


Relational Reinforcement Learning

Prasad Tadepalli, Alan Fern, Kristan Kersting
Decision-Theoretic Planning and Learning in Relational Domains
Twenty-Third AAAI Conference on Artificial Intelligence,
Chicago, Illinois, USA, July 13–17 2008



Roadmap

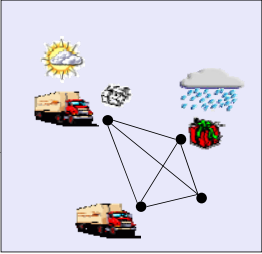
- Relational MDPs and RL
- Function approximation
- Relational Value Function Learning
 - ▲ Propositionalization
 - ▲ Relational Regression
- Relational Policy Learning
 - ▲ Approximate Policy Iteration

Real Time Strategy Games




- Many objects of various types in complex interactions
- Good players can generalize across situations involving distinct object configurations

The Logistics Domain



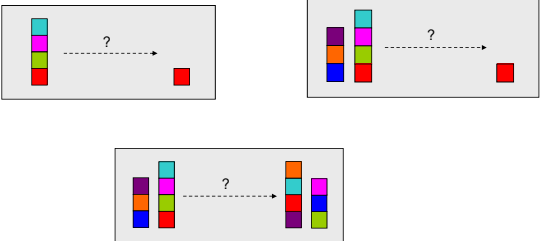
- Move many objects around with many other objects
- Identities and numbers of objects always changing

Robot Soccer



- Reasoning about relationship between objects (players and ball) key to good play

and of course Blocksworld



- Would like a policy that is independent of number of objects/blocks

Relational MDPs (RMDPs)

- RMDPs are MDPs that make the notion of object explicit in the state and action spaces
- **States:** each state is described by set of objects, object properties, and relations among objects. E.g.
 - ▲ Objects: agent1, agent2, weapon1, location1, location2, ...
 - ▲ Properties: strength(agent1,100), empty(location1)
 - ▲ Relations: holding(agent1,weapon1), at(agent2,location2)
- **Actions:** parameterized by objects
 - ▲ Abstract actions: Attack(X,Y), Pickup(X,Y)
 - ▲ Ground actions: Attack(agent1,agent2), Pickup(agent1,weapon1)
 - ▲ Effects of actions can change properties of and relationships among objects, or sometimes even create new objects
- **Reward:** depends on object properties and relations
 - ▲ # of packages delivered, # of enemies killed

Relational RL

- RMDPs with fixed set of objects can be “propositionalized”
 - ▲ Describe states via traditional feature vectors that list values of all properties and relations.
 - [on(a,b)=true, on(b,a)=false, ontable(a)=false, ontable(b)=true]
- Can then directly apply feature-based RL to this representation
 - ▲ Loses the relational structure provided by objects
 - ▲ Policies can't be applied directly to new object domains
 - ▲ Can be difficult to learn from such large feature vectors
- Relational RL attempts to learn policies that directly exploit relational structure:
 - ▲ Faster learning w.r.t. propositionalization even with fixed # of objects
 - ▲ Learn policies that generalize across object domains

Large Relational State Spaces

- When a problem has a large state space we can not longer represent the V or Q functions as explicit tables
 - ▲ Generally the case for RMDPs with a non-trivial numbers of objects
- Even if we had enough memory
 - ▲ Never enough training data!
 - ▲ Learning takes too long
- What to do??

Roadmap

- Relational MDPs and RL
- Function Approximation
- Relational Value Function Learning
 - ▲ Propositionalization
 - ▲ Relational Regression
- Relational Policy Learning
 - ▲ Approximate Policy Iteration

Prasad Tadepalli, Alan Fern, Kristian Kersting
Decision-Theoretic Planning and Learning in Relational Domains
Twenty-Third AAAI Conference on Artificial Intelligence,
Chicago, Illinois, USA, July 13--17, 2009



Function Approximation

- Never enough training data!
 - ▲ Must **generalize** what is learned from one situation to other “similar” new situations
- Basic Idea:
 1. Represent Q-function using a compact representation
 - Function encoding size much smaller than table
 2. Learn compact function from experience instead of table
- Q-function updates arising from experience in one state can influence Q-estimate in other similar states
 - ▲ Facilitates generalization of experience
- We will first consider feature-based approximation

Feature Based Function Approx.

- Define a set of n state-action features $f_1(s,a), \dots, f_n(s,a)$
 - ▲ The features are used as our representation of state-action pairs
 - ▲ State-action pairs with similar features will be considered similar
 - ▲ In RRL s and a are relational states and actions
- **Example Representation:** linear approximator

$$\hat{Q}_\theta(s, a) = \theta_0 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \dots + \theta_n f_n(s, a)$$
- More generally one can use any form of function approximator in terms of these features
 - ▲ Regression trees
 - ▲ Kernel regression
 - ▲ Neural networks
 - ▲ etc.

Q-Learning for Linear Approximators

1. Start with initial parameter values
2. Take action according to an **explore/exploit policy**
3. Perform Q-update for each parameter

$$\theta_i \leftarrow ?$$
4. Goto 2

13

Aside: Gradient Descent for Squared Error

- Suppose that we have a sequence of states-action pairs with target Q-values $\langle s_1, a_1, q(s_1, a_1) \rangle, \langle s_2, a_2, q(s_2, a_2) \rangle, \dots$
- Our goal is to minimize the sum of squared errors between our estimated function and each target value:

$$E_j = \frac{1}{2} (\hat{Q}_\theta(s_j, a_j) - q(s_j, a_j))^2$$

squared error of example j our estimated value for j'th state target value for j'th state

- After seeing j'th state the **stochastic gradient descent rule** tells us to update all parameters by:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i}, \quad \frac{\partial E_j}{\partial \theta_i} = \frac{\partial E_j}{\partial \hat{Q}_\theta(s_j, a_j)} \frac{\partial \hat{Q}_\theta(s_j, a_j)}{\partial \theta_i}$$

learning rate

14

Aside: continued

$$\theta_i \leftarrow \theta_i + \alpha \frac{\partial E_j}{\partial \theta_i} = \theta_i + \alpha \underbrace{(q(s_j, a_j) - \hat{Q}_\theta(s_j, a_j))}_{\frac{\partial E_j}{\partial \hat{Q}_\theta(s_j, a_j)}} \frac{\partial \hat{Q}_\theta(s_j, a_j)}{\partial \theta_i}$$

depends on form of approximator

- For a linear approximation function:

$$\hat{Q}_\theta(s, a) = \theta_1 + \theta_2 f_1(s, a) + \theta_3 f_2(s, a) + \dots + \theta_n f_n(s, a)$$

$$\frac{\partial \hat{Q}_\theta(s_j)}{\partial \theta_i} = f_i(s_j, a_j)$$

- Thus the update becomes:

$$\theta_i \leftarrow \theta_i + \alpha (q(s_j, a_j) - \hat{Q}_\theta(s_j, a_j)) f_i(s_j, a_j)$$

15

Q-Learning for Linear Approximators

1. Start with initial parameter values
 2. Take action according to an **explore/exploit policy**
 3. Perform Q-update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha (R(s) + \beta \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)) f_i(s, a)$$
 4. Goto 2
- Predicted Target q(s,a) Current estimate

16

Q-learning w/ Non-linear Approximators

$\hat{Q}_\theta(s, a)$ is sometimes represented by a non-linear approximator such as a neural network

1. Start with initial parameter values
2. Take action according to an **explore/exploit policy** (should converge to greedy policy, i.e. GLIE)
3. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha (R(s) + \beta \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)) \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

calculate closed-form

4. Goto 2

- Typically the space has many local minima and we no longer guarantee convergence
- Often works well in practice

17

Roadmap

- Relational MDPs and RL
- Function approximation
- Relational Value Function Learning
 - Feature Engineering
 - Relational Regression
- Relational Policy Learning
 - Approximate Policy Iteration

Relational RL via Feature Engineering

- What if our states and actions are relational?
- One approach is to engineer a fixed set of “relational features”
 - ▲ Each feature returns a value for any relational state-action pair
 - ▲ Features should be well defined regardless of the number of objects
 - Unlike naïve propositionalization approach
 - ▲ Ideally feature values should be similar for similar relational states
- With such a feature representation, can use feature-based Q-learning to learn a relational Q-function.
 - ▲ Success relies critically on ability to define appropriate features
 - ▲ Often requires significant effort and insight into problem

19

Example: Tactical Battles in Wargus

- Wargus is real-time strategy (RTS) game
 - ▲ Tactical battles are a key aspect of the game




5 vs. 5
10 vs. 10

- **RL Task:** learn a policy to control n friendly agents in a battle against m enemy agents
 - ▲ Policy should be applicable to tasks with different sets and numbers of agents
 - ▲ That is, policy should be relational

20

Example: Tactical Battles in Wargus

- **Relational States:** contain information about the locations, health, and current activity of all friendly and enemy agent
- **Relational Actions:** Attack(F,E)
 - ▲ causes friendly agent F to attack enemy E
- **Policy Structure:** each decision cycle loop through each friendly agent F and use a learned Q-function to select enemy to attack
 - ▲ I.e. select enemy E for F that maximizes $Q(s, \text{Attack}(F,E))$
- $Q(s, \text{Attack}(F,E))$ is relational since any agents can be substituted for F and E
 - ▲ We used a linear function approximator with Q-learning

21


Example: Tactical Battles in Wargus

$$\hat{Q}_\theta(s, a) = \theta_1 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \dots + \theta_n f_n(s, a)$$

- Engineered a set of relational features $\{f_1(s, \text{Attack}(F,E)), \dots, f_n(s, \text{Attack}(F,E))\}$
- **Example Features:**
 - ▲ # of other friendly agents that are currently attacking E
 - ▲ Health of friendly agent F
 - ▲ Health of enemy agent E
 - ▲ Difference in health values
 - ▲ Walking distance between F and E
 - ▲ Is E the enemy agent that F is currently attacking?
 - ▲ Is F the closest friendly agent to E?
 - ▲ Is E the closest enemy agent to E?
 - ▲ ...
- Features are well defined for any number of agents

22


Example: Tactical Battles in Wargus



Initial random policy

23

Example: Tactical Battles in Wargus

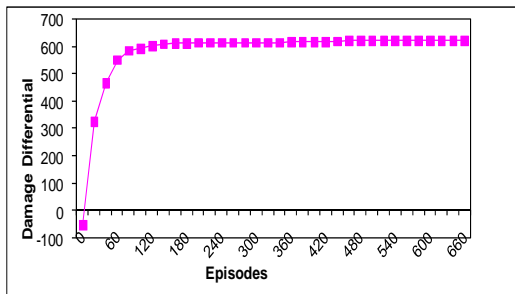


Learned Policy after 120 battles

24

Example: Tactical Battles in Wargus

- Linear Q-learning in 5 vs. 5 battle



25

Example: Tactical Battles in Wargus

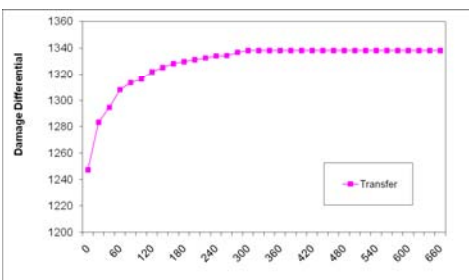


Policy learned on 5 vs. 5

26

Example: Tactical Battles in Wargus

- Initialize Q-function for 10 vs. 10 to one learned for 5 vs. 5
 - ▲ Initial performance is very good which demonstrates relational generalization from 5 vs. 5 to 10 vs. 10



27

Roadmap

- Relational MDPs and RL
- Function approximation
- Relational Value Function Learning
 - ▲ Feature Engineering
 - ▲ Relational Regression
- Relational Policy Learning
 - ▲ Approximate Policy Iteration

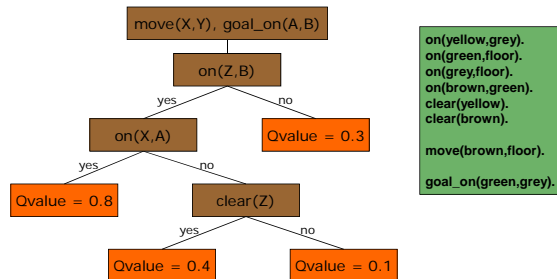
Prasad Tadepalli, Alan Fern, Kristian Kersting
Decision-Theoretic Planning and Learning in Relational Domains
Twenty-Third AAAI Conference on Artificial Intelligence,
Chicago, Illinois, USA, July 13–17, 2009



Relational Regression

- The previous approach relies on feature engineering to reduce a relational problem to a propositional one
 - ▲ Requires significant effort and trial-error
- Can we learn a relational value function automatically without propositionalization?
- Several relational regression algorithms for batch supervised learning
 - ▲ Relational regression trees
 - ▲ Gaussian processes
 - ▲ Nearest neighbors

Relational Regression Trees



- Internal nodes can have relational tests
 - Tests can involve variables
 - Can depend on input objects (X,Y,A,B)
 - TILDE is a popular top-down regression tree learner [Blockeel AIJ 101]

RRL Example

- Use current policy until a goal-state is reached

RRL Example

- Back propagate Q-value estimates

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

RRL Example

- Store (state,action,qvalue) triples as batch training set
- Give to TILDE to learn a tree

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

state(s3). action(a). qvalue(0.56).	state(s3). action(d). qvalue(0.81).
state(s1). action(b). qvalue(0.66).	state(s5). action(e). qvalue(0.90).
state(s2). action(c). qvalue(0.73).	state(s4). action(f). qvalue(1.00).

The TG algorithm

Based on the G-tree algorithm and the Tilde algorithm

$$\frac{n_p}{n} \sigma_p^2 + \frac{n_n}{n} \sigma_n^2 \quad \text{vs.} \quad \sigma_{total}^2$$

RRL-TG algorithm

- No need to accumulate examples generated during RRL
 - Simply update tree incrementally

initialize tree to a **single root node**

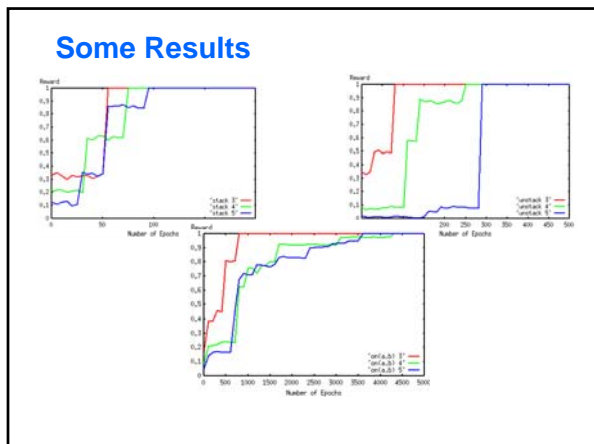
while (true)

 generate episode through the use of a **standard Q-learning** algorithm using the current tree as Q-function

generate example (s_i, a_i, q_i) for each state-action pair encountered

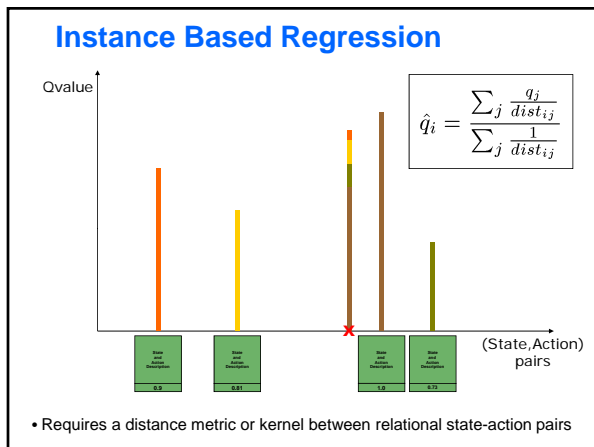
update tree using **the TG-algorithm** and the generated examples

Blocks World

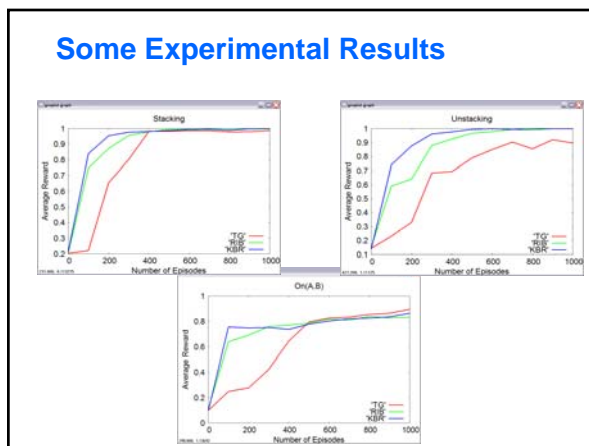
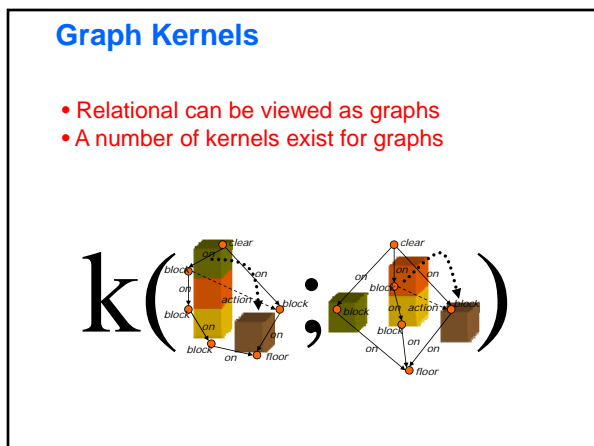


Some Results: Timings

		3 blocks	4 blocks	5 blocks
Batch RRL	Stack (30 epochs)	6.16 min	62.4 min	306 min
	Unstack (30 epochs)	8.75 min	Not Stated	Not Stated
	On(a,b) (30 epochs)	20 min	Not Stated	Not Stated
RRL-TG	Stack (200 epochs)	19.2 sec	26.5 sec	39.3 sec
	Unstack (500 epochs)	1.10 min	1.92 min	2.75 min
	On(a,b) (5000 epochs)	25.0 min	57 min	102 min



- ### Instance-Based Regression
- Instance based regression typically
 - ▶ Stores past examples (s,a,q)
 - ▶ Given a new example (s',a') interpolate wrt example set to estimate q'
 - Require a "kernel" K(x,x') that measures distances between any examples x and x'
 - A variety of algorithms exist that turn a kernel function into a regression method
 - ▶ Gaussian Processes
 - ▶ Support Vector Regression
 - ▶ K-NN methods
- Kernel for relational data?**

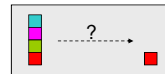


Roadmap

- Relational MDPs and RL
- Function approximation
- Relational Value Function Learning
 - ▲ Feature Engineering
 - ▲ Relational Regression
- Relational Policy Learning
 - ▲ Approximate Policy Iteration

Challenge Problem

Consider the following class of stochastic blocks world problems:



Goal: clear off blocks in the goal

- Optimal policy is:
 - ▲ obvious to us
 - ▲ simple and compact
 - ▲ independent of number of blocks

4
6

Policy for a Simple Domain



A compact policy for this domain:

1. If holding a block, put it down on the table, else...
2. Pick up a clear block above a block that is clear in the goal.



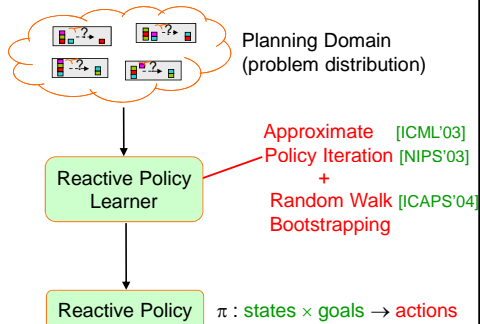
A **control policy** maps states and goals to actions.
 $\pi: \text{states} \times \text{goals} \rightarrow \text{actions}$

Formal Policy for Simple Domain

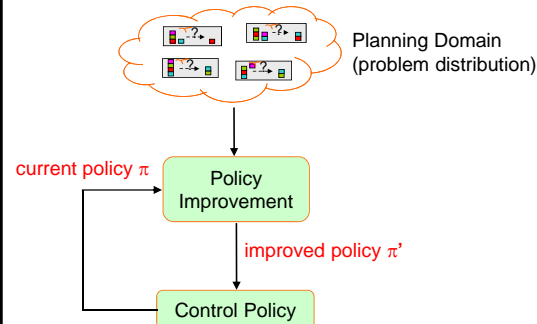
English Decision List	Taxonomic Syntax
1. "blocks being held" : putdown 2. "clear blocks above gclear blocks" : pickup	1. holding : putdown 2. clear \cap (on* gclear) : pickup
1. 	2. 

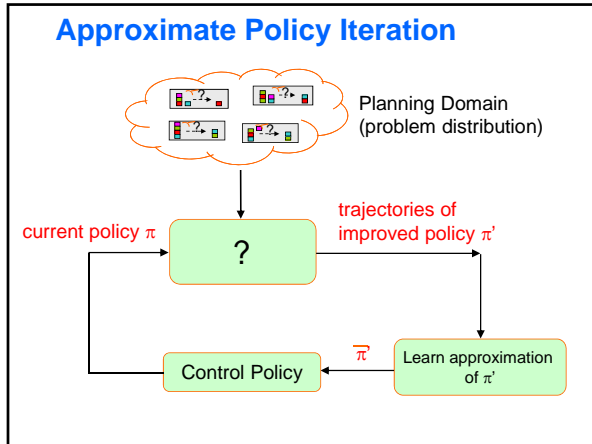
Policy language choice restricts the approachable domains

Learning Domain-Specific Policies



Policy Iteration



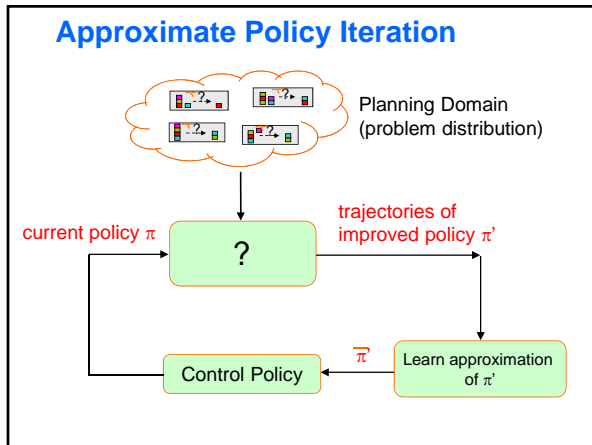


Relating Planning and Classification

[Kharden, MLJ'99] theoretically links classification and planning performance.

Consider class of policies C .
 Observe $O(\log |C|)$ **trajectories** of target policy in C .
 If policy π in C is consistent with trajectories then quality of π is "probably close" to quality of target.

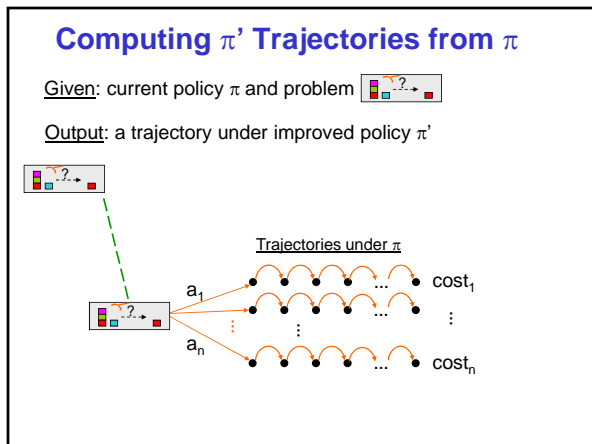
Obtain an improved policy by observing an improved policy



Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'



Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'

Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'

Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'

Computing π' Trajectories from π

Given: current policy π and problem

Output: a trajectory under improved policy π'

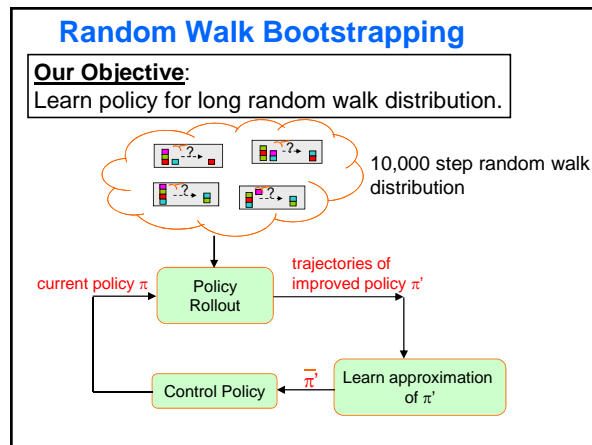
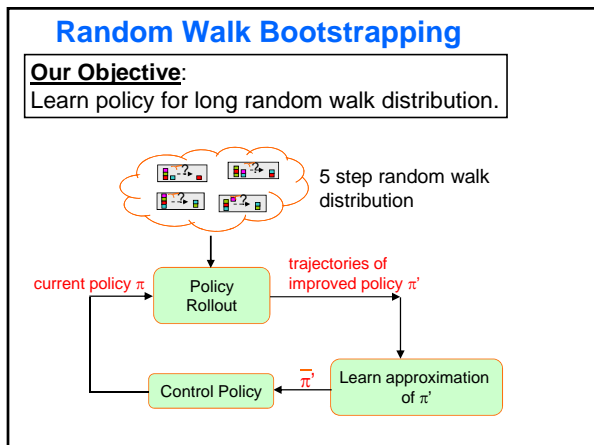
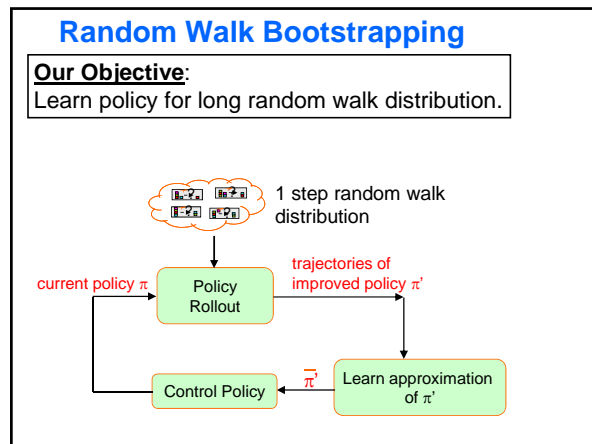
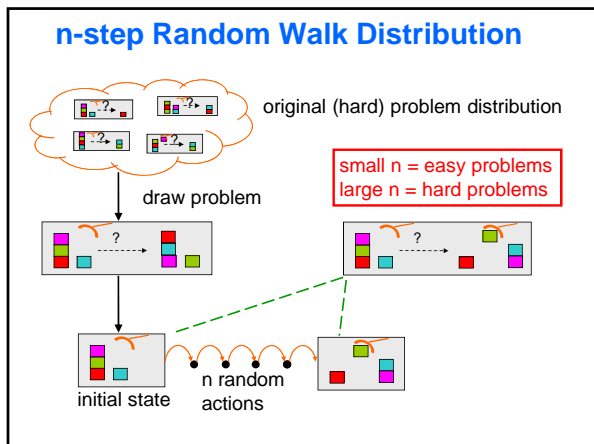
Policy Learning Algorithm

- Training data: generate π' trajectories and save state-action pairs $\langle s_1, A_1 \rangle, \langle s_2, A_2 \rangle, \langle s_3, A_3 \rangle, \dots$.
- Use a Rivest-style decision-list learning approach (induce one rule at a time, in order).
 - ▲ While there are training instances remaining,
 - Find a good rule
 - Remove instances covered by the rule
- Rules found by heuristically guided beam search
 - ▲ heuristic combines consistency and coverage
 - ▲ rules searched from small to large

Finding a Good Rule

- Find rule for each **action type**, then return best one.
- Find rule by a heuristically guided beam search
 - ▲ Search from general to specific by intersecting C_i 's.
 - ▲ Heuristic trades-off consistency and coverage.

Bootstrapping API



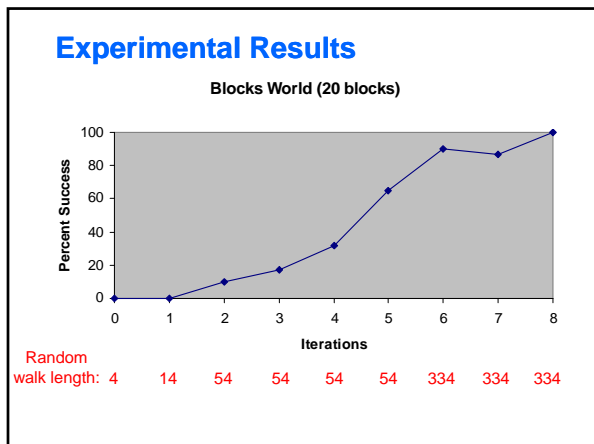
Random Walk Bootstrapping

Our Objective:
Learn policy for long random walk distribution.

- Long random walk policies often perform well on target problem distributions.
- Target distribution can be very different from long random walk distribution.
 - ▲ e.g. grid world with doors and keys

Experiments

- Evaluate on the seven domains from AIPS-2000 + TL-PLAN planning competition.
 - ▲ 5 domains : can represent good policies
 - ▲ 2 domains : can not represent good policies
- Compare against state-of-the-art planner FF
 - ▲ FF's heuristic is very good for most of these domains.
- We equal or better FF's performance when we can represent policies.



Domains with Good Policies

Success Percentage

	Blocks	Elevator	Schedule	Briefcase	Gripper
API	100	100	100	100	100
FF-Plan	28	100	100	0	100

Typically our solution lengths are comparable to FF's.

- ### Recap
- RMDPs describe the world in terms of objects, their properties, and object relations
 - Relational RL algorithms attempt to exploit the relational structure
 - Faster learning
 - Policies generalize across object domains
 - Can exploit relational regression and classification methods for model-free RRL
 - Feature engineering
 - Relational regression trees (RRL-TG)
 - Relational decision lists (API)

RMDPs: The Logistics Domain

Abstract Actions

```

load(Bloc :k, Truck :t, City :c1)
  = Success Probability: if (BlocIn(k, c1) = TruckIn(t, c1)) then 0 else 0
  + Add Effects on Success: {BlocOut(k, c1)}
  + Delete Effects on Success: {BlocIn(k, c1)}

unload(Bloc :k, Truck :t, City :c1)
  = Success Probability: if (BlocIn(k, c1) = TruckIn(t, c1)) then 0 else 0
  + Add Effects on Success: {BlocIn(k, c1)}
  + Delete Effects on Success: {BlocOut(k, c1)}

drive(Truck :t, City :c1, City :c2)
  = Success Probability: if (TruckIn(t, c1) = c2) then 1 else 0
  + Add Effects on Success: {TruckIn(t, c2)}
  + Delete Effects on Success: {TruckIn(t, c1)}

map
  = Success Probability: 1
  + Add Effects on Success: 0
  + Delete Effects on Success: 0
    
```

Objects & Relations

Domain Object Types (i.e., sorts): *Bloc, Truck, City = {paris, ...}*

Relations (with parameter sorts):
BlocIn(Bloc, City), TruckIn(Truck, City), BlocOut(Bloc, Truck)

Abstract Rewards

Reward: if $\exists k, \text{BlocIn}(k, \text{paris})$ then 10 else 0

Prasad Tadepalli, Alan Fern, Kristian Kersting
 Decision-Theoretic Planning and Learning in Relational Domains
 Twenty-Third AAAI Conference on Artificial Intelligence,
 Chicago, Illinois, USA, July 13–17, 2008

Relational Regression Tree

- Uses relational tests in internal nodes
 - Tests include variables
 - Tree depends on input objects
- Leafs make numeric predictions
- TILDE is a popular regression tree learner [Blockeel AIJ 101]
 - Top-down induction
 - Large # of possible tests

Tree for Q's Prop(X)

Prasad Tadepalli, Alan Fern, Kristian Kersting
 Decision-Theoretic Planning and Learning in Relational Domains
 Twenty-Third AAAI Conference on Artificial Intelligence,
 Chicago, Illinois, USA, July 13–17, 2008