

RL en espacios continuos

(con diapositivas de Ali Nouri)

Assumptions for Continuous Spaces

- Transition can be written in the following form:

$$S_{t+1} = f(s_t, a_t) + \omega$$

Where ω is drawn from a known distribution.

[Note: a recent work shows learning the noise is not very detrimental to algorithms [BLLLR08]

- Transition and reward functions are Lipschitz continuous [CT91]:

$$|| f(s_1, a) - f(s_2, a) || < C_T || s_1 - s_2 ||$$

$$|| R(s_1) - R(s_2) || < C_R || s_1 - s_2 ||$$

MDP Properties Cntd.

- Optimal value function satisfies the following Bellman equation [P94]:

Finite State Space:
$$V^*(s) = R(s) + \max_a \gamma \sum_{s'} T(s'|s, a) V^*(s')$$

Continuous State Space:
$$V^*(s) = R(s) + \max_a \gamma \int_{s'} T(s'|s, a) V^*(s') ds'$$

Solving MDPs

- Finite state space [P94]:
 - Value iteration
 - Policy iteration
 - Linear programming
 - Continuous state space:
 - Convert to finite MDP by discretization, solve accordingly
 - Fitted value iteration [G95]
 - Forward search: sparse sampling, UCT [KMN02, KS06]
- ← Relatively very expensive

RL en espacios continuos

- Todos los algoritmos que mencionamos hasta ahora usan tablas para almacenar los parámetros del problema.
- Imposible en espacios continuos con estados y/o acciones infinitas.
- Hay que usar algún método de generalización.
- Podemos reemplazar las tablas por un aproximador de funciones, aunque puede traer problemas.

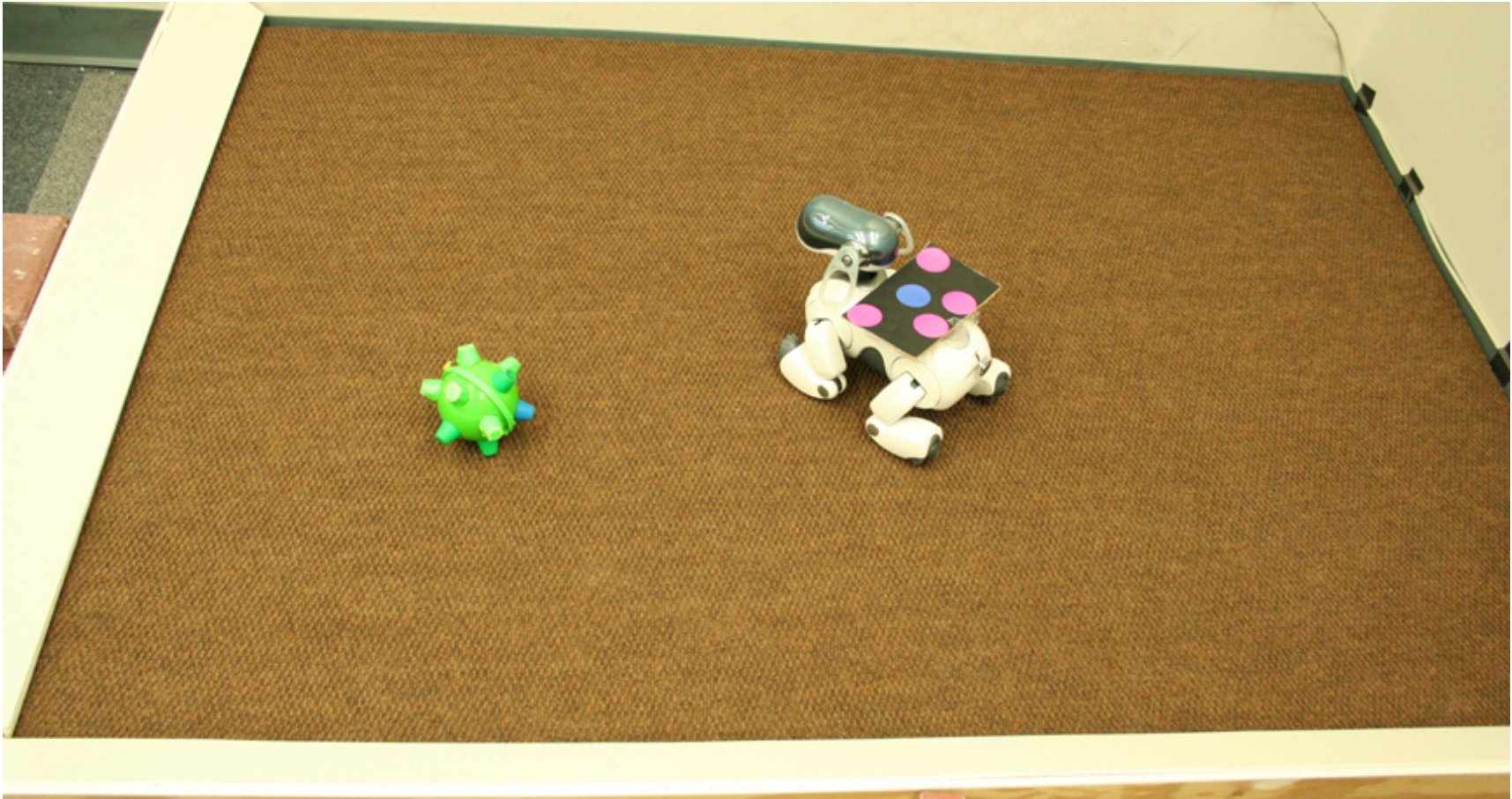
Value-function Approximation

- Several researchers have tried to apply function approximation to values of states a long time ago [T95, BM95]
- Boyan and Moore reported that some function approximators are not stable and result in divergence [BM95].
- Gordon showed a very restricted set of function approximators are stable [G95].

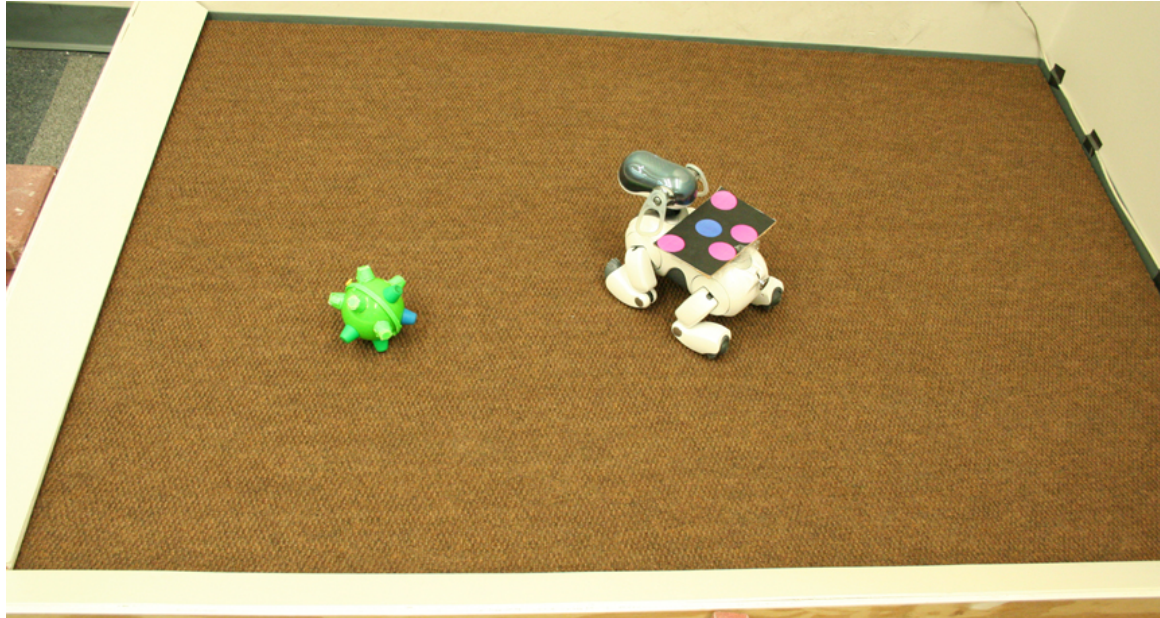
Model Approximation

- Metric E3 [KKL03] provides an algorithm schema for how generalization can be done in a model-based setting without losing convergence guarantees.
- A few realization of metric E3 exist for a subset of environments [SL07,JS06].

An Experimental Domain



Factored State Spaces



$$S = S^1 \times S^2 \times \dots \times S^m$$

$$S1 = (s^1_1, s^1_2, \dots, s^1_{m1})$$

$$S2 = (s^2_1, s^2_2, \dots, s^2_{m2})$$

...

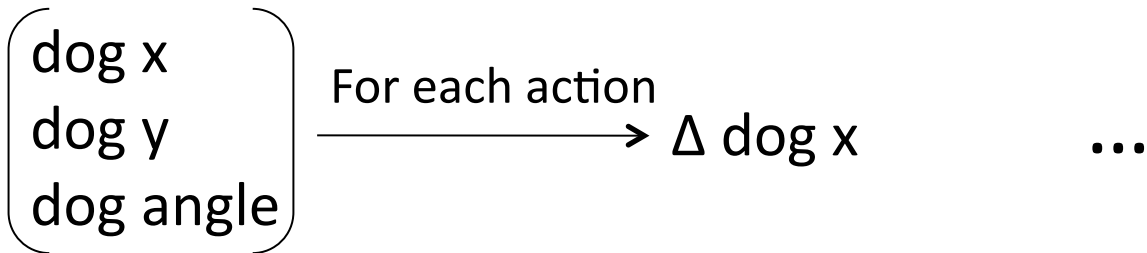
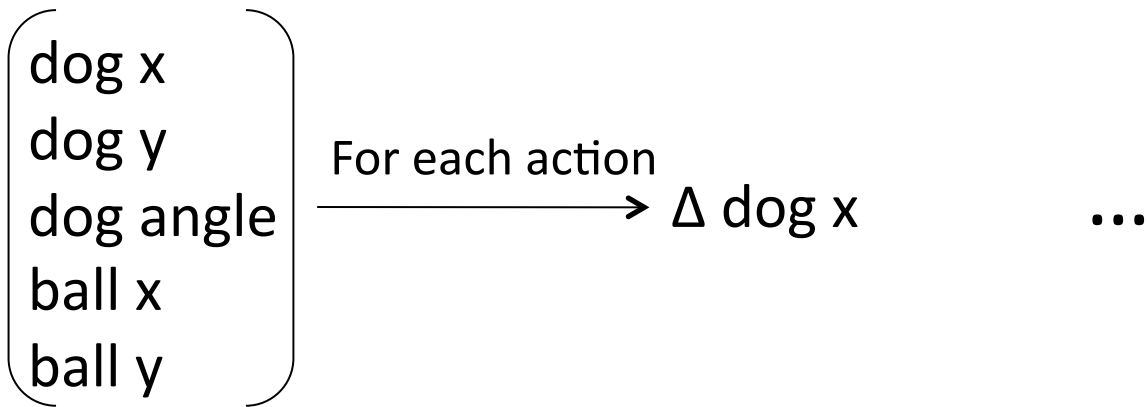
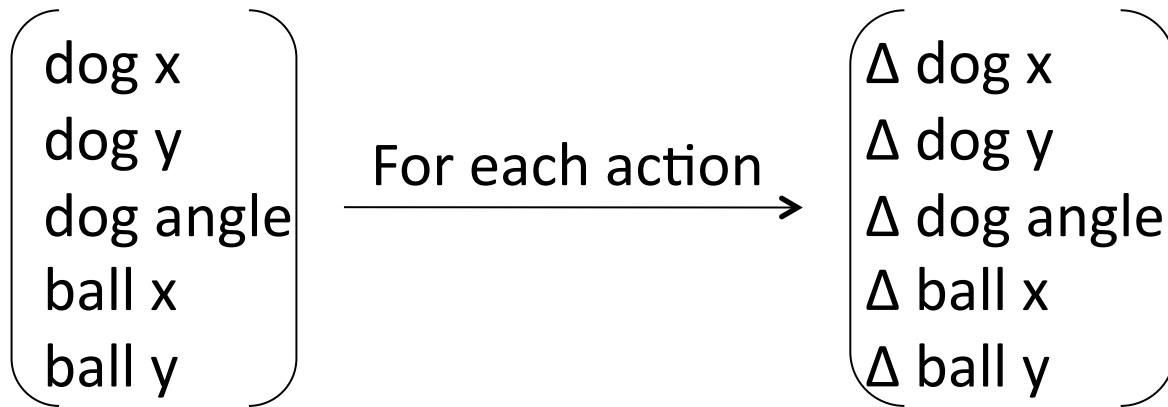
State: $\left(\begin{array}{l} \text{dog } x \\ \text{dog } y \\ \text{dog angle} \\ \text{ball } x \\ \text{ball } y \end{array} \right)$

Model Parameter Approximation (MPA)

A factored learner for continuous spaces

Factored Learner in Continuous Domains

- Estimate environment in delta model [JS06]:
 - Instead of learning a mapping from S to S , learn a mapping from S to \mathbb{R}^m such that $s_{t+1} = s_t + f(s_t, a_t) + \omega$
- Input the dependency graph of variables in the form of a DBN.
- Construct a function approximator for each state variable.
- Train each function approximator with samples in delta transition model.
- Allow function approximators to return “I don’t know” value, if there’s not enough support.



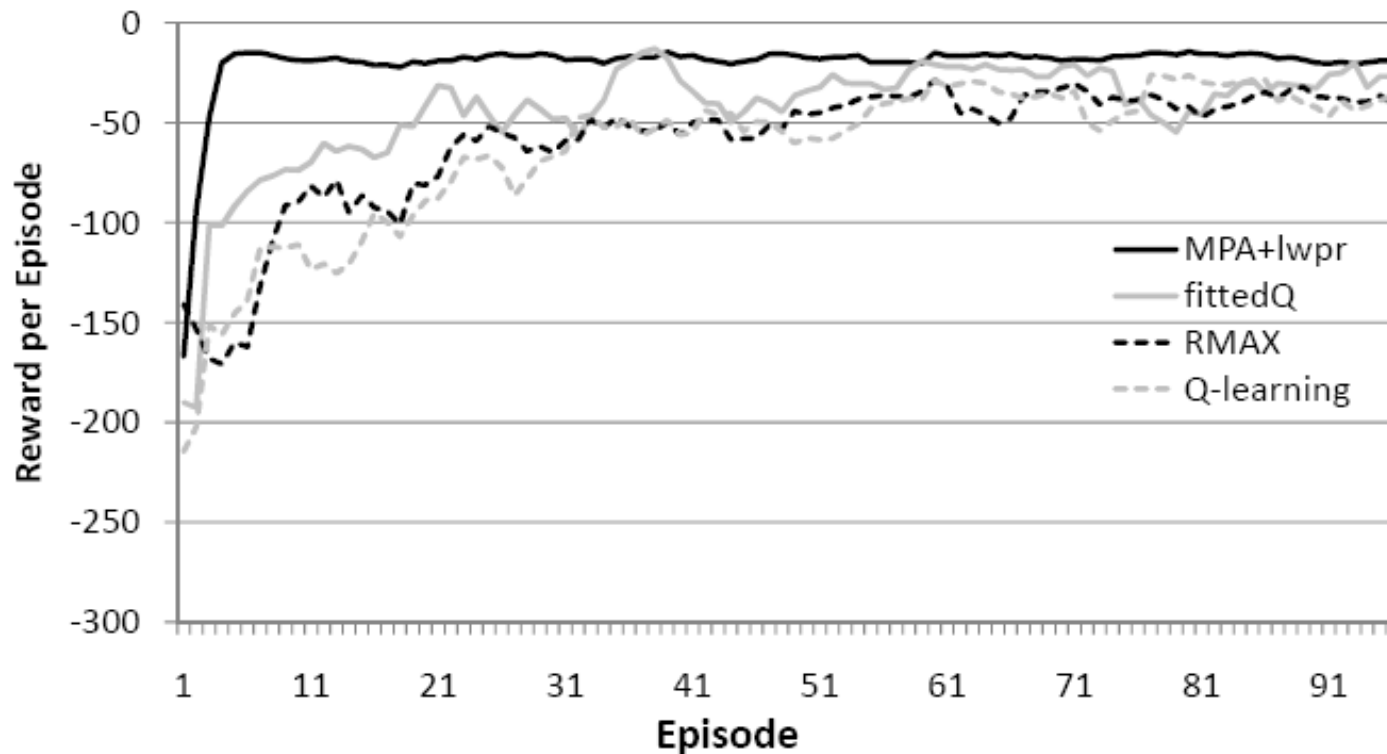
MPA

- Input dependency graphs $d_{i,j}$ for each action i and target state variable j .
- Create transition function approximator $\theta_{i,j}$ for each action i and target state variable j .
- Create reward function approximator ϕ .
- While learning continues:
 - observe an interaction in the form of $(s_t, a_t, s_{t+1}, r_{t+1})$
 - Train $\theta_{at,i}$ with $(d_{at,i}(s_t), s_{t+1}(i) - s_t(i))$
 - Train ϕ with (s_{t+1}, r_{t+1})
 - If $t = \text{intervalTime}$ then
 - Let $\pi = \text{plan}(\Theta, \phi, R_{\max})$
 - Return $\pi(s_{t+1})$

MPA Planning

- Input Θ , ϕ and R_{\max}
- Partition state space using non-overlapping cells ζ with resolution σ .
- Create fictitious state s^f with reward R_{\max} and self-looping actions.
- For each cell ζ in ζ , generate k uniformly distributed samples (O^1, \dots, O^k).
- Let $R'(O^i) = \phi(O^i)$
- For each action a_i , Let $P^{ij} = \theta(O^j, a_i)$ and $\text{cell}(s)$ be the cell containing s .
- Using P^{ij} 's, construct maximum likelihood transition function $T'(\zeta, a_i)$
- If $P^{ij} = \text{IDK}$ then $P^{ij} = s^f$
- Construct finite MDP $M' = \langle \zeta, A, T', R', \gamma \rangle$
- Solve M' using conventional value iteration algorithm.
- Return the best policy

Results for Robot Navigation

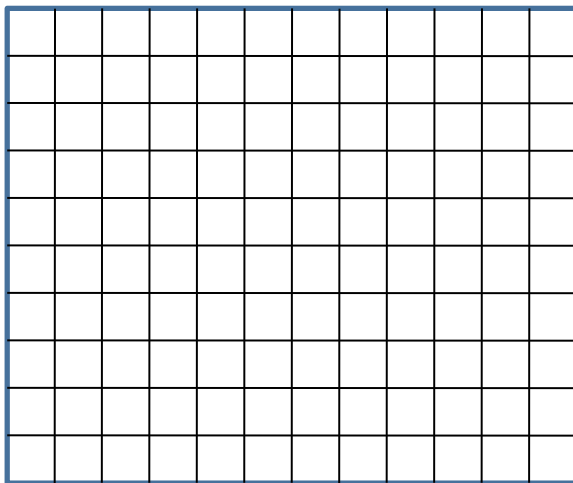


Results when BumbleBall is NOT in the environment
LWPR used as function approximation [VSS00]
Results averaged over 10 runs

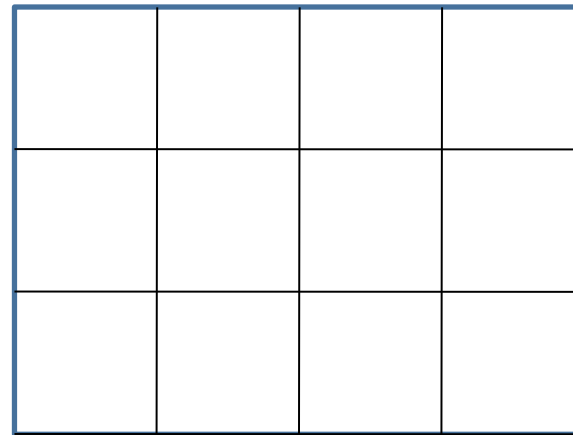
Multi-resolution Exploration (MRE)

A discretization-based exploration

- Discretize state space and use samples in each cell to tag that region as known or unknown.
- We can use it for implementing metric E3
- It's computationally faster than maintaining hyper spheres.

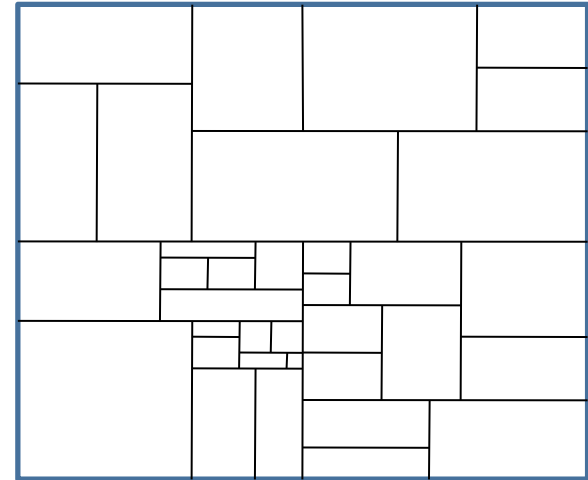


More accurate
less generalization



More generalization
Less accurate

- Allow different levels of generalization depending on how many samples exist in the neighborhood.
- Get more accurate estimations in parts of state space where we care more.
- Allow more generalization for places where we don't need much accuracy



Kd-tree Structure

- It partitions the state space into variable-sized regions.
- The root covers the whole space, each node selects one of the dimensions and splits the space into two half-spaces, producing two children.
- We split at the median of the points along that direction.
- We stop once the number of points inside a regions is less than a threshold.

Knownness for MRE

- Make *knownness* a function of the size and the number of points in the cell:
 - Define a target resolution, τ , and number of desired points in cells, η , based on ε , λ , and smoothness parameters.
 - Define smallness of a cell to be a function that goes from 0 to 1 as the size of the cell decreases from $||S||$ to τ .
 - Define knownness to be
$$\text{knownness}(\zeta) = (|O_\zeta| / \eta) \cdot (\text{smallness}(\zeta))$$

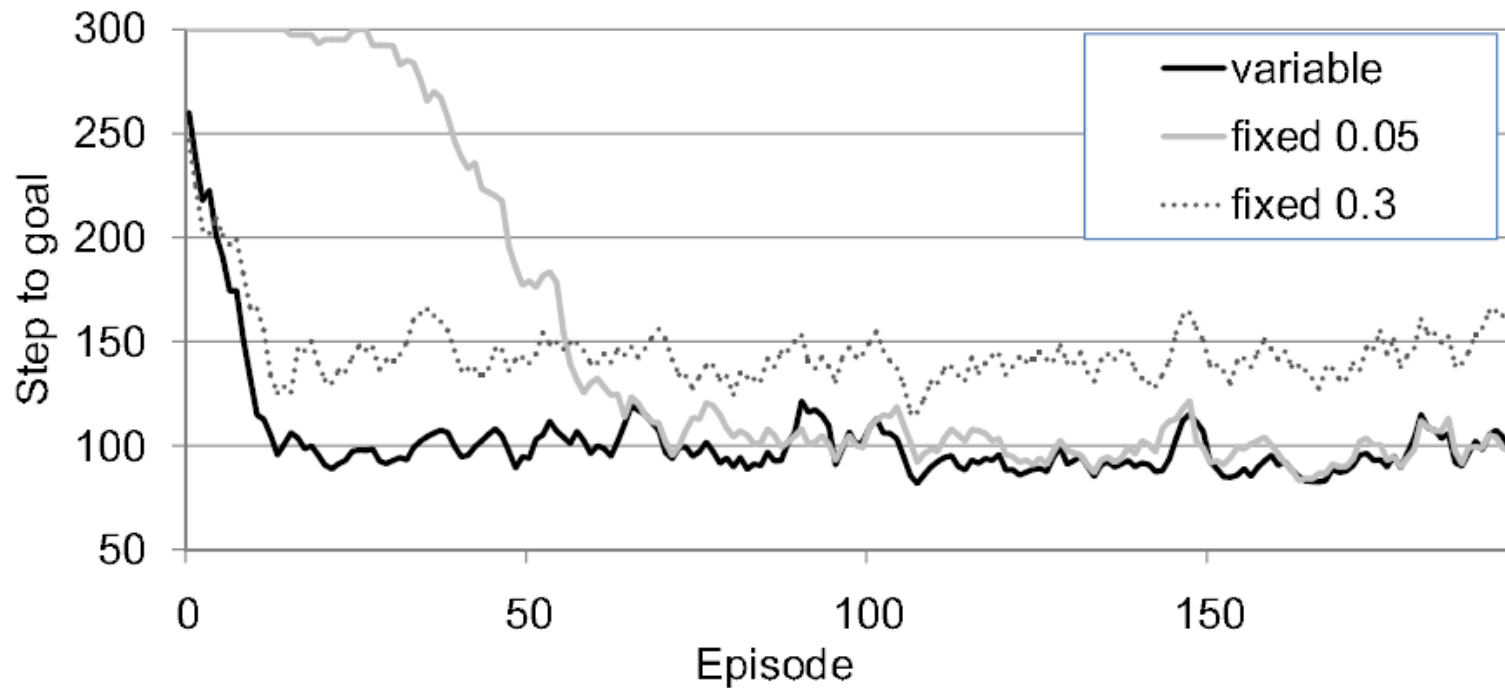
Model-based MRE

- Let O_{sa} be the set of samples for s/a pair.
- Let s^f be a fictitious state with value= V_{\max} .
- Build the transition function as follows:
- $T(s'|s,a) = k(s,a) T^{\wedge}(s'|s,a)$
- $T(sf|s,a) = 1 - k(s,a)$

Propiedades de Model-based MRE

- Es independiente del aproximador utilizado.
- Es independiente del algoritmo de planning.
- *Resultado: es PAC.*

Results for Mountain Car



MountainCar is a 2D environment
Results averaged over 20 runs

