caelan / **pddlstream** Public

PDDLStream: Integrating Symbolic Planners and Blackbox Samplers

arxiv.org/abs/1802.08705

pddlstream

PDDLStream is a planning framework comprised of an action language and suite of algorithms for Artificial Intelligence (AI) planning in the presence of sampling procedures. PDDLStream extends Planning Domain Definition Language (PDDL) by introducing streams, declarative specifications of sampling procedures. PDDLStream algorithms are domain independent and solve PDDLStream problems with only a blackbox description of each sampler. The motivating application of PDDLStream was for general-purpose robot Task and Motion Planning (TAMP).

The default **pddlstream** branch (main) is the newest stable "release" of **pddlstream**. The downward **pddlstream** branch is the most recent and advanced version of **pddlstream** but also is somewhat experimental.

Publications

- PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning

Citation

Caelan R. Garrett, Tomás Lozano-Pérez, Leslie P. Kaelbling. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning, International Conference on Automated Planning and Scheduling (ICAPS), 2020.

Contact

Caelan Garrett: [username]@mit.edu

History

PDDLStream is the "third version" of the PDDLStream/STRIPStream planning framework, intended to supersede previous versions:

1. https://github.com/caelan/stripstream
2. https://github.com/caelan/ss

PDDLStream makes several representational and algorithmic improvements over these versions. Most notably, it adheres to PDDL conventions and syntax whenever possible and contains several new algorithms.

Installation

```
$ git clone --recursive --branch main git@github.com:caelan/pddlstream.git
$ cd pddlstream
pddlstream$ git submodule update --init --recursive
pddlstream$ ./downward/build.py
```

If necessary, see FastDownward's documentation for more detailed installation instructions.

PDDLStream actively supports python2.7 as well as the most recent version of python3.

Make sure to recursively update **pddlstream**'s submodules when pulling new commits.

```
pddlstream$ git pull --recurse-submodules
```

Examples

This repository contains several robotic and non-robotic PDDLStream example domains.

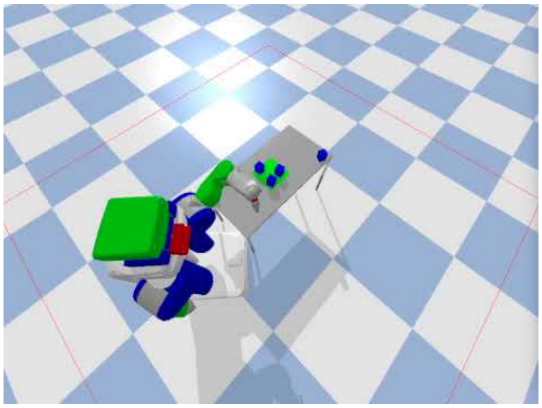
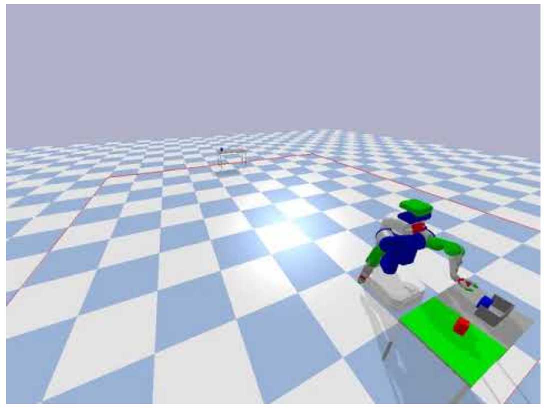
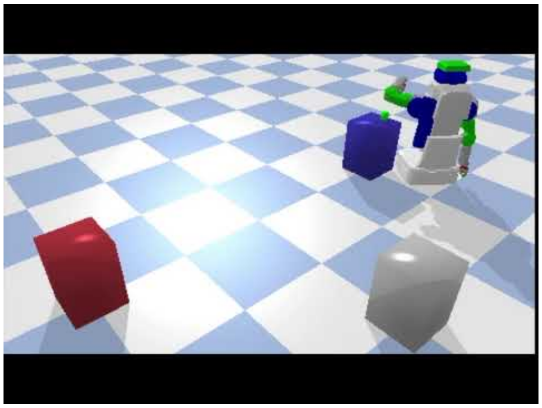
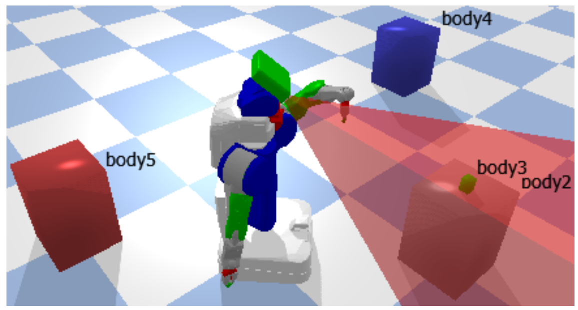
PyBullet

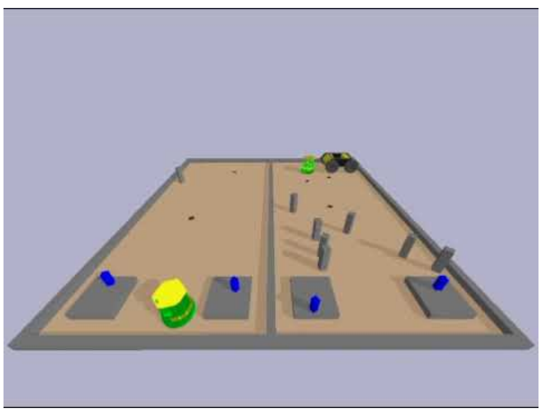
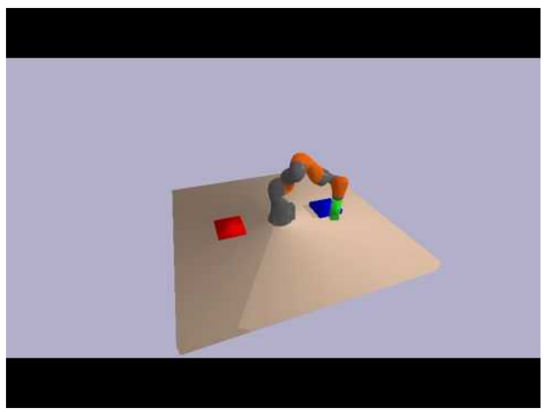
Install PyBullet on OS X or Linux using:

```
$ pip install pybullet numpy scipy
```

Examples:

- PR2 TAMP: `pddlstream$ python -m examples.pybullet.tamp.run`
- PR2 Cleaning and Cooking: `pddlstream$ python -m examples.pybullet.pr2.run`
- Turtlebot Rovers: `pddlstream$ python -m examples.pybullet.turtlebot_rovers.run`
- PR2 Rovers: `pddlstream$ python -m examples.pybullet.pr2_rovers.run`
- PR2 Planning and Execution: `pddlstream$ python -m examples.pybullet.pr2_belief.run`
- Kuka Cleaning and Cooking: `pddlstream$ python -m examples.pybullet.kuka.run`

See https://github.com/caelan/pybullet-planning for more information about my PyBullet planning primitives library.

Python TKinter

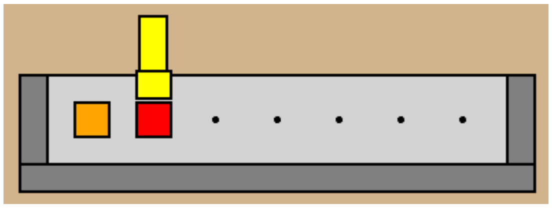
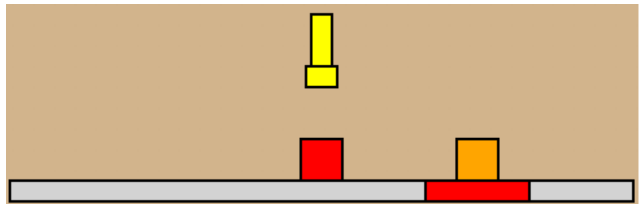
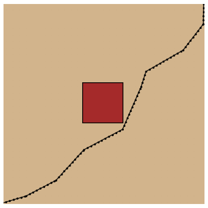
Install numpy and Python TKinter on Linux using:

```
$ sudo apt-get install python-tk
$ pip install numpy
```

Examples:

- 1D Continuous TAMP: `pddlstream$ python -m examples.continuous_tamp.run`
- 2D Motion Planning: `pddlstream$ python -m examples.motion.run`

- Discrete TAMP: `pddlstream$ python -m examples.discrete_tamp.run`
- Discrete TAMP with pushing: `pddlstream$ python -m examples.discrete_tamp.run`

Pure Python

Simple examples that can be run without additional dependencies:

- Blocksworld: `pddlstream$ python -m examples.blocksworld.run`
- Blocksworld with Derived Predicates: `pddlstream$ python -m examples.blocksworld.run_derived`
- Kitchen (debug streams): `pddlstream$ python -m examples.kitchen.run`

Advanced Functionality

Test cases or advanced (and undocumented) functionality:

- Action Description Language (ADL): `pddlstream$ python -m examples.advanced.adl.run`
- Deferred streams (postponed evaluation): `pddlstream$ python -m examples.advanced.defer.run`
- Exogenous streams (observations): `pddlstream$ python -m examples.advanced.exogenous.run`
- Fluent streams (state constraints): `pddlstream$ python -m examples.advanced.fluent.run`
- Constraint satisfaction: `pddlstream$ python -m examples.advanced.satisfy.run`
- Wild streams (ad hoc certification): `pddlstream$ python -m examples.advanced.wild.run`

International Planning Competition (IPC)

Unmodified PDDL IPC examples solved using PDDLStream's modified translator:

- Rovers: `pddlstream$ python -m examples.ipc.rovers.run`
- Satellites: `pddlstream$ python -m examples.ipc.satellites.run`

Applications

External projects that make use of PDDLStream:

- Online TAMP - https://github.com/caelan/SS-Replan
- Automated Construction - https://github.com/caelan/pb-construction
- Learning + TAMP (LTAMP) - https://github.com/caelan/LTAMP

Algorithms

PDDLStream is a planning framework comprised of a **single** planning language but **multiple** planning algorithms. Some of the algorithms are radically different than others (e.g. Incremental vs Focused) and thus the planning time can also substantially vary. The **Adaptive** algorithm typically performs best for domains with many possible sampling pathways, such as robot manipulation domains.

The meta procedure solve(...) allows the user to toggle between avaliable algorithms using

Property descriptions:

- **Method**: the python function that calls the algorithm

- **Negated streams**: whether the algorithm supports inverting test streams
- **Fluent streams**: whether the algorithm supports fluent streams that additionally condition on the fluent state
- **Wild streams**: whether the algorithm supports streams that additionally can certify ad hoc facts

Adaptive

- **Method**: solve_adaptive(...)
- **Negated streams**: supported
- **Fluent streams**: supported
- **Wild streams**: supported

Binding

- **Method**: solve_binding(...)
- **Negated streams**: supported
- **Fluent streams**: supported
- **Wild streams**: supported

Focused

- **Method**: solve_focused_original(...)
- **Negated streams**: supported
- **Fluent streams**: supported
- **Wild streams**: supported

Incremental

- **Method**: solve_incremental(...)
- **Negated streams**: not supported
- **Fluent streams**: not supported
- **Wild streams**: supported

Search Subroutines

Many (but not all) **pddlstream** algorithms have a discrete planning phase that can be implemented using any finite state-space search algorithm, such as Breadth-First Search (BFS) and Uniform-Cost Search (UCS). However, because **pddlstream** extends PDDL, this planning phase can also be implemented by state-of-the-art classical planning algorithms, which leverage the factored structure of action languages such as PDDL to vastly improve empirical planning efficiency. Best-first heuristic search algorithms, which automatically derive heursitics in a *domain-independent* manner, are one example class of these algorithms.

FastDownward

pddlstream comes pre-packaged with FastDownward, a prolific library that contains many best-first heuristic search PDDL planning algorithms. I've preconfigured a small number of effective and general search algorithms in SEARCH_OPTIONS, which can be toggled using the keyword argument planner=?. I've roughly ranked them in order of least lazy (lowest cost) to most lazy (lowest runtime):

- Dijkstra/Uniform-Cost Search (UCS) - optimal but slowest
- hmax A* - optimal but slow

- lmcut A* - optimal but slow
- ff eager astar
- ff eager {1,...,5} weighted astar - recommended (w=2)
- ff lazy {1,...,5} weighted astar
- ff eager greedy
- ff lazy greedy

The runtime of the discrete planning phase varies depending on the selected search algorithm. For many non-adversarial problems, these algorithms will either solve a problem instantenously or, if they aren't greedy enough, not terminate within 10 minutes. I recommend starting with a greedier configuration and moving torward a less greedy one if desired.

Other PDDL Planners

Any PDDL planning algorithm could be used in the place of FastDownward; however, a caveat is that some of these planners are only implemented to support a limited set of representational features (e.g. no conditional effects, no derived predicates, etc.), which can make both modeling more difficult and ultimately planning less efficient in many real-world (non-IPC) planning domains. While I heavily recommend FastDownward, some PDDL planners that I've interfaced with in the past with some success include:

Classical Planners

- FastDownward
- FastForward
- pyplanners
- Cerberus
- LAPTK
- YSHAP

Numeric Planners:

- Metric FF
- SMTPlan

Temporal Planners:

- Temporal FastDownward
- TPSHE

Diverse Planners:

- Forbid Iterative
- kstar
- symk

Resources

- Recent Talk
- Recent Overview
- Recent Tutorial
- PDDLStream Tutorial
- The AI Planning & PDDL Wiki
- Planning Domain Definition Language (PDDL)
- Derived Predicates
- FastDownward

Retired

"Retired" folders indicate code that no longer is continuously supported and thus is likely outdated.

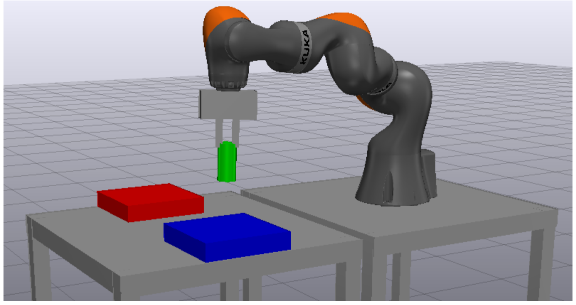
Drake

Install Drake on OS X or Ubuntu by following the following instructions: http://drake.mit.edu/installation.html.

Alternatively, install Drake through docker by following the following instructions: http://manipulation.csail.mit.edu/install_drake_docker.html. Use the appropriate docker_run_bash script with docker tag drake-20181128.

Examples:

- Kuka IIWA task and motion planning: `~/pddlstream$ python -m examples.drake.run`



Additional PDDLStream + Drake examples can be found at: https://github.com/RobotLocomotion/6-881-examples.

Releases

No releases published

Packages

No packages published

Contributors 3

caelan Caelan Garrett

cpaxton Chris Paxton

aidan-curtis Aidan Curtis

Languages

- **Python** 100.0%