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

– Introduction and motivation

– Hierarchical POMDPs
  • POMDP
  • FSC

– Algorithms
  • A*
  • UCT

– Implementation

– Evaluation

– Summary
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

getPassenger

down  openDoor  closeDoor
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

- 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)

\[
S = \{-d, d\}, \quad d = \text{door is open}
\]

\[
A = \{\text{od}\}, \quad o = \text{open door}
\]

\[
T = P(s' \mid a, s)
= \{P(\neg d \mid \text{od}, d) = 0.05, \ldots\}
\]

\[
P(\neg d \mid \text{od}, d) = 0.95
\]

\[
P(\neg d \mid \text{od}, \neg d) = 0.05
\]
System model using POMDPs

**Partial Observable Markov Decision Process:**

- Partially observable
- Probabilistic actions/observations

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

\[ O = \{d\text{Obs}\} \], \( d\text{Obs} = \text{door open observation} \)

\[ Z = P(o^i | a, s) \]
\[ = \{P(d\text{Obs} | od, d) = 0.99, \ldots\} \]

- \( P(d\text{Obs} | od, d) = 0.99 \)
- \( P(\neg d\text{Obs} | od, d) = 0.01 \)
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(¬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 q0 = 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 $a = \{q1 \rightarrow \text{move up}, \ldots\}$
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 \( a = \{ q1 \rightarrow \text{move up}, \ldots \} \)

Transition function \( \delta = \{ (q1,q2) \rightarrow (p = \text{personWaitingObs}), \ldots \} \)
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

\[ \neg p \land \neg top \]

\[ \neg p \land \neg top \]

\[ p \land T \]
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

\[ \neg p \land \neg \text{top} \rightarrow \text{move up} \]

\[ p \rightarrow \text{open} \]

\[ T \rightarrow \text{open/close} \]
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 received observations: {}

\[ \neg p \land \neg top \]

move up

\[ top \]

open

\[ p \]

open/close

\[ T \]
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 received observations: \{p\}

$$\neg p \land \neg top \quad \text{move up} \quad \begin{cases} \top \quad \text{open} \\ p \quad \text{open/close} \end{cases}$$
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 received 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 received observations: {top}
Policy model using FSCs

What is the quality of policy $\pi$?

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

BUT: domain is stochastic $\rightarrow V(\pi)$ 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 $m = (A^i_a, \text{partial plan FSC})$

• decomposition: replace $A^i_a$ with implementation
HTN-style planning

Hierarchical Task Network Planning

- Exploitation of expert knowledge
- Method $m = (A_a^i, \text{partial plan } FSC)$
- Decomposition: replace $A_a^i$ with implementation
HTN-style planning

Hierarchical Task Network Planning

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

Initial plan:

Implementation for Go Up:

\[ \neg p \land \neg top \]

move up

open/close

Go up \(T\) Open

\(T\)
Hierarchical POMDPs

Initial plan:

- Go up $\rightarrow$ Open

Implementation for Go Up:

- $\neg p \land \neg top$ $\rightarrow$ move up
- $top$ $\rightarrow$ open/close
- $T$ $\rightarrow$ $p$
- $p$ $\rightarrow$ Open

decomposed plan:

- $\neg p \land \neg top$ $\rightarrow$ move up
- $T$ $\rightarrow$ $p$
- $p$ $\rightarrow$ open/close
- $T$ $\rightarrow$ $p$
- $p$ $\rightarrow$ Open
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 $\pi : f(\pi) = g(\pi) + h(\pi)$
A* - Algorithm

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

cost function $g(\pi)$: guaranteed expected costs for all decompositions
A* - Algorithm

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

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 an approximate optimal policy $\pi^+$ by interacting with a domain simulator.
UCT - Algorithm

Upper Confidence Tree – Algorithm (UCT)

Idea: calculate an approximate optimal policy $\pi^+$ by interacting with a domain simulator.

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

Simplification: find the best primitive policy out of a given set of primitive policies by using the simulator
Algorithms - UCT

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

Trivial approach:

- Simulate every policy $k$ times
Algorithms - UCT

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

Trivial approach:

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

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

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

Trivial approach:

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

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

- Additive Chernoff Bound:
Algorithms - UCT

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

Trivial approach:

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

\[ \bar{V}_{sim}^{k}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j \]

- Additive Chernoff Bound:

\[ P \left( \left| V(\pi) - \bar{V}_{sim}^{k}(\pi) \right| \geq \varepsilon \right) \]
Algorithms - UCT

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

Trivial approach:

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

\[
\bar{V}_{\text{sim}}^{k}(\pi) = \frac{1}{k} \cdot \sum_{j=1}^{k} v_j
\]

- Additive Chernoff Bound:

\[
P\left( \left| V(\pi) - \bar{V}_{\text{sim}}^{k}(\pi) \right| \geq \varepsilon \right) \leq \exp\left( -\varepsilon^2 \cdot k \right)
\]
Algorithms - UCT

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

Exploitation of previous simulation results:

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

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

Exploitation of pervious simulation results:

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

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

$n$: total simulation count

$n(a_i)$: simulation count for policy $\pi_i$
Algorithms - UCT

Simplification: find the best primitive plan out of a given set of primitive 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 ↔ Exploration

\( Q(a_i) \): Average simulation value for the policy \( \pi_i \)

\( n \): total simulation count

\( n(a_i) \): simulation count for policy \( \pi_i \)
**Algorithms - UCT**

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

Exploitation of previous simulation results:

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

Exploitation ↔ Exploration

\(Q(a_i): \) Average simulation value for the policy \(\pi_i\)

\(n: \) total simulation count

\(n(a_i): \) simulation count for policy \(\pi_i\)

→ Transfer to a larger hierarchy possible
Implementation

HPOMDP-Planner

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

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:

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

Evaluation results of the elevator domain:

<table>
<thead>
<tr>
<th>Instance</th>
<th>NoOp/Random</th>
<th>A*</th>
<th>UCT[10s]</th>
<th>SPPerseus</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 floors</td>
<td>-44.4 / -52.3</td>
<td>timeout n/a</td>
<td>-32.6</td>
<td>-35.4</td>
</tr>
<tr>
<td>4 floors</td>
<td>-89.0 / -100.6</td>
<td>timeout n/a</td>
<td>-74.3</td>
<td>timeout n/a</td>
</tr>
<tr>
<td>5 floors</td>
<td>-133.8 / -147.7</td>
<td>timeout n/a</td>
<td>-113.2</td>
<td>timeout n/a</td>
</tr>
<tr>
<td>6 floors</td>
<td>-177.9 / -193.0</td>
<td>timeout n/a</td>
<td>-148.8</td>
<td>timeout n/a</td>
</tr>
</tbody>
</table>

→ A* unexpected inefficient
# Evaluation

Evaluation results of the elevator domain:

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

→ A* unexpected inefficient
→ Quality is very hierarchy dependant
## Evaluation

Evaluation results of the elevator domain:

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

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

UCT convergence:

→ After 10 seconds less than 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.