Task and Motion Planning

Dylan Hadfield-Menell

UC Berkeley CS 287 Guest Lecture
Planning for Complex Tasks
Outline

- Task Planning
  - Formulation
  - Fast-Forward

- Task and Motion Planning
  - Forward Search
  - Plan Skeletons
  - Extension: Partial observability
Example Domain
Motion Planning

- **Initial State:**

- **Goal State:**
  - Target robot pose
Motion Planning++

- Initial State:

- Goal State:
  - Set of Robot Configurations
  - In(Robot, Room2)
  - In(Robot, Room3)?
Task and Motion Planning

- **Initial State:**
  - In(A, Room3) 
  - In(B, Room2)

- **Goal State:**
  - In(A, Room3) ^ In(B, Room2)
Early Robotics: Shakey the Robot
Task Planning: State Representation

- Represent state of the world as list of true properties

```
In(Robot, R0)
In(A, R1)
In(C, R2)
In(B, R0)
Holding(Robot, None)
Blocks(B, R0, R3)
```

Diagram:

- A
- C
- B
- Robot

An operator is defined by 3 attributes:

- **Name**: Identifier for the action
- **Preconditions**: List of fluents that must be true in order to take the action
- **Effects**:
  - Add list: fluents that become true after the action
  - Delete list: fluents that become false after the action

**Preconditions**
- In(robot, R0)
- Connected(R0, R1)
- ~Blocks(A, R0, R1)
- ~Blocks(B, R0, R1)
- ~Blocks(C, R0, R1)

**Effects**
- In(robot, R1)
- ~In(robot, R0)
Task Planning: More Actions

**Pick(A, R0)**

- **Preconditions**
  - Holding(None)
  - In(A, R0)
  - In(robot, R0)

- **Effects**
  - ~Holding(None)
  - Holding(A)

**Clear(B, R0, R1)**

- **Preconditions**
  - Blocks(B, R0, R1)
  - In(robot, R0)
  - Holding(None)

- **Effects**
  - ~Blocks(B, R0, R1)
  - ~Holding(None)

**MoveHolding(A, R0, R1)**

- **Preconditions**
  - In(robot, R0)
  - Holding(A)
  - Connected(R0, R1)
  - ~Blocks(A, R0, R1)
  - ~Blocks(B, R0, R1)
  - ~Blocks(C, R0, R1)

- **Effects**
  - In(robot, R1)
  - ~In(robot, R0)
  - In(A, R1)
  - ~In(A, R0)
Planning Domain Description Language

- Standardized format to represent planning problems
- Used for International Planning Competitions
  - Lots of published code that can read this representation
- Domain file defines
  - Fluents, object types, operator schemas
- Problem file defines
  - Objects, Initial state, Goal condition
Example PDDL Domain

(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b)
    (gripper ?g)
    (at-robby ?r)
    (at ?b ?r) (free ?g)
    (carry ?o ?g))

(:action move
  :parameters (?from ?to)
  :precondition (and (room ?from)
    (room ?to)
    (at-robby ?from))
  :effect (and (at-robby ?to)
    (not (at-robby ?from))))

Example from Manuela Veloso
Example PDDL Domain (cont’d)

(:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj)
                     (room ?room)
                     (gripper ?gripper)
                     (at ?obj ?room)
                     (at-robby ?room)
                     (free ?gripper))
  :effect (and (carry ?obj ?gripper)
              (not (at ?obj ?room)) (not (free ?gripper))))

(:action drop
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj)
                     (room ?room)
                     (gripper ?gripper)
                     (carry ?obj ?gripper)
                     (at-robby ?room))
  :effect (and (at ?obj ?room)
              (free ?gripper)
              (not (carry ?obj ?gripper))))

Example from Manuela Veloso
Example PDDL Problem

(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma) (room roomb)
    (ball ball1) (ball ball2)
    (gripper left) (gripper right)
    (at-robby rooma)
    (free left) (free right)
    (at ball1 rooma) (at ball2 rooma))
  (:goal (at ball1 roomb)))

Solution:
  pick(ball1 rooma left)
  move(rooma roomb)
  drop(ball1 roomb left)
Algorithms for Task Planning

1959: GPS

1972: STRIPS

1995: GraphPlan

2001: Fast Forward

2006: Fast Downward

1998: First IPC

2014: IPC-8
50+ submissions

Lots of intermediate approaches

Not to scale
Preprocessing Step before planning

Can reveal natural structure in problem

Compute over-approximation of reachable set of literals
Planning Graph [Blum & Furst ‘95]

$L_0 \leftarrow \text{all facts true in initial state}$

$t \leftarrow 0$

While $goal \notin L_t$

$L_t \leftarrow \text{facts from } L_{t-1}$

For each action with $pre(a) \in L_{t-1}$

$L_t = L_t \cup eff(a)$

$t \leftarrow t + 1$

Theorem: $L_t$ is a superset of reachable set of fluents for plans of length $t$
Early use of plan graphs analyzed the plan graph to extract a sequence of actions

Fast-Forward: use the length of the planning graph as a heuristic inside of a forward search
  - Actually use relaxed planning graph, which ignores delete effects
  - Some modifications to handle very slow heuristic computation

Fast-Forward [Hoffmann 2001]
\[ Q \leftarrow \text{PriorityQueue}() \]
\[ Q.push(\text{init}, 0) \]

While goal not found
\[ s \leftarrow Q.pop() \]
\[ pg \leftarrow \text{RelaxPlanGraph}(s, \text{goal}) \]
\[ \text{for } c \text{ in } s.\text{children} \]
\[ Q.push(c, \text{len}(pg)) \]
Enforced hill climbing
- Greedy search + breadth-first search to account for plateaus

Push children with heuristic evaluated on parent
- 1 heuristic evaluation/expansion
- Alternative is 1 heuristic evaluation/child

Helpful actions
- When planning graph terminates, we can extract a plan with simultaneous actions
- Search those actions first
Binary State Representation
- Properties of the world that change over time

Actions defined by preconditions and effects

State-of-the-art relies on heuristic forward search with domain independent heuristics
Task Planning for Robots (the hope)

Binary Planning Representation

Continuous Full Representation

In(Robot, R0) In(A, R1) In(C, R2) In(B, R0) Holding(Robot, None) Blocks(B, R0, R3)

Task Planning

Motion Planning

Execution
Binary Planning Representation

<table>
<thead>
<tr>
<th>In(Robot, R0) In(A, R1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>In(C, R2) In(B, R0)</td>
</tr>
<tr>
<td>Holding(Robot, None)</td>
</tr>
<tr>
<td>Blocks(B, R0, R3)</td>
</tr>
</tbody>
</table>

Continuous Full Representation

No Plan Found!

Task Planning

Motion Planning

Execution
Executing a Task Plan

- Each high level action encodes a motion planning problem
- Ex. Move(R0, R1)
  - Initial State: Current robot pose
  - Goal State: anything in R1
- Motion plan each step in sequence
  - Issue: dependency between intermediate steps of plan
Dependency for intermediate states

Move(R0, R1)  Move(R1, R2)

Solution: Try several intermediate poses for each action
What if the task plan itself is wrong?
A Continuous Representation

- **Goal:** Holding(robot, A)

- **High-Level Actions**
  - Grasp(robot, r_pose, obj, o_pose, grasp)
  - Move(robot, pose1, pose2)
  - Place(robot, r_pose, obj, grasp, obj_pose)

- Grasps, poses, and locations are all continuous
A Continuous Operator

Grasp(robot, r_pose, obj, o_pose, grasp)

Preconditions:

GraspPose(r_pose, o_pose, grasp)
At(robot, r_pose)
At(obj, o_pose)
Holding(robot, None)

Effects:

~At(obj, o_pose)
Holding(robot, obj)
~Holding(robot, None)
\( \forall p1, p2 \sim \text{Obstructs}(obj, p1, p2) \)
Task and Motion Planning Approaches

- **Forward Search**

- **Hierarchical TAMP**

- **Plan Skeleton**
Strawman TAMP Algorithm: Discretize

- Replace each continuous value with a set of discrete options
- Compute all relevant properties
- Run your favorite task planner
  - Now it sets intermediate poses as well
- Issues?
  - Curse of dimensionality
  - Lots of irrelevant motion planning
Main idea: lazily discretize values and compute properties during search

TAMP via Forward Search

- Move
- Grasp
Forward Search

\[ Q \leftarrow \text{PriorityQueue}() \]
\[ Q.\text{push}(\text{init}, 0) \]

While goal not found

\[ s \leftarrow Q.\text{pop}() \]

\[ pg \leftarrow \text{RelaxPlanGraph}(s, \text{goal}) \]

for each applicable action, \( a \) s.t. \( \text{pre}(a) \in s \)

\[ \text{children} \leftarrow \text{Discretize}(s, a) \]

for \( c \in \text{children} \)

\[ Q.\text{push}(c, h(c)) \]

Challenge: What goes here?
Node expansions are very slow

- >95% of running time is spent answering motion planning queries
- Efficient caching strategies can help a lot
- [aSyMov ‘05] interleave PRM iterations with search iterations

Useful heuristic information

- Obtaining useful heuristic information has been a primary bottleneck
- Recent work investigates efficient computation of plan graph heuristic [Garrett ‘15]
Task and Motion Planning Approaches

**Forward Search**


**Hierarchical TAMP**


**Plan Skeleton**

Initially plan with abstract representation that ignores continuous dynamics

Output can be thought of as a continuous constraint satisfaction problem
  - Preconditions $\rightarrow$ constraints

Algorithm sketch
  - Generate task plan
  - Attempt to solve CSP
  - If failure, generate new plan
A Continuous Operator

Continuous parameters

\text{Grasp}(\text{robot, } r\text{\_pose, obj, } o\text{\_pose, grasp})

Preconditions:
\begin{align*}
\text{GraspPose}(r\text{\_pose, o\_pose, grasp}) \\
\text{At(\text{robot, } r\text{\_pose})} \\
\text{At(}\text{obj, } o\text{\_pose}) \\
\text{Holding(\text{robot, None})}
\end{align*}

Effects:
\begin{align*}
\neg\text{At(}\text{obj, } o\text{\_pose}) \\
\text{Holding(\text{robot, obj})} \\
\neg\text{Holding(\text{robot, None})} \\
\forall p1, p2 \neg\text{Obstructs(}\text{obj, p1, p2})
\end{align*}
Replace continuous values with symbolic references

Leave these values *uninstantiated* during task planning

*Refine* task plan to pick values for continuous parameters

---

**Symbols**

- $P_A$: “object pose where A is”
  - type: object pose
- $G_A$: “grasp we can use for A”
  - type: grasp
- $GP_A$: “pose with a valid grasp for A”
  - type: robot pose
- $P_R$: “initial robot pose”
  - type: robot pose

**Properties**

- $\text{At(robot, } P_R\text{)}$
- $\text{At } (A, P_A)$
- $\text{GraspPose}(GP_A, P_A, G_A)$

Planning with an Interface

Goal: Holding(robot, A)

Plan Skeleton:
1. Move(robot, P_R, GP_A)
2. Grasp(robot, GP_A, A, P_A, G_A)

Symbols
P_A, G_A, ...

Properties
At(robot, P_R)
...

Interface

<table>
<thead>
<tr>
<th>P_R</th>
<th>(0,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_A</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>G_A</td>
<td>(0.1, 0.2)</td>
</tr>
<tr>
<td>GP_A</td>
<td>(1.1, 1.2)</td>
</tr>
</tbody>
</table>

Symbols
P_A, G_A, ...

Proper*es
At(robot, P_R) ...

Task Planner

Fail

Motion Planner
Planning with an Interface

Goal: Holding(robot, A)

Plan Skeleton:
1. Move(robot, P_R, GP_A)
2. Grasp(robot, GP_A, A, P_A, G_A)

Symbols
P_A, G_A, ...
Properties
At(robot, P_R)
...

Final Plan:
1. Move(robot, (0,0), (0.9, 0.8))
2. Grasp(robot, (0.9, 0.8), A, (1,1), (0.1, 0.2))
What do we lose with symbol references?

- High level can’t know anything that depends on specific values of parameters
- E.g. what if B blocks A

Solution:

- Interface queries motion planner to determine failure
- Updates high level
Error Propagation

Plan Skeleton:
1. Move(robot, \( P_R, GP_A \))
2. Grasp(robot, \( GP_A, A, P_A, G_A \))
3. Obstructs(B, \( P_R, GP_A \))

Plan Skeleton:
1. Move(robot, \( P_R, GP_B \))
2. Grasp(robot, \( GP_B, B, P_B, G_B \))
3. Move(robot, \( GP_B, PDP_B \))
4. Place(robot, \( PDP_B, B, PDP_B, G_B \))
5. ...

Interface

Task Planner

Motion Planner
Plan Refinement via Local Search

Plan Skeleton

Move(robot, \(P_R, GP_A\))
\(~\text{Obstructs}(A, P_R, GP_A)\)
\(~\text{Obstructs}(B, P_R, GP_A)\)

Grasp(robot, \(GP_A, A, P_A, G_A\))

GraspPose(\(GP_A, P_A, G_A\))

Holding(robot, None)

Preconditions constrain potential values of symbols

Calls to Motion Planner

Initialize symbols

Determine violated constraint

Modify symbols of violated constraints
Plan Refinement via Local Search

- Initialize symbols:
  - \( P_R \), \( P_A \), \( G_A \), \( GP_A \)
  - Positions: (0,0), (1,1), (0.1,0.2), (1.1,1.2)

- Determine violated constraint:
  - \( \text{Obstructs}(A, P_R, GP_A) \)
  - \( \sim \text{GraspPose}(GP_A, P_A, G_A) \)

- Modify symbols of violated constraints:
Plan Refinement via Local Search

- Initialize symbols
- Determine violated constraint
- Modify symbols of violated constraints

Obstructs(A, P_R, GP_A)
\sim GraspPose(GP_A, P_A, G_A)

Constraint ordering

Conditional distribution over symbol values
Searching over Plan Skeletons

- Using the failure information to generate the next state defines a graph
  - Nodes are plan skeletons
  - Edges are failure explanations
- Interleave node expansion (failure propagation) and node refinement (motion planning)

[Guided Search for Task and Motion Plans Using Learned Heuristics Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Pieter Abbeel. ICRA, 2016 (under review)].
Searching over Plan Skeletons

Plan Skeleton:
1. Move(robot, P_R, GP_A)
2. Grasp(robot, GP_A, A, P_A, G_A)

Plan Skeleton:
1. Move(robot, P_R, GP_B)
2. Grasp(robot, GP_B, B, P_B, G_B)
3. Move(robot, GP_B, PDP_B)
4. Place(robot, PDP_B, B, PDP_B, G_B)
5. ...

Challenge: need useful heuristics to effectively search this graph.
Solution: learn a heuristic (details at final project presentations)
Task and Motion Planning Summary

- Pure Task Planning doesn’t work directly because of
  - Abstracted continuous dynamics
  - Long horizons

- Solution methods
  - Discretize and represent everything logically
  - Discretize lazily and run motion planning during search
  - Plan abstractly and fill in continuous values later
    - Get a new plan if that doesn’t work
Extension: Partial Observability

Belief State

- Proposal: treat beliefs like poses
  - Symbolic references let us reason about and plan with continuous state

Physical State

A

\[
\begin{align*}
P &= 0.3 \\
P &= 0.7 \\
P &= 0.05 \\
P &= 0.95
\end{align*}
\]
Challenge: Non-determinism

- Observations depend on physical state
  - Which we don’t know!

- Approximate solution:
  - Assume that each belief state deterministically generates its maximum likelihood observation\(^1\)
  - Re-plan if necessary

\[ P = 0.3 \quad P = 0.7 \]

Belief State

\(^1\) Platt et al. "Belief space planning assuming maximum likelihood observations." RSS (2010).
Challenge: Non-determinism

Belief State

\[ P = 0.3 \quad P = 0.7 \]

\[ P = 0.7 \]

\[ P = 0.3 \]
A Partially Observed Move

Move(robot, r_p1, r_p2)

Preconditions:
- At(robot, r_p1)
- $\forall$ obj $\neg$Obstructs(obj, r_p1, r_p2)

Effects:
- $\neg$At(robot, r_p1)
- At(robot, r_p2)

PO-Move(robot, r_p1, r_p2)

Preconditions:
- At(robot, r_p1)
- $\forall$ obj $\neg$BOBstructs(obj, r_p1, r_p2)

Effects:
- $\neg$At(robot, r_p1)
- At(robot, r_p2)

obj blocks trajectory
Achieved by Pick

w.h.p. obj blocks trajectory
Achieved by Pick OR Observe
Logical Belief State Dynamics

Should I pick up B or observe it??

$P = 0.3$

$P = 0.7$
In the POMDP formulation, answering this question is complicated...

Key Idea: observation will only be useful if it lets us conclude that B is not in the way

- We’ve assumed maximum likelihood observations, so this is tractable

Should I pick up B or observe it??
Split properties of belief states into 2 cases
- Properties of maximum likelihood states
- Properties of associated uncertainty

Interface determines which caused failure and updates high level

PO-Move(robot, r_p1, r_p2)

Preconditions:
At(robot, r_p1)

Effects:
~At(robot, r_p1)
At(robot, r_p2)

Achieved by Observe
Achieved by Pick
Refining a Plan Skeleton in Belief Space

Plan Skeleton:
1. PO-Move(robot, P_R, GP_A )
2. PO-Grasp(robot, GP_A, A, BP_A, G_A )

Weighted # of collisions < safety threshold

Belief State: Sampled Obstacles

Belief Query: Sampled Obstacles

Interface:
- sampled obstacles
- bp_a, g_a
- grasp success

Motion Planner:
- p_r, gp_a
- success
Error Propagation in Belief Space

Plan Skeleton:
1. PO-Move(robot, P_R, GP_A)
2. PO-Grasp(robot, GP_A, A, BP_A, G_A)

UObstructs(B, P_R, GP_A)

Belief Query
Sampled Obstacles
ML Query
ML Obstacles

Sampled Obstacles p_r, gp_a
fail
ML p_r, Obstacles gp_a
success
Error Propagation in Belief Space

Plan Skeleton:
1. PO-Move(robot, P_R, GP_A )
2. PO-Grasp(robot, GP_A, A, BP_A, G_A )

ML Query
MLObstructs(B, P_R, GP_A )

Sampled Obstacles
ML Obstacles
fail

Sampled Obstacles
ML Obstacles

ML Obstacles

Belief State
Interface
Motion Planner