STRIPS PLANNING IN INFINITE DOMAINS

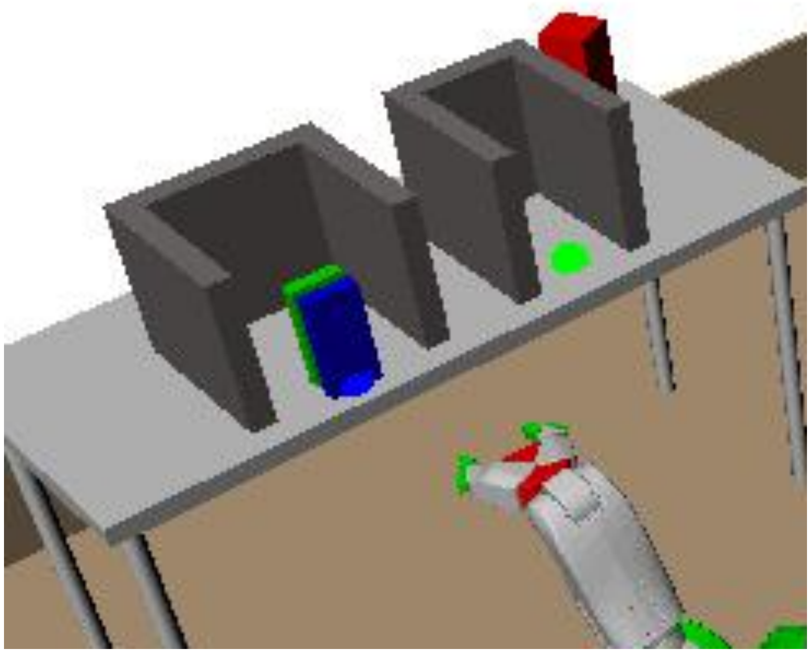
Caelan Garrett, Tomás Lozano-Pérez, and Leslie Kaelbling
MIT CSAIL - ICRA PlanRob 2017

Task and Motion Planning

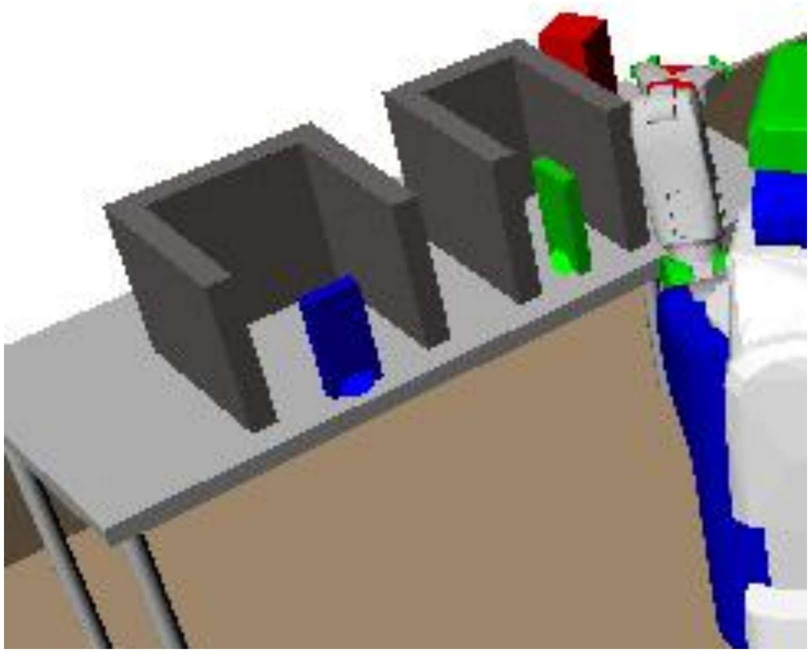
- **Task planning (AI planning)**
  - Discrete actions - pick, place, …
- **Motion planning**
  - Robot movements - trajectories
- **Continuous variables**
  - Poses, grasps, configurations trajectories
- **Geometric constraints affect plan feasibility**
  - Kinematic, motion, collision, …

Geometric Constraints Affect Plan

- **Goal conditions**
  - Green block on green dot
  - Blue block on blue dot (its initial pose)



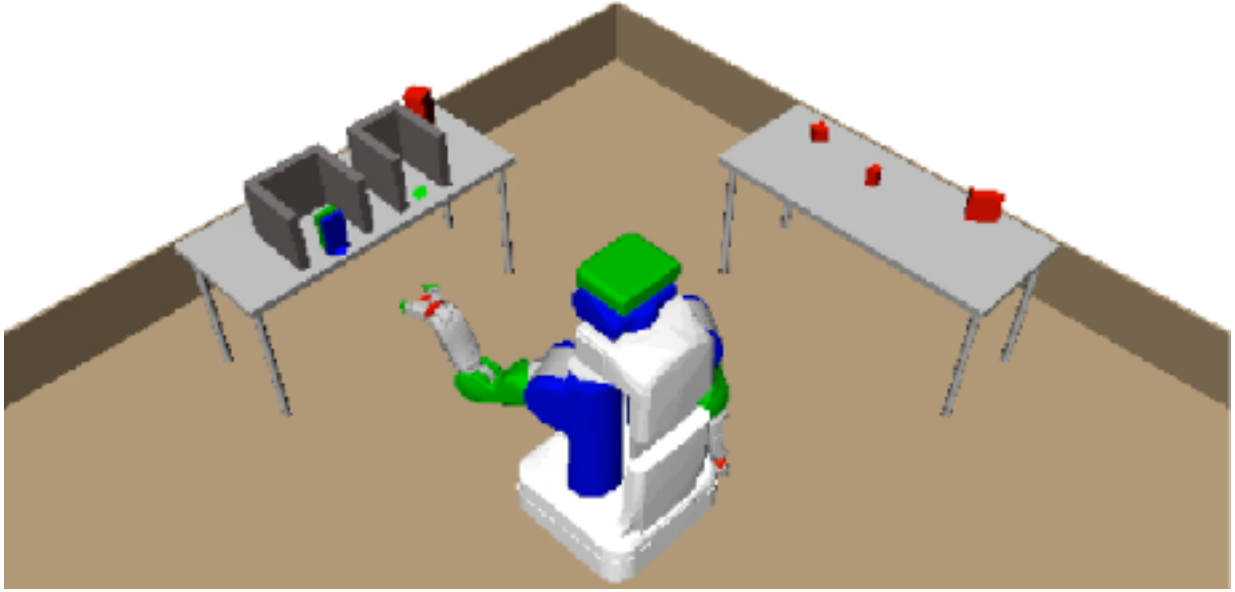
Initial state



Goal state

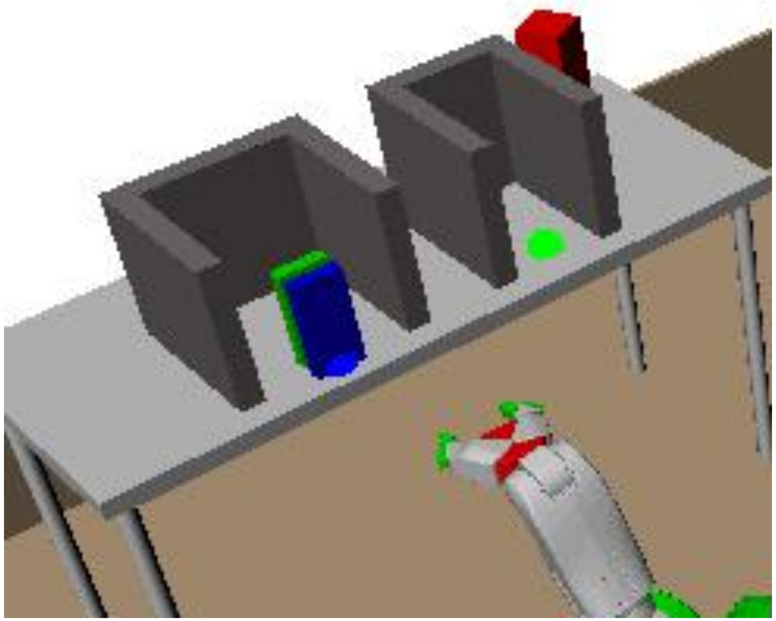
Geometric Constraints Affect Plan

- **Solutions must**
  - Move blue block out of the way
  - Regrasp the green block to change grasps
  - Return the blue block to its initial pose



No a Priori Discretization

- **Values given at start**
  - 1 initial configuration
  - 2 initial poses
  - 2 goal poses
- **Planner needs to find**
  - 1 additional pose
  - 3 grasps
  - 8 grasp configurations
  - 8 manipulator trajectories
  - 8 base trajectories



We Introduce STRIPStream

- Extend STRIPS for **integrated specification** of problems with discrete and continuous variables
- Takes advantage of
  - **Factored representations** and **efficient search**
  - **Dynamic sampling** of continuous spaces
- **Domain-independent**

- Two **probabilistically complete** algorithms that reduce planning to a **sequence of finite problems**
- **Software** - https://github.com/caelan/stripstream

Prior Work

- **Task and Motion Planning**
  - Cambon et al., Dornhege et al., Plaku & Hager, Erdem et al., Kaelbling & Lozano-Perez, Lagriffoul et al., Pandey et al., de Silva et al., Srivastava et al., Garrett et al., Toussaint, Dantam et al., …
  - STRIPStream generalizes themes in these approaches
- **PDDL+** [Fox & Long]
  - Extends PDDL+ to incorporate continuous variables dependent on time
  - Algorithms limited to simple dynamics models

Pick-and-Place STRIPS Actions

- **Discrete** - block (B)
- **Continuous** - pose (P), grasp (G) configuration (Q), trajectory (T)
- Given a sufficient set of samples, we could do STRIPS

- Pick(B, P, G, Q, T):
  - **static**: {*IsBlock*(B), *IsPose*(P), *IsGrasp*(P), *IsConf*(Q), *IsTraj*(Q), <u>*IsKin*(P, G, Q, T)</u>}
  - **pre**: {*AtPose*(B, P), *HandEmpty*(), *AtConf*(Q), *Safe*(b1, B, G, T), …, *Safe*(b10, B, G, T)}
  - **eff**: {*Holding*(B), **not** *AtPose*(B, P), **not** *HandEmpty*()}

Pick-and-Place STRIPS Axioms

- Similar actions for Place and Move

- Use axioms to evaluate *Safe*(B2, B, G, T)
  - **Factors collision checking**

- SafeAxiom(B2, P2, B, G, T):
  - **static**: {*IsBlock*(B), *IsPose*(P2), *IsBlock*(B), *IsGrasp*(G), *IsTraj*(G), <u>*IsCollisionFree*</u>(B2, P2, B, G, T)}
  - **pre**: {*AtPose*(B2, P2)} **or** {Holding(B2)}
  - **eff**: {*Safe*(B2, B, G, T)}

Need to Produce Samples

- How do we…
  - Obtain poses, grasps, configurations, trajectories?
  - Evaluate *IsCollisionFree*(B2, P2, B, G, T)?
  - Produce values that satisfy *IsKin*(P, G, Q, T)?

- Want to avoid producing many unnecessary samples
- Extend STRIPS to give capability of dynamically generating samples

Streams

- **Generator**
  - Finite or infinite sequence of values
  - Specified by a blackbox procedure (e.g. Python)

- **Stream(Y1, …, Yn | X1, …, Xm)**
  - **Inputs** X1, …, Xm, **outputs** Y1, …, Yn
  - **Conditional generator (gen)**
    - Function from x1, …, xm to generator producing y1, …, yn
  - Input (**inp**) /output (**out**) static atoms certify facts

Streams as Samplers

- PoseStream(P | ())
  - **gen: lambda:** (sample-pose() **for** i in **range**(0, **Inf**))
  - **inp:** {}
  - **out:** {*IsPose*(P)}

- No inputs, pose output
- sample-pose randomly samples a stable object pose

- All procedures (e.g. sample-pose) are Python functions using the OpenRAVE robotics simulator

Streams as Tests

- CollisionFreeStream(() | B2, P2, B, G, T)
  - **gen**: **lambda** b2, p2, b, g, t: [()] **if not any**(<u>collision</u>(b2, p2, b, g, q) **for** q **in** t) **else** []
  - **inp**: {*IsBlock*(B2), *IsPose*(P2), *IsBlock*(B), *IsGrasp*(G), IsTraj(T)}
  - **out**: {*IsCollisionFree*(B2, P2, B, G, T)}

- Several inputs, no outputs
- Certifies B1, P1 is not in collision with B2, G, T
- <u>collision</u> calls a collision checker

Streams as Conditional Samplers

- KinStream(Q, T | P, G)
  - **gen: lambda** p, g: (<u>sample-manipulation</u>(p*g^-1) **for** i in **range**(0, **Inf**))
  - **inp:** {*IsPose*(P), *IsGrasp*(G)}
  - **out:** {*IsConf*(Q), *IsTraj*(T), *IsKin*(P, G, Q, T)}

- Pose & grasp inputs, configuration & trajectory outputs
- <u>sample-manipulation</u> uses an inverse kinematic solver and a motion planner

Mobile Manipulation in STRIPStream

```
CONF, BLOCK, POSE, GRASP, TRAJ = Type(), Type(), Type(), Type(), Type()

AtConfig = Pred(CONF)
HandEmpty = Pred()
AtPose = Pred(BLOCK, POSE)
Holding = Pred(BLOCK, GRASP)
Safe = Pred(BLOCK, TRAJ)
IsPose = Pred(BLOCK, POSE)
IsGrasp = Pred(BLOCK, GRASP)
IsKin = Pred(BLOCK, POSE, GRASP, CONF, TRAJ)
IsCollisionFree = Pred(BLOCK, POSE, TRAJ)

O, P, G, Q, T = Param(BLOCK), Param(POSE), Param(GRASP), Param(CONF), Param(TRAJ)
Q1, Q2, OB = Param(CONF), Param(CONF), Param(BLOCK)

actions = [
    Action(name='pick', parameters=[O, P, G, Q, T],
      condition=And(AtPose(O, P), HandEmpty(),
        IsKin(O, P, G, Q, T), AtConfig(Q),
        ForAll([OB], Or(Equal(O, OB), Safe(OB, T)))),
      effect=And(AtPose(O, None), Holding(O, G),
        Not(HandEmpty()), Not(AtPose(O, P)))),
    Action(name='place', parameters=[O, P, G, Q, T],
      condition=And(AtPose(O, None), Holding(O, G),
        IsKin(O, P, G, Q, T), AtConfig(Q),
        ForAll([OB], Or(Equal(O, OB), Safe(OB, T)))),
      effect=And(AtPose(O, P), HandEmpty(),
        Not(AtPose(O, None)), Not(Holding(O, G))))]
```

Mobile Manipulation in STRIPStream

```
actions += [
    Action(name='move', parameters=[Q1, Q2],
      condition=AtConfig(Q1),
      effect=And(AtConfig(Q2), Not(AtConfig(Q1))))]

axioms = [
    Axiom(effect=Safe(O, T),
          condition=Exists([P], And(AtPose(O, P), IsCollisionFree(O, P, T))))]

cond_streams = [
    GenStream(inputs=[O], outputs=[P], conditions=[], effects=[IsPose(O, P)],
              generator=sample_poses),
    GenStream(inputs=[O], outputs=[G], conditions=[], effects=[IsGrasp(O, G)],
             generator=sample_grasps),
    GenStream(inputs=[O, P, G], outputs=[Q, T], conditions=[IsPose(O, P), IsGrasp(O, G)],
             effects=[IsKin(O, P, G, Q, T)], generator=sample_motion),
    TestStream(inputs=[O, P, T], conditions=[IsPose(O, P)],
              effects=[IsCollisionFree(O, P, T)], test=collision_free)]

constants = []
initial_atoms = [AtConfig(initial_config), HandEmpty()]
for obj, pose in initial_poses.iteritems():
  initial_atoms += [AtPose(obj, pose), IsPose(obj, pose)]
goal_literals = []
for obj, pose in problem.goal_poses.iteritems():
  goal_literals.append(AtPose(obj, pose))
  initial_atoms.append(IsPose(obj, pose))
return STRIPStreamProblem(initial_atoms, goal_literals, actions + axioms, streams, constants)
```

STRIPStream Algorithms

- Propose two algorithms - **incremental** & **focused**
- Both reduce STRIPStream planning to a sequence of finite problems
- Any **search subroutine** can be used to solve each finite problem
  - We compile to PDDL and use FastDownward
- Given the approapriate streams, both are **probabilistically complete**

Incremental Algorithm

- Call **K** streams and check if a solution exists
- Generalizes a Probabilistic Roadmap (PRM) from motion planning
- Unnecessarily calls many streams and produces many gratuitous samples
- Use AI actions to determine useful streams

Focused Algorithm

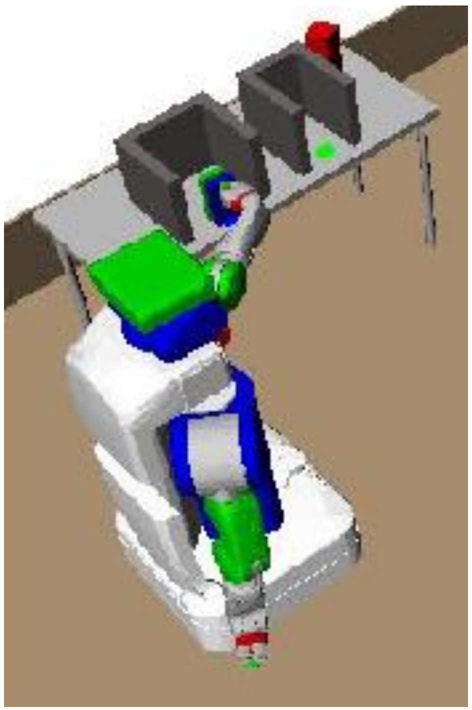
- Plan with **abstract values** before concrete values

- Create **optimistic** finite domain problem that **mixes abstract values and existing samples**
- After finding plan, call associated streams
- Terminate after finding a plan with no abstract objects

- Generalizes lazy PRM except it lazily produce entires samples
- Solves on first iteration if all streams succeed

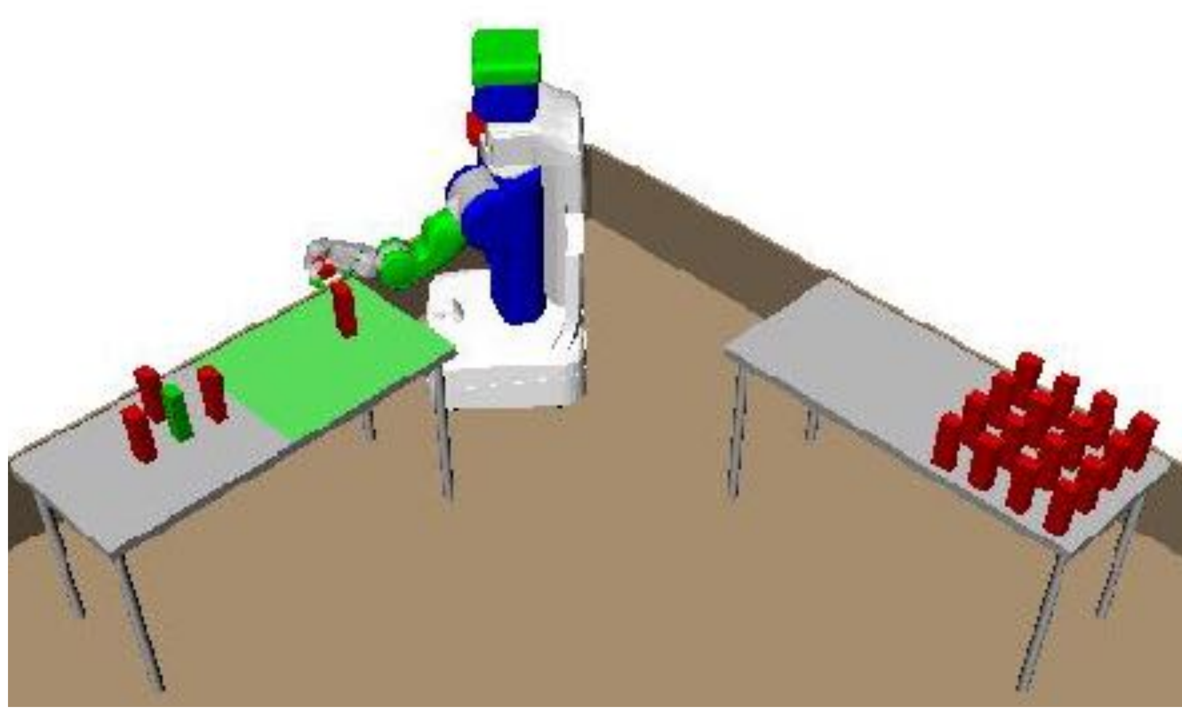
Incremental vs Focused Experiments

Problem 1



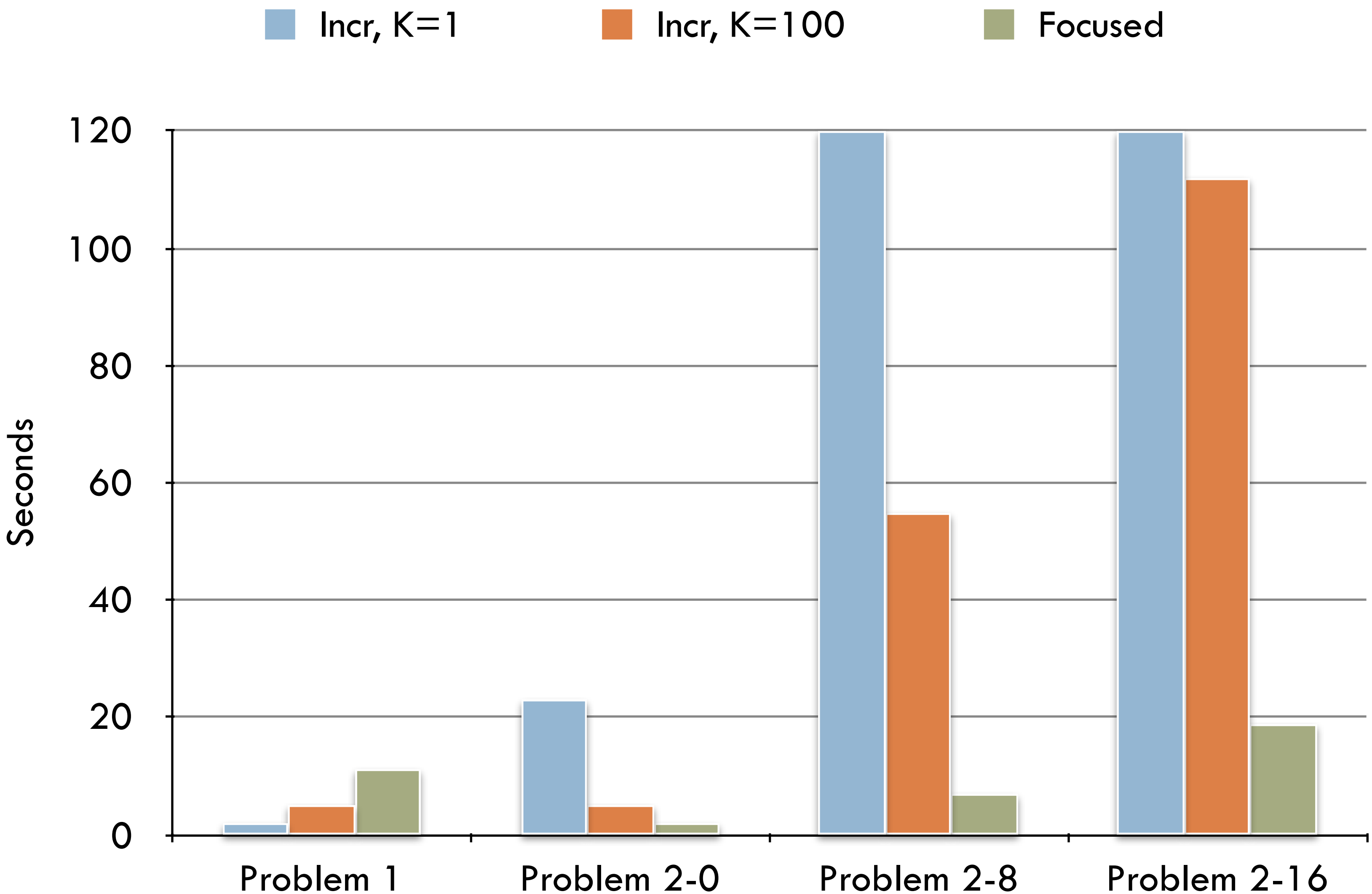
Regrasp required

Problem 2-16

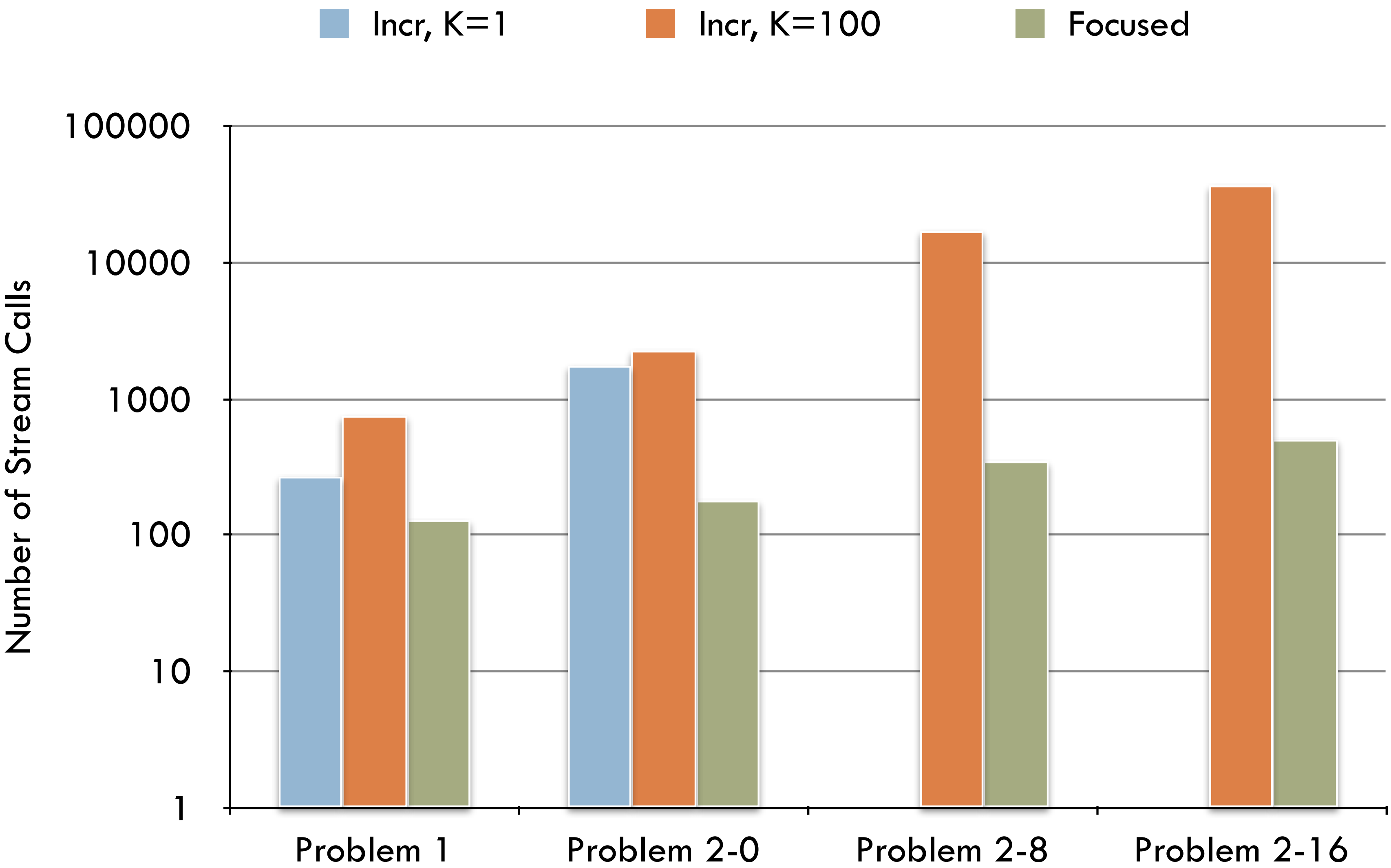


Many distracting objects

Average Total Runtime (120s Timeout)



Stream Calls



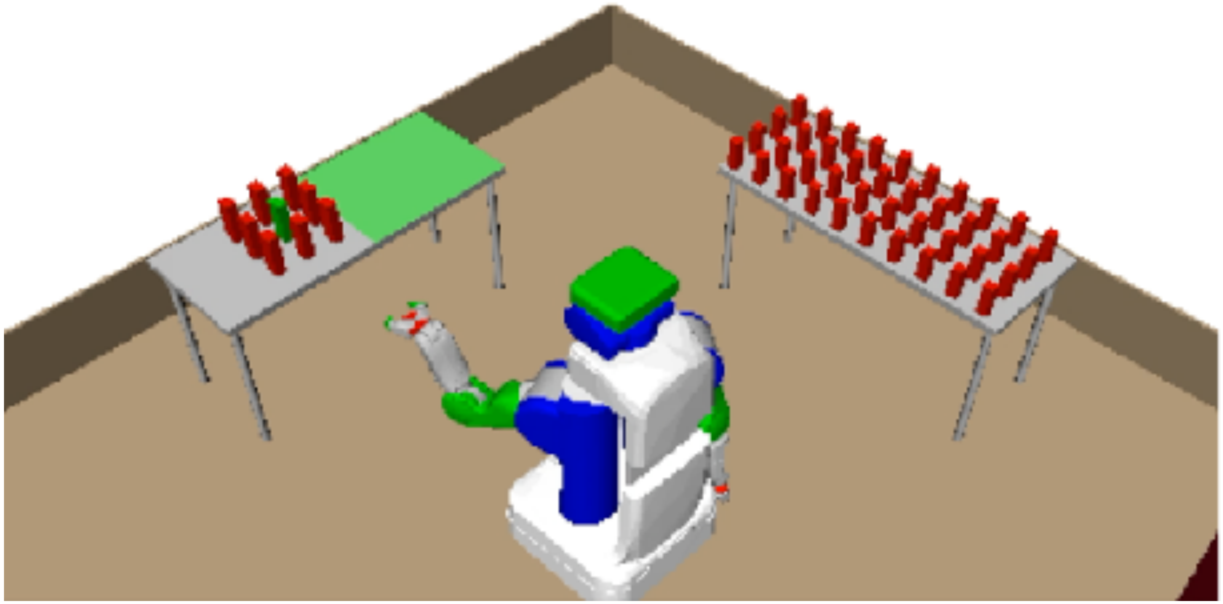
Takeaways

- **STRIPStream = STRIPS + Streams**
  - Streams implement samplers, tests, and conditional samplers
  - Domain-independent
  - Can model task and motion planning domains

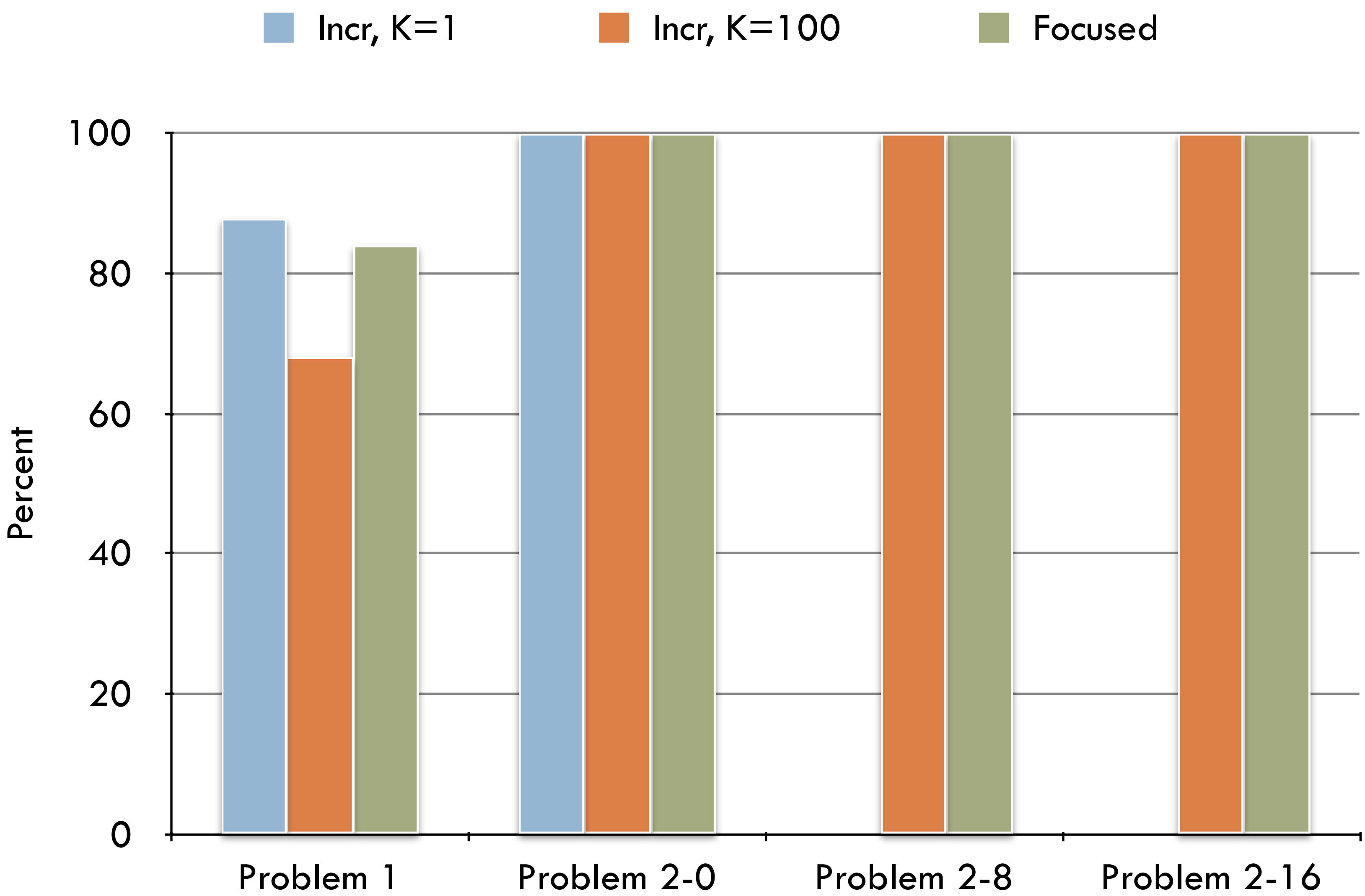
- Focused algorithm able to avoid producing many unnecessary samples
  - Probabilistically complete

Any Questions?

- STRIPStream implementation and examples
  - https://github.com/caelan/stripstream
- Contact information
  - caelan@csail.mit.edu
- Thank you!



Success Rate (25 Trials)



Full Experimental Results

II	increm. $K = 1$			increm. $K = 100$			focused		
	%	t	c	%	t	c	%	t	c
1	88	2	268	68	5	751	84	11	129
2-0	100	23	1757	100	9	2270	100	2	180
2-8	0	-	-	100	55	17217	100	7	352
2-16	0	-	-	100	112	36580	100	19	506

- 4 problems
- 25 trials per algorithm and problem
- Timeout of 120 seconds
- Python implementation uses OpenRAVE