

Relational Reinforcement Learning



Do you want the left cookie?

Kurt Driessens

**joint work with Saso Dzeroski,
Luc De Raedt and Jan Ramon**

Overview

- **Automated Behaviour Generation**
 - Why and How
- **Reinforcement Learning & Q-learning**
 - Foundations, Algorithm, Variations
- **Relational Reinforcement Learning**
 - Algorithm, Benefits



Why learning?

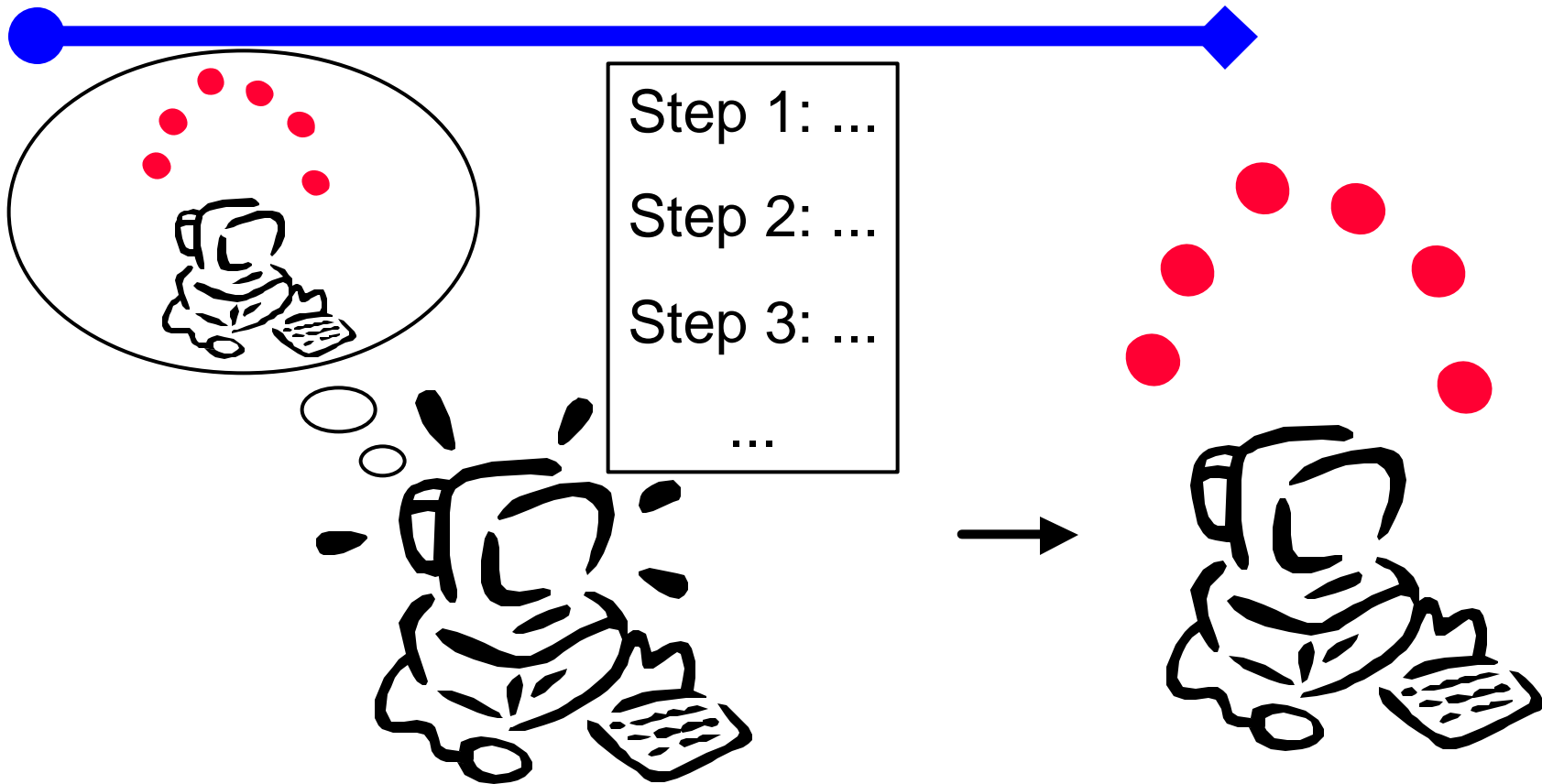
- “Hard Task for a Computer”
 - CPU Requirements?
 - Memory Requirements?
 - Other Requirements?
- “Hard Task for a Programmer”

How Learning?

- Doing the task is not difficult
- Describing **how** to do the task is difficult
- Figuring out how to do a task:
 - Planning
 - Behavioral Cloning
 - Reinforcement Learning



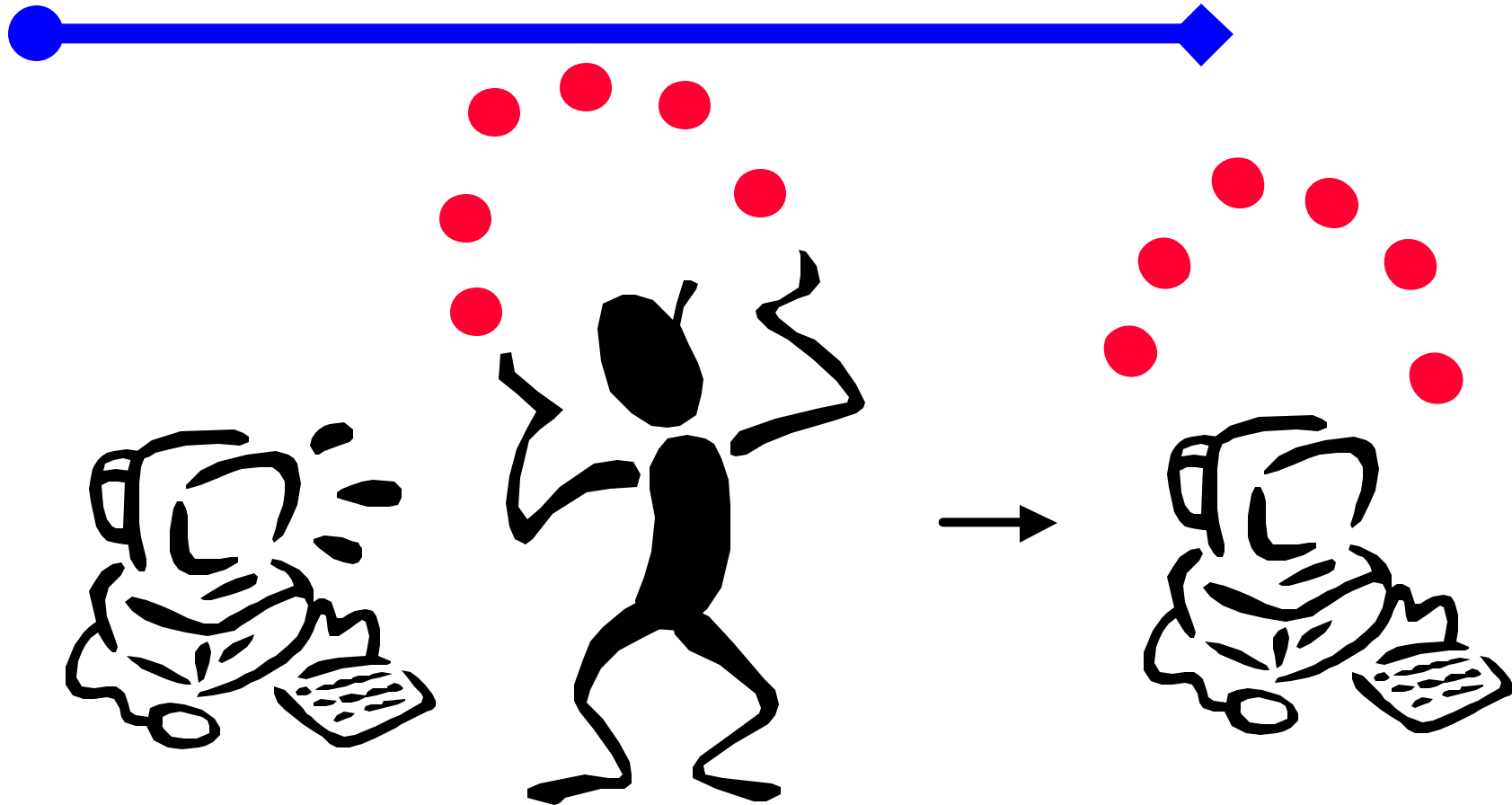
Planning



Planning

- Given a model of the environment (results of actions, ...),
→ Build plan that reaches desired state
- Requires a full model of the environment
 - Often harder to describe than the task at hand

Behavioral Cloning



3 July, 2001



Relational Reinforcement Learning



7

Behavioral Cloning

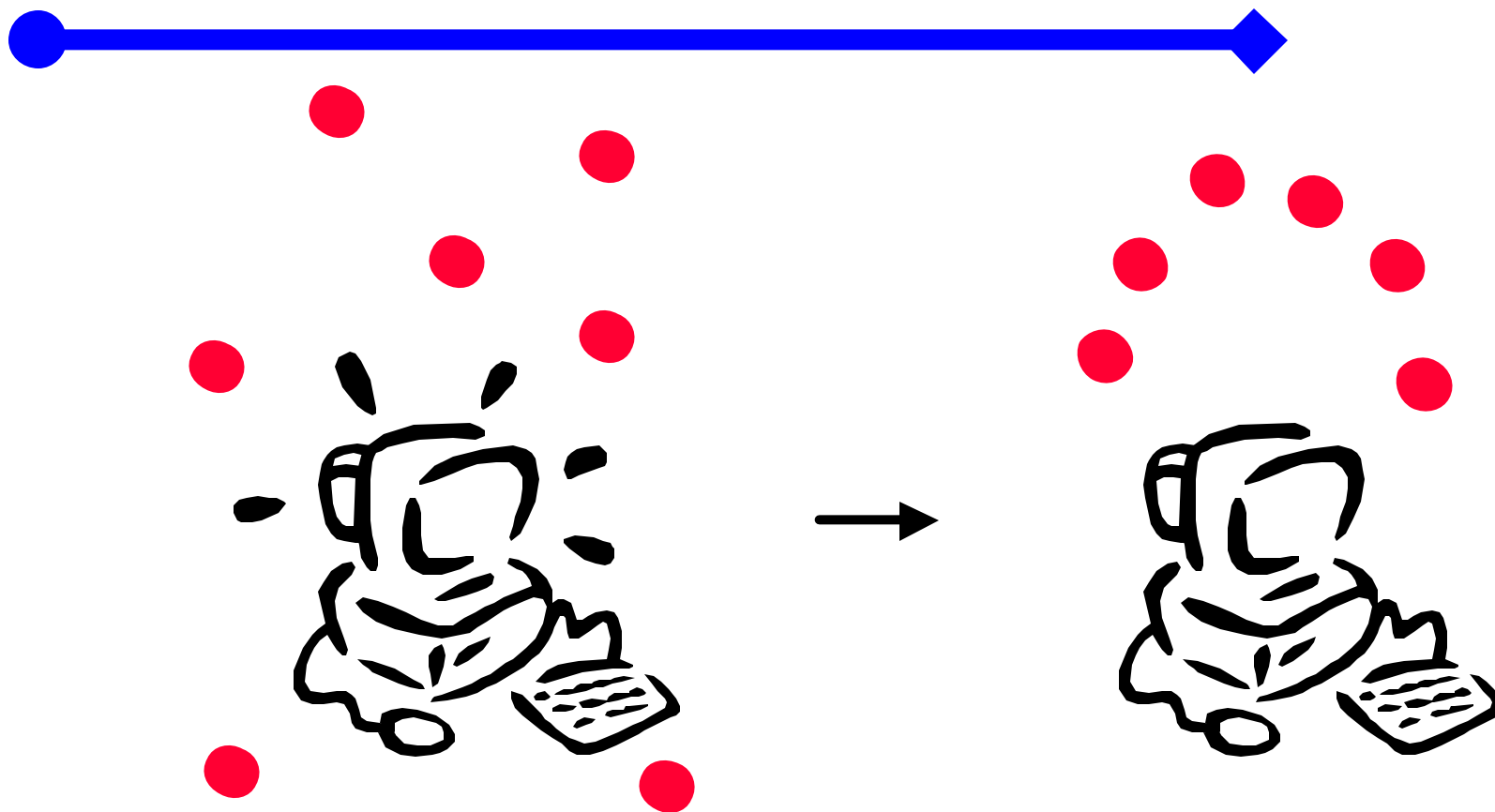
- Imitation of a policy
- Mapping:

State \longrightarrow Action

- Improvement of policy only through “cleanup” effect.

Needs a policy to create a policy

Reinforcement Learning



Reinforcement Learning

- On-line learning algorithm to learn behavior in a state based world.
- Same principle as used by parents on their kids, owners on dogs, ...
 - Let kid/dog/agent act in world.
 - Reinforce (reward/punish) when needed.
 - Observe improvement in behavior.



Task Description

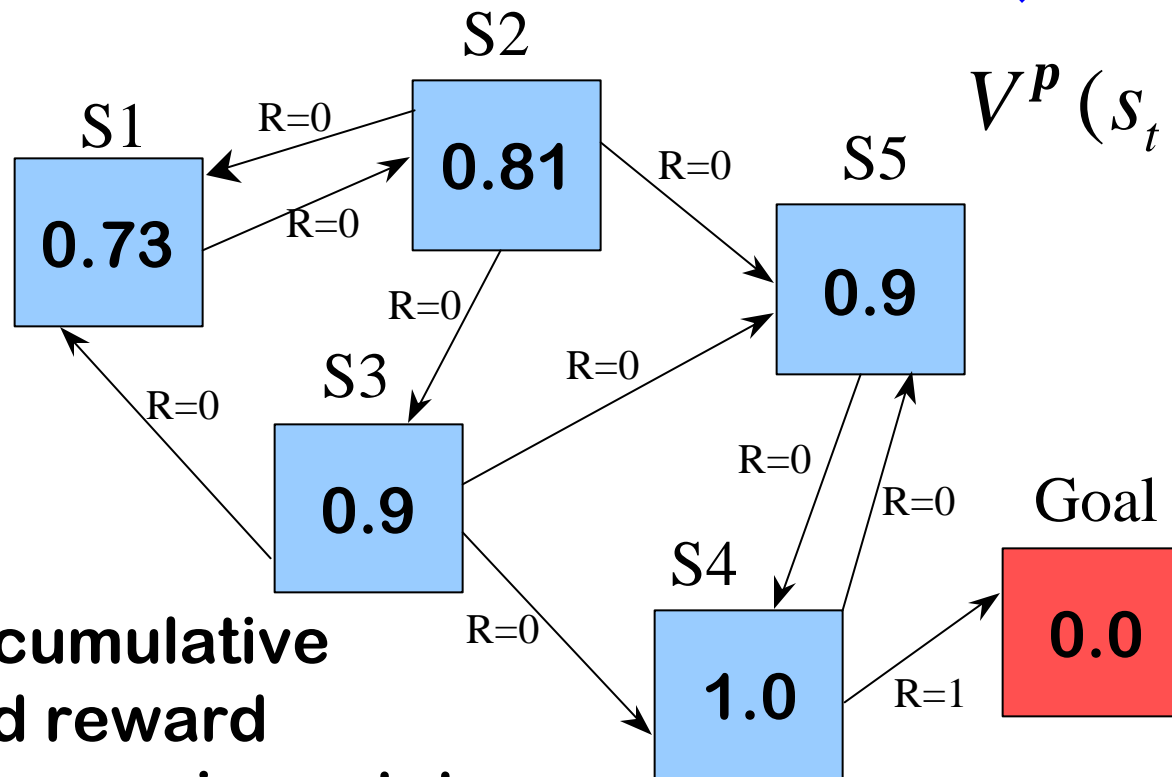
- Given:
 - A set of States S
 - A set of possible Actions A
 - A Transition function $d : S \times A \rightarrow S$
 - A (to the agent) unknown Reward function

$$r : S \times A \rightarrow \mathcal{R}$$

- Find the optimal Policy i.e. the one that maximizes $V^P (s_t) = \sum_{i=0}^{\infty} g^i r_{t+i}$



Utility of States



$$V^P(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Expected cumulative discounted reward starting from a given state.




Utility of States (2)

- Cumulative: global view
- Expected: non-deterministic worlds
- Discounted: finite sum & shortest path
- To find the **optimal** policy: Maximize!

$$V^*(s_t) = \arg \max_p (V^p(s_t))$$

Utility to Policy

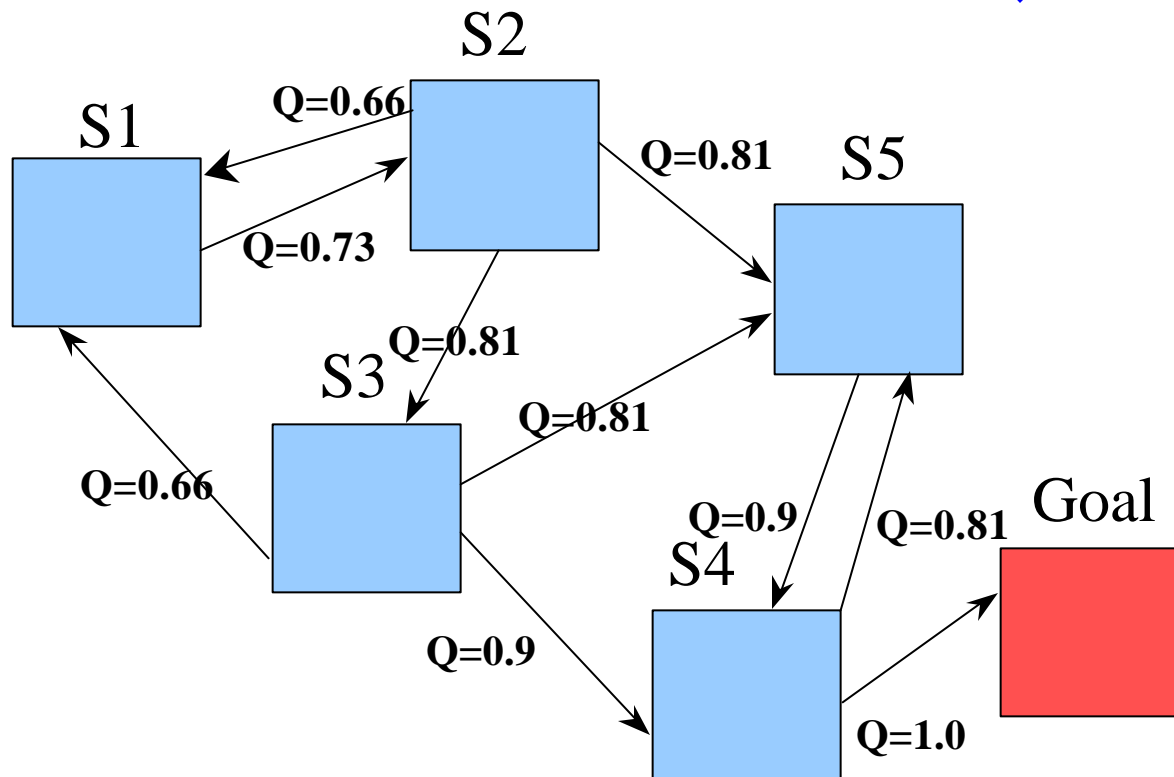

$$p^*(s) = \arg \max_a \left(r(s, a) + gV^*(d(s, a)) \right)$$

- Needs model of environment
 - Which action results in what state?

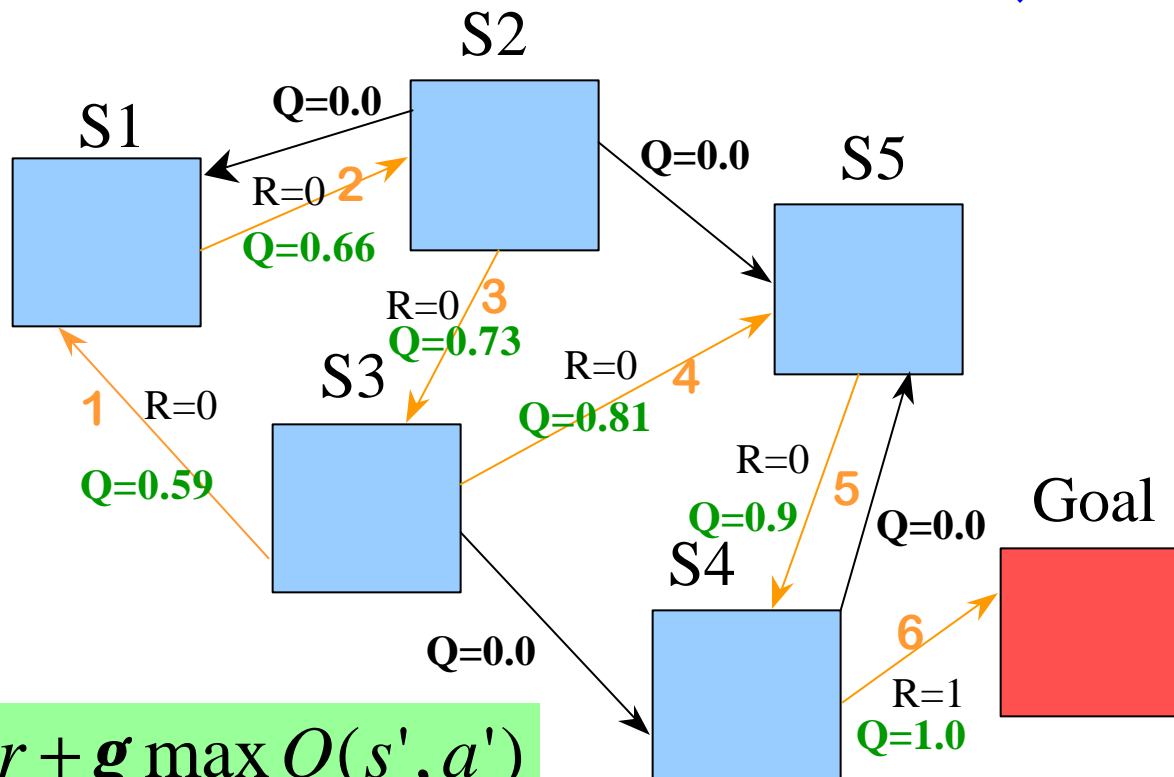
- Q-function:

$$Q(s, a) = r(s, a) + gV^*(d(s, a))$$

Q-Function



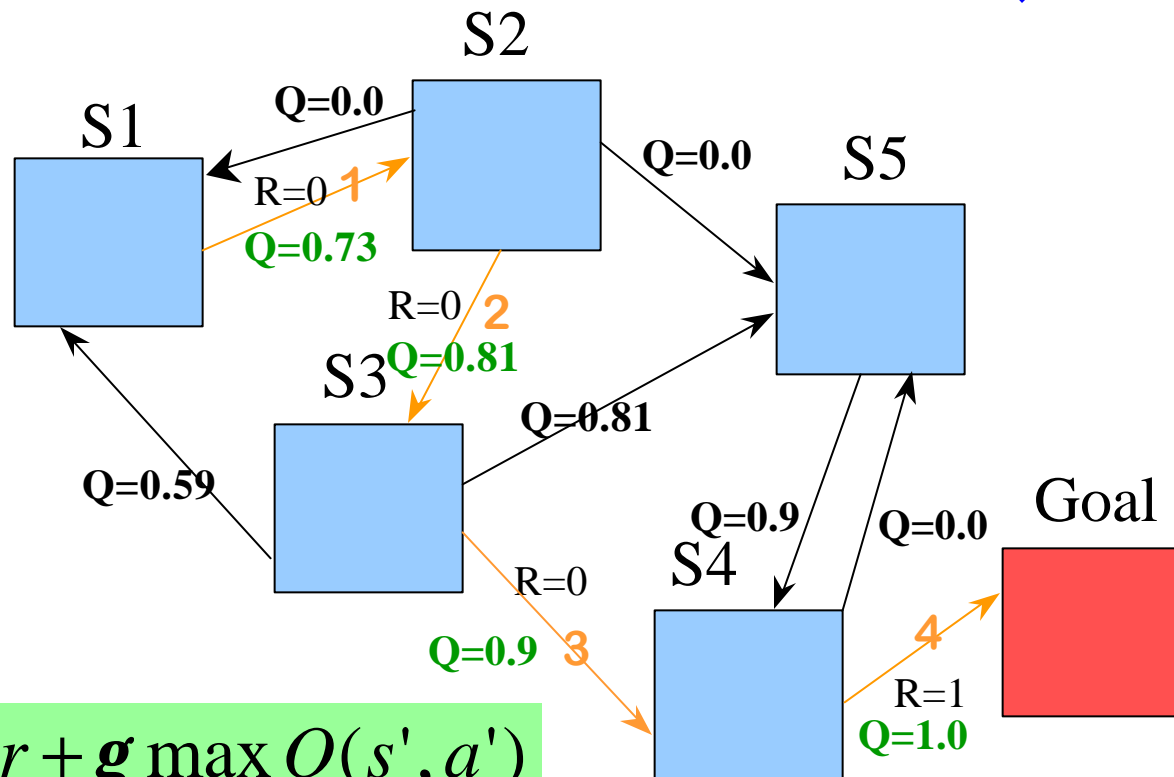
Q-algorithm



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




Q-algorithm (2)



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

Update Rule(s)


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

- **Stochastic Processes**

$$Q(s, a) \leftarrow (1 - \mathbf{a})Q(s, a) + \mathbf{a}[r + \mathbf{g} \max_{a'} Q(s', a')]$$

$$\text{with } \mathbf{a} = \frac{1}{1 + \text{visits}(s, a)}$$

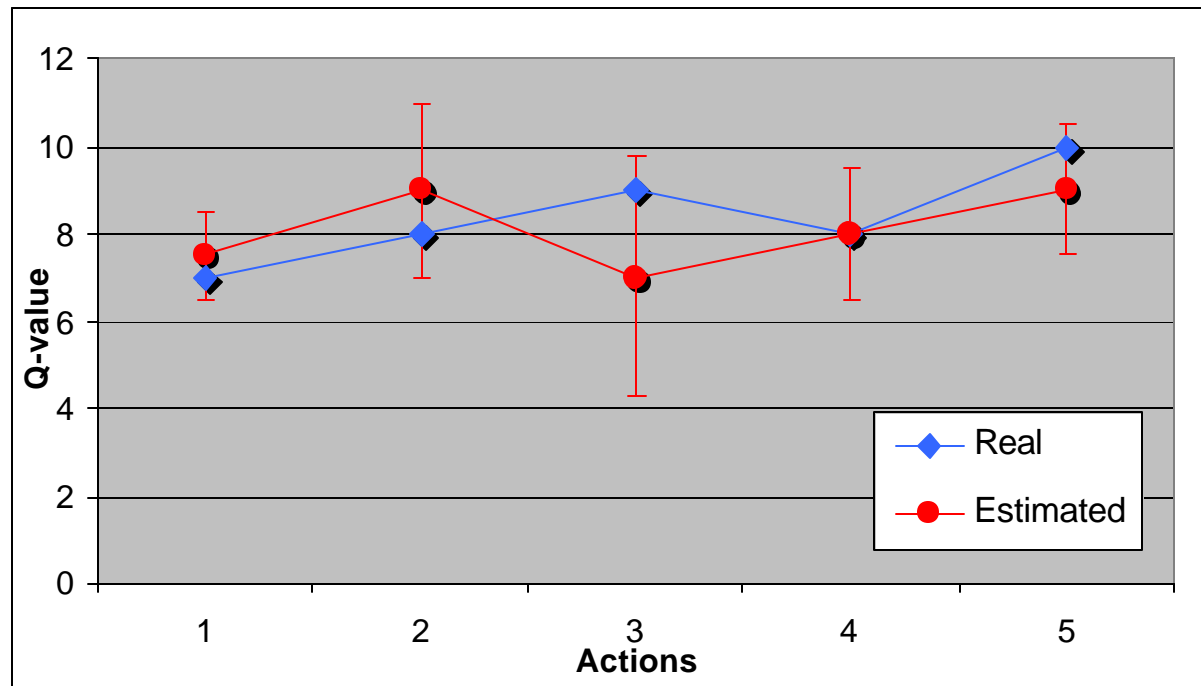
Exploration vs. Exploitation

- “Visit each state-action pair an infinite number of times”
 - infinite number \longrightarrow often enough
- Greedy and ϵ -Greedy
- Boltzmann


$$P(a | s) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$$

Exploration vs. Exploitation

- Interval Based



Some Demos

- 
- Acrobot
 - BlackJack



Overview (again)

- 
- **Automated Behaviour Generation**
 - Why and How

- **Reinforcement Learning & Q-learning**

- Foundations, Algorithm, Variations

- 
- **Relational Reinforcement Learning**
 - Algorithm, Benefits

Overview (again)

- Automated Behaviour Generation
 - Why and How

- Reinforcement Learning

Relational Reinforcement Learning (RRL)

- Tetris example: problems
- RRL-Algorithm
- P-learning
- Demo

Generalizations

- **Size of lookup table:**

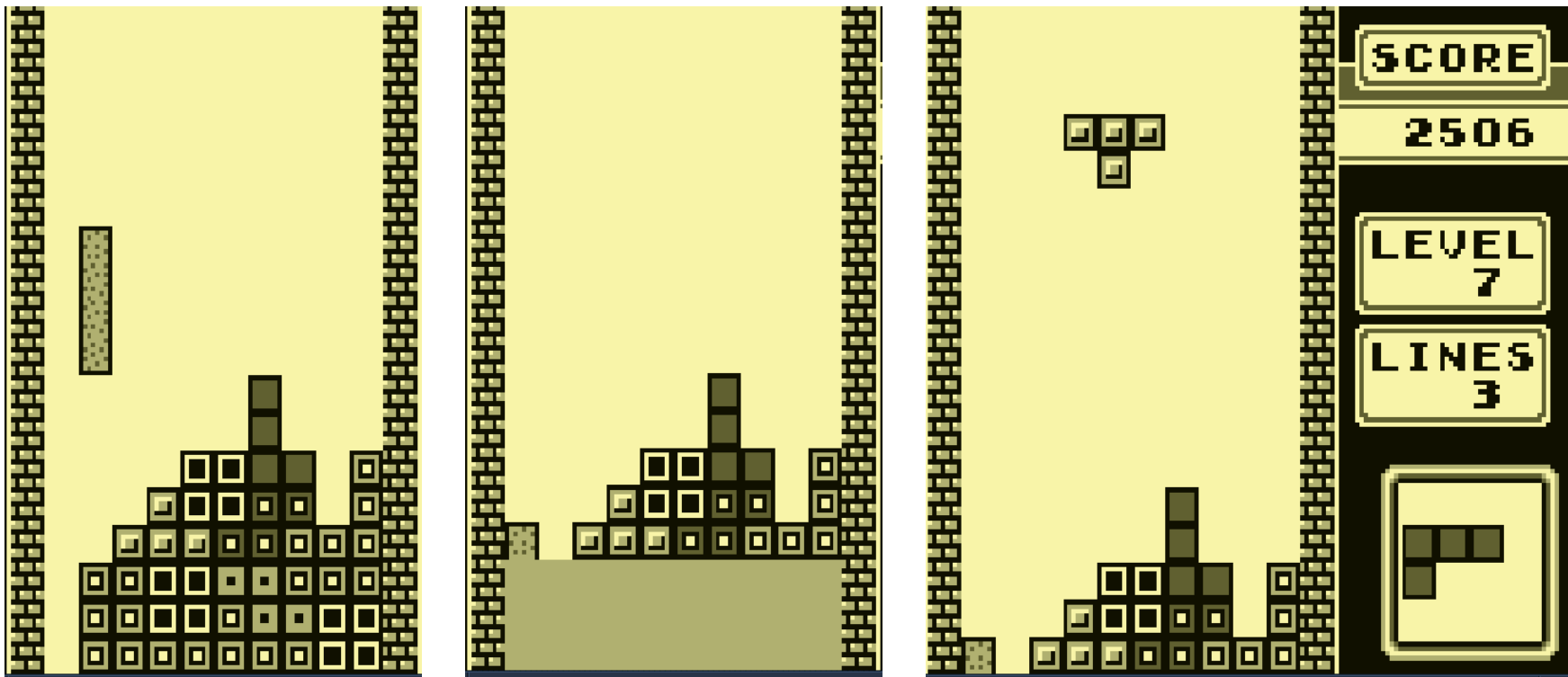
Number of States x

Number of Actions per State

Curse of dimensionality:

**Table size increases exponentially
in the number of attributes**

Example: Tetris



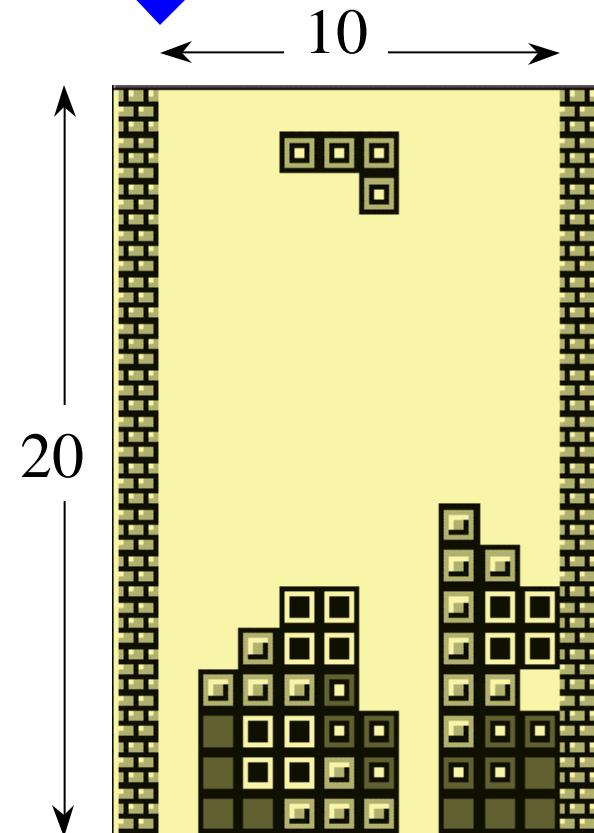
Tetris Properties

- **Large State Space:**

- $2^{200} * 7 * 4 * 10 (* 7)$

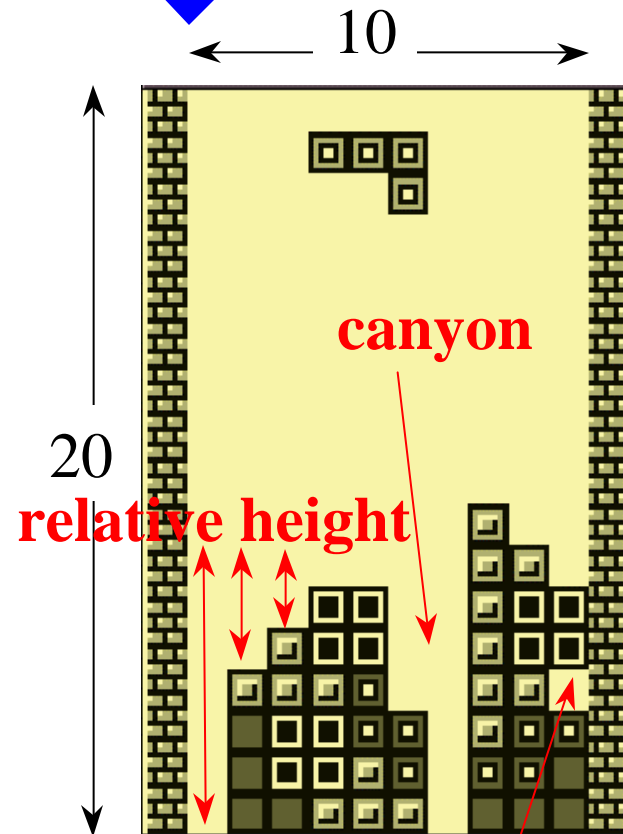
 - Overestimate $\Rightarrow 10^{60}$

- **Similar states, actions and subgoals**



Tetris Properties (2)

- Relative features
 - Local Canyons
 - Depth
 - Shape
 - Relative Height
 - Hole Existence and Placement
 - ...



Tetris Properties (3)

- **Two parts**
 - **Strategic part**
 - **Possible action:** drop(Orientation, Column)
 - **Stochastic:** What blocks will follow?
 - **Arcade part**
 - **Possible actions:** turn, left, right, drop
 - **Deterministic** → planning problem

Tetris Properties (4)

- Similar actions and subgoals
 - Drop(X,Y)
 - Block type
 - Orientation
 - Column
 - Turn, Left, Right, Drop
 - Exact column
- ⇒ Needs relative information only(?)

Q-table Generalizing

Any Regression Technique (given certain constraints)

→ Function Approximation

- Neural Networks
 - Tesauro's BackGammon
- First Order Logic Regression Trees
 - Relational Reinforcement Learning

Relational Reinforcement Learning

- Reinforcement learning +
 inductive logic programming
 - Structural representations
 - Abstraction from specific goals
 - Incremental reinforcement learning

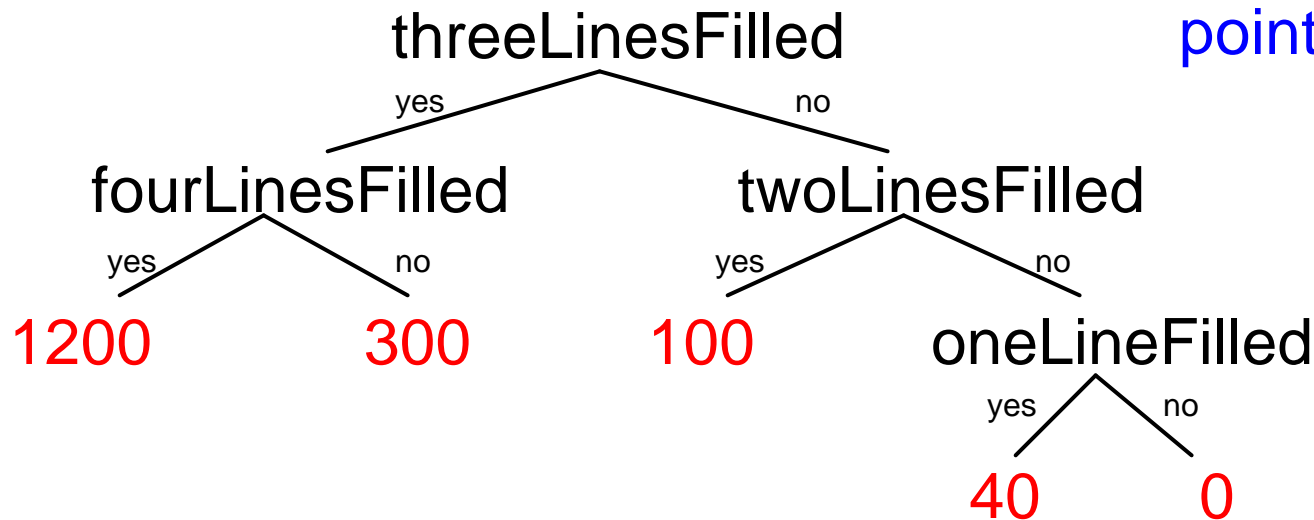
RRL

- **Generalization of Q-table**
 - Relational Representation
- **Represents Q-function as a**
First Order Regression Tree

Decision Trees

- Internal nodes = tests
- Leaves = prediction

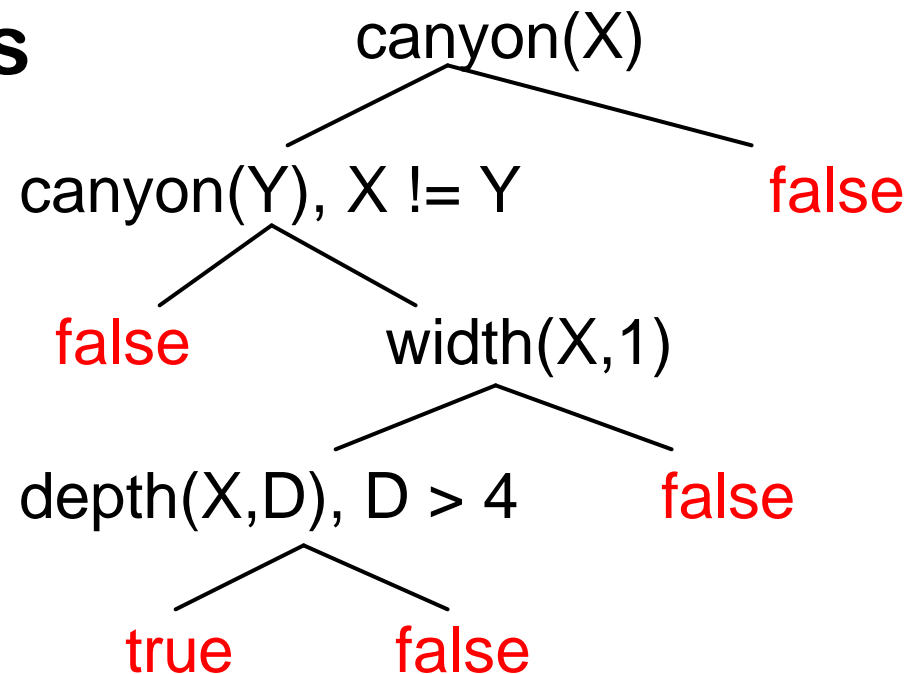
Number of
points scored?



First Order Decision Tree

- Variables in tests
- Relational tests

Possibility for scoring a Tetris?



RRL-algorithm



initialize tree to **a single root node**
while (true)

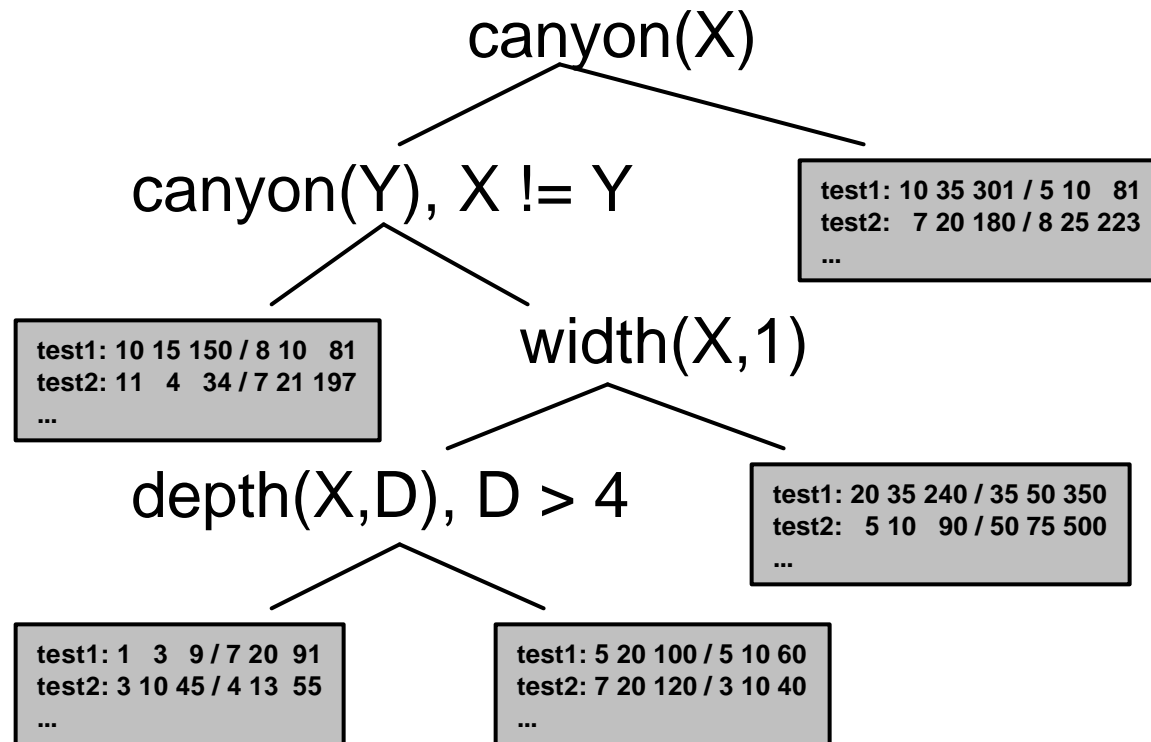
generate episode through the use
of a **standard Q-learning algorithm**
using the current tree as Q-function
generate example (s_i, a_i, q_i) for each
state-action pair encountered
update tree using **the TG-algorithm** and
the generated examples

TG-algorithm



```
create an empty leaf
while (examples available)
  sort example down to leaf
  update statistics in leaf
  if (split needed)
    create two empty leafs
```


TG-algorithm (2)



TG-algorithm (3)

- **Statistics:**
 - **Per leaf and per test:**
 - Number of positive and negative examples
 - Sum of q-values
 - Sum of squared q-values
 - **Allows the calculation of significance of tests**

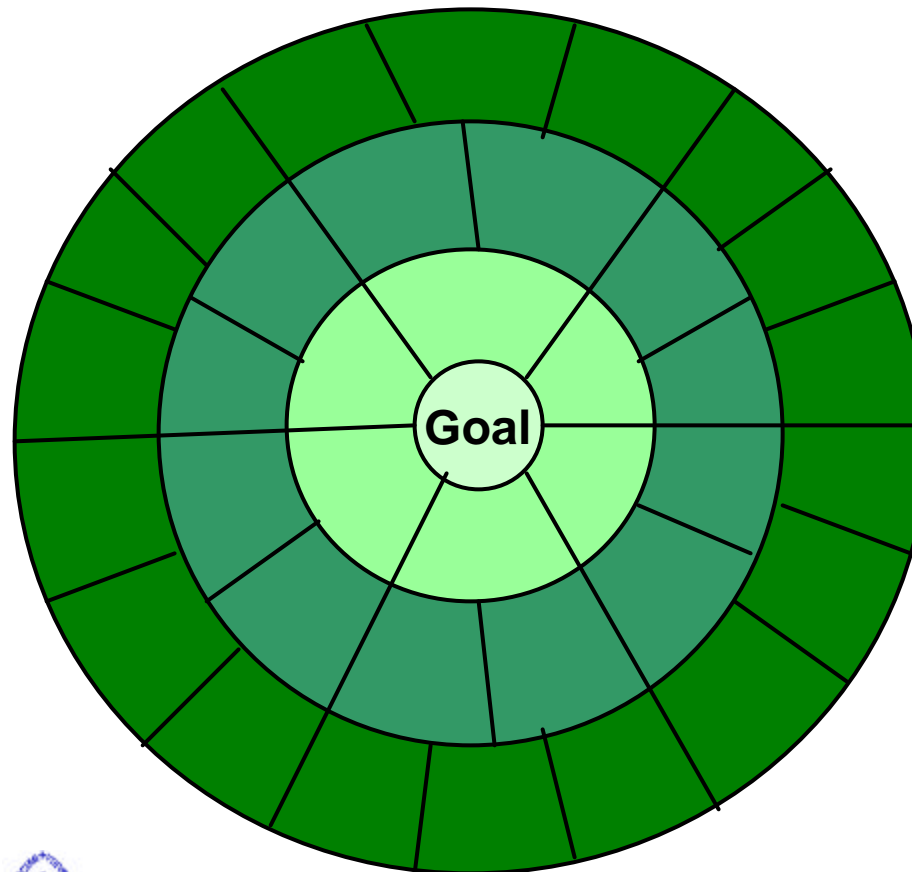
Q-tree to Policy

- 
- **Observe state**
 - **For each possible action**
 - Use Q-tree to predict q-value
 - Choose action with highest q-value

P(olicy)-learning

- **Generate a P-tree that predicts the optimality of an action**
 - True if action is part of optimal policy
 - False otherwise
- **Easier to represent than Q-function**
 - Q-function \sim distance to reward

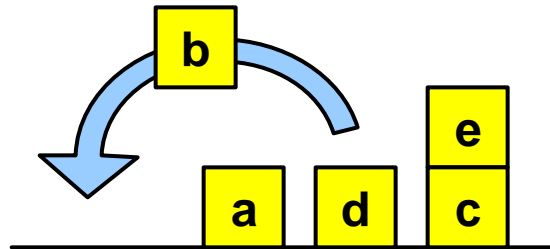
Q-Function and Distance



P-learning

- For each state visited
 - Lookup all the possible actions and predict q-values
 - Use TG-algorithm to generate a first order classification tree to predict optimal or not
- Generalizes better than Q-function

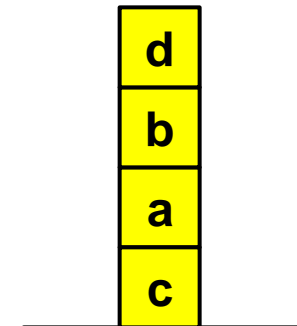
Blocks World



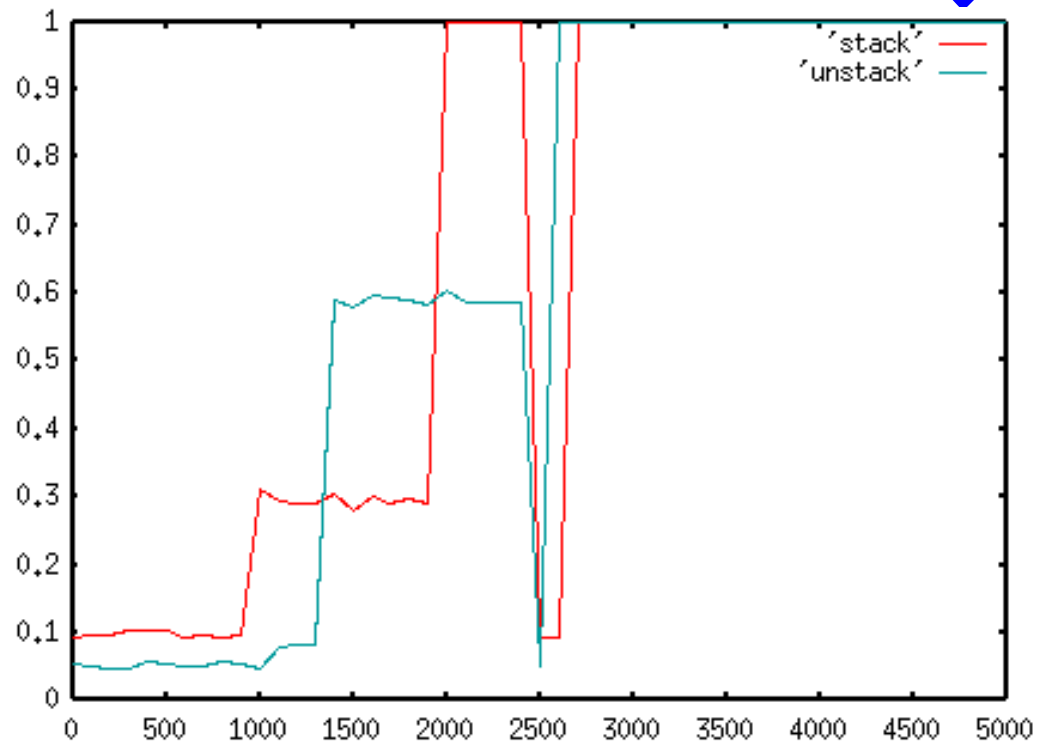
Unstacking



Stacking



Some Results



P-learning started at episode 2500

Some Results (2)

- DiggerDemo

Where to go now?

- References: in the book
- Slides:
<http://www.cs.kuleuven.ac.be/~kurtd/>
- Prague: after lunch

