg's key insight: Power-law distributed neighbours

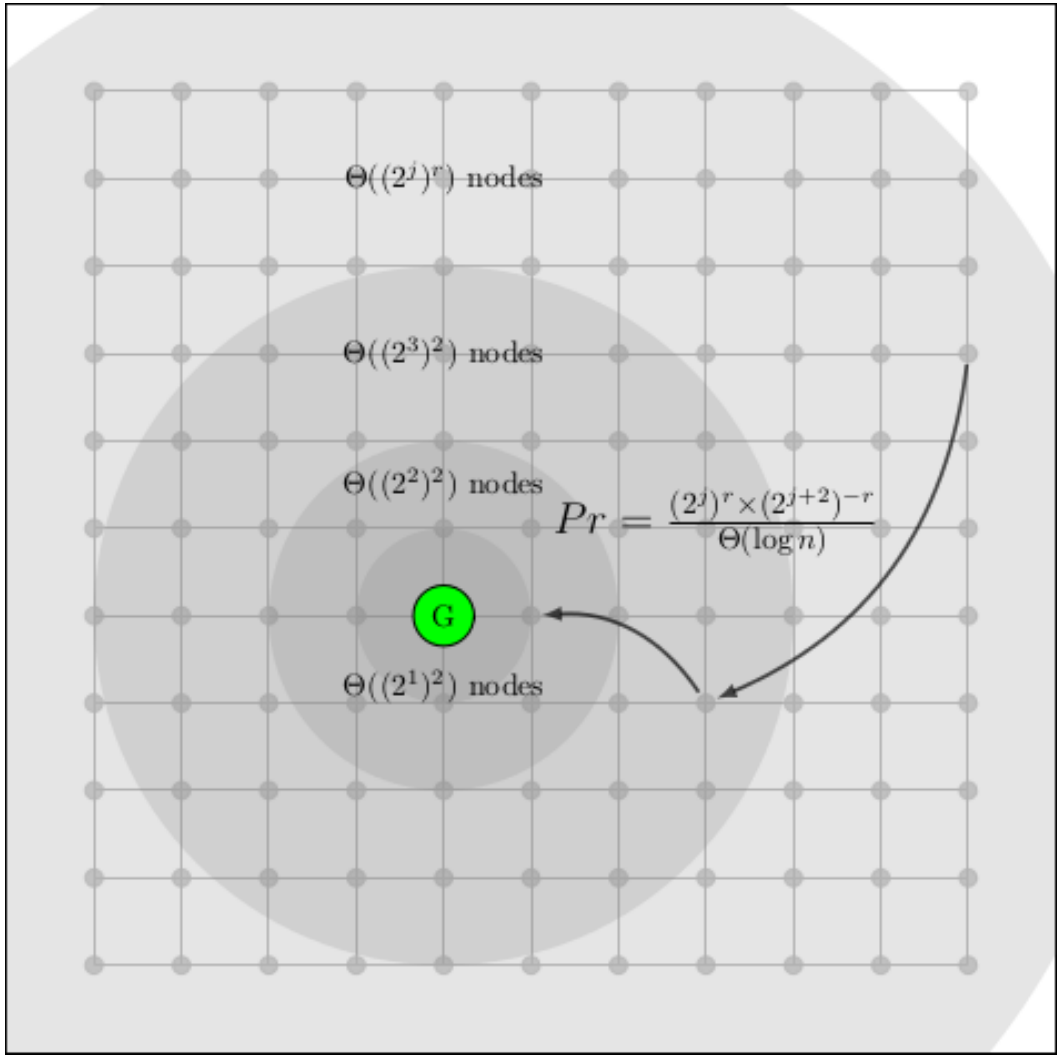
Small Worlds

- A r-dimensional lattice graph
- Edges distributed inversely proportional to distance



Small Worlds

- A r-dimensional lattice graph
- Edges distributed inversely proportional to distance
- A greedy agent will move from one neighbourhood to another in log(n) time

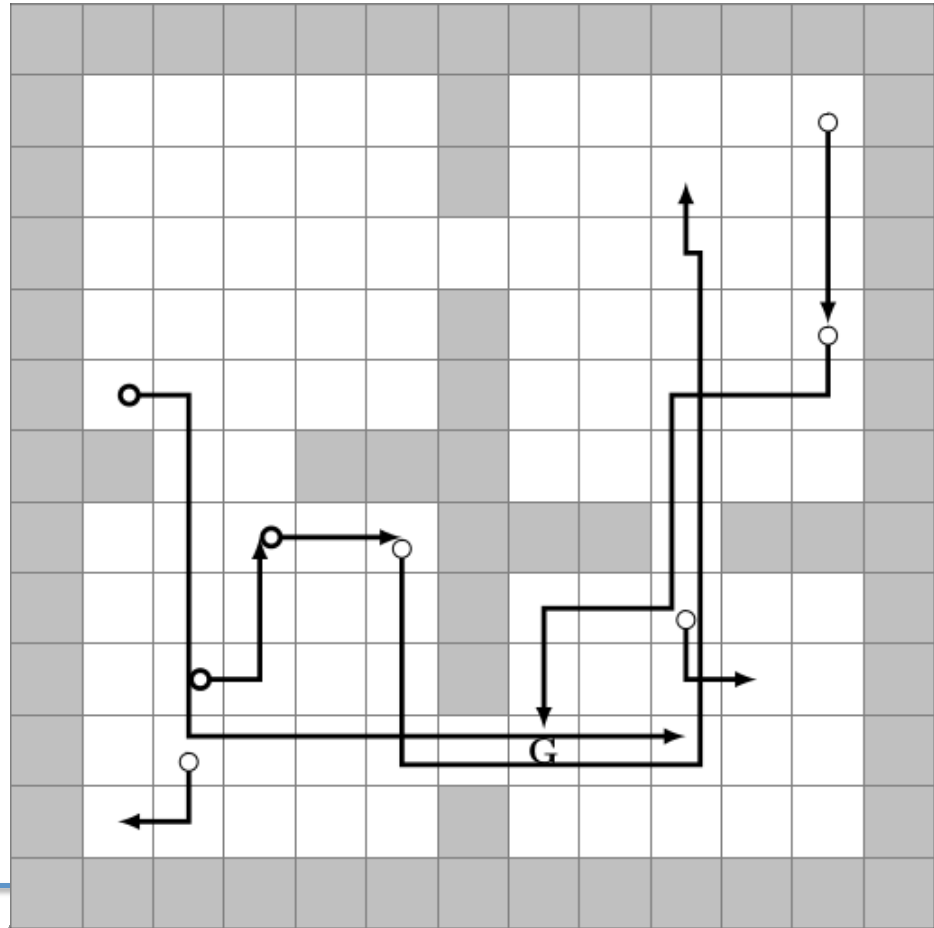


Small Worlds in RL

- Construct "path options" that take an agent from state *s* to *s′*.

- *s′* is chosen according to the power-law.
- Which distance based?
- Value and state-space distance are related

$$k_1\|u-v\|-c_1 \leq \|u-v\|_V \leq k_2\|u-v\|$$



Small Worlds in RL

- Many options required; how do we effectively learn them?
  - *Note: We add only one extra option per state*
- Key Insight: The important point is to move to an exponentially smaller neighbourhood of target.
  - Use cheap, possibly inaccurate options
- Algorithm:
  - Train an agent on T different tasks.
  - For each task, save path options using two states distributed according to the power-law, and following the policy gradient.
- Does not require complete knowledge of the MDP, nor does it need to build a model

Algorithm

Algorithm 2 QOptions: Options from a Q-Value Function

Require: Q, r, n

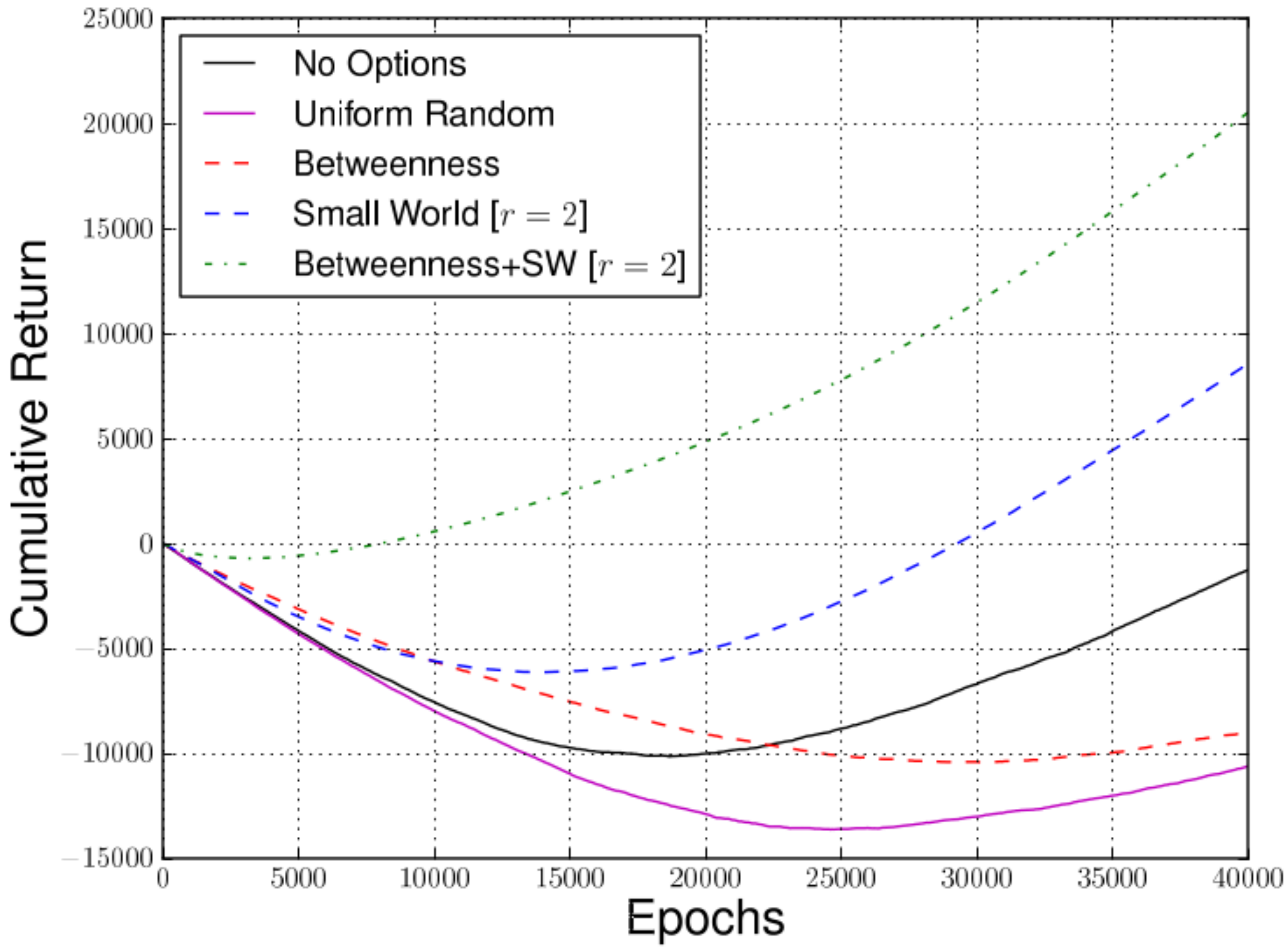
1: $O \leftarrow \emptyset$
2: $\pi \leftarrow$ greedy policy from Q
3: **for all** s in S **do**
4: Choose an s' according to P_r
5: **if** $Q(s', \pi(s')) > Q(s, \pi(s))$ **then**
6: $O \leftarrow O \cup \langle\{s\}, \pi, \{s'\} \cup \{t \mid Q(s', \pi(s')) < Q(t, \pi(t))\}\rangle$
7: **end if**
8: **end for** s in S
9: **return** A random subset of n options from O

Experiments

1. No options
2. 200 options generated uniformly randomly
3. Betweenness based options
4. 200 small world options
5. Betweenness + Small World

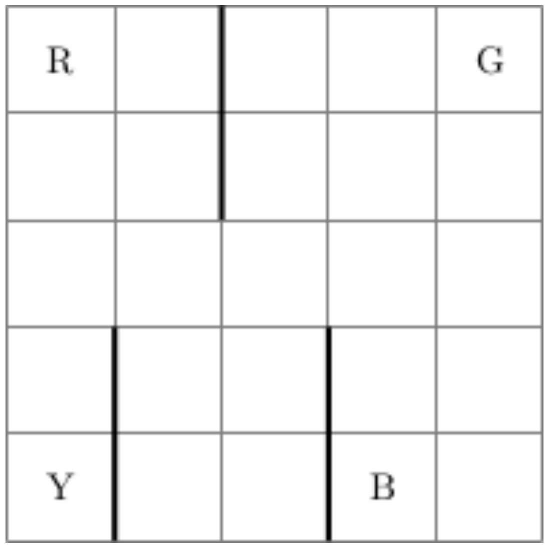
Experimental Results

Rooms: Cumulative Return with 200 options



Experimental Results

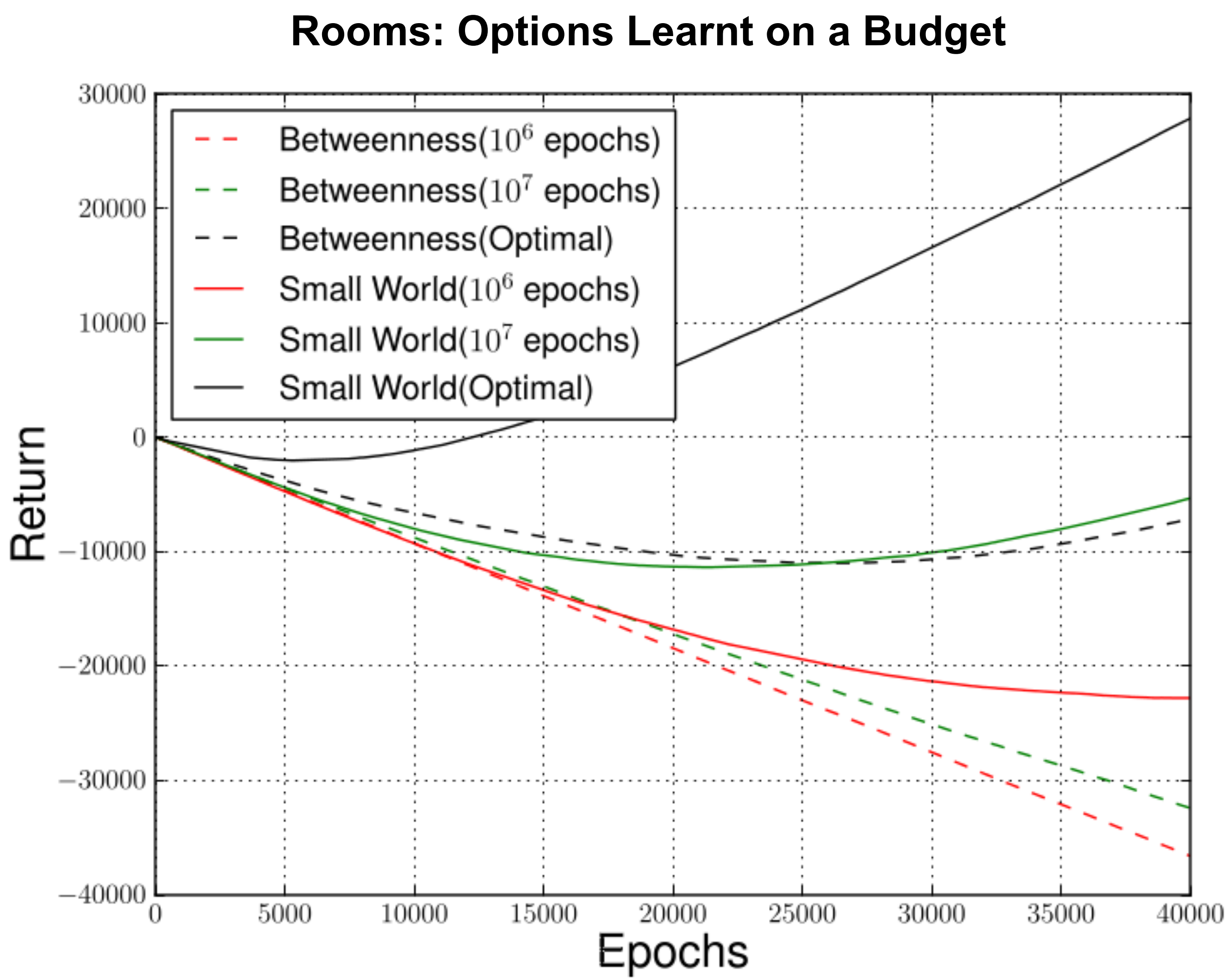
	Arbitrary Navigation	Rooms	Taxi
None	-31.82	-1.27	-16.90
Random	-31.23	-10.76	-18.83
Betweenness	-18.28	-8.94	**80.48**
Small World	**-14.24 [r = 4]**	**8.54 [r = 2]**	0.66 [r = 0.75]



Learning on a budget

- Differing amounts of effort required to learn options
- Fewer betweenness options than small world options
- Equalize budget for both settings
  - 10^6 epochs
  - 10^7 epochs
  - Optimal solutions
- Recall: It is sufficient if the small world option "hops" to closer region

Experimental Results

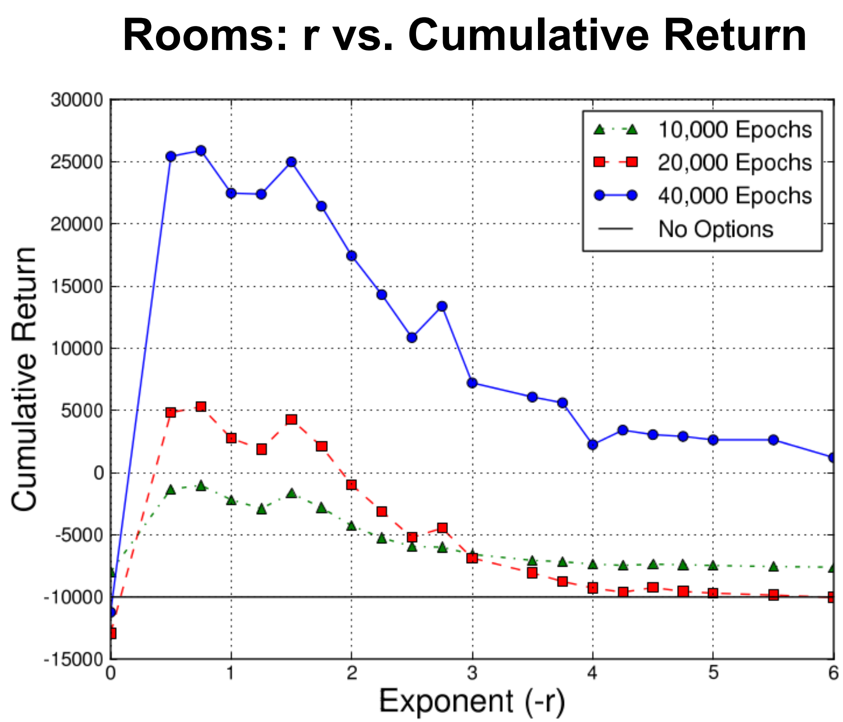


Conclusions

- A new option generation scheme with theoretically motivated generalisation properties.
- An efficient algorithm to learn “small world” options solely from experience on a few tasks.
- Competitive performance on standard domains

Future Work

- Extending small world options to continuous domains
- Experiments on more diverse domains
- What factors affect the power-law exponent r?
- Adding options online
- Exploring implications on learning bounds



Questions?

- http://www.cse.iitm.ac.in/~ravi