Task and Motion Planning (TAMP)

Caelan Reed Garrett
Advisors: Tomás Lozano-Pérez and Leslie Pack Kaelbling
08/29/2019 @ NVIDIA Seattle Robotics Lab

web.mit.edu/caelan/
github.com/caelan/pddlstream
(Probable) Roadmap

1. Background
   1. Task Planning
   2. Motion Planning

2. Hybrid Planning
   1. Prediscretized & Numeric Planning
   2. Multi-Modal Motion Planning
   3. Integrated TAMP

3. PDDLStream Language and Algorithms
   4. Temporal TAMP
   5. TAMP under Uncertainty

[Fig from Erion Plaku]
Planning for Autonomous Robots

- Robot must select both **high-level** actions & **low-level** controls
- **Application areas**: semi-structured and human environments
Task and Motion Planning (TAMP)

- Plan in a **factored, hybrid** space
  - **Discrete** and **continuous** variables & actions

- **Variables**
  - **Continuous**: robot configuration, object poses, door joint positions,
  - **Discrete**: is-on, is-in-hand, is-holding-water, is-cooked, ...

- **Actions**: move, pick, place, push, pull, pour, detect, cook, ...
Cooking and Stacking
Preparing Coffee
Automated Fabrication

- Plan sequence of **306** 3D printing extrusions (actions)
- Collision, kinematic, **stability** and **stiffness** constraints

[Huang, Garrett, & Mueller 2018]
Problem Class

- **Discrete-time**
  - Plans are finite sequences of controls
- **Deterministic** (for now)
  - Actions always produce the intended effect
  - Solutions are *plans* (instead of policies)
- **Observable** (for now)
  - Access to the full world state
- **Hybrid**
  - States & controls composed of *mixed discrete-continuous variables*
Task Planning
Classical (Task) Planning

- Key focus: **discrete** problems with **many variables**
  - Often enormous, but **finite**, state-spaces

- Problems typically described using an **action language**
  - **Propositional Logic** (STRIPS) [Fikes 1971]  [Aeronautiques 1998]
  - **Planning Domain Description Language** (PDDL)

- Develop **domain-independent** algorithms
- Can apply to **any problem** expressible using PDDL
- Exploit **factored** and **sparse** structure to develop efficient algorithms
First-Order Action Languages

- **Predicate**: boolean function  
  
  (On ?b1 ?b2)=True/False

- **Facts (literals)**: instantiated predicates  
  
  (On D C)=True

- **State**: set of facts  
  
  \{(On A B)=False, (On D C)=True, \ldots\}

- Equivalently, boolean state variables

- **Closed-world** assumption: unspecified facts are false

- **Example**: Blocksworld domain

**Facts**: on(x, y), onTable(x), clear(x), holding(x), armEmpty().

**Initial state**: \{onTable(E), clear(E), \ldots, onTable(C), on(D, C), clear(D), armEmpty()\}.

**Goal**: \{on(E, C), on(C, A), on(B, D)\}.  

**Actions**: stack(x, y), unstack(x, y), putdown(x), pickup(x).  

[Figs from Hector Geffner]
(Lifted) Action Schema

- A tuple of free parameters
- A precondition formula tests applicability
- An effect formula modifies the state
- Logical conjunctions enable factoring
- Effects are deltas

```
(:action stack
 :parameters (?b1 ?b2)
 :precondition (and
               (Holding ?b1) (Clear ?b2))
 :effect (and
           (ArmEmpty)
           (On ?b1 ?b2) (Clear ?b1)
           (not (Holding ?b1))
           (not (Clear ?b2))))

(:action unstack
 :parameters (?b1 ?b2)
 :precondition (and
               (ArmEmpty) (On ?b1 ?b2) (Clear ?b1))
 :effect (and
           (Holding ?b1) (Clear ?b2)
           (not (Clear ?b1))
           (not (ArmEmpty))
           (not (On ?b1 ?b2))))
```
Planning Approaches

- **State-space search:** [Bonet 2001] [Hoffman 2001] [Helmert 2006]
  - Progression (forward) or regression (backward)
  - Best-first **heuristic search** algorithms
- **Partial-order planning** [Penberthy 1992]
  - Search directly over plans (**plan-space**)
- Planning as **Satisfiability** [Kautz 1999]
  - Compile to **fixed-horizon** SAT instance
  - SAT is **NP-Complete**
  - Planning is **PSPACE-Complete**
  - Increase horizon if formula unsatisfiable
Forward Best-First Search

- For a state $s$
  - Path cost: $g(s)$
  - Heuristic estimate: $h(s)$
  - Open list sorted by priority $f(s)$
- Weighted A*: $f(s) = g(s) + wh(s)$
  - Uniform cost search: $w = 0 \implies f(s) = g(s)$
  - A* search: $w = 1 \implies f(s) = g(s) + h(s)$
  - Greedy best-first search: $w = \infty \implies f(s) = h(s)$
- How do we estimate $h(s)$?
- No obvious metric (no metric-space embedding)
Predict the Minimum Plan Length

- Can stack / unstack anywhere on the ground
- Hint: is an **even** number

```
Initial State

Goal State
```
Solution (length=6):
- (unstack D C)
- (stack D B)
- (unstack C ground)
- (stack C A)
- (unstack E ground)
- (stack E C)
Predict the Minimum Plan Length

Initial State

Goal State
Domain-Independent Heuristics

- Estimating $h(s)$ is nontrivial
- Can we do it in an a domain-independent manner?
- Solve a related, approximate planning problem
  - Primary focus for almost all of classical planning

- Suggestions for how to do this?
  - **Independently** plan for each goal
  - **Remove** some action preconditions  [Helmert 2006]
  - Remove negative (delete) effects  [Bonet 2001] [Hoffman 2001]
  - ...

[Bonet 2001] [Hoffman 2001] [Helmert 2006]
Delete-Relaxation Heuristics

- Remove all negative (not) effects
- Solving optimally is NP-Complete
- Can greedily find a short plan in polynomial time
- Basis for both admissible and greedier, non-admissible heuristics

```
(:action stack
  :parameters (?b1 ?b2)
  :precondition (and
      (Holding ?b1) (Clear ?b2))
  :effect (and
      (ArmEmpty)
      (On ?b1 ?b2) (Clear ?b1)
      (not (Holding ?b1))
      (not (Clear ?b2))))

(:action unstack
  :parameters (?b1 ?b2)
  :precondition (and
      (ArmEmpty) (On ?b1 ?b2)
      (Clear ?b1))
  :effect (and
      (Holding ?b1) (Clear ?b2)
      (not (Clear ?b1))
      (not (ArmEmpty))
      (not (On ?b1 ?b2))))
```
Predict the Minimum Delete-Relaxed Plan Length

- Can stack / unstack anywhere on the ground
- Hint: is no greater than 6

Initial State

Goal State
Predict the Minimum Delete-Relaxed Plan Length

- **Solution** (length=6):
  - (unstack D C)
  - (stack D B)
  - (unstack C ground)
  - (stack C A)
  - (unstack E ground)
  - (stack E C)
Can stack / unstack anywhere on the ground

Hint: is an **even** number
Predict the Minimum Plan Length

- **Solution** (length=12):
  - (unstack E C)
  - (stack E ground)
  - (unstack C A)
  - (stack C ground)
  - (unstack E ground)
  - (stack E C)
  - (unstack B D)
  - (stack B ground)
  - (unstack D ground)
  - (stack D A)
  - (unstack B ground)
  - (stack B D)

![Diagram of initial and goal states]
Predict the Minimum Delete-Relaxed Plan Length

- Can stack / unstack anywhere on the ground
- Hint: is **no greater** than 12
Predict the Minimum Delete-Relaxed Plan Length

- **Solution** (length=5):
  - (unstack E C)
  - (unstack C A)
  - (unstack B D)
  - (unstack D ground)
  - (stack D A)
Motion Planning
Robotics Terminology

- **Pose**: (position, orientation)
  - **Position**: [x, y, z]
  - **Orientation**: [roll, pitch, yaw]

- **Config(uration)**: robot degrees-of-freedom (DOFs)
  - **Base**: [x, y, yaw]
  - **Arm**: [joint\(_1\), …, joint\(_n\)]

- **Trajectory**: sequence of robot configurations
- **Grasp**: relative pose between gripper & object
- **Inverse kinematics**: find a config that reaches a pose

[Fig from Katharina Muelling]
Plan a path for a robot from an initial configuration to a goal configuration that avoids obstacles

- Sequence of continuous configurations
- Configurations often are high-dimensional
- Example: 7 DOFs

High-level approaches:
- Geometric decomposition
- Sampling-based
- Optimization-based
Configuration Space

- Reduce robot to a **point** moving through collision-free **configuration space** [Lozano-Pérez 1979]
- Obstacles are **inflated** by the robot’s geometry
- Example: configuration $q = (x, y, \theta)$

[Fig from Jyh-Ming Lien]
Sampling-Based Motion Planning

- **Discretize** configuration space by **sampling**
  - Sampling be deterministic or **random**
- **Implicitly** represent the collision-free configuration space using an blackbox **collision checker**
- **Abstracts away** complex robot geometry

- **Algorithms**
  - **Probabilistic Roadmap (PRM)**
  - Rapidly-Exploring Random Tree (RRT)
  - Bidirectional RRT (BiRRT)

[Kavraki 1994][Kuffner 2000][LaValle 2006]
Probabilistic Roadmap (1/7)

Find a path from init to goal that avoids the obstacles

[Fig from Erion Plaku]
Probabilistic Roadmap (2/7)

Sample a set of configurations
Probabilistic Roadmap (3/7)

Remove configurations that collide with the obstacles

[Fig from Erion Plaku]
Connect nearby configurations
Prune connections that collide with the obstacles
The resulting structure is a finite roadmap (graph)
Search for the shortest-path on the roadmap
Collision Checking is Expensive

- Collision checking **dominates** runtime
- **Complex geometries & fine resolutions** (for safety)
- Many edges clearly do not lie on a low-cost path
- **Optimistically** plan **without collisions**
- Check collisions **lazily** only by only evaluating candidate plans
Construct a PRM ignoring collisions
Lazy PRM (2/10)

Search for the shortest-path on the roadmap

[Fig from Erion Plaku]
Lazy PRM (3/10)

Remove plan edges that collide with obstacles

[Fig from Erion Plaku]
Search for the new shortest-path on the roadmap
Check the edges on the plan for collisions
Lazy PRM (6/10)

Check the edges on the plan for collisions (with increased resolution)

[Fig from Erion Plaku]
Lazy PRM (7/10)

Remove plan edges that collide with obstacles

[Fig from Erion Plaku]
Search for the new **shortest-path** on the roadmap
Lazy PRM (9/10)

Check the edges on the plan for collisions

[Fig from Erion Plaku]
Return the **current path** as a solution
Lazy Motion Planning

- **Defer** collision checking until a path is found
- **Remove** colliding edges path from the roadmap
- **Repeat** this process with a new path
- **Terminate** when a collision-free path is found

[Bohlin 2000][Dellin 2016]
Theoretical Properties

- Sampling-based algorithms cannot prove infeasibility nor even solve every feasible problem.
- Robustly feasible: a problem that admits a solution for which all local perturbations are also solutions.
- Probabilistic complete: an algorithm that solves any robustly feasible problem with probability 1.
Trajectory Optimization

- Frame motion planning as a **non-convex constrained optimization** problem & solve for **local minima**

  minimize $f(x)$
  
  subject to
  
  $g_i(x) \leq 0, \quad i = 1, 2, \ldots, n_{ineq}$
  
  $h_i(x) = 0, \quad i = 1, 2, \ldots, n_{eq}$

- Collision constraints enforced via **signed distance** $(sd)$

  ![Diagram showing signed distance](image)

  $sd > 0 \quad sd < 0$

  [Ratliff 2009][Schulman 2013]
Task and Motion Planning (TAMP)
Shakey the Robot (1969)

- **First autonomous mobile manipulator** (via pushing)
- Visibility graph, A* search, and STRIPS!
- **Decoupled** task and motion planning
- Task planning **then** motion planning

```
type(robot robot) type(object object)  
name(robot shakey) name(object box1) 
at(robot 4.1 7.2) at(object 3.1 5.2) 
theta(robot 90.1) inroom(object r1) 
shape(object wedge) radius(object 3.1)
```

GOTHRU(d,r1,r2)

**Precondition** INROOM(ROBOT,r1) ∧ CONNECTS(d,r1,r2)

**Delete List** INROOM(ROBOT,$)

**Add List** INROOM(ROBOT,r2)

[Nilsson 1984]

[Fikes 1971]
Obstacle Blocks Shakey’s Path

- What if a movable block *prevented* Shakey from safely moving into the adjacent room?
- Shakey could *push* it out of the way or *go around* it
- What’s more efficient? How to push it? …
Decoupled vs Integrated TAMP

- **Decoupled**: discrete (task) planning *then* continuous (motion) planning
- Requires a strong *downward refinement* assumption
- *Every* correct discrete plan can be *refined* into a correct continuous plan (from hierarchal planning)
- **Integrated**: *simultaneous* discrete & continuous planning
Geometric Constraints Affect Plan

- **Inherits challenges** of both motion & classical planning
  - **High-dimensional, continuous** state-spaces
  - State-space *exponential* in number of variables
  - **Long horizons**

- Continuous constraints limit high-level strategies
  - Kinematics, reachability, joint limits, collisions, grasp, visibility, stability, stiffness, torque limits, …
Pouring Among Obstacles
Preparing a Meal for Two
Breaking Down “Preparing a Meal”

- Clean 3 blue cups and clean/cook 2 green cabbages
- 64 continuous and 10 discrete variables

1. High-dimensional
2. Long horizon
3. Discrete state
4. Geometric constraints

■ Clean 3 blue cups and clean/cook 2 green cabbages
■ 64 continuous and 10 discrete variables

Remove obstructing radishes
Clean each cabbage
Replace the radishes
Cook each cabbage
Serve the cabbage
Robot forced to **regrasp** the object

- Change from a **top** grasp to a **side** grasp

- **Non-monotonic**

- Plan must **undo goals** to solve

- **Open then close** the drawer & cabinet door
Hybrid Planning Spectrum

Purely Discrete

Task Planning

Prediscretized Planning

Hybrid

Numeric Planning

Integrated Task and Motion Planning

Purely Continuous

Motion Planning

Multi-Modal Motion Planning
Prediscretized & Numeric Planning
Prediscretized Planning

- Assumes that a **finite set** of object placements, object grasps, and (sometimes) robot configurations are **given**
- Can **directly** perform discrete task planning
- Still need to evaluate **reachability**
  - Eagerly in **batch** [Lozano-Pérez 2014][Garrett 2017][Ferrer-Mestres 2017]
  - Eagerly during **search** [Dornhege 2009]
  - **Lazily** [Erdem 2011][Dantam 2018][Lo 2018]
Discrete-Control Numeric Planning

- Classical planning with **real-valued variables** and **durative actions**
  - **Examples**: time and energy
- Most planners only support **linear/polynomial dynamics**
- **Non-linear** dynamics addressed by **discretizing time**
- **Example**: battery domain

\[
\frac{d\delta}{dt} = \frac{i(t)}{c} - k'\delta
\]
\[
\frac{d\gamma}{dt} = -i(t)
\]

\[
\delta(t) = \frac{I}{c} \cdot \frac{1 - e^{-k't}}{k'}
\]
\[
\gamma(t) = C - It
\]

[Fox 2003][Hoffmann 2003][Eyerich 2009]
Continuous-Control Numeric Planning

- **Continuous control** parameters
- Tackle **convex dynamics** using cone programming
- Non-convexity handled by **partitioning** the state-space

- **In contrast,** TAMP is often:
  - High-dimensional
  - Non-convex
  - 3D collision constraints
  - Less sophisticated dynamically

[Deits 2015][Shoukry 2016]
[Fernandez-Gonzalez 2018]
Multi-Modal Motion Planning
Multi-Modal Motion Planning

- Collision-free configuration space changes when objects are manipulated
- Use a sequence of motion planning problems each defined by a mode
- **Mode**: a set of motion constraints
  - Gripper is empty
  - Relative object pose remains constant

Low-dimensional Intersections

- Need samples that connect adjacent modes
- Intersection of two modes is often low-dimensional
- Special-purpose samplers are needed
- Example: transition from gripper empty to holding
- Configurations at the intersection obtained using inverse kinematics (IK)

\[ \text{Hauser 2011} \]
1. Sample from the set of modes
2. Sample at the low-dimensional intersection of adjacent modes
3. Sample a roadmap within each mode
4. Discrete search on the multi-modal roadmap

[Hauser 2011]
Optimization-Based Multi-Modal Motion Planning

- Discrete search over sequences of **mode switches**
- Sequences have **varying length**, cannot compactly model as a Mixed Integer Program (MIP)
- Each sequence induces a **non-convex constrained optimization problem**
- Sequences can be pruned using **lower bounds** obtained by **relaxing** constraints

\[
\min_{x,a_{1:K},s_{1:K}} \int_0^T f_{\text{path}}(\dot{x}(t)) \, dt + f_{\text{goal}}(x(T))
\]

s.t.  
\[
x(0) = x_0, \quad h_{\text{goal}}(x(T)) = 0, \quad g_{\text{goal}}(x(T)) \leq 0, \\
\forall t \in [0,T]: \quad h_{\text{path}}(\dot{x}(t), s_{k(t)}) = 0, \\
\quad g_{\text{path}}(\dot{x}(t), s_{k(t)}) \leq 0 \\
\forall k \in \{1,\ldots,K\}: \quad h_{\text{switch}}(\dot{x}(t_k), a_k) = 0, \\
\quad g_{\text{switch}}(\dot{x}(t_k), a_k) \leq 0, \\
\quad s_k \in \text{succ}(s_{k-1}, a_k). 
\]
Integrated TAMP

- Geometric search **guided** by classical planning
  - Both heuristic and sampling guidance [Gravot 2005][Plaku 2010]
- Task and motion planning **interface**
  - Maintain **separate** discrete and continuous descriptions
- Custom interface to communicate between the two
- How are failures **diagnosed**?
  [Erdem 2011][De Silva 2013]
  [Srivastava 2014][Dantam 2018]

- Direct search in **combined** state-space
  [Kaelbling 2011] [Garrett 2018a] [Garrett 2018b]
Hybrid Planning Spectrum Revisited

Discrete
- Task Planning
  - Prediscretized Planning

Hybrid
- Numeric Planning
  - Integrated Task and Motion Planning

Continuous
- Motion Planning
  - Multi-Modal Motion Planning
Our Approach: STRIPStream

- No general-purpose, flexible framework for planning in a variety of TAMP domains
- Extends PDDL to incorporate sampling procedures
  - Can model domains with infinitely-many actions

- Develop domain-independent algorithms that treat the samplers as blackbox inputs
- Algorithms solve a sequence of finite PDDL problems
  - Leverage existing classical planners as subroutines
- Algorithms are particularly fast when downward refinement holds while remaining complete
STRIPStream Language
Benefits of Extending PDDL

- **Standardized** action description language
- Emphasis on describing and solving problems in a **domain-independent** way

- Large wealth of efficient, **existing algorithms** that exploit **factored** state & action structure

- Encodes the **difference** between two states using preconditions & effects
  - Most variables are **unchanged**
  - Actions can be described using **few parameters**
Solved Using the Same Algorithm

- Framework not specific to a single robot or robotics at all!
Motivating Pick & Place Example

- Single object prevents a goal object from being reachable
- Focus on a compact 2D version
- Formulation almost the same for 3D
- Algorithms agnostic to number of DOFs
2D Pick-and-Place Example

- **Goal**: block A within the red region
- Robot and block poses are continuous \((x, y)\) pairs
- Block B obstructs the placement of A
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
- move, pick B, move, place B,
  move, pick A, move, place A
Some constants are numpy arrays

Static initial facts - value is constant over time
- (Block, A), (Block, B), (Region, red), (Region, grey), (Conf, [-7.5 5.]), (Pose, A, [0. 0.]), (Pose, B, [7.5 0.]), (Grasp, A, [0. -2.5]), (Grasp, B, [0. -2.5])

Fluent initial facts - value changes over time
- (AtConf, [-7.5 5.]), (HandEmpty), (AtPose, A, [0. 0.]), (AtPose, B, [7.5 0.])

Goal formula: \((\exists p) \ (\text{and} \ (\text{Contained} \ A \ p \ \text{red}) \ (\text{AtPose} \ A \ p))\)
2D Pick-and-Place Actions

- Typical PDDL action description except that arguments are **high-dimensional & continuous**!
- To use the actions, must **prove** the following **static facts**:

  \[(\text{Motion } ?q1 \ ?t \ ?q2), \ (\text{Kin } ?b \ ?p \ ?g \ ?q)\]

```prolog
(:action move
 :parameters (?q1 ?t ?q2)
 :precondition (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
 :effect (and (AtConf ?q2) (not (AtConf ?q1))))

(:action pick
 :parameters (?b ?p ?g ?q)
 :precondition (and (Kin ?b ?p ?g ?q)
                    (AtConf ?q) (AtPose ?b ?p) (HandEmpty))
 :effect (and (AtGrasp ?b ?g)
              (not (AtPose ?b ?p)) (not (HandEmpty))))
```
BFS in Discretized State-Space

Suppose we were **given** the following additional static facts:

- \((\text{motion}, [-7.5, 5.], \tau_1, [0. 2.5]), (\text{motion}, [-7.5, 5.], \tau_2, [-5. 5.]), (\text{motion}, [-5. 5.], \tau_3, [0. 2.5]), (\text{kin}, A, [0. 0.], [0. -2.5], [0. 2.5]), \ldots\)

**Initial State**

- \((\text{move}, [-7.5, 5.], \tau_1, [0. 2.5])\)
- \((\text{move}, [-7.5, 5.], \tau_2, [-5. 5.])\)

**Diagram**

- \((\text{atconf}, [0. 2.5])\)
- \((\text{atpose}, A, [0. 0.])\)
- \((\text{atpose}, B, [7.5 0.])\)
- \((\text{handempty})\)

- \((\text{pick}, A, [0. 0.], [0. -2.5], [0. 2.5])\)
- \((\text{move}, [-5. 5.], \tau_3, [0. 2.5])\)

- \((\text{move}, [-7.5, 5.], \tau_1, [0. 2.5])\)
- \((\text{move}, [-7.5, 5.], \tau_2, [-5. 5.])\)
No a Priori Discretization

- **Values given at start:**
  - 1 initial configuration: (Conf, [-7.5 5.])
  - 2 initial poses: (Pose, A, [0. 0.]), (Pose, B, [7.5 0.])
  - 2 grasps: (Grasp, A, [0. -2.5]), (Grasp, B, [0. -2.5])

- **Planner needs to find:**
  - 1 pose within a region: (Contain A ?p red)
  - 1 collision-free pose: (CFree A ?p ? B ?p2)
  - 4 grasping configurations: (Kin ?b ?p ?g ?q)
  - 4 robot trajectories: (Motion ?q1 ?t ?q2)
What Samplers Do We Need?

- **Low-dimensional** placement stability constraint \((\text{Contain})\)
  - i.e. 1D manifold embedded in 2D pose space
  - Directly sample values that satisfy the constraint
  - May need arbitrarily many samples
  - Gradually enumerate an **infinite sequence**
Intersection of Constraints

- **Kinematic constraint** \( (\text{Kin}) \) involves poses, grasps, and configurations
- **Conditional samplers** - samplers with inputs

![Diagram of kinematic constraints involving poses, grasps, and configurations](image)
Composing Conditional Samplers

- Outputs of one conditional sampler are the inputs to another
- Directed acyclic graph (DAG) of conditional samplers
Stream: a function to a generator

- **Advantages**
  - Programmatic implementation
  - Compositional
  - Supports infinite sequences

- **Stream** - function from an input object tuple \((x_1, x_2, x_3)\) to a (potentially infinite) sequence of output object tuples \([(y_1, y_2), (y'_1, y'_2), \ldots]\)

```python
def stream(x1, x2, x3):  
    i = 0  
    while True:  
        y1 = i*(x1 + x2)  
        y2 = i*(x2 + x3)  
        yield (y1, y2)  
        i += 1
```

[Kaelbling 2011][Srivastava 2014][Garrett 2018a][Garrett 2018b]
Stream Certified Facts

- Objects alone aren’t helpful: what do they represent?
  - Communicate semantics using **predicates**!

- Augment stream specification with:
  - **Domain facts** - static facts declaring legal **inputs**
    - e.g. only configurations can be motion inputs
  - **Certified facts** - static facts that all **outputs** satisfy with their corresponding **inputs**
    - e.g. poses sampled from a region are within it
Sampling Contained Poses

\[
(:\text{stream} \ \text{sample-region} \\
  :\text{inputs} \ (\ ?b \ ?r) \\
  :\text{domain} \ \text{(and)} \ (\text{Block} \ ?b) \ (\text{Region} \ ?r)) \\
  :\text{outputs} \ (\ ?p) \\
  :\text{certified} \ \text{(and)} \ (\text{Pose} \ ?b \ ?p) \ (\text{Contain} \ ?b \ ?p \ ?r))
\]

```python
def sample_region(b, r):
    x_min, x_max = REGIONS[r]
    w = BLOCKS[b].width
    while True:
        x = random.uniform(x_min + w/2, x_max - w/2)
        p = np.array([x, 0.])
        yield (p,)
```

Block b

Region r

sample-region

Pose [(p), (p'), (p''), ...]
Sampling IK Solutions

- **Inverse kinematics** (IK) to produce robot grasping configuration
- Trivial in 2D, non-trivial in general (e.g. 7 DOF arm)

```
(stream sample-ik
  :inputs (?b ?p ?g)
  :domain (and (Pose ?b ?p) (Grasp ?b ?g))
  :outputs (?q)
  :certified (and (Conf ?q) (Kin ?b ?p ?g ?q)))
```
Calling a Motion Planner

- “Sample” (e.g. via a PRM) multi-waypoint trajectories
- Include joint limits & fixed obstacle collisions, but not movable object collisions

`(:stream sample-motion
 :inputs (?q1 ?q2)
 :domain (and (Conf ?q1) (Conf ?q2))
 :outputs (?t)
 :certified (and (Traj ?t) (Motion ?q1 ?t ?q2)))`
2D Place Collisions

- Add parameters for the pose of each block - bad!
- Use a derived predicate for whether currently unsafe
  - Predicate defined by logical formula
  - Enables lightweight logical inference
  - Decomposes collision checking into a logical AND

```prolog
(:action place
 :parameters (?b ?p ?g ?q)
 :precondition (and ... (not (UnsafePose ?b ?p)))
 :effect (and ...)

(:derived (UnsafePose ?b1 ?p1)
Check Block Collisions

- **Test stream:** stream without output objects
- Return True if **collision-free** placement (e.g., via querying a collision checker)

```
(:stream test-cfree
 :inputs (?b1 ?p1 ?b2 ?p2)
 :domain (and (Pose ?b1 ?p1) (Pose ?b2 ?p2))
 :outputs ()
 :certified (CFree ?b1 ?p1 ?b2 ?p2))
```

Test stream: stream without output objects

Return True if collision-free placement (e.g., via querying a collision checker)
STRIPStream = STRIPS + Streams

- **Domain dynamics (domain.pddl):** declares actions
- **Stream properties (stream.pddl):**
  - Declares stream inputs, outputs, and certified facts
- **Problem and stream implementation (problem.py):**
  - Initial state, Python constants, & goal formula
- **Stream implementation using Python generators**

User provides:
- Domain
- Streams
- Init & Goal

[Garrett 2018b]
STRIPStream Algorithms
Two STRIPStream Algorithms

- STRIPStream planners decide which streams to use
- Algorithms alternate between searching & sampling:
  1. Search a finite PDDL problem for plan
  2. Modify the PDDL problem (depending on the plan)
- Search implemented using off-the-shelf algorithms
- Off-the-shelf AI planner - FastDownward
  - Exploits factoring in its search heuristics (e.g. $h_{FF}$)
- Probabilistically complete given sufficient samplers

[Garrett 2018a]
[Garrett 2018b]
Incremental Algorithm

- Incrementally construct all possible initial facts
- Periodically check if a solution exists
- Repeat:
  1. **Compose** and **evaluate** a finite number of streams to unveil more facts in the initial state
  2. **Search** the current PDDL problem for plan
  3. **Terminate** when a plan is found

[Garrett 2018a]
[Garrett 2018b]
Incremental: Sampling Iteration 1

**Iteration 1** - 14 stream evaluations

- **Sampled:**
  - 2 new robot configurations:
  - 4 new block poses:
  - 2 new trajectories:
Incremental: Search Iteration 1

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient
Incremental: Sampling Iteration 2

**Iteration 2 - 54 stream evaluations**

- **Sampled:**
  - 4 new robot configurations:
  - 4 new block poses:
  - 10 new trajectories:
Incremental: Search Iteration 2

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient
Incremental Example: Iterations 3-4

**Iteration 3** - 118 stream evaluations

**Iteration 4** - 182 stream evaluations

**Solution:**

1) move [-7.5, 5.] [[-7.5, 5.], [-7.5, 5.], [7.5, 5.], [7.5, 2.5]] [7.5, 2.5]
2) pick B [7.5, 0.] [0., -2.5] [7.5, 2.5]
3) move [7.5, 2.5] [[7.5, 2.5], [7.5, 5.], [10.97, 5.], [10.97, 2.5]] [10.97, 2.5]
4) place B [10.97, 0.] [0., -2.5] [10.97, 2.5]
5) move [10.97, 2.5] [[10.97, 2.5], [10.97, 5.], [0.5], [0.25]] [0.25]
6) pick A [0.0] [0. -2.5] [0.25]
7) move [0.25] [[0.25], [0.5], [7.65, 5.], [7.65, 2.5]] [7.65, 2.5]
8) place A [7.65, 0.] [0. -2.5] [7.65, 2.5]

- **Drawback** - many unnecessary samples produced
- **Computationally expensive** to generate
- **Induces large discrete-planning problems**
Optimistic Stream Outputs

- Many TAMP streams are exceptionally **expensive**
- Inverse kinematics, motion planning, collision checking
- **Only** query streams that are **identified** as useful
- Plan with **optimistic hypothetical** outputs  [Srivastava 2014]
- Inductively create **unique placeholder** output objects for each stream instance (has # as its prefix)

Optimistic evaluations:

1. **s-region:**(b0, red)->(#p0)
2. **s-ik:**(b0, [0. 0.], [0. -2.5]) ->(#q0),
3. **s-ik:**(b0, #p0, [0. -2.5]) ->(#q2)

[Garrett 2018a]  [Garrett 2018b]
Focused Algorithm

- **Lazily** plan using optimistic outputs **before** real outputs
- **Recover** set of streams used by the optimistic plan
- Repeat:
  1. Construct active **optimistic** objects
  2. **Search** with real & **optimistic** objects
  3. If only real objects used, return plan
  4. **Sample** used streams
  5. **Disable** used streams

[Garrett 2018a][Garrett 2018b]
**Focused Example 1**

**Optimistic Plan:**

\[
\text{move}([-5. \ 5.], \#t0, \#q0), \ \text{pick}(A, [0. 0.], [-0. -2.5], \#q0), \\
\text{move}(\#q0, \#t2, \#q1), \ \text{place}(A, \#p0, [-0. -2.5], \#q1)
\]

**Constraints:**

\[
(\text{kin, } A, \#q0, \#p0, [-0. -2.5]), \\
(\text{kin, } A, \#q1, [0. 0.], [-0. -2.5]), \\
(\text{motion, } [-5. 5.], \#t1, \#q1), \\
(\text{motion, } \#q1, \#t2, \#q0), \\
(\text{contain, } A, \#p0, \text{red})
\]
Focused Example 2: Iteration 1

Optimistic Plan:
\begin{align*}
\text{move}([-5. \ 5.], \#t0, \#q0), & \ \text{pick}(A, [0. \ 0.], [-0. \ -2.5], \#q0), \\
\text{move}(\#q0, \#t2, \#q1), & \ \text{place}(A, \#p0, [-0. \ -2.5], \#q1)
\end{align*}

Constraints:
\begin{align*}
(\text{cfree}, A, \#p0, B, [7.5 \ 0.]), & \ (\text{contain}, A, \#p0, \text{red}), \\
(\text{kin}, A, \#q0, [0. \ 0.], [-0. \ -2.5]), & \ (\text{kin}, A, \#q1, \#p0, [-0. \ -2.5]), \\
(\text{motion}, \#q0, \#t2, \#q1), & \ (\text{motion}, [-5. \ 5.], \#t0, \#q0)
\end{align*}

Stream evaluations:
1. \text{s-region}:(A, \text{red})\rightarrow[[[8.21 \ 0.]]] 
2. \text{t-cfree}:(A, [8.21 \ 0.], B, [7.5 \ 0.]) = \text{False}

These stream instances are \textbf{removed} from subsequent searches.
Focused Example: Iteration 2

Optimistic Plan:

\text{move}([-5. \ 5.], \ #t4, \ #q2), \ \text{pick}(B, [7.5 \ 0.], [-0. \ -2.5], \ #q2),
\text{move}(#q2, \ #t9, \ #q3), \ \text{place}(B, \ #p1, [-0. \ -2.5], \ #q3),
\text{move}(#q3, \ #t6, \ #q0), \ \text{pick}(A, [0. \ 0.], [-0. \ -2.5], \ #q0),
\text{move}(#q0, \ #t8, \ #q4), \ \text{place}(A, [8.21 \ 0.], [-0. \ -2.5], \ #q4)

\text{t-cfree}:(A, [8.21 \ 0.], \ B, [7.5 \ 0.]) \ \text{previously failed}
\text{t-cfree}:(A, [8.21 \ 0.], \ B, \ #p1) \ \text{might succeed}

\text{s-region}(B, \ \text{grey})\rightarrow(#p1)

\text{t-cfree}:(B, \ #p1, \ A, [0. \ 0.])\rightarrow()

\text{t-cfree}:(A, [8.21 \ 0.], \ B, \ #p1)\rightarrow()

\text{s-ik}(B, [7.5 \ 0.], [-0. \ -2.5])\rightarrow(#q3)

\text{s-motion}([-5. \ 5.], \ #q3)\rightarrow(#t4)
Focused Outperforms Incremental

Incremental: ~20 s
Focused: ~10 s

Incremental: N/A
Focused: ~25 s

[Garrett 2018a]
Multi-Robot TAMP
Centralized Scheduling of Robots

- PDDL rovers domain with **visibility** and **reachability**
- How to plan for **simultaneous** execution?
- Use a **temporal planner** as search subroutine (e.g. Temporal FastDownward)

[Eyerich 2009]
Swap Initial Configurations
Temporal Task & Motion Planning

- Temporally annotated preconditions and effects
  - “at start”, “over all”, and “at end” (PDDL2.1) [Fox 2003]

(:durative-action move
 :parameters (?r ?q1 ?t ?q2)
 :duration (= ?duration (/ (Distance ?t) (Speed ?r))))

:condition (and
 (at start (Robot ?r))
 (at start (Motion ?q1 ?t ?q2))
 (at start (AtConf ?r ?q1))
 (over all (not (UnsafeTraj ?r ?t))))

:effect (and
 (at start (not (AtConf ?r ?q1)))
 (at start (OnTraj ?r ?t))
 (at end (not (OnTraj ?r ?t)))
 (at end (AtConf ?r ?q2))))
Robots might collide **during** the execution of their trajectories

- Planner doesn’t know exact position along trajectory

- Conservatively, **check all configuration pairs** per segment

(over all (not (UnsafeTraj ?r ?t)))

- **Derived predicate** evaluated at each time event

Swap with Rechargeable Battery
Robot movement **depletes battery charge** proportional to distance traversed

**Infinitely-many** possible move action instances

```prolog
(:durative-action move
 :parameters (?r ?q1 ?t ?q2)
 :duration (= ?duration (/ (Distance ?t) (Speed ?r)))

 :condition (and ...
 (at start (<= (* (ConsumeRate ?r) ?duration)
 (Energy ?r)))

 :effect (and ...
 (at start (decrease (Energy ?r)
 (* (ConsumeRate ?r) ?duration)))))
```
Robots can recharge battery via solar power

Can perform **3D robotic planning** while benefiting from state-of-the-art **numeric heuristics**

```
(:durative-action recharge
 :parameters (?r ?q)
 :duration (= ?duration (/ (- (Capacity ?r) (Energy ?r)) (RechargeRate ?r))))

:condition (...)

:effect (and ...
  (at end (increase (Energy ?r)
  (* (RechargeRate ?r) ?duration))))
```
TAMP Under Uncertainty
Probabilistic TAMP

- **Hybrid** Markov Decision Process (**MDP**)
  - Actions have *stochastic effects*
  - Agent might arrive at a state off its intended plan
  - Need a **policy** (mapping from each state to an action) instead of a **plan**
  - Computing an policy **offline** is intractable

- Synthesize policy **online** by **replanning**
- Action **determinization** approximation [Yoon 2007]
  - Planner deterministically selects the action outcome
  - Unlikely outcomes penalized by **high action costs**
Partially-Observable TAMP

- Hybrid Partially-Observable Markov Decision Process (POMDP) [Kaelbling 1998]
- Reduce POMDP to belief-state (distribution) MDP
- Belief distribution representation - Multivariate Gaussian, Discretized, Factored, Mixture, …

[Kaelbling 2013]

[Hadfield-Menell 2015]
**Goal**: high confidence that the green block is on the blue table

**Information gathering actions**: scan room & detect

- Robot arm must not obstruct observations
Belief-Space TAMP System

- Convolutional Neural Network (CNN) Object Detector
- Point cloud **plane estimation** to identify surfaces
- Point cloud **pose estimation** for objects
- **Occupancy grid** for non-manipulable
- Plan, execute, & observe in **real time**
Takeaways

- **Task and Motion Planning (TAMP):** hybrid planning where continuous constraints affect discrete decisions
- **Sampling** is powerful for exploring continuous spaces
- **STRIPStream:** planning language that supports sampling procedures as blackbox streams
- **Domain-independent algorithms**
- **Lazy/optimistic** planning intelligently queries only a small number of samplers (focused algorithm)
  - [github.com/caelan/pddlstream](https://github.com/caelan/pddlstream)
- Ongoing work involving cost-sensitive, multi-agent, probabilistic & partially observable TAMP
Questions? (and Outtakes!)
Task Planning

Motion Planning

Prediscretized Planning


Numeric Planning


Multi-Modal Motion Planning


Task and Motion Planning


Probabilistic & Partially-Observable


