ulm university universität uulm

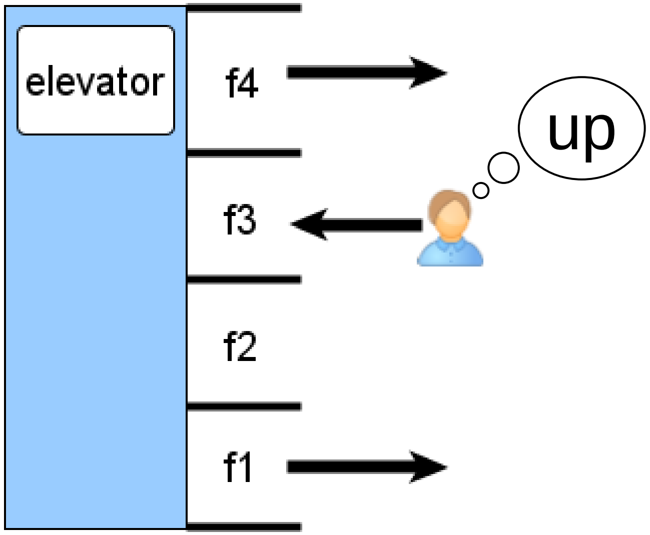
Implementation and evaluation of a hierarchical planning-system for factored POMDPs

Christian Späth | 11.10.2011 | summing-up

Content

Introduction

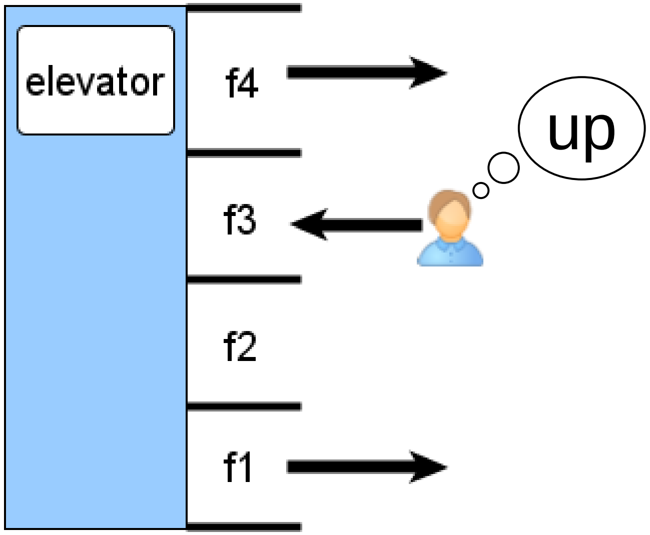
Planning under uncertainty



Introduction

Planning under uncertainty

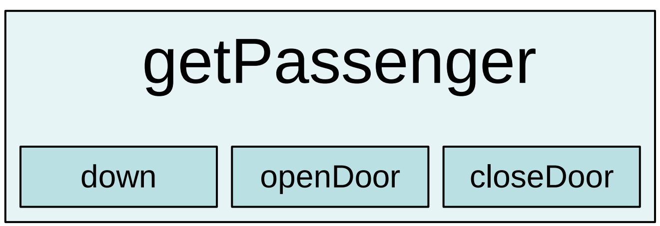
- Large domains
- Hierarchical domain structure
- Several solutions for one problem

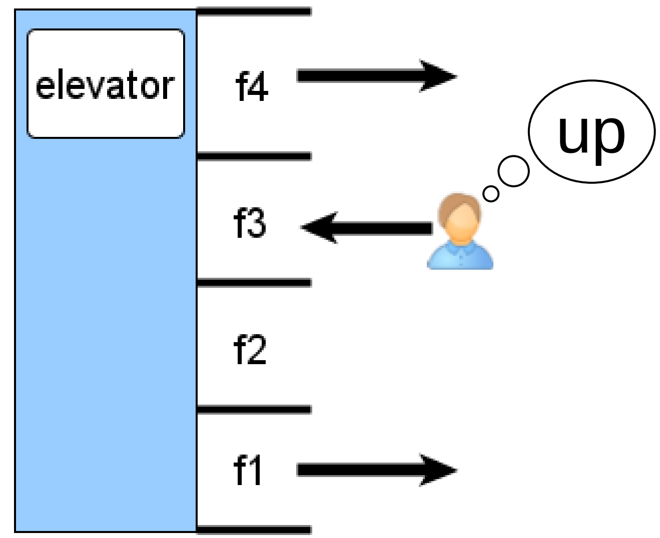


Introduction

Planning under uncertainty

- Large domains
- Hierarchical domain structure
- Several solutions for one problem

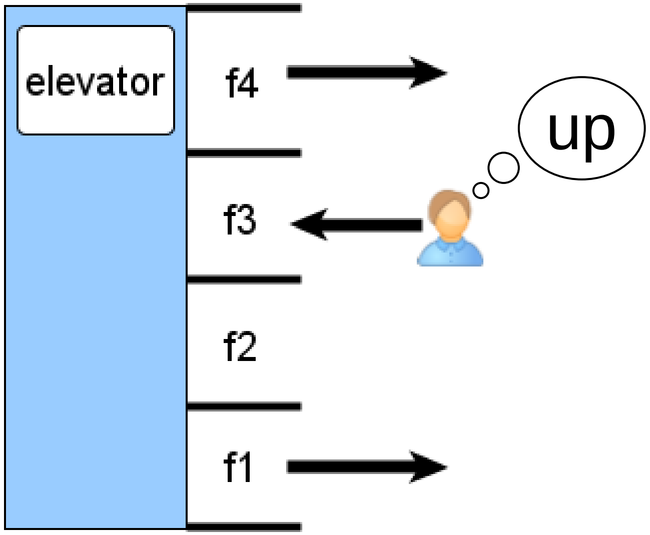




Introduction

Planning under uncertainty

- Large domains
- Hierarchical domain structure
- Several solutions for one problem

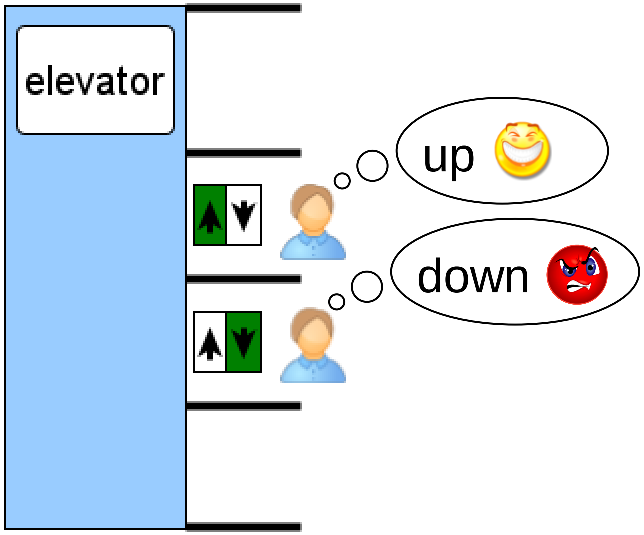


Introduction

Planning under uncertainty

- Large domains
- Hierarchical domain structure
- Several solutions for one problem

- Aspects can be merely partially observable
→ uncertain information about the state



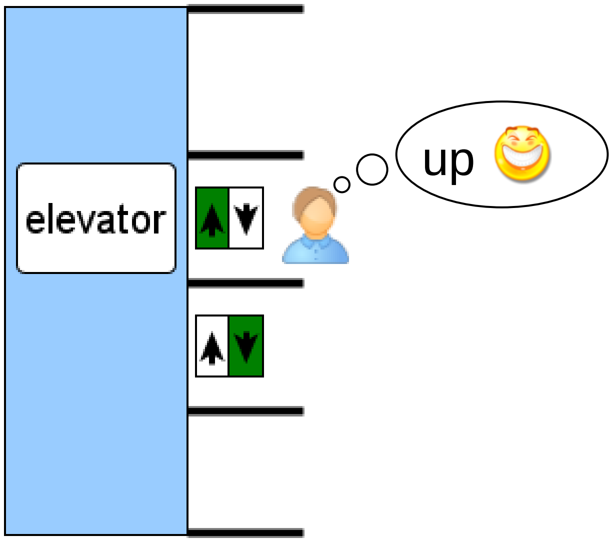
Introduction

Planning under uncertainty

- Large domains
- Hierarchical domain structure
- Several solutions for one problem

- Aspects can be merely partially observable

→ uncertain information about the state



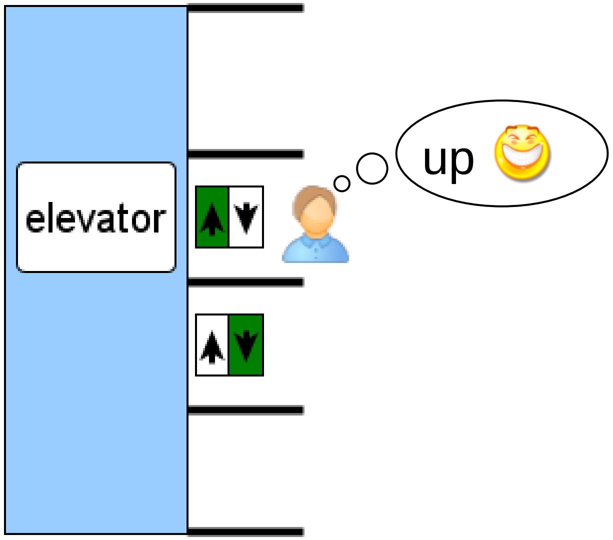
Introduction

Planning under uncertainty

- Large domains
- Hierarchical domain structure
- Several solutions for one problem

- Aspects can be merely partially observable
→ uncertain information about the state

- Actions do have probabilistic effects
→ uncertain information about the progress



Motivation

Common way of modeling: POMDP

- BUT: in praxis rarely used for planning
  → manually designed solutions instead (Expert knowledge)

- solution finding is expensive (PSPACE-complete / undecidable)

Idea: optimize solution-finding by exploiting expert-knowledge
using HTN-methods and FSC

→ Hierarchical factored POMDPs

System model using POMDPs

Partial Observable Markov Decision Process:

- Partially observable
- Probabilistic actions/observations

POMDP = (S, A, T, R, O, Z, h, γ)

System model using POMDPs

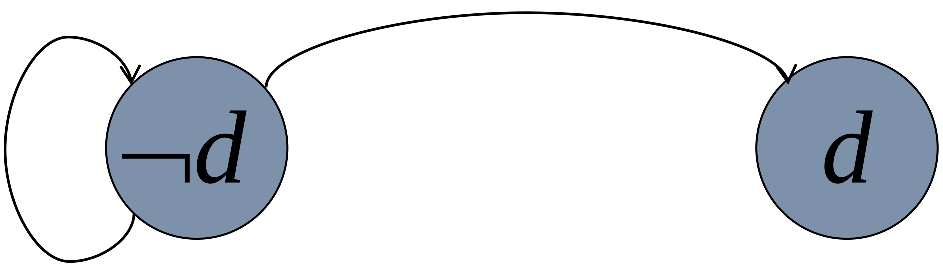
Partial Observable Markov Decision Process:

- Partially observable
- Probabilistic actions/observations

POMDP = (**States**, **Actions**, **Transition function**, R, O, Z, h)

S = {$\neg d$, d } , d = door is open
A = {od} , o = open door

T = P(s‘ | a, s)
= {P($\neg d$ | od, d) = 0.05, …}



System model using POMDPs

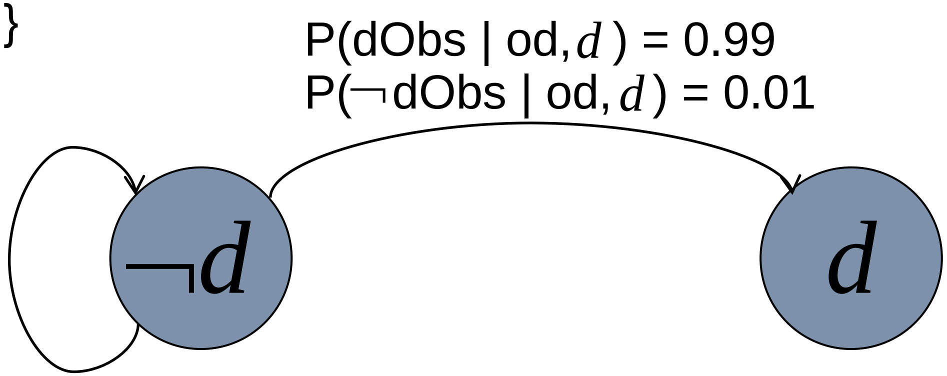
Partial Observable Markov Decision Process:

- Partially observable
- Probabilistic actions/observations

POMDP = (S, A, T, R, **Observations**,**Observation function**, h)

O = {dObs}   , dObs = door open observation

Z = P(o^i | a, s)
   = {P(dObs | od, d) = 0.99, …}



System model using POMDPs

Partial Observable Markov Decision Process:

- Partially observable
- Probabilistic actions/observations

POMDP = (S, A, T, **Reward function**, O, Z, **horizon**)

Reward function: R(s,a)
Bsp: R($\neg d$, od) = -1
R(d, od) = -10

Horizon h: max. number of actions

Policy model using FSCs

Policy representation using Finite State Controller (FSC)

- Generalization of action sequences
- Compact representation using observation formulas

FSC = (Q, q0, α, δ)

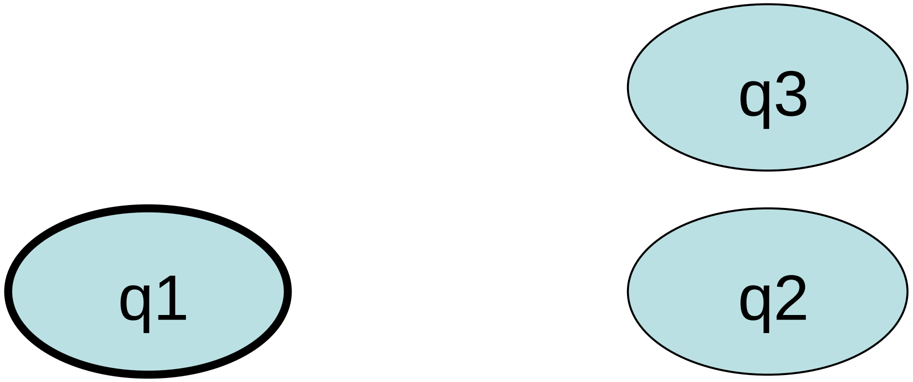
Policy model using FSCs

Policy representation using Finite State Controller (FSC)

- Generalization of action sequences
- Compact representation using observation formulas

FSC = (**Q, q0**, α, δ)

Controller nodes Q = {q1,q2,q3}
Start node q_0 = q1



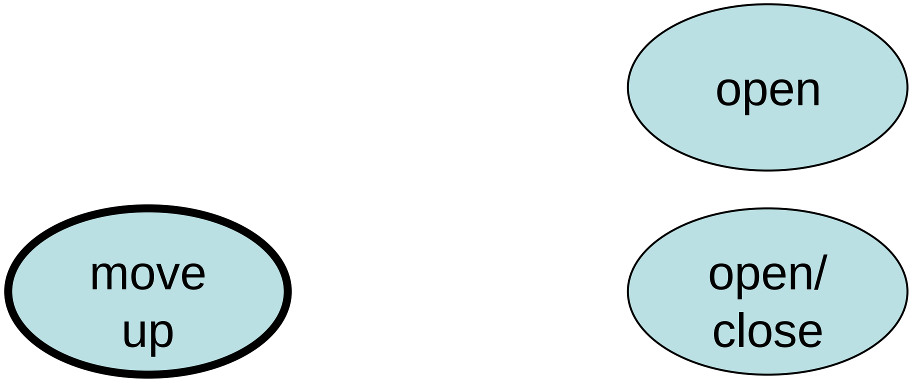
Policy model using FSCs

Policy representation using Finite State Controller (FSC)

- Generalization of action sequences
- Compact representation using observation formulas

FSC = (Q, q0, **α**, δ)

Action association function a = {q1 → move up, ...}



Policy model using FSCs

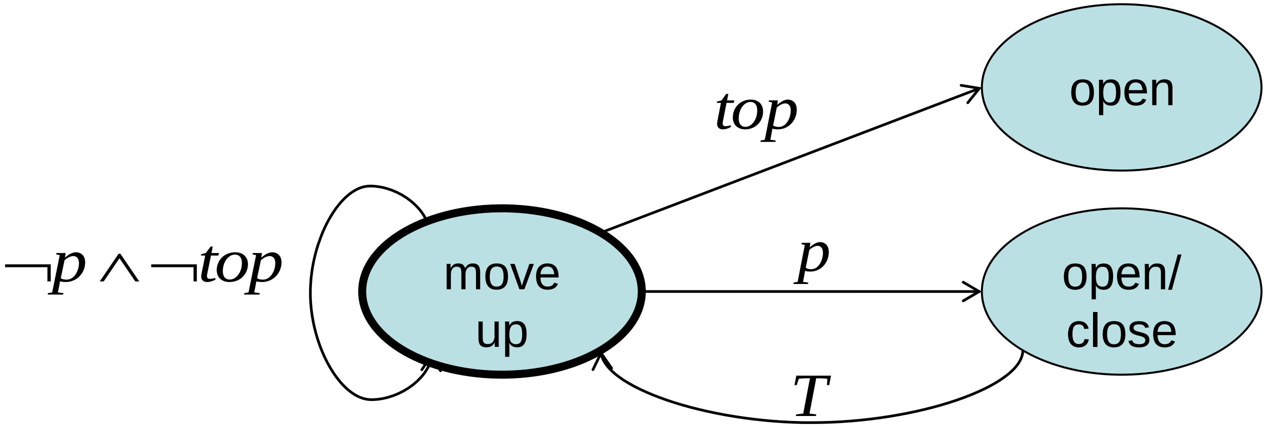
Policy representation using Finite State Controller (FSC)

- Generalization of action sequences
- Compact representation using observation formulas

FSC = (Q, q0, α, **δ**)

Action association function a = {q1 → move up, ...}

Transition function δ = {(q1,q2) → (p = personWaitingObs), ... }

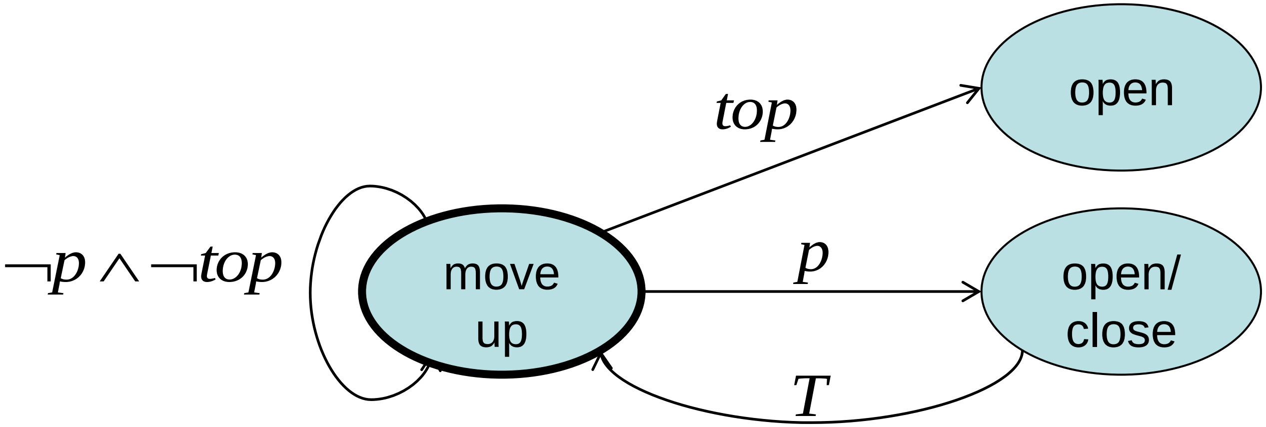


Policy model using FSCs

Policy representation using Finite State Controller

Execution of a FSC:

- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula

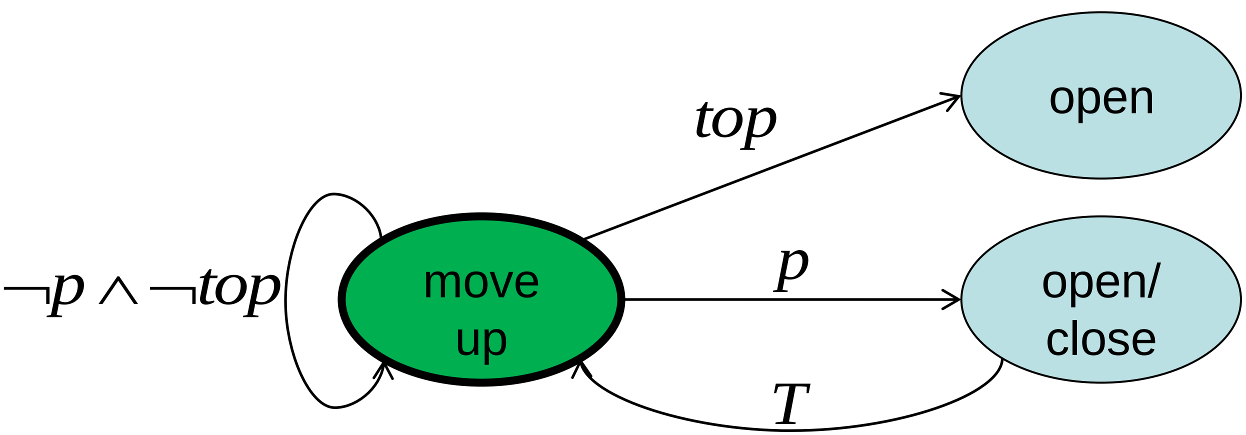


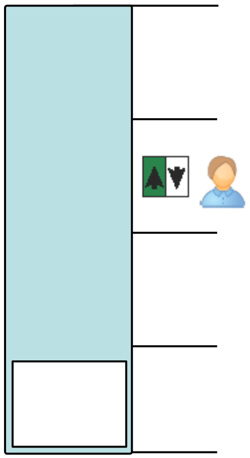
Policy model using FSCs

Policy representation using Finite State Controller

Execution of a FSC:

- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula



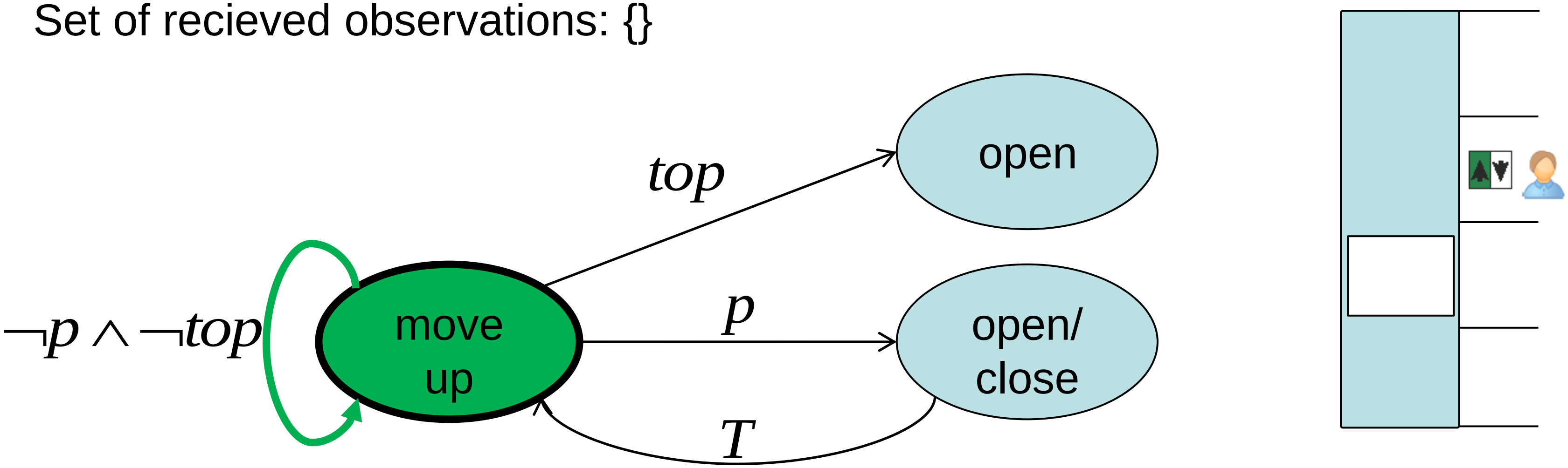


Policy model using FSCs

Policy representation using Finite State Controller

Execution of a FSC:
- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula

Set of recieved observations: {}



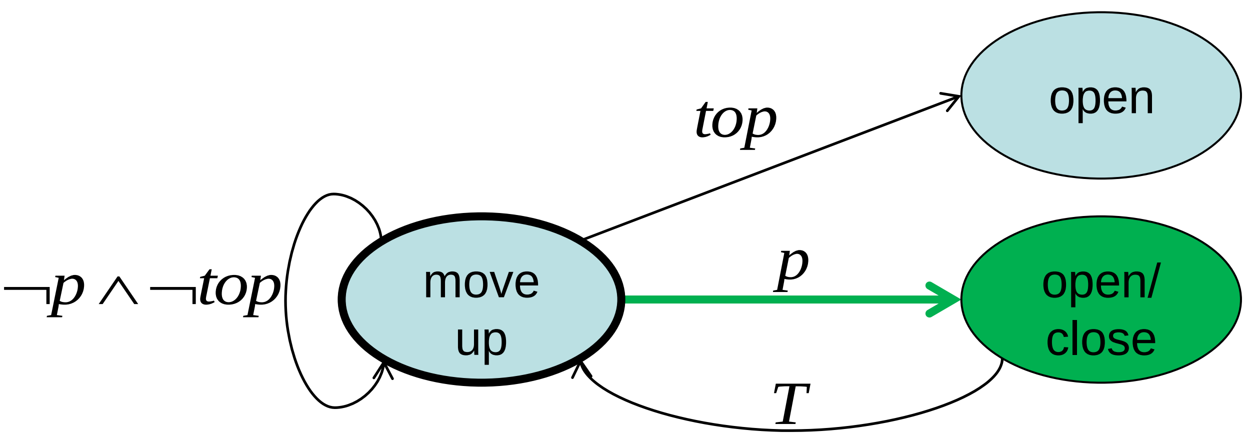
Policy model using FSCs

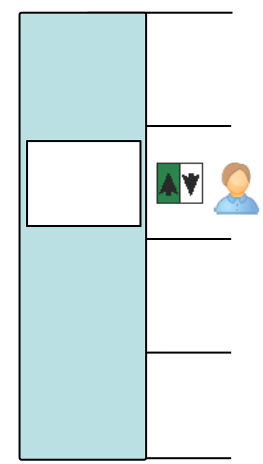
Policy representation using Finite State Controller

Execution of a FSC:

- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula

Set of recieved observations: {p}

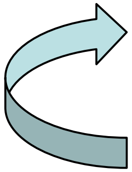




Policy model using FSCs

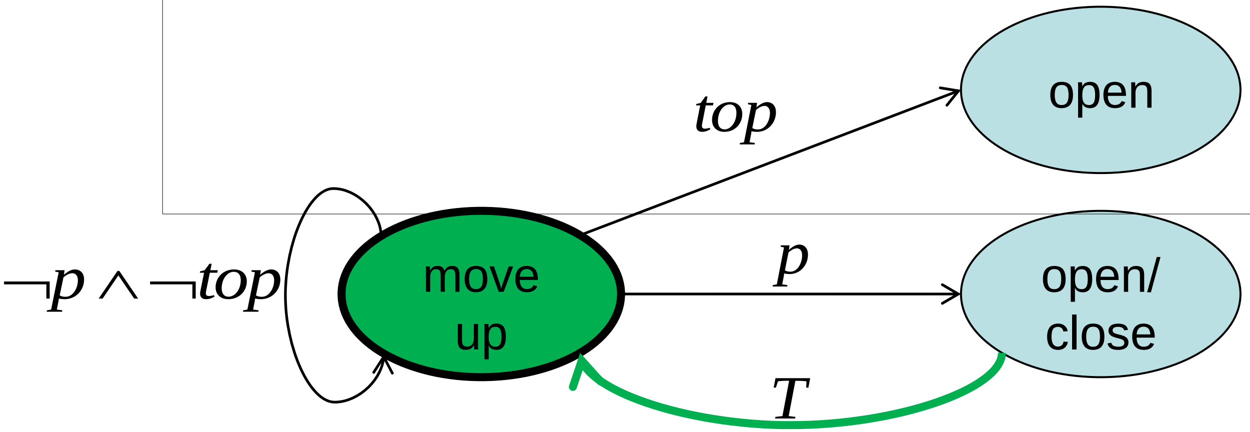
Policy representation using Finite State Controller

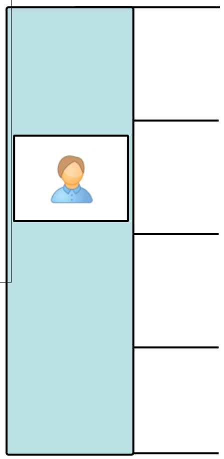
Execution of a FSC:



- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula

Set of recieved observations: {}





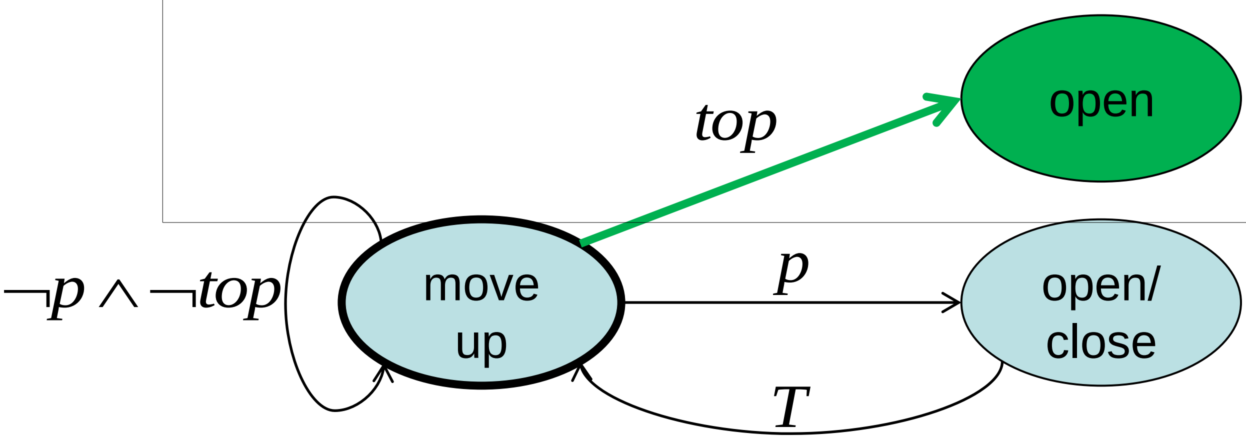
Policy model using FSCs

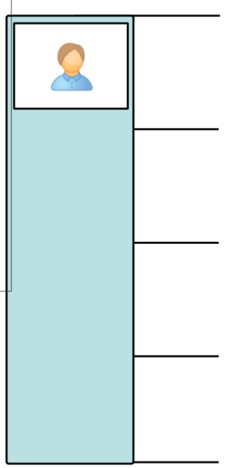
Policy representation using <u>F</u>inite <u>S</u>tate <u>C</u>ontroller

Execution of a FSC:

- Execute action of current node
- Receive a set of observations depending on prev. action
- Advance to next node according to observation formula

Set of recieved observations: {top}





Policy model using FSCs

What is the quality of policy π ?

Value of executing a policy: $V(\pi) = \sum_{t=0}^{T} R(s_t, a_t)$

BUT: domain is stochastic → $V(\pi)$ is also stochastic

Solution: $V(\pi)$ defined as expected execution value

HTN-style planning

Hierarchical Task Network Planning

- Exploitation of expert knowledge

HTN-style planning

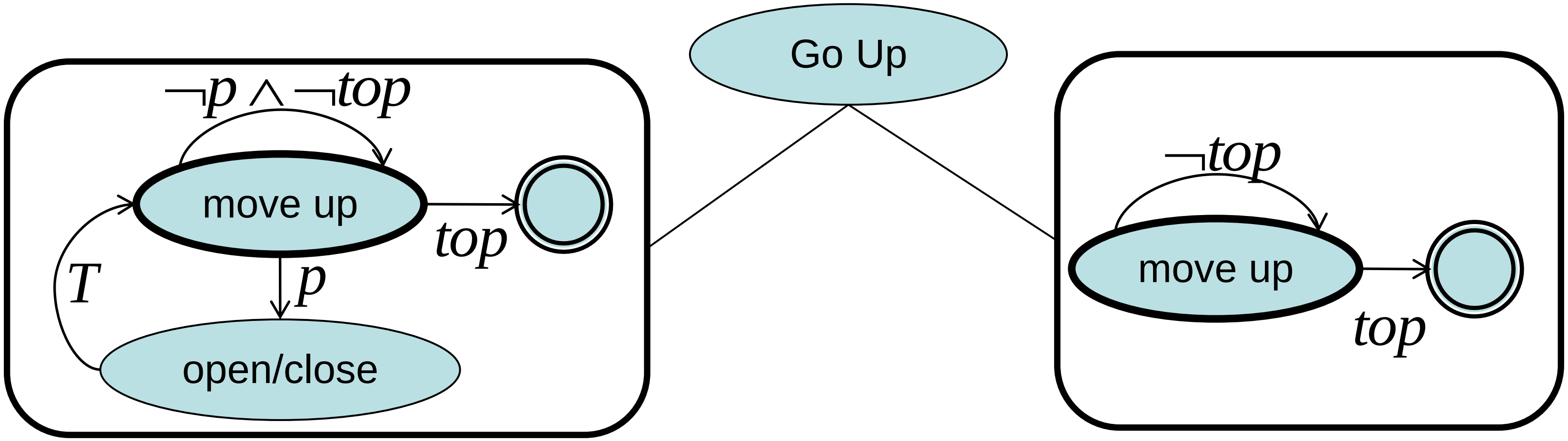
Hierarchical Task Network Planning

- Exploitation of expert knowledge
- Method m = (A_a^i, partial plan FSC)
- decomposition: replace A_a^i with implementation

HTN-style planning

Hierarchical Task Network Planning

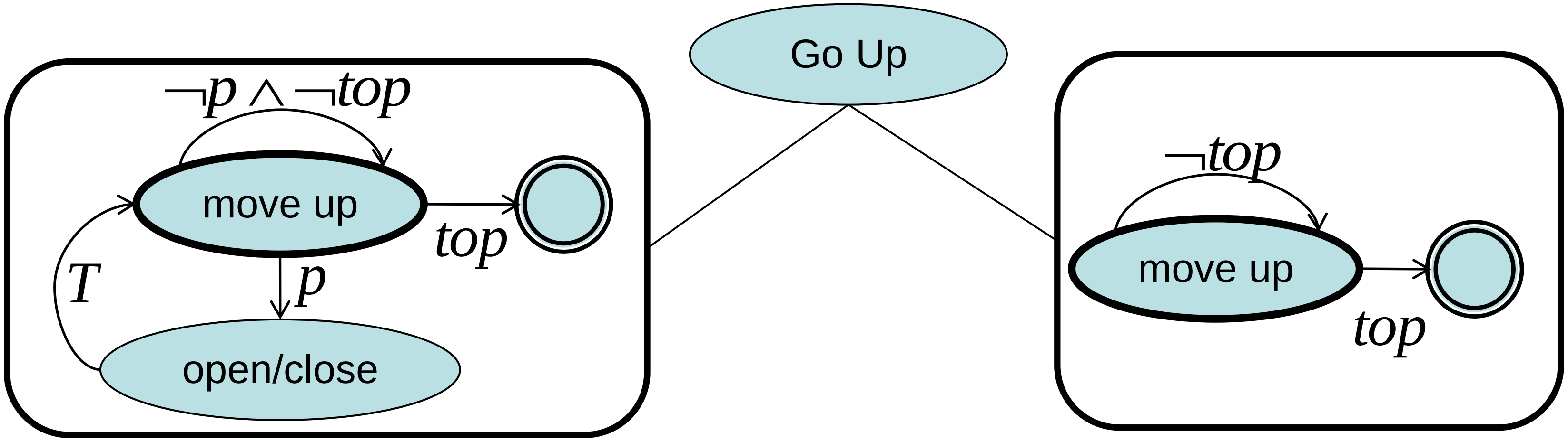
- Exploitation of expert knowledge
- Method m = (A_a^i, partial plan FSC)
- decomposition: replace A_a^i with implementation



HTN-style planning

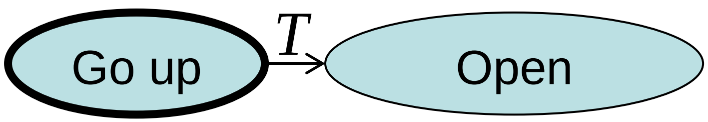
Hierarchical Task Network Planning

- Exploitation of expert knowledge
- Method m = (A_a^i, partial plan FSC)
- decomposition: replace A_a^i with implementation
- goal: decompose initial plan until it is primitive

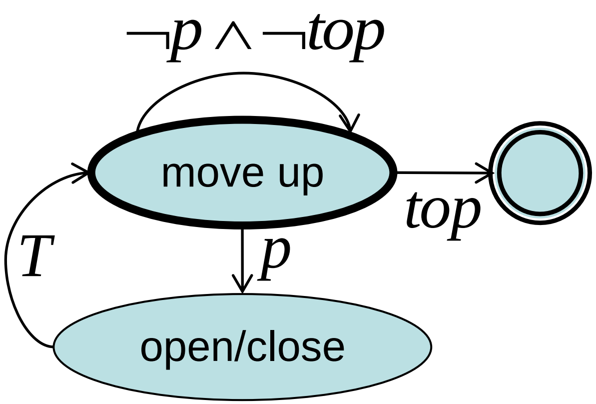


Hierarchical POMDPs

Initial plan:

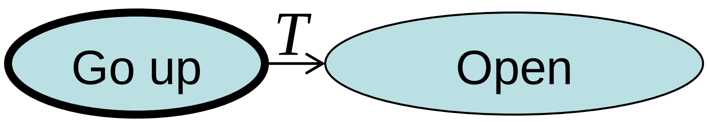


Implementation for Go Up:

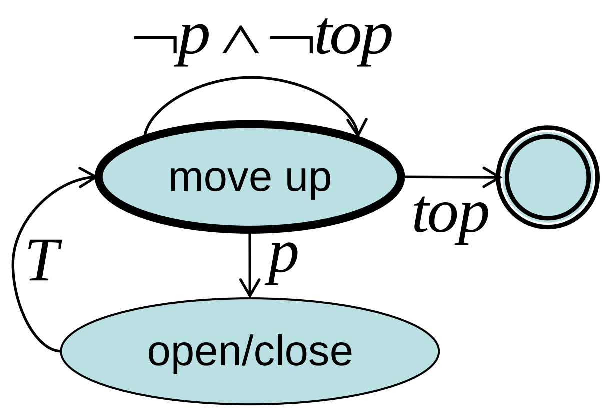


Hierarchical POMDPs

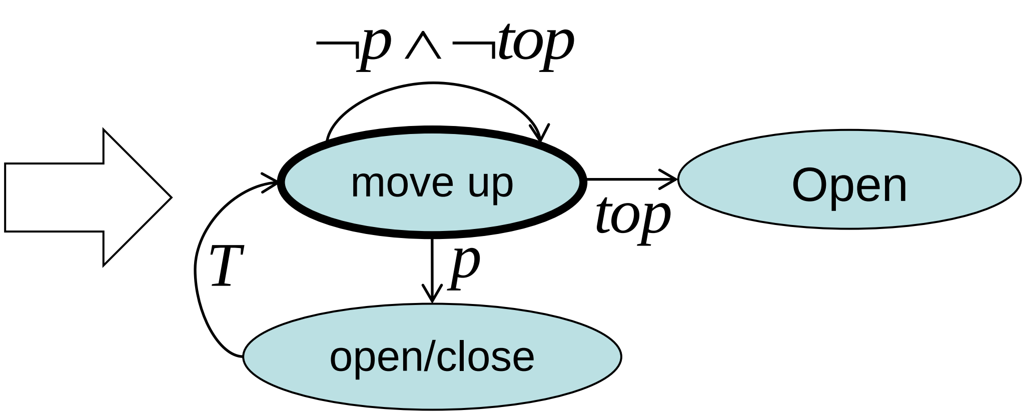
Initial plan:



Implementation for Go Up:



decomposed plan:



Algorithms

Adaption of two known search-algorithms to HPOMDP:

→ Policy representation using FSCs
→ Policy modification by applying methods
→ search in policy space

Algorithms

Adaption of two known search-algorithms to HPOMDP:

→ Policy representation using FSCs
→ Policy modification by applying methods
→ search in policy space

A* : an optimal efficient algorithm

- optimal solution with respect to a given hierarchy
- cost function and heuristic estimate for FSC

Algorithms

Adaption of two known search-algorithms to HPOMDP:

→ Policy representation using FSCs
→ Policy modification by applying methods
→ search in policy space

A* : an optimal efficient algorithm

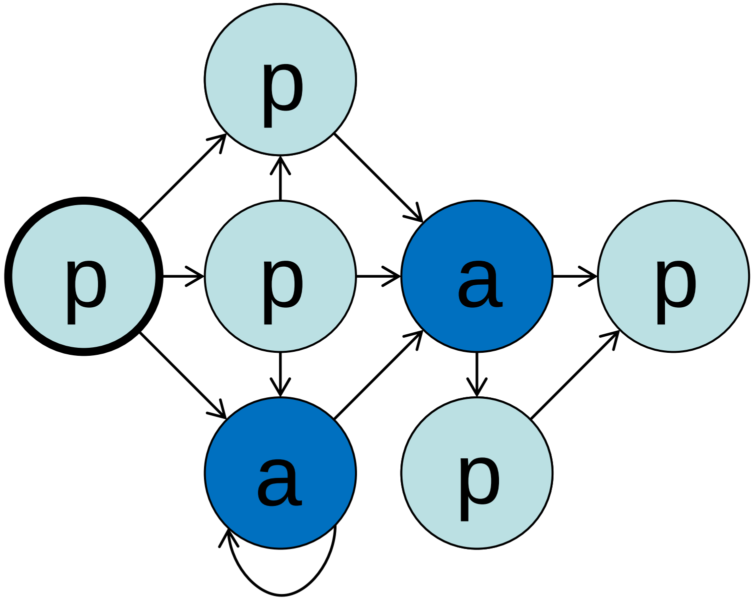
- optimal solution with respect to a given hierarchy
- cost function and heuristic estimate for FSC

UCT: based on monte-carlo tree search

- probabilistic approach
- approximative optimal solution with respect to a given hierarchy
- anytime property

A* - Algorithm

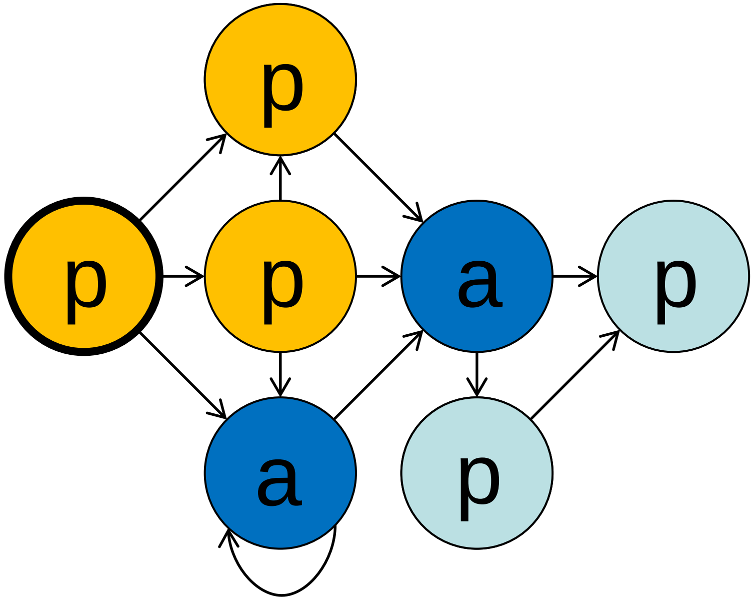
Evaluate every policy π : $f(\pi) = g(\pi) + h(\pi)$



A* - Algorithm

Evaluate every policy π: $f(\pi) = g(\pi) + h(\pi)$

cost function $g(\pi)$: guaranteed expected costs for all decompositions

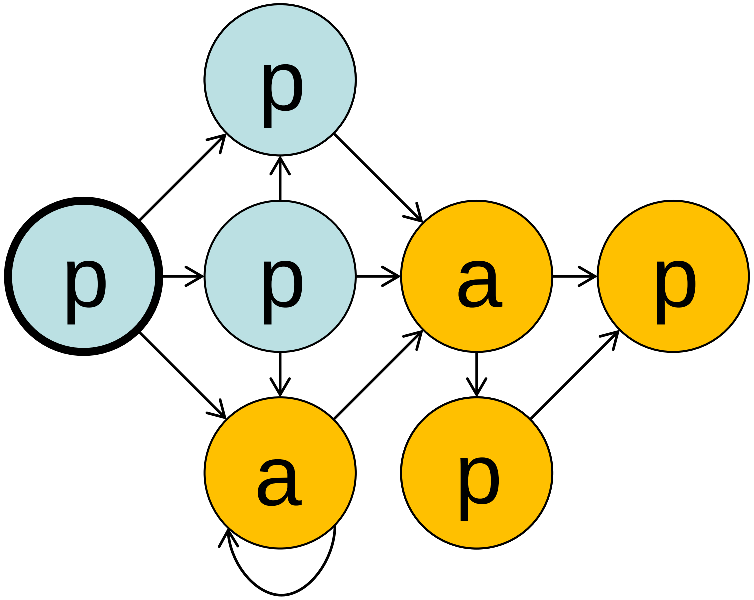


A* - Algorithm

Evaluate every policy π: $f(\pi) = g(\pi) + h(\pi)$

cost function $g(\pi)$: guaranteed expected costs for all decompositions

heuristic estimate $h(\pi)$: minimal costs of the (partially) abstract part



UCT - Algorithm

Upper Confidence Tree – Algorithm (UCT)

Idea: calculate a approximate optimal policy π^+ by interacting with a domain simulator

UCT - Algorithm

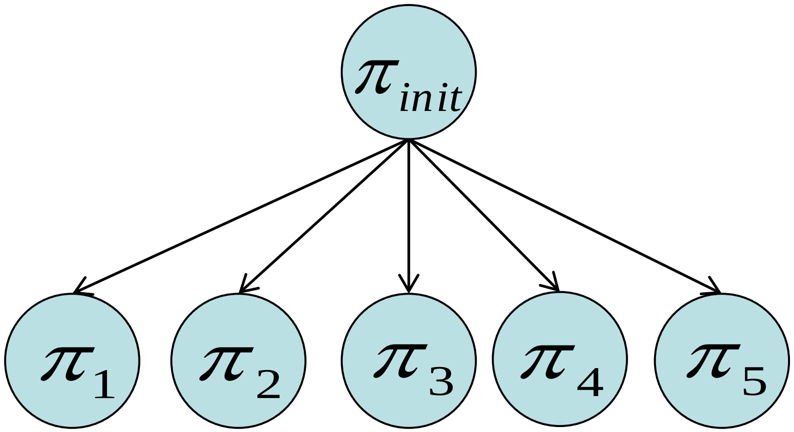
Upper Confidence Tree – Algorithm (UCT)

Idea: calculate a approximate optimal policy π^+ by interacting with a domain simulator

Simulator can simulate the execution of a primitive policy π
→ sampled simulation value $V_{sim}(\pi)$

Algorithms - UCT

Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

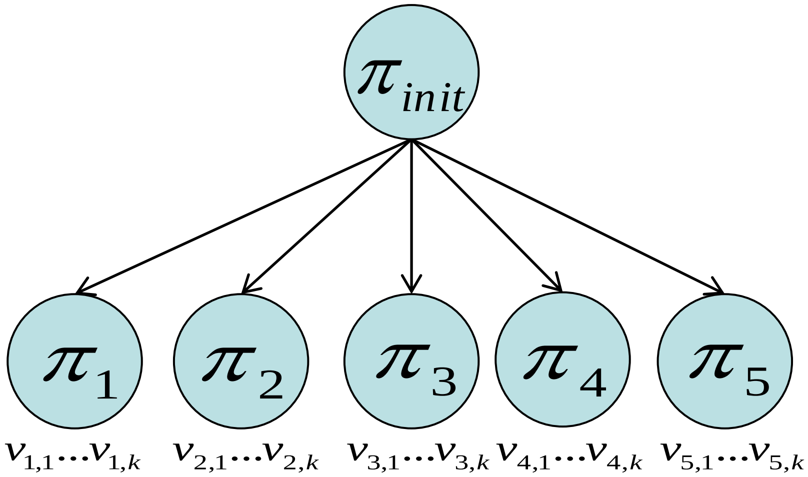


Algorithms - UCT

Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

Trivial approach:

- Simulate every policy k times



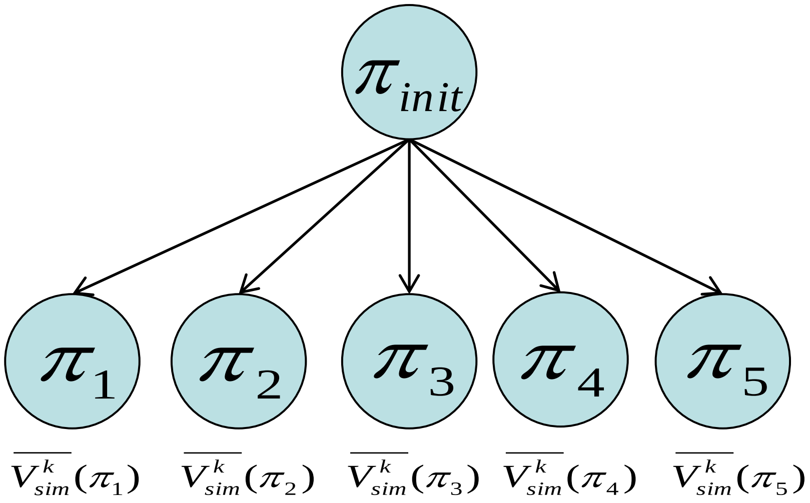
Algorithms - UCT

Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

Trivial approach:

- Simulate every policy k times
- Return the policy with the highest average value

$$\overline{V_{sim}^{k}}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j$$



Algorithms - UCT

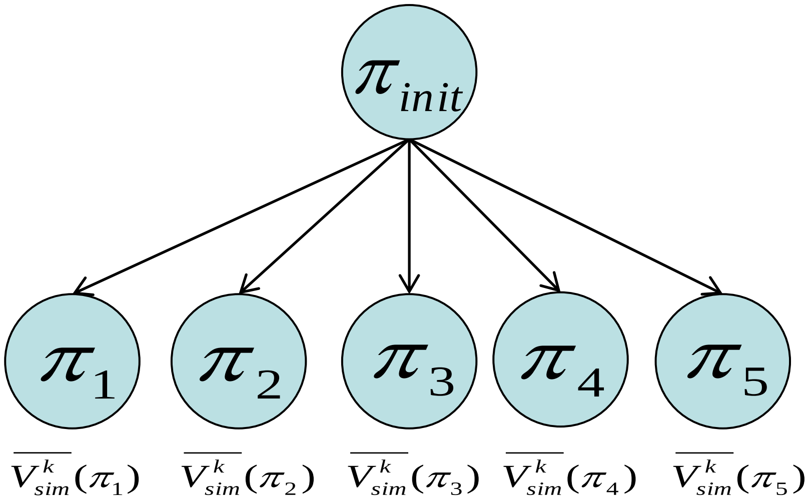
Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

Trivial approach:

- Simulate every policy k times
- Return the policy with the highest average value

$$\overline{V_{sim}^{k}}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j$$

- Additive Chernoff Bound:



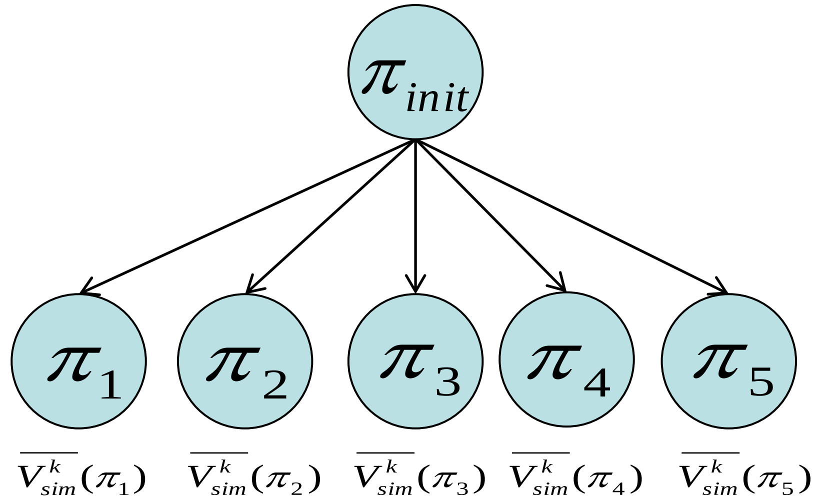
Algorithms - UCT

Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

Trivial approach:

- Simulate every policy k times
- Return the policy with the highest average value

$$\overline{V_{sim}^{k}}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j$$



- Additive Chernoff Bound:

$$P\left(| V(\pi) - \overline{V_{sim}^{k}}(\pi) | \geq \varepsilon \right)$$

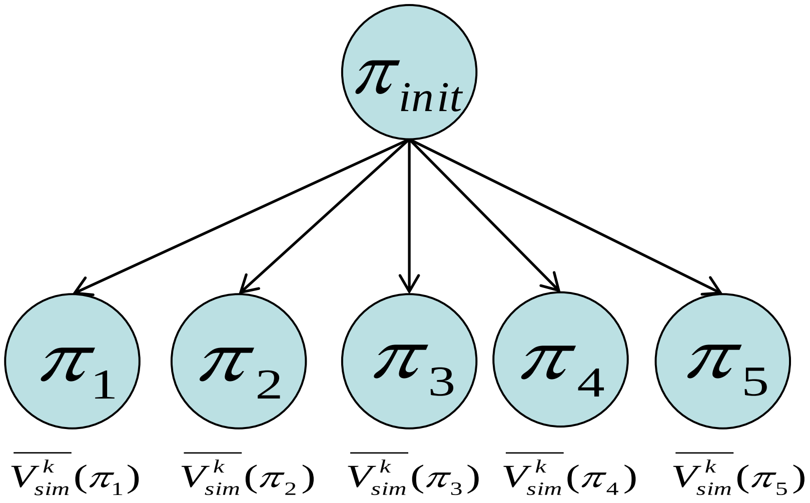
Algorithms - UCT

Simplification: find the best primitve policy out of a given set of primitve policies by using the simulator

Trivial approach:

- Simulate every policy k times
- Return the policy with the highest average value

$$\overline{V_{sim}^{k}}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j$$



- Additive Chernoff Bound:

$$P\left(| V(\pi) - \overline{V_{sim}^{k}}(\pi) | \geq \varepsilon \right) \leq \exp\left(-\varepsilon^2 \cdot k \right)$$

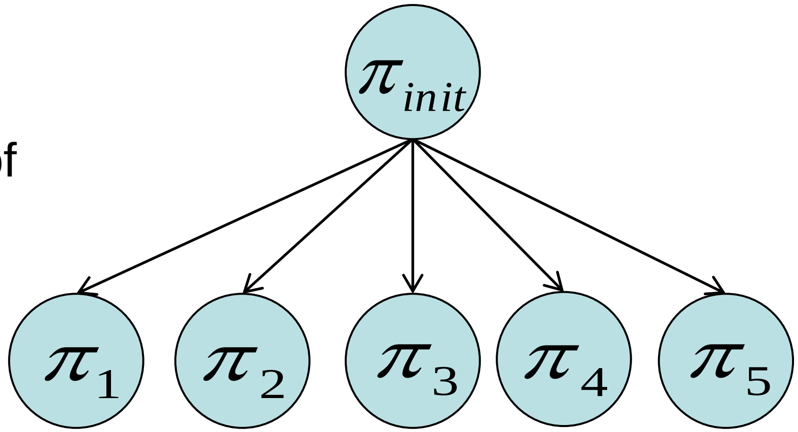
Algorithms - UCT

Simplification: find the best primitve plan out of a given set of primitve policies by using the simulator

Exploitation of previous simulation results:

- Tendency after few simulations

→ Avoid unnecessary simulations of bad policies
→ Less simulation then $5 \cdot k$ for same accuracy
→ Increase simulation count for good policies

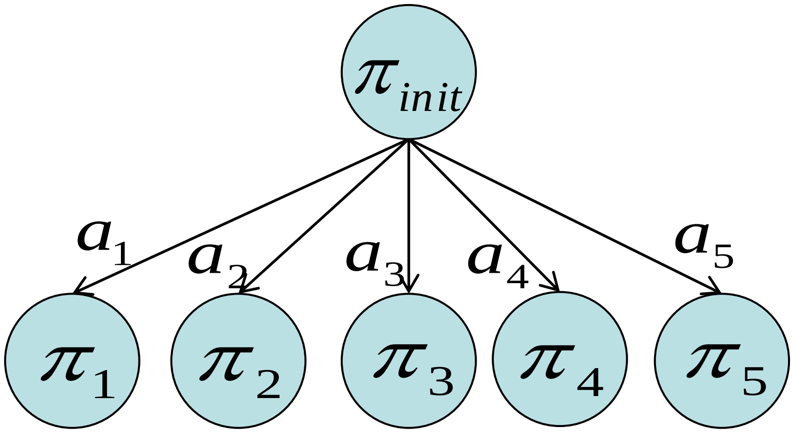


Algorithms - UCT

Simplification: find the best primitve plan out of a given set of primitve policies by using the simulator

Exploitation of pervious simulation results:

- Exploitation: use previous values
- Exploration: minimal simulation count for every policy

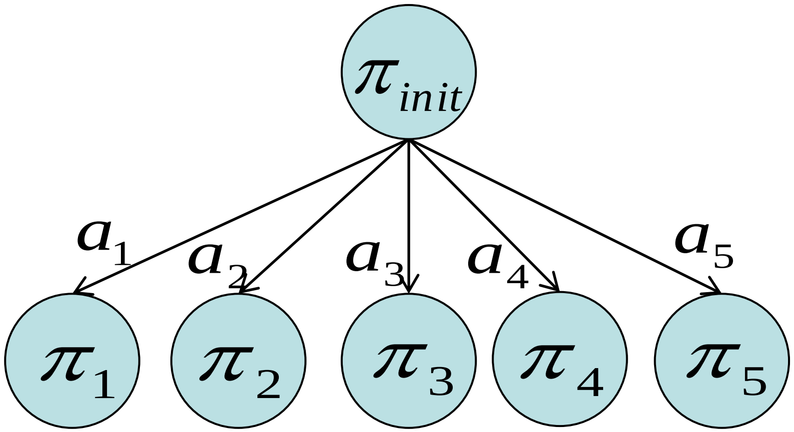


Algorithms - UCT

Simplification: find the best primitve plan out of a given set of primitve policies by using the simulator

Exploitation of pervious simulation results:

$$a^* = \max_{a \in A} Q(a) + \sqrt{\frac{2 \cdot \ln(n)}{n(a)}}$$



$Q(a_i)$: Average simulation value for the policy π_i

n : total simulation count

$n(a_i)$: simulation count for policy π_i

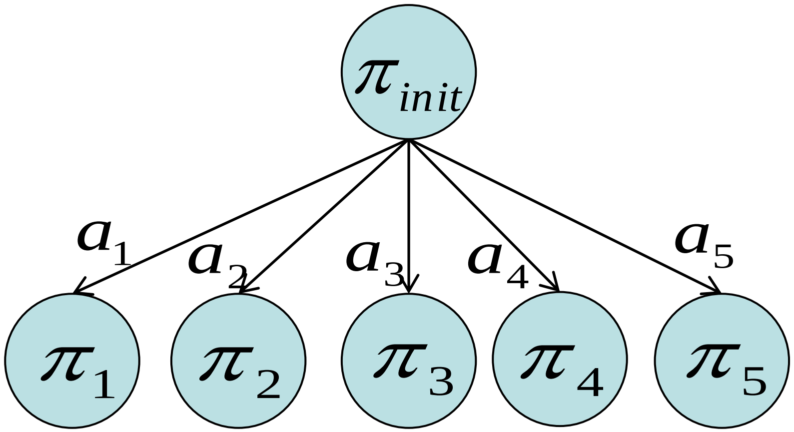
Algorithms - UCT

Simplification: find the best primitve plan out of a given set of primitve policies by using the simulator

Exploitation of pervious simulation results:

$$a^* = \max_{a \in A} Q(a) + \sqrt{\frac{2 \cdot \ln(n)}{n(a)}}$$

Exploitation $\leftrightarrow$ Exploration



$Q(a_i)$: Average simulation value for the policy π_i

n : total simulation count

$n(a_i)$: simulation count for policy π_i

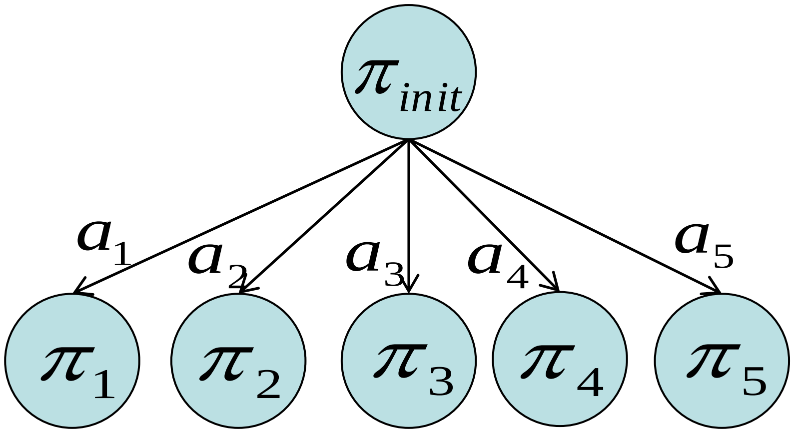
Algorithms - UCT

Simplification: find the best primitve plan out of a given set of primitve policies by using the simulator

Exploitation of pervious simulation results:

$$a^* = \max_{a \in A} Q(a) + \sqrt{\frac{2 \cdot \ln(n)}{n(a)}}$$

Exploitation $\leftrightarrow$ Exploration



$Q(a_i)$: Average simulation value for the policy π_i

n : total simulation count

$n(a_i)$: simulation count for policy π_i

→ Transfer to a larger hierarchy possible

Implementation

HPOMDP-Planner

- datastructure (FSC, Formula, …)
- A*-Searchalgorithm
  - cost function
  - heuristic estimate
- UCT-Searchalgorithm

Integration of RDDLSim for evaluation purpose

Demo: 1) solution of the 6 floors elevator instance, using the UCT-algorithm with 100 seconds

2) simulate execution via RDDLSim

Evaluation

„Is it possible to optimize the solution-finding for partially observable, stochastic domains, by exploiting expert knowledge with HPOMDPs?“

Evaluation

Evaluation domains:

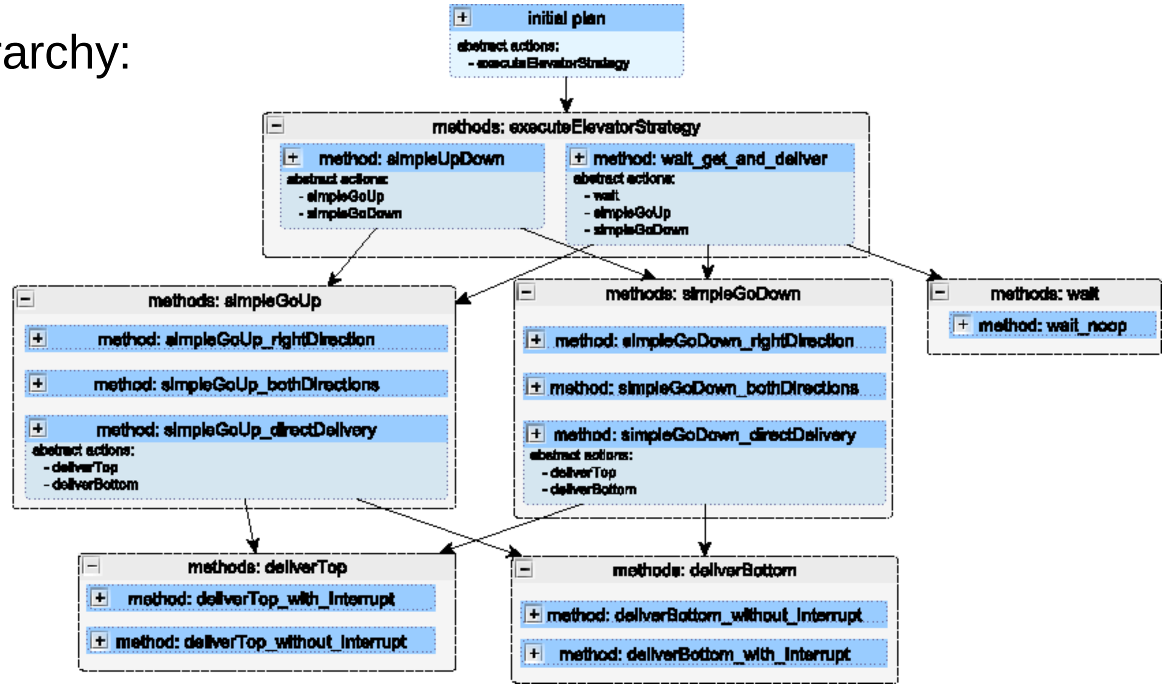
- Five stochastic and partially observable evaluation domains
- Hierarchy for the domains
- Several instances per domain, e.g. 3-6 floors for the elevator domain

Evaluation

Evaluation domains:

- Five stochastic and partially observable evaluation domains
- Hierarchy for the domains
- Several instances per domain, e.g. 3-6 floors for the elevator domain

Elevator hierarchy:



Evaluation

References:
- External non-hierarchical planner Symbolic Perseus (participant of the IPPC 2011)
- NoOp- and Random-Policy

Neutral evaluation platform:
- RDDLSim, the official competition platform of the IPPC 2011

Evaluation setting:
- fixed calculation time (2h) and memory (4GB)

Evaluation criteria:
- calculation time
- RDDLSim value

Evaluation

Evaluation results of the elevator domain:

Instance	NoOp/Random	A*	UCT[10s]	SPerseus	
3 floors	-44.4 / -52.3 n/a	timeout n/a	-32.6 10.9	-35.4 651.9	value time
4 floors	-89.0/ -100.6 n/a	timeout n/a	-74.3 12.2	timeout n/a	
5 floors	-133.8/ -147.7 n/a	timeout n/a	-113.2 14.5	timeout n/a	
6 floors	-177.9/ -193.0 n/a	timeout n/a	-148.8 23.9	timeout n/a	

Evaluation

Evaluation results of the elevator domain:

Instance	NoOp/Random	A*	UCT[10s]	SPerseus
3 floors	-44.4 / -52.3 -	timeout n/a	-32.6 10.9	-35.4 651.9
4 floors	-89.0/ -100.6 -	timeout n/a	-74.3 12.2	timeout n/a
5 floors	-133.8/ -147.7 -	timeout n/a	-113.2 14.5	timeout n/a
6 floors	-177.9/ -193.0 -	timeout n/a	-148.8 23.9	timeout n/a

→ A* unexpected inefficient

Evaluation

Evaluation results of the elevator domain:

Instance	NoOp/Random	A*	UCT[10s]	SPerseus
3 floors	-44.4 / -52.3 -	timeout n/a	-32.6 10.9	-35.4 651.9
4 floors	-89.0/ -100.6 -	timeout n/a	-74.3 12.2	timeout n/a
5 floors	-133.8/ -147.7 -	timeout n/a	-113.2 14.5	timeout n/a
6 floors	-177.9/ -193.0 -	timeout n/a	-148.8 23.9	timeout n/a

→ A* unexpected inefficient
→ Quality is very hierarchy dependant

Evaluation

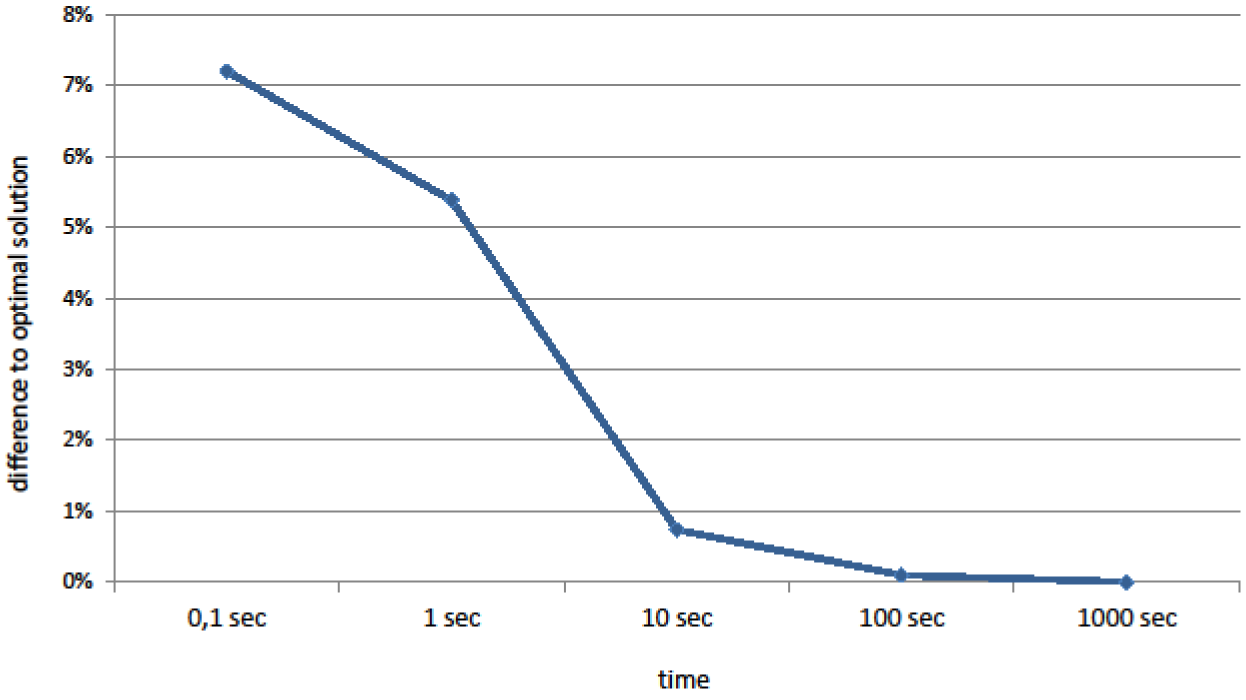
Evaluation results of the elevator domain:

Instance	NoOp/Random	A*	UCT[10s]	SPerseus
3 floors	-44.4 / -52.3 -	timeout n/a	-32.6 10.9	-35.4 651.9
4 floors	-89.0/ -100.6 -	timeout n/a	-74.3 12.2	timeout n/a
5 floors	-133.8/ -147.7 -	timeout n/a	-113.2 14.5	timeout n/a
6 floors	-177.9/ -193.0 -	timeout n/a	-148.8 23.9	timeout n/a

→ A* unexpected inefficient
→ Quality is very hierarchy dependant
→ UCT scales by far best

Evaluation

UCT convergence:



→ After 10 seconds less then 1% difference to optimal solution of the given hierarchy

Summary

- Challenges of partial observable, stochastic domains
  → Motivation for HPOMDPs
- HPOMDP
  - POMDP as system representation
  - FSC as policy and implementation representation
  - decomposition
- Search algorithms
  - A*
  - UCT
- Evaluation
  → HPOMDPs with UCT significantly optimize the solution-finding for partial observable, stochastic domains.