16-843 – Manipulation Algorithms
Task Planning 2

K. Muelling
Task Planning Summary

What have we learned so far about task planning?

What is task planning?
Task Planning: Problem Definition

Decision making problem in which we have to choose a sequence of actions that will transform the current state of the world, step by step, such that it will satisfy some desired goal constraints.
Task Planning Summary

What have we learned so far about task planning?

What is task planning?

How is it different from motion planning?
Example Problem: Clean Up

[Kaelbling et al. Hierarchical Planning in the Now]
# Task Planning: Problem Definition

Decision making problem in which we have to choose a **sequence of actions** that **will transform the current state** of the world, step by step, such that it will satisfy some **desired goal** constraints.

## Motion Planning

States a robot motion planning problem:
- How to move from one robot configuration into another
- Only considers the robots configuration space

## Task Planning

About solving a complex task that
- Involves a potential large number of objects
- Long time horizons
- State space includes all objects in environment
Task Planning Summary

What have we learned so far about task planning?

What is task planning?

How is it different from motion planning?

Why can we not use the same algorithms as in motion planning?
Main types of Planners

Three main types of planners:

- Domain specific
  - Developed for specific problems
  - Don’t work outside their specific domains
- Domain independent
  - Idea: work with every domain, no problem specific knowledge
  - Makes simplification necessary: classical planning
- Configurable
  - Hierarchical Task Networks (HTN)
Main Types of Planners

Domain dependent task planners

Mars Rover Explorer

Board games: Back Gammon

Wikimedia.org

Wikipedia
Main types of Planners

Domain independent task planners

General goal of AI Planning:
Aim for domain independence, i.e., planner generates plans independent of its application and environment

But: so far, it hardly works in real-life problems

Simplifications that lead to classical planning
Classical Planning

The Planning Problem

Given:
• An Initial State
• A characterization of the Goal
• A set of possible actions

Synthesize:
• Sequence of actions that transform the initial state into a state that fulfills the goal characteristic
# Classical Planning

<table>
<thead>
<tr>
<th>Environment</th>
<th>Fully observable</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Finite</td>
</tr>
<tr>
<td>Actions</td>
<td>Finite</td>
</tr>
<tr>
<td>Objects</td>
<td>Finite</td>
</tr>
<tr>
<td>Time</td>
<td>Actions only have one time step</td>
</tr>
<tr>
<td>Planning</td>
<td>Offline</td>
</tr>
</tbody>
</table>

## Real World
Classical Planning

The Planning Problem

Given:
• An Initial State
• A characterization of the Goal
• A set of possible actions

Synthesize:
• Sequence of actions that transform the initial state into a state that fulfills the goal characteristic
Planning Languages

1971

**STRIPS**: Stanford Research Institute Problem Solver
- Formalism for logic based planning algorithm
- Developed for a robotic system at SRI International

1987

**ADL**: Action Description Language
- Automated planning and scheduling system
- Improve STRIPS by allowing effects of an operator to be conditional

1998

**PDDL**: Planning Domain Definition Language
- Standard AI planning language for the IPC (International Planning Competition)

“For a greater reuse of research and [...] direct comparison of systems and approaches” M. Fox and D. Long
Planning Languages

Components

- **Objects** or **Instances**: List of objects in the world, e.g., table glass, bottle.

- **Predicates**: Properties of the objects. Binary value functions, i.e., can be *true* or *false*. E.g.: On(glass,table).

- **Initial State**: The initial state of the world.

- **Goal**: Set of predicates that specify the conditions to be true and false for the program to terminate.

- **Operators** or **Actions**: Functions that change the state of the world
Specifying a task in PDDL

Two components:

1) **Domain file**: specify predicates and actions

2) **Problem file**: specify objects, initial state and goal specifications for the problem we want to solve
PDDL

Specifying a task in PDDL

Domain Files:

\[
\text{(define (domain domain\_name)}
\text{:requirements :strips)}
\text{(:predicates (p\_1 \?x) \ldots (p\_n \?x \?y))}
\text{(:action action\_name)}
\text{:parameters (?v\_1 ?v\_2 \ldots ?v\_m)}
\text{:precondition (and (p\_2 ?v\_2 ?v\_3) \ldots (not(p\_k ?v\_1)))}
\text{:effect (and (p\_1 ?v\_1) \ldots))}
\]

List of predicates

Action definition
Specifying a task in PDDL

Domain Files:

(define (domain flash_light)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (in ?x ?y))
  (:action remove_cap
   :parameters (?cap ?flash)
   :precondition ((on ?cap ?flash))
   :effect (not (on ?cap ?flash)))
  (:action insert
   :parameters (?bat ?flash ?cap)
   :precondition (and (not (in ?bat, ?flash))
                    (not (on ?cap, ?flash)))
   :effect (in ?bat, ?flash))

Battery1
Battery2
Flashlight
Cap
PDDL

Specifying a task in PDDL

Problem Files:

```
(define (problem problem_name)
 (:domain domain_name)
 (:objects i_1, i_2, ..., i_l)
 (:init (p_1 i_2 i_b)
       ...
       (p_d i_1))
 (:goal (and (p_2 i_2 i_6)
                ...
                (p_d i_5)))
```

- List of objects in your environment
- Initial State
- Goal criteria
PDDL

Specifying a task in PDDL

Problem Files:

(define (problem flash_light_1)
  (:domain flash_light)
  (:objects cap, flash, bat1, bat2)
  (:init (on cap flash)
    )
  (:goal (and (in bat1 flash)
    (in bat2 flash)
    (on cap flash)
    )))
Classical Planning

The Planning Problem

**Given:**
- An Initial State
- A characterization of the Goal
- A set of possible actions

** Synthesize:**
- Sequence of actions that transform the initial state into a state that fulfills the goal characteristic
Classical Planning

How do we do the planning?

Search for plans: Search for paths in directed graph
Planning through Search

Only apply effective actions

Forward Search:

```
On(Cap, Flash)
removeCap(Cap, Flash)
{ }
Insert(Bat1, Flash)
Insert(Bat2, Flash)
Place(Cap, Flash)
```

Forward (progression) state-space search: Start at initial state and use only actions that are relevant, i.e., actions whose pre-condition match a subset of the state
Planning through Search

Only apply effective actions

Backward Search:

Backward (regression) state-space search: Start at final state and use only actions that are relevant inversely (i.e., actions whose effects match a subset of the state) and consistent.
Planning graph: sequence of levels that correspond to time steps

Each level contains set of literals that could be true at this time step

Each level contains set of actions that could be applied
Planning through Search

How do we search in the graph?

Blind Search:
• Depth First Search
• Dijkstra

Informed/Heuristic Search:
• A*
• Hill Climbing
• Best First Search

Usually more effective!

How do we get the heuristic automatically?
Planning through Search

FF
The Fast-Forward Planning System

Joerg Hoffmann
Overview Planning Algorithm

- STRIPS 1972
- GraphPlan 1995
- FF 2001
- 1. IPC 1998
- 8. IPC 2014
- 2006 FastDownward
- Domain Independent Heuristics
Pancakes

Solution with FF:

ff: found legal plan as follows

step 0: ADD FLOUR BLENDER TABLE
1: ADD MILK BLENDER TABLE
2: ADD SUGAR BLENDER TABLE
3: ADD EGG1 BLENDER TABLE
4: ADD EGG2 BLENDER TABLE
5: TURN_ON BLENDER

Why is this not enough?
How do we replace a tire?
Multi-Step Manipulation Planning
Multi-Step Manipulation Planning

FF: remove tire

Involves:
- dual arm manipulation
- reachability of places
- orientation sensitive

0: MOVE R_ARM from PIAR to PDB
1: MOVE R_ARM from PDB to PWG
2: MOVE L_ARM from PIAL to PD
3: GRASP L_ARM DRIVER at PD
4: MOVE_OBJ DRIVER with L_ARM from PD to PWG
5: RELEASE L_ARM DRIVER at PWG
6: GRASP R_ARM DRIVER at PWG
7: ROTATEOBJECT R_ARM DRIVER from (NY,NX) to (Z,NY)
8: ASSEMBLEDRILL: R_ARMR DRIVER BASE from PWG at PDB (Z, NY)
9: MOVEDRILLTOOBJECT: R_ARM from PDB to PL1
10: DRILL: R_ARM LUG1 at PL1 (Z, NY)
11: MOVEDRILLFROMOBJECTTOOBJECT: R_ARM from PL1 to PL2
12: DRILL: R_ARM LUG2 at PL2 (Z, NY)
13: MOVEDRILLFROMOBJECTTOOBJECT: ARMR from PL2 to PL3
14: DRILL: R_ARM DRILL LUG3 at PL3 (Z, NY)
15: MOVEDRILLFROMOBJECTTOOBJECT: R_ARMR DRILL from PL3 to PL4
16: DRILL: R_ARM DRILL LUG4 at PL4 (Z, NY)
17: MOVEDRILLFROMOBJECT: R_ARM DRILL from PL4 to PIAR
18: MOVE: L_ARM from PWG to PW
19: GRASP: L_ARM WHEEL at PW
20: MOVE: ARMR DRILL from PIAR to P3
21: RELEASE R_ARM DRIILL at P3
22: MOVE R_ARM from P3 to PW
23: GRASP R_ARM WHEEL with at PW
24: MOVE WHEEL with R_ARM and L_ARM from PW to PWG
25: RELEASE L_ARM WHEEL at PWG
26: RELEASE R_ARM WHEEL at PWG
27: MOVE L_ARM from PWG to PIAL
28: MOVE R_ARM from PWG to PIAR

Time: 2.06 seconds
Task Planning for Robots

PDDL definition

```
(define (domain domain_name)
  (:requirements :strips)
  (:predicates (p1 ?x)
    ....
    (pn ?x ?y))
  (:action action_name
    :parameters (?v1 ?v2 ... ?vm)
    :precondition (and (p2 ?v2 ?v3)
                     ...
                     (not(pk ?v1)))
    :effect (and (p1 ?v1)
                 ....))
)
(:action action_name2 ...)
```

Task Plan:
- RemoveCap(Cap,Flashlight)
- Insert(Bat1,Flashlight)
- Insert(Bat2,Flashlight)
- PutCap(Cap,Flashlight)

Symbolic Planner

Motion Planner

Execution
Task Planning for Robots

PDDL definition

(define (domain domain_name)
  (:requirements :strips)
  (:predicates (p1 ?x)
    ....
    (pn ?x ?y))
  (:action action_name
    :parameters (?v1 ?v2 ... ?vm)
    :precondition (and (p2 ?v2 ?v3)
      ....
      (not(pk ?v1)))
    :effect (and (p1 ?v1)
      ....))
)
(:action action_name2 ...)

Task Plan:
RemoveCap(Cap,Flashlight)
Insert(Bat1,Flashlight)
Insert(Bat2,Flashlight)
PutCap(Cap,Flashlight)

Symbolic Planner

Motion Planner

Execution

Error
# Task Planning for Robots

<table>
<thead>
<tr>
<th></th>
<th>Classical Planning</th>
<th>Real World</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Environment</strong></td>
<td>Fully observable</td>
<td>Partial observable</td>
</tr>
<tr>
<td></td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td><strong>States</strong></td>
<td>Finite</td>
<td>Non-finite, continuous</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>Finite</td>
<td>Non-finite</td>
</tr>
<tr>
<td></td>
<td>Deterministic</td>
<td>Non-deterministic</td>
</tr>
<tr>
<td><strong>Objects</strong></td>
<td>Finite</td>
<td>Non-finite</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>Actions only have one time step</td>
<td>We have the notion of time, actions can take different amount of times</td>
</tr>
<tr>
<td><strong>Planning</strong></td>
<td>Offline</td>
<td>Online</td>
</tr>
</tbody>
</table>
Can we just interpret the low level planning problem into the task planning?
Task And Motion Planning Approaches

- **Forward Search:**
  - Dornhege et al., Semantic attachments for domain-independent planning systems, 2014
  - Garret et al., FFRob: An efficient heuristic for task and motion planning, 2015

- **Hierarchical TAMP**
  - Kaelbling et al., Hierarchical Planning in the now, 2011

- **Plan Skeleton**
  - Lozano-Perez, A constraint based method for solving sequential manipulation problems, 2014

- **MDP approaches**
Task And Motion Planning Approaches

- **Forward Search:**
  - Dornhege et al., Semantic attachments for domain-independent planning systems, 2014
  - Garret et al., FFRob: An efficient heuristic for task and motion planning, 2015
- **Hierarchical TAMP**
  - Kaelbling et al., Hierarchical Planning in the now, 2011
- **Plan Skeleton**
  - Lozano-Perez, A constraint based method for solving sequential manipulation problems, 2014
- **MDP approaches**
Task And Motion Planning Approaches

- Forward Search:
  - Dornhege et al., Semantic attachments for domain-independent planning systems, 2014
  - Garret et al., FFRob: An efficient heuristic for task and motion planning, 2015

- Hierarchical TAMP
  - Kaelbling et al., Hierarchical Planning in the now, 2011

- Plan Skeleton
  - Lozano-Perez, A constraint based method for solving sequential manipulation problems, 2014

- MDP approaches
Problem: Pick and Place with many objects

Motion Planning:
- Traditional motion planning algorithms cannot handle problems where the configuration space contains more than robot config

Symbolic Planning:
- Efficient in constructing plans with many entities
- Cannot incorporate kinesthetic and geometric constraints
FFRob: An efficient heuristic for task and motion planning

**Problem:** Pick and Place with many objects

**Specifically:**

**PICK**($C_1,O,G,P,C_2$):
- **pre:** HandEmpty, Pose($O,P$), RobotConf ($C_1$), CanGrasp($O;P;G;C_2$), Reachable($C_1;C_2$)
- **add:** Holding($O;G$), RobotConf ($C_2$)
- **delete:** HandEmpty, RobotConf ($C_1$)

**PLACE**($C_1,O,G,P,C_2$):
- **pre:** Holding($O;G$), RobotConf ($C_1$), CanGrasp($O;P;G;C_2$), Reachable($C_1;C_2$)
- **add:** HandEmpty, Pose($O,P$), RobotConf ($C_2$)
- **delete:** Holding($O;G$), RobotConf ($C_1$)

*High-dimensional continuous*  
*Many conditional effects*
FFRob: An efficient heuristic for task and motion planning

Idea:
Extend heuristic ideas from FF planner to motion planning using probabilistic roadmaps
FFRob: An efficient heuristic for task and motion planning
Summary Task Planning

General goal of AI Planning:
Aim for domain independence, i.e., planner generates plans independent of its application and environment

But: so far, it hardly works in real-life problems

Simplifications that lead to classical planning
Summary Task Planning

Components of planning languages

• **Objects** or **Instances**: List of objects in the world, e.g., table glass, bottle.

• **Predicates**: Properties of the objects. Binary value functions, i.e., can be *true* or *false*. E.g.: On(glass, table).

• **Initial State**: The initial state of the world.

• **Goal**: Set of predicates that specify the conditions to be true and false for the program to terminate.

• **Operators** or **Actions**: Functions that change the state of the world

PDDL
Summary Task Planning

Search for plans: Search for paths in directed graph

Blind Search:
• Depth First Search
• Dijkstra

Informed/Heuristic Search:
• A*
• Hill Climbing
• Best First Search

1. 1972 STRIPS
2. 1995 GraphPlan
3. 2001 FF
4. 2006 FastDownward
5. 2014 8. IPC

Domain Independent Heuristics

1972
1995
2001
2014
Summary Task Planning

Problem with current planning

Motion Planning:
- Traditional motion planning algorithms cannot handle problems where the configuration space contains more than robot config

Symbolic Planning:
- Efficient in constructing plans with many entities
- Cannot incorporate kinesthetic and geometric constraints
Summary Task Planning

Combine Motion and Task Planning

• Forward Search:
  • Dornhege et al., Semantic attachments for domain-independent planning systems, 2014
  • Garret et al., FFRob: An efficient heuristic for task and motion planning, 2015

• Hierarchical TAMP
  • Kaelbling et al., Hierarchical Planning in the now, 2011

• Plan Skeleton
  • Toussaint, Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning, 2015
  • Lozano-Perez, A constraint based method for solving sequential manipulation problems, 2014

• MDP approaches