TRIAL-BASED HEURISTIC TREE SEARCH FOR FINITE HORIZON MDPS

Thomas Keller and Malte Helmert

Presented by: Ryan Berryhill

Outline

- Motivation
- Background
- THTS framework
- THTS algorithms
- Results

Motivation

- Advances the state-of-the-art in finite-horizon MDP search based on Monte Carlo planning and heuristics
  - Finite-horizon MDP: specialization of MDPs
- Unifies existing techniques into a common framework
  - Monte Carlo, heuristic search, dynamic programming
  - How can we talk about algorithms for this problem in a unified way?
- Focus on anytime-optimal algorithms
  - Converge towards an optimal solution given enough time
  - Give reasonably good results whenever they are stopped

Background

- Finite-horizon MDPs
  - An MDP with an added parameter *H* (the *horizon*)
  - At most *H* transitions between initial state and terminal states
  - The horizon can be thought of as the agent's lifespan
  - Decision nodes where the agent makes a decision
  - Chance nodes where the environment makes a "decision"

- Planning problem: maximize reward obtained in *H* steps
  - We get a *non-stationary policy*
  - Agent may act differently if it has 5 seconds to live versus 50 years

Rollout-Based Monte Carlo Planning

- Rollout-based Monte Carlo planning
  - Run a number of episodes starting from the current state
  - Episodes are generated by sampling actions in each state visited
  - Take the best action observed across all episodes

- Simple approach: sample actions uniformly

- UCT: treat states as multi-armed bandits when sampling
  - Use the UCB1 algorithm to solve the bandit problem
  - UCB1: take the action that optimizes $\overline{x_j} + \sqrt{\frac{2 \ln n}{n_j}}$

THTS Framework

- Goal: Make algorithms fit into this framework
- Algorithms specify:
  - heuristic function
  - backup function
  - action selection
  - outcome selection
  - trial length

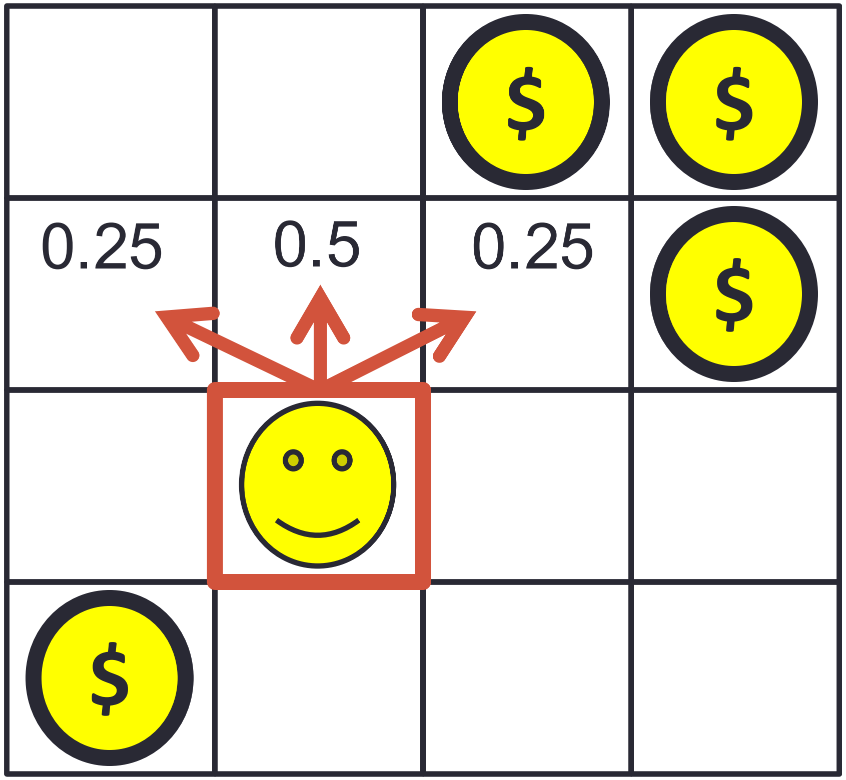
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T)**:**
    $n_0 \leftarrow$ getRootNode(M)
    while not solved(n_0) and time() $< T$ **do**
        visitDecisionNode(n_0)
    return greedyAction(n_0)

visitDecisionNode(Node n_d)**:**
    if n_d was never visited **then** initializeNode(n_d)
    $N \leftarrow$ selectAction(n_d)
    for $n_c \in N$ **do**
        visitChanceNode(n_c)
    backupDecisionNode(n_d)

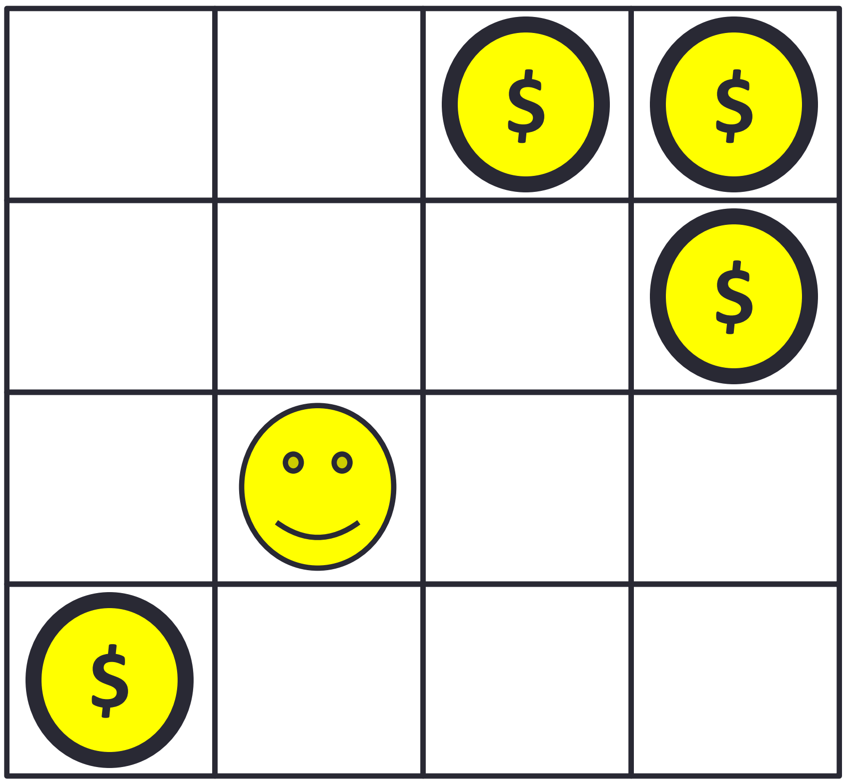
visitChanceNode(Node n_c)**:**
    $N \leftarrow$ selectOutcome(n_c)
    for $n_d \in N$ **do**
        visitDecisionNode(n_d)
    backupChanceNode(n_c)

Example



- Grid with rewards
- Assume *H* = 5
- Actions: move in four directions
  - 50% chance of ending up where you wanted to go
  - 25% change of going to one of the two neighbors
  - E.g. Going up

Example



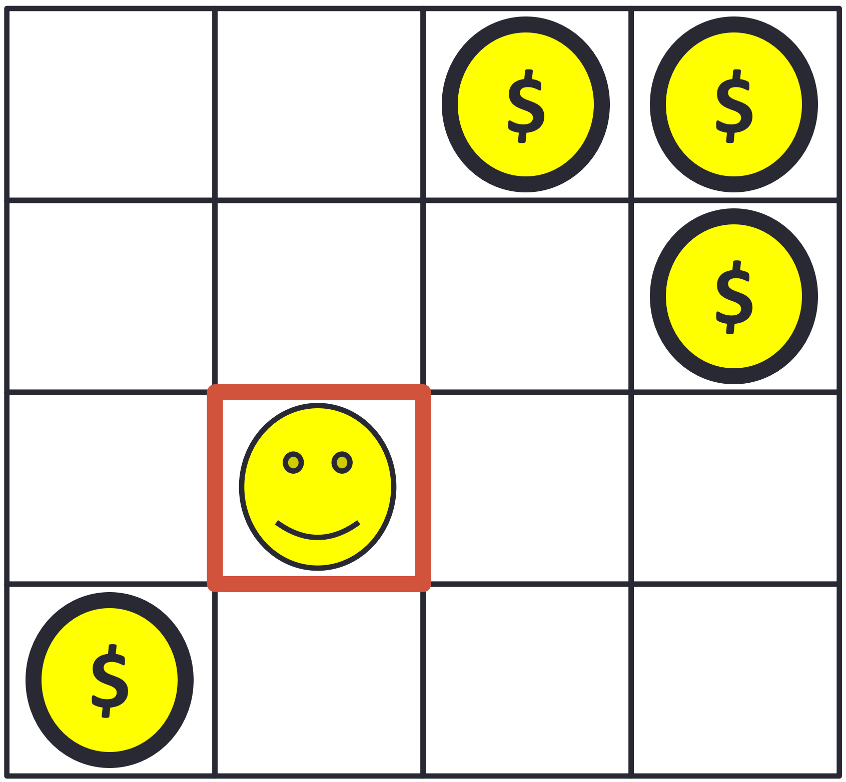
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
  $n_0 \leftarrow$ getRootNode(M)
  while not solved(n_0) and time() $< T$ **do**
    visitDecisionNode(n_0)
  return greedyAction(n_0)

visitDecisionNode(Node n_d):
  if n_d was never visited **then** initializeNode(n_d)
  $N \leftarrow$ selectAction(n_d)
  for $n_c \in N$ **do**
    visitChanceNode(n_c)
  backupDecisionNode(n_d)

visitChanceNode(Node n_c):
  $N \leftarrow$ selectOutcome(n_c)
  for $n_d \in N$ **do**
    visitDecisionNode(n_d)
  backupChanceNode(n_c)

Example



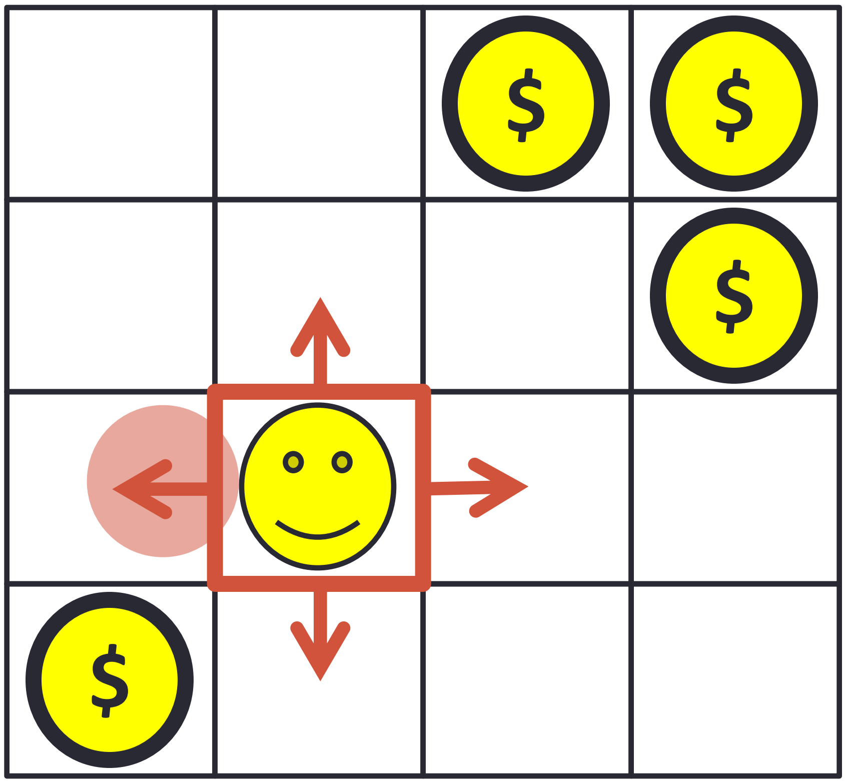
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
    $n_0 \leftarrow$ getRootNode(M)
    while not solved(n_0) and time() $< T$ **do**
        visitDecisionNode(n_0)
    return greedyAction(n_0)

visitDecisionNode(Node n_d):
    if n_d was never visited **then** initializeNode(n_d)
    $N \leftarrow$ selectAction(n_d)
    for $n_c \in N$ **do**
        visitChanceNode(n_c)
    backupDecisionNode(n_d)

visitChanceNode(Node n_c):
    $N \leftarrow$ selectOutcome(n_c)
    for $n_d \in N$ **do**
        visitDecisionNode(n_d)
    backupChanceNode(n_c)

Example



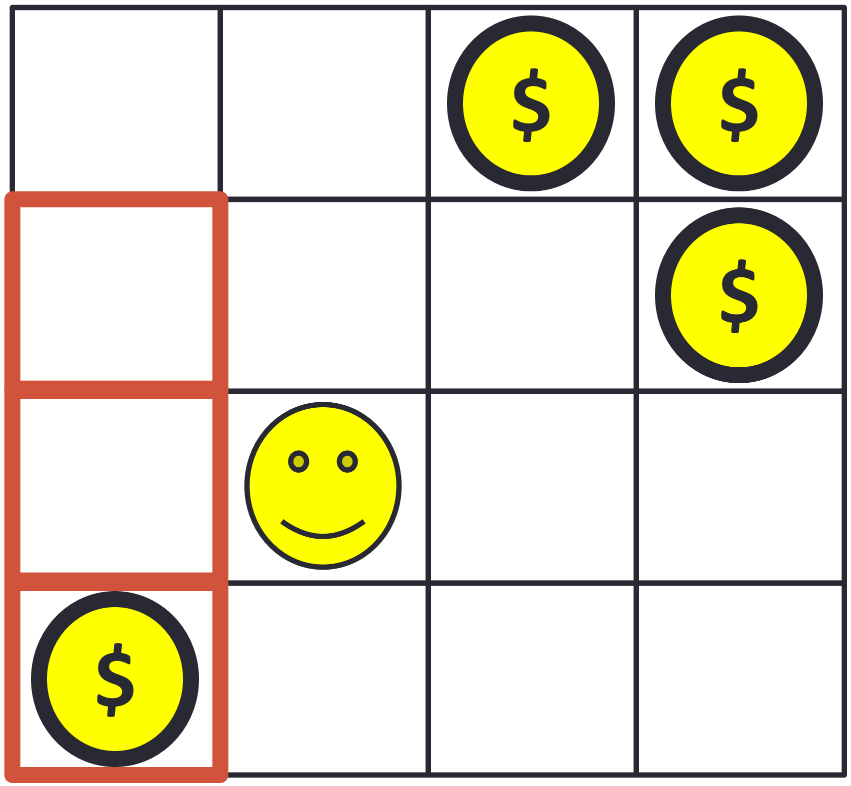
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
  $n_0 \leftarrow$ getRootNode(M)
  while not solved(n_0) and time() $< T$ **do**
    visitDecisionNode(n_0)
  return greedyAction(n_0)

visitDecisionNode(Node n_d):
  if n_d was never visited **then** initializeNode(n_d)
  $N \leftarrow$ selectAction(n_d)
  for $n_c \in N$ **do**
    visitChanceNode(n_c)
  backupDecisionNode(n_d)

visitChanceNode(Node n_c):
  $N \leftarrow$ selectOutcome(n_c)
  for $n_d \in N$ **do**
    visitDecisionNode(n_d)
  backupChanceNode(n_c)

Example



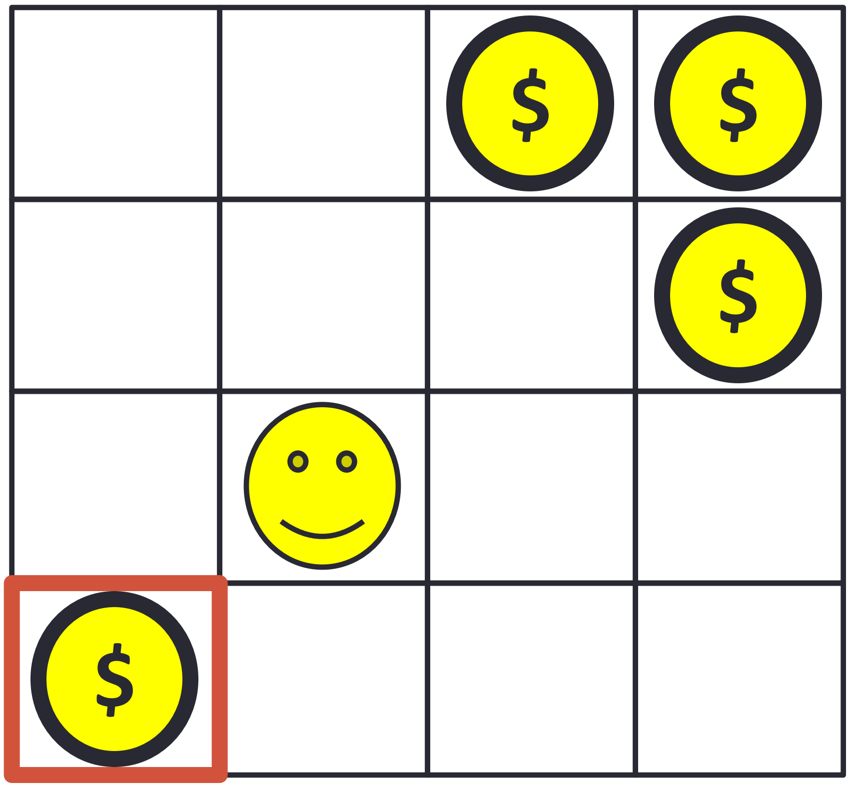
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
    $n_0 \leftarrow$ getRootNode(M)
    while not solved(n_0) and time$() < T$ **do**
        visitDecisionNode(n_0)
    return greedyAction(n_0)

visitDecisionNode(Node n_d):
    if n_d was never visited **then** initializeNode(n_d)
    $N \leftarrow$ selectAction(n_d)
    for $n_c \in N$ **do**
        visitChanceNode(n_c)
    backupDecisionNode(n_d)

visitChanceNode(Node n_c):
    $N \leftarrow$ selectOutcome(n_c)
    for $n_d \in N$ **do**
        visitDecisionNode(n_d)
    backupChanceNode(n_c)

Example



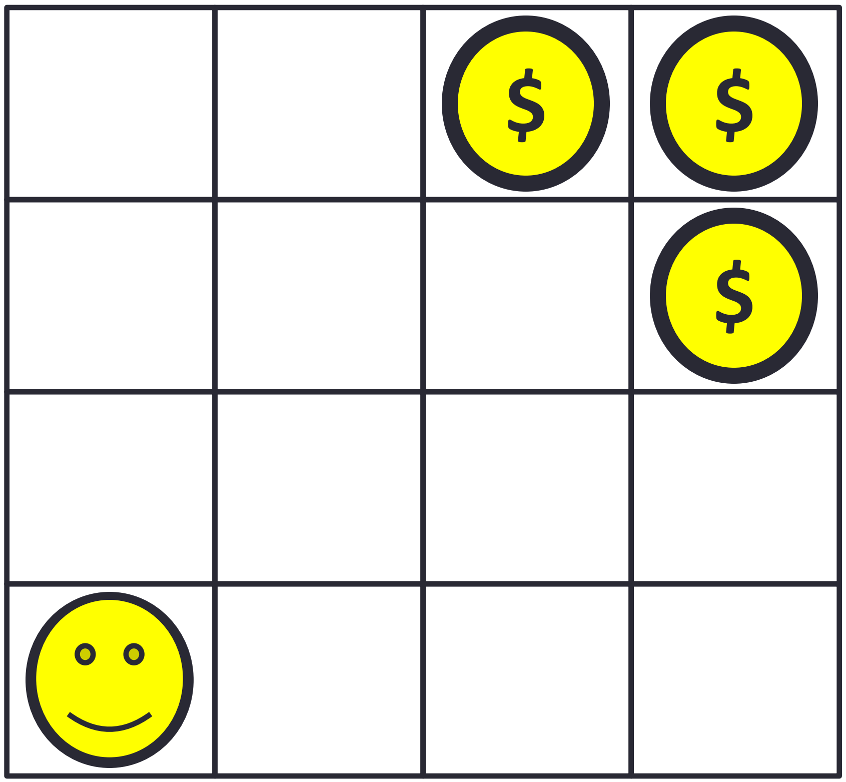
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
    $n_0 \leftarrow$ getRootNode(M)
    while not solved(n_0) and time() $< T$ **do**
        visitDecisionNode(n_0)
    return greedyAction(n_0)

visitDecisionNode(Node n_d):
    if n_d was never visited **then** initializeNode(n_d)
    $N \leftarrow$ selectAction(n_d)
    for $n_c \in N$ **do**
        visitChanceNode(n_c)
    backupDecisionNode(n_d)

visitChanceNode(Node n_c):
    $N \leftarrow$ selectOutcome(n_c)
    for $n_d \in N$ **do**
        visitDecisionNode(n_d)
    backupChanceNode(n_c)

Example



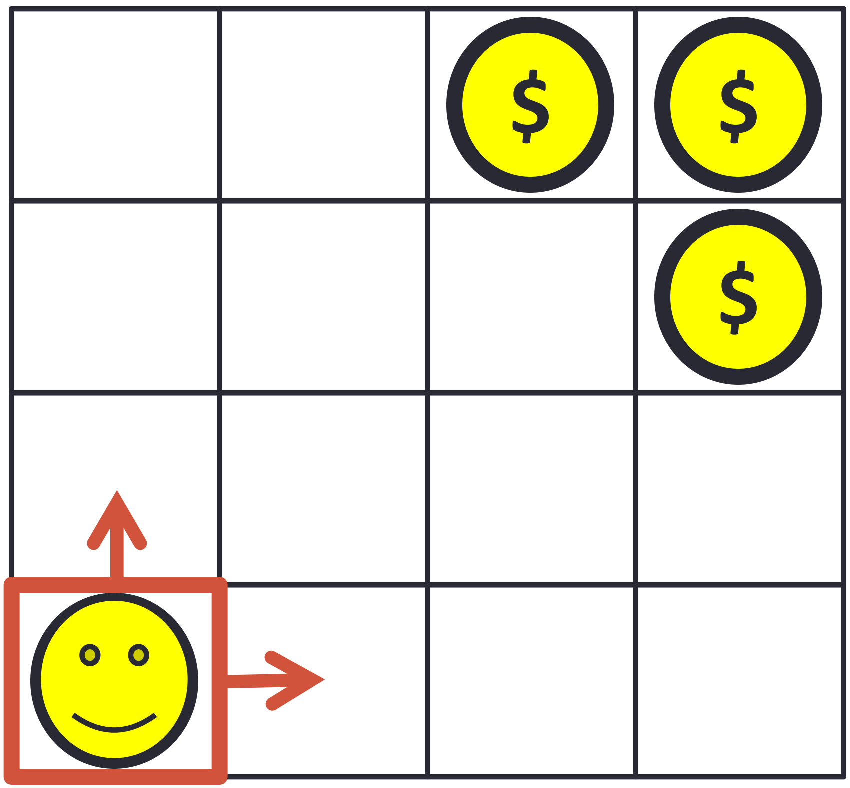
Algorithm 1: The THTS schema.

THTS(MDP M, timeout T):
  $n_0 \leftarrow$ getRootNode(M)
  while not solved(n_0) and time() $< T$ **do**
    visitDecisionNode(n_0)
  return greedyAction(n_0)

visitDecisionNode(Node n_d):
  if n_d was never visited **then** initializeNode(n_d)
  $N \leftarrow$ selectAction(n_d)
  for $n_c \in N$ **do**
    visitChanceNode(n_c)
  backupDecisionNode(n_d)

visitChanceNode(Node n_c):
  $N \leftarrow$ selectOutcome(n_c)
  for $n_d \in N$ **do**
    visitDecisionNode(n_d)
  backupChanceNode(n_c)

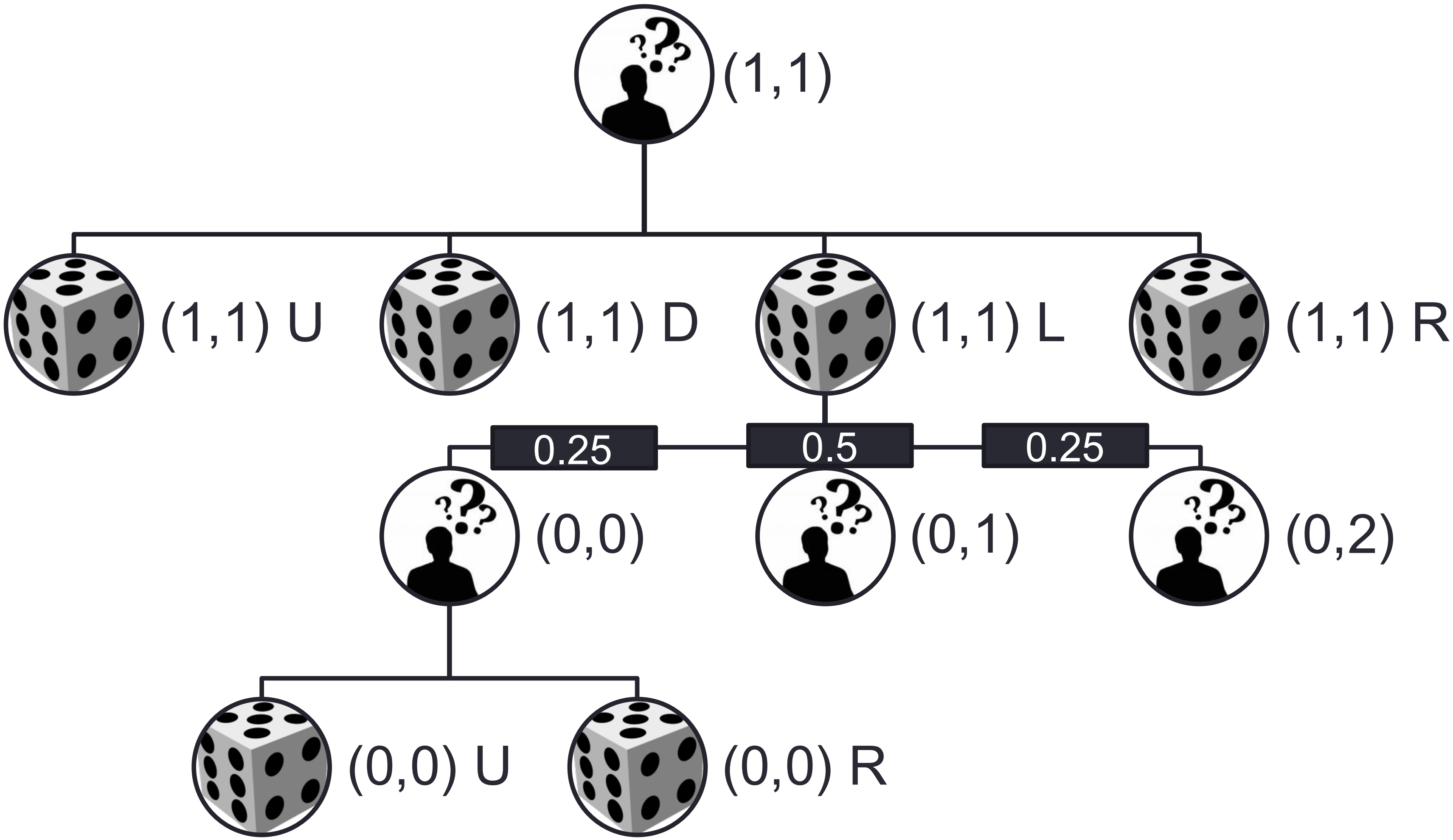
Example



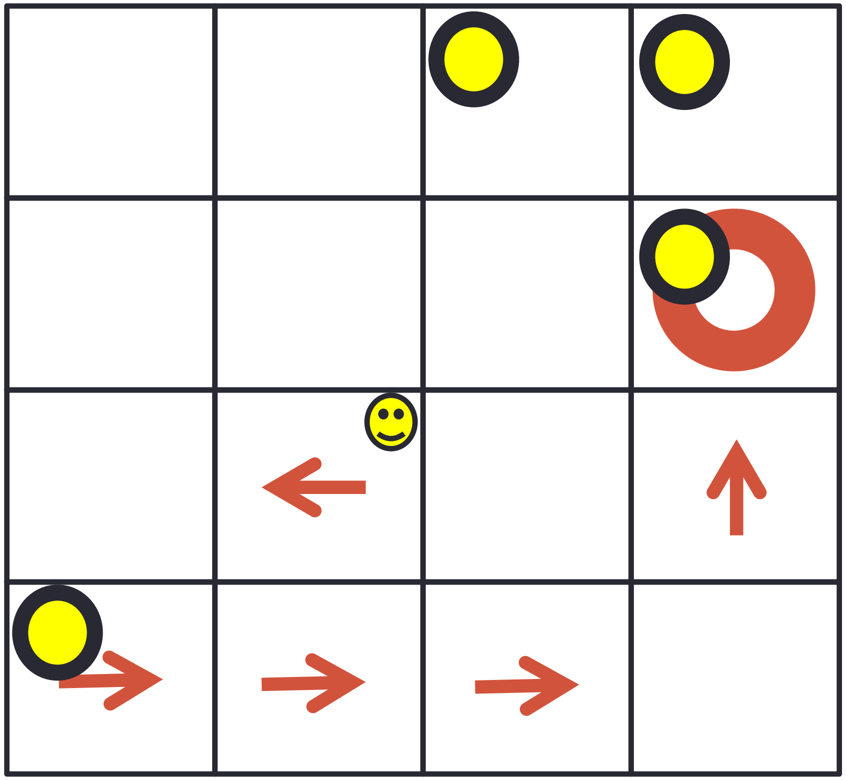
Algorithm 1: The THTS schema.

```
THTS(MDP M, timeout T):
    n_0 ← getRootNode(M)
    while not solved(n_0) and time() < T do
        visitDecisionNode(n_0)
    return greedyAction(n_0)
visitDecisionNode(Node n_d):
    if n_d was never visited then initializeNode(n_d)
    N ← selectAction(n_d)
    for n_c ∈ N do
        visitChanceNode(n_c)
    backupDecisionNode(n_d)
visitChanceNode(Node n_c):
    N ← selectOutcome(n_c)
    for n_d ∈ N do
        visitDecisionNode(n_d)
    backupChanceNode(n_c)
```

Example



Example

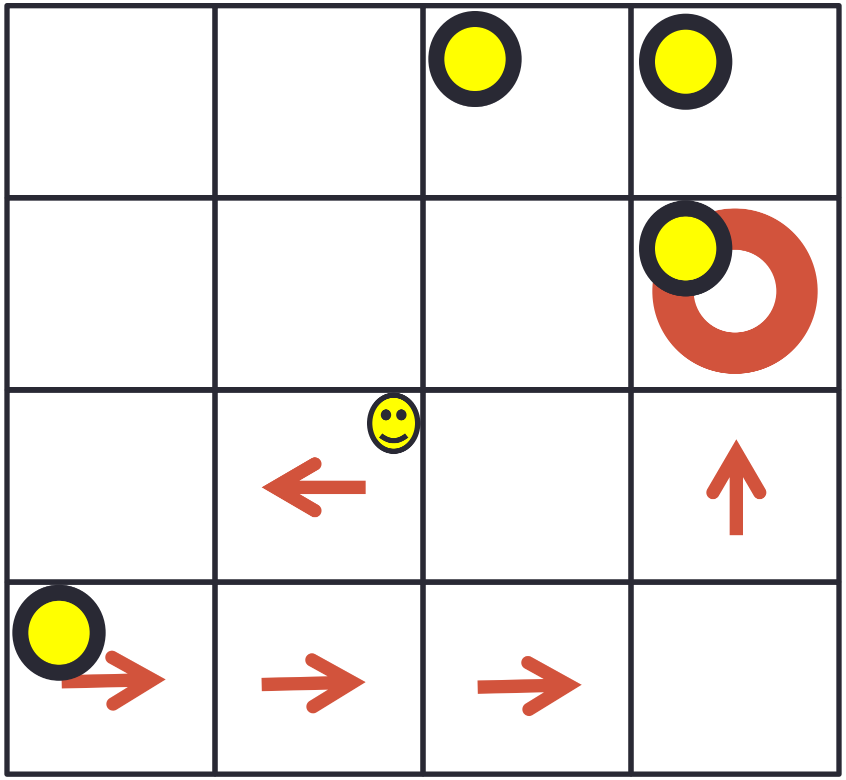


- Say it took the shown actions and ended up at **O**
- Reward is 2
- We need to push this information back up the tree

visitDecisionNode(Node n_d)**:**
    if n_d was never visited **then** initializeNode(n_d)
    $N \leftarrow$ selectAction(n_d)
    for $n_c \in N$ **do**
        visitChanceNode(n_c)
    backupDecisionNode(n_d)
visitChanceNode(Node n_c)**:**
    $N \leftarrow$ selectOutcome(n_c)
    for $n_d \in N$ **do**
        visitDecisionNode(n_d)
    backupChanceNode(n_c)

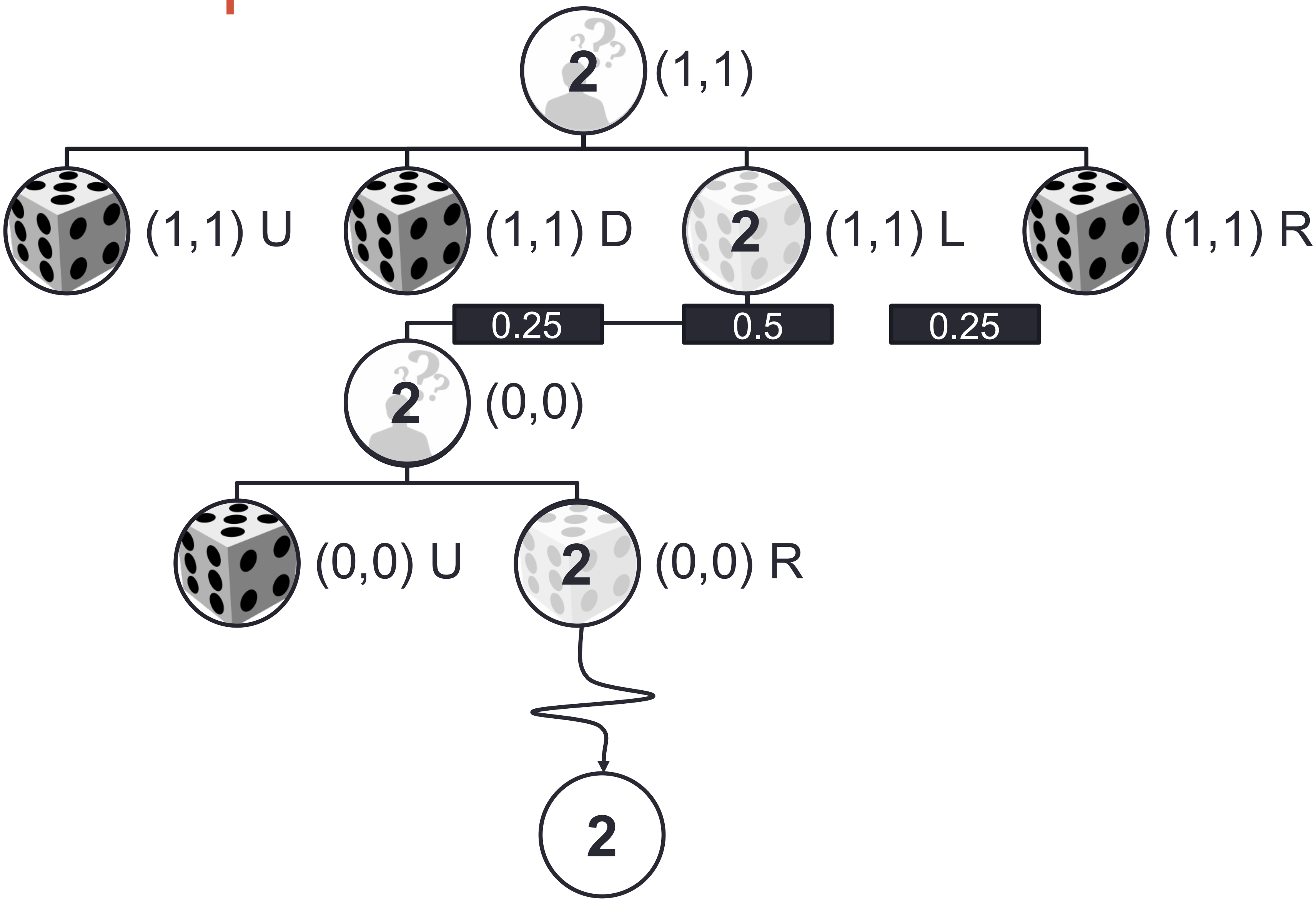
Example

- Use standard techniques again
  - E.g. Monte Carlo backups – average over all previous trials
  - In this example, the agent learns:

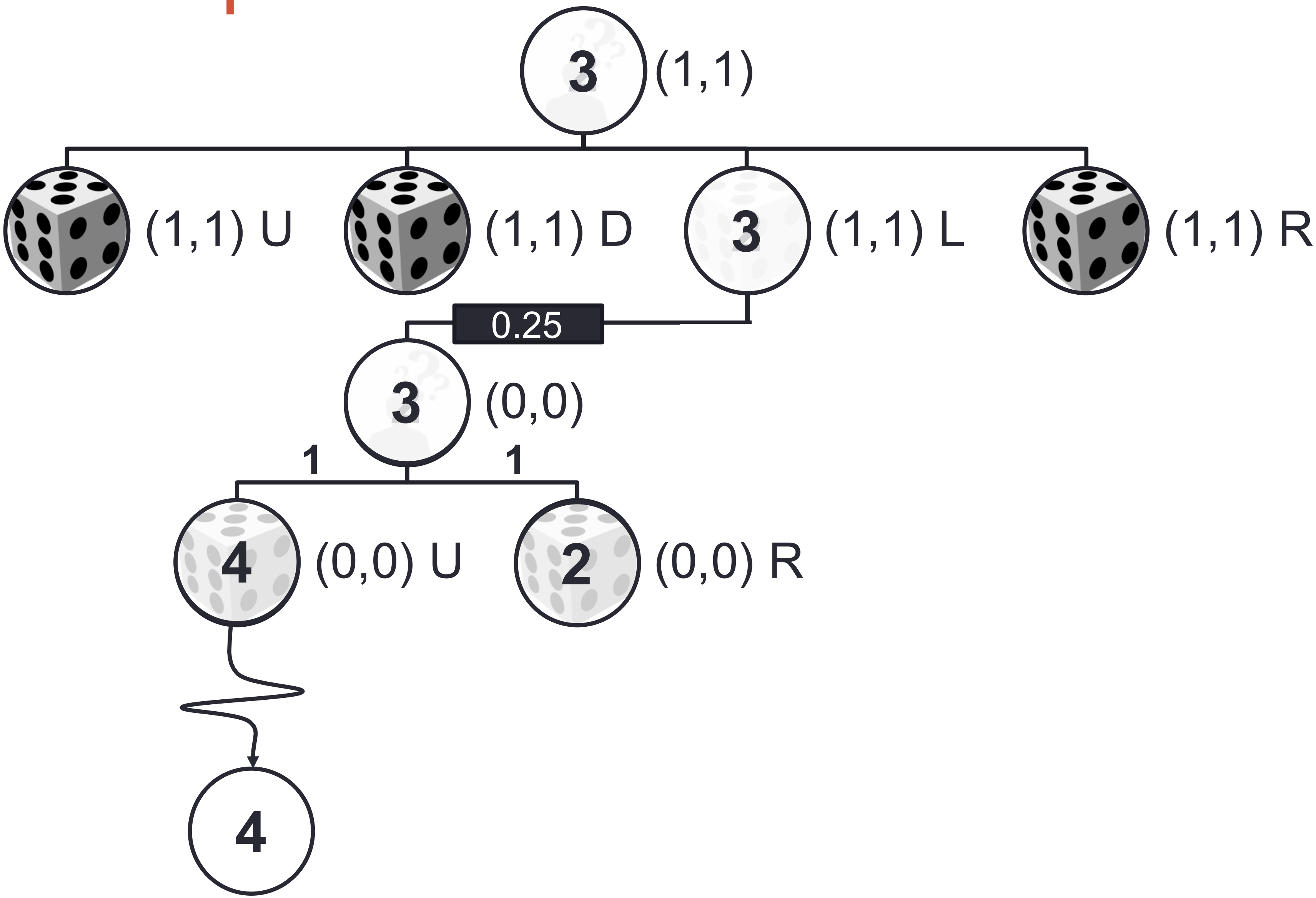


- The expected reward of going up in position (4,2) at time 5 is worth 2
- The expected reward of state (4,2) at time 5 is 2
- Right at (3,1) at time 4 is worth 2
- etc.

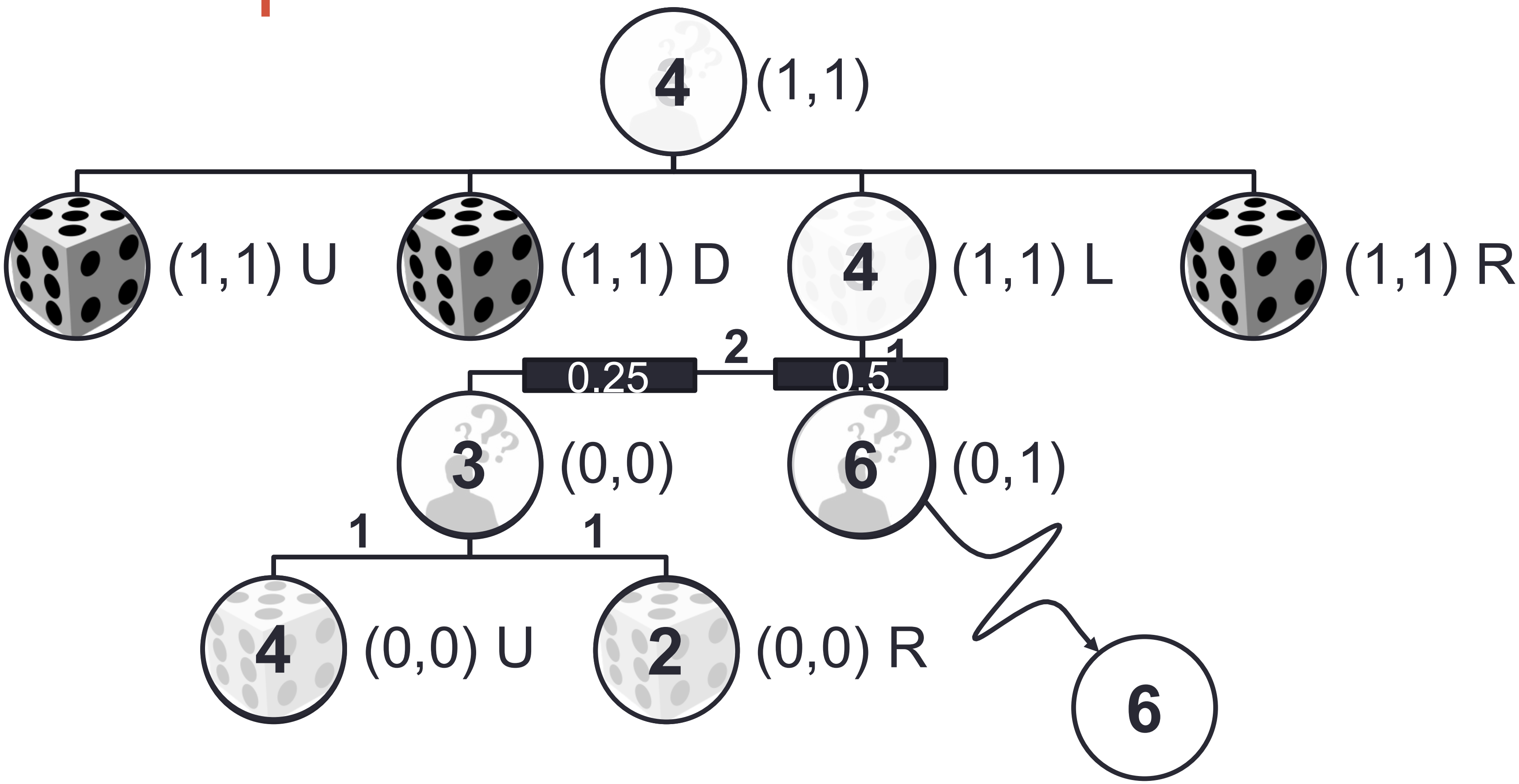
Example



Example



Example



THTS Framework

- Now we can describe this simple THTS algorithm:
  - Heuristic: blind
  - Backup: Monte Carlo
  - Action selection: uniform random
  - Outcome selection: Monte Carlo
  - Trial length: until terminal is hit
- How do other algorithms vary?
  - This framework makes it easy to describe algorithms that fit into it

UCT as a THTS

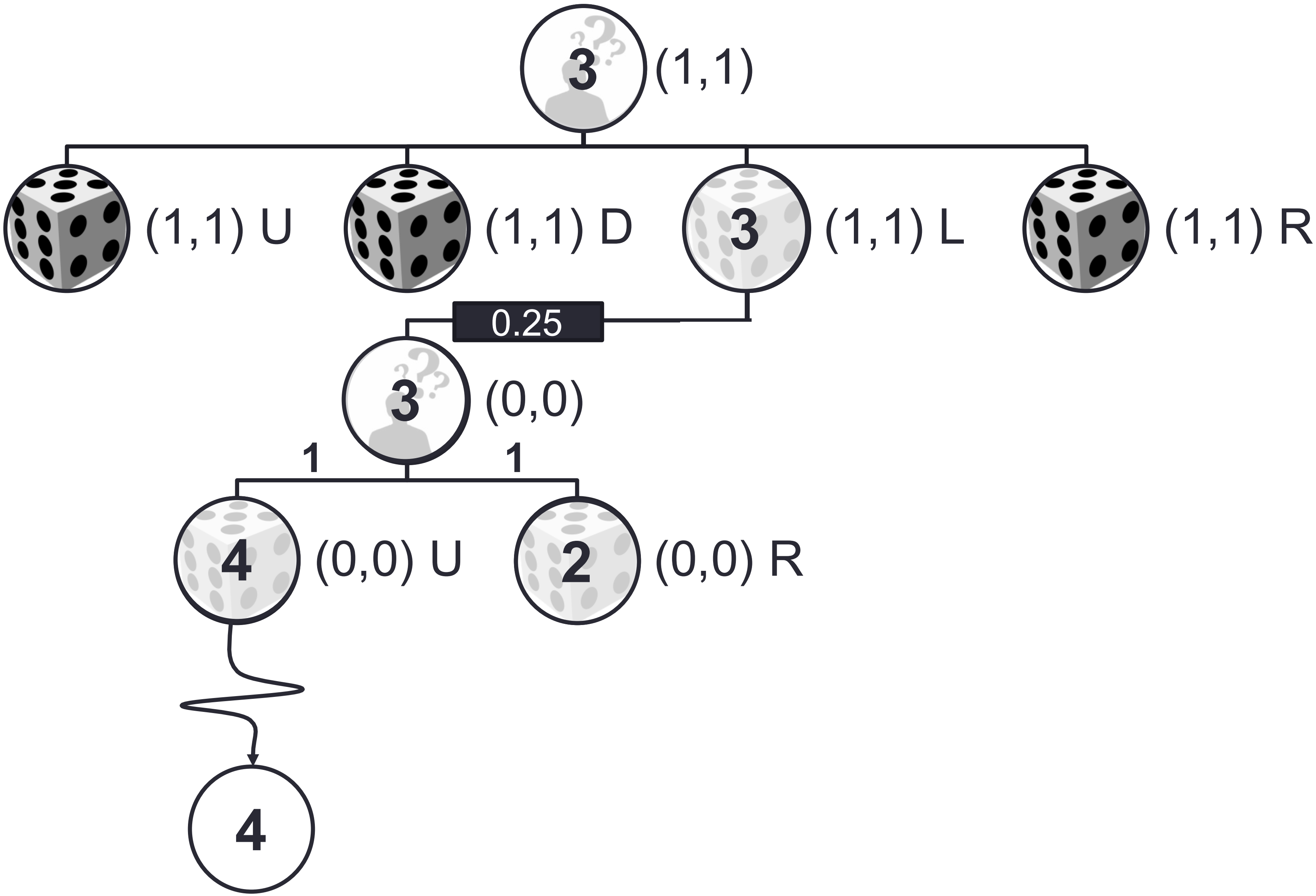
- UCT:
  - **Heuristic: any reasonable choice (even inadmissible)**
  - Backup: Monte Carlo
  - **Action selection: UCB1**
  - Outcome selection: Monte Carlo
  - Trial length: until terminal is hit

Max-UCT

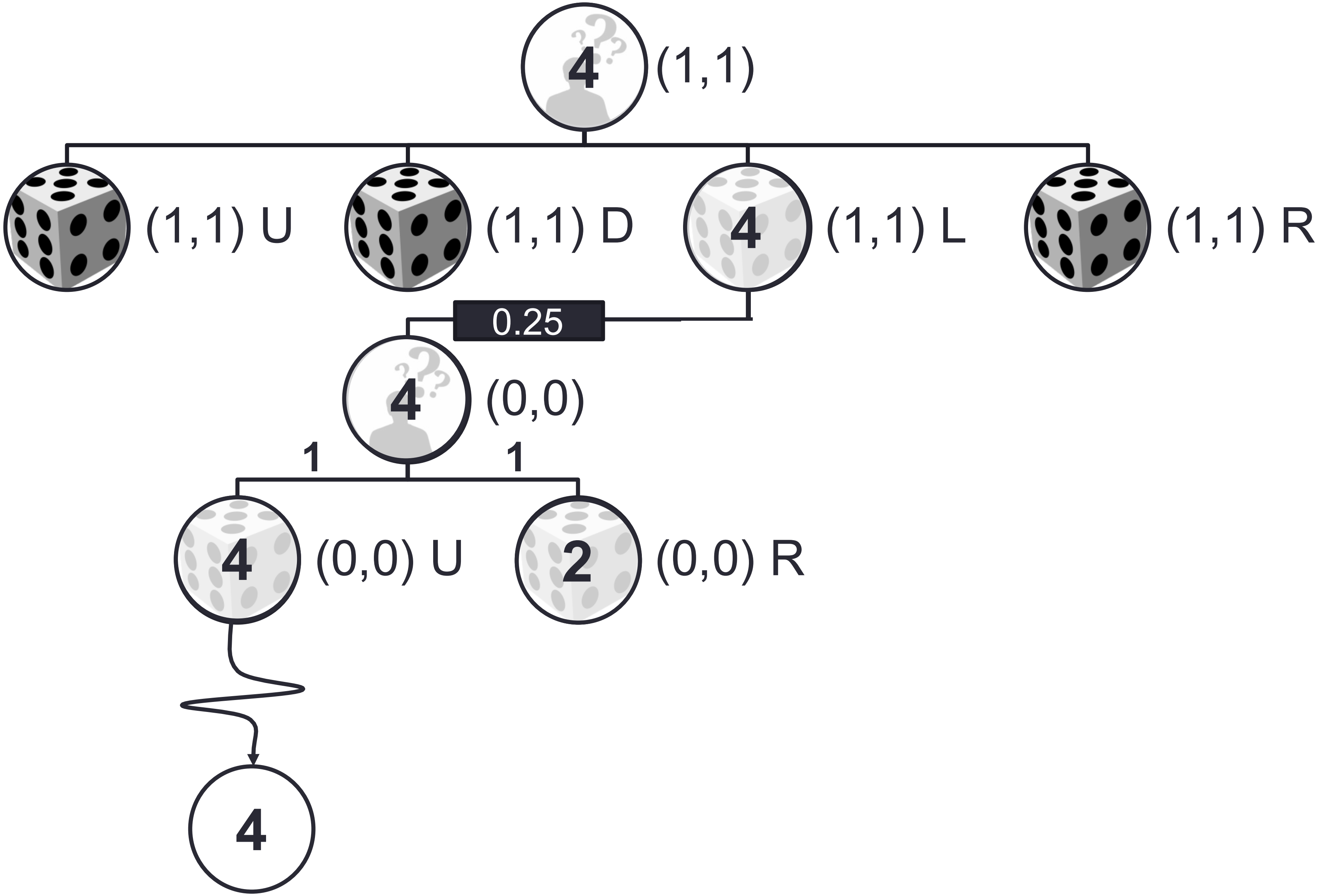
- Uses a different backup function called Max Monte Carlo
- Backup for decision nodes is based on the best child
- This is generally preferable

Monte Carlo	$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \frac{\sum_{n_c \in \mathcal{S}(n_d)} C^k(n_c) \cdot Q^k(n_c)}{C^k(n_d)} & \text{otherwise,} \end{cases}$ $Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} C^k(n_d) \cdot V^k(n_d)}{C^k(n_c)}.$
Max Monte Carlo	$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \max_{n_c \in \mathcal{S}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$ $Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} C^k(n_d) \cdot V^k(n_d)}{C^k(n_c)}.$

Max-UCT



Max-UCT

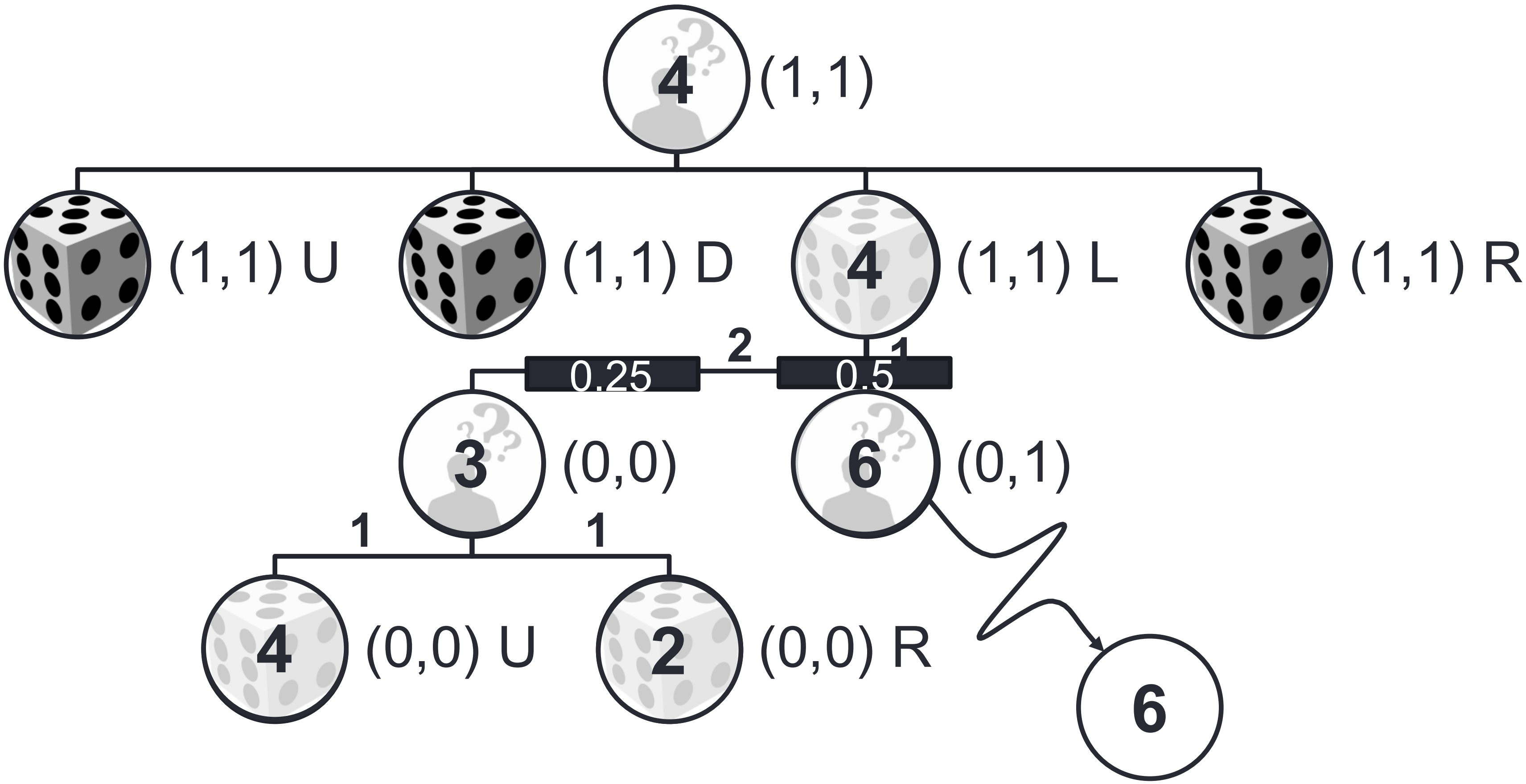


DP-UCT

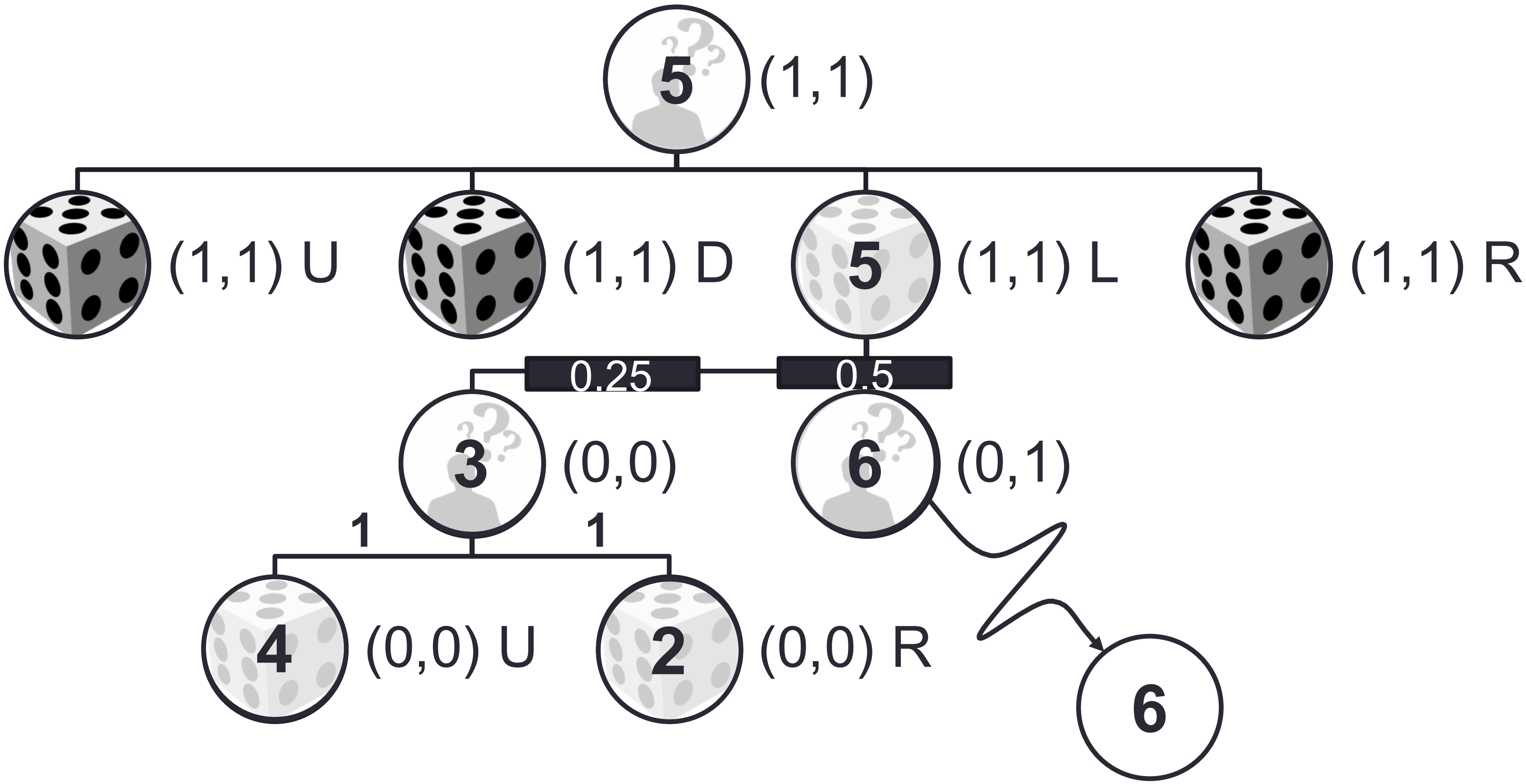
- Modify the backup function for chance nodes
  - Partial Bellman
- A step towards the full Bellman approach used in dynamic programming
  - Without having to explicate every possible outcome

Max Monte Carlo	$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \max_{n_c \in \mathcal{S}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$ $Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} C^k(n_d) \cdot V^k(n_d)}{C^k(n_c)}.$
Partial Bellman	$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d) \text{ is terminal} \\ \max_{n_c \in \mathcal{S}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$ $Q^k(n_c) = R(n_c) + \frac{\sum_{n_d \in \mathcal{S}(n_c)} P(n_d \mid n_c) \cdot V^k(n_d)}{P^k(n_c)},$

DP-UCT



DP-UCT



UCT*

- We don’t want a complete policy, just the next decision
  - Uncertainty grows with distance from the root node
  - Therefore, things closer to the root are more important
- UCT, builds a tree skewed towards more promising parts of the solution space
  - Because of the UCB1 action selection
  - However, it does not consider depth as a deterrent
- DP-UCT + trial length change
  - Trial ends when we expand a new node
  - Encourages more exploration in shallower parts of the tree

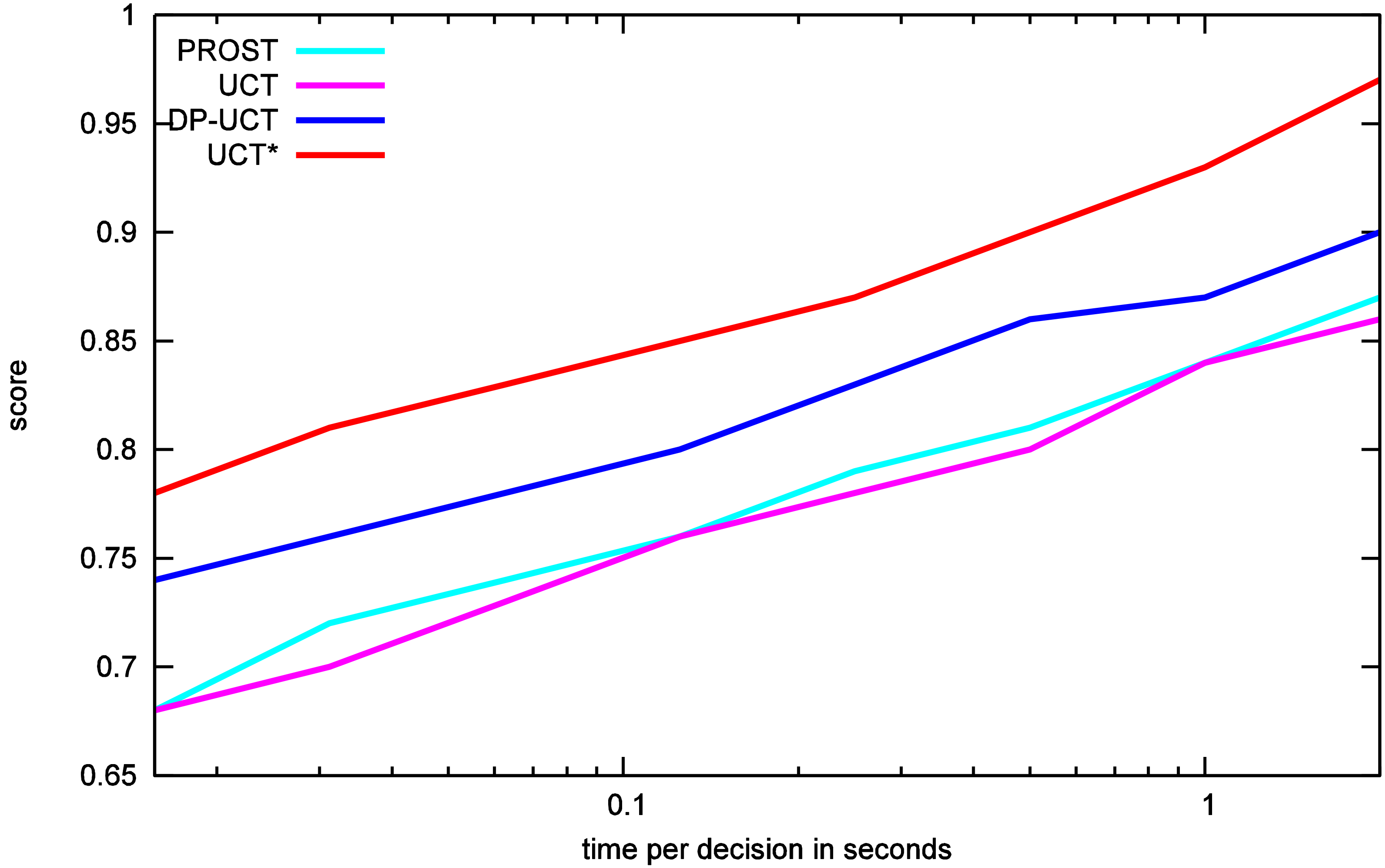
Results

- IPCC 2011 benchmarks: 10 problems per domain
- Results averaged over 100 runs
- Heuristic used: inadmissible, same in all planners
- Compared against Prost (winner of IPCC 2011)

Results - Summary

Domain	UCT	Max-UCT	DP-UCT	UCT*	Prost
ELEVATORS	0.93	**0.97**	**0.97**	**0.97**	0.93
SYSADMIN	0.66	0.71	0.65	**1.00**	0.82
RECON	**0.99**	0.88	0.89	0.88	**0.99**
GAME	0.88	0.90	0.89	**0.98**	0.93
TRAFFIC	0.84	0.86	0.87	**0.99**	0.93
CROSSING	0.85	0.96	0.96	**0.98**	0.82
SKILL	0.93	0.95	**0.98**	0.97	0.97
NAVIGATION	0.81	0.66	**0.98**	0.96	0.55
Total	0.86	0.86	0.9	**0.97**	0.87

Results – Anytime Performance



Graphic borrowed from Thomas Keller's presentation at ICAPS 2013

Conclusion

- A uniform framework to describe this type of algorithm
  - Must specify five elements
- Three novel algorithms:
  - Max-UCT: better handling of highly destructive actions
  - DP-UCT: benefits of dynamic programming without the expense
  - UCT*: encourage more exploration close to the decision at hand