# Иерархическое планирование поведения

Подоляко Илья Александрович БПМИ-176

## Версии PDDL и их различия.

В данной части рассмотрим некоторые отличия PDDL разных версий. Стоит сказать, что главными особенностями была поддержка языка для STRIPS-планировщика(не языка). Т.е. формально это попытка объединить все хорошие особенности предыдущих языков(как я понял, ADL и STRIPS-язык) и создать единый формат, который бы был хотя бы совместим со STRIPS-планировщиком.(в связи с тем, что STRIPS-язык не поддерживает условыне эффекты были бы проблемы, но PDDL имеет декларацию о условных эффектах и мы сразу понимаем о его совместимости). Также многие планировщики имеют дополнительную аннотацию называемую советами(advice - подсказка куда что использовать и др), но так как это загромождает описание и обязательно не для каждого планировщика, то создатели PDDL решили свести это к минимуму и поэтому создана отдельная синтаксическая конструкция. Девиз PDDL - physics, not advice, что означает, что приоритет отдан физике(Домену), описывающей предикаты действий, состояний и тд.
Сразу оговорим особенности, чтобы не писать их в таблице первой версии(PDDL 1.2), а описать в таблице непосредственно изменения следующих версий.
Во-первых, PDDL описывает домены планирования(физика наших задач). Домен декларирует типы объектов в нашей системе, виды отношений между объектами и их свойства, константы системы, схемы действий и схемы аксиом.
Во-вторых, PDDL декларирует задачу планирования(task). Это разделение удобно тем, что для одной и той же системы всегда можно применить разные таски, без введения данных о мире заново. В задаче декларируют принадлежность к домену, множество объектов системы, начальное состояние системы и конечное состояние.
По мимо этого PDDL ещё разрешает декларировать следующие конструкции:
1) Флюент; (Флюента (fluent) — это функция, областью определения которой является множество всех возможных ситуаций.)
2) Вычисляемых выражений;
3) Условных эффектов действий;
4) Иерархических структур действий (для иерархических планировщиков);
5) Ограничений сохранности (Своего рода дополнительных целей, которые должны быть истинны в целевом состоянии, если они были истины в начальном состоянии. Если ограничение сохранности нарушается где-то в процессе планирования, то оно должно быть восстановлено в дальнейшем.);
6) Ограничений на длину целевого плана.

| PDDL 2.1 | PDDL 2.2 | PDDL 3.0 |
|---|---|---|
| 1) Поддержка старых версий (При условии корректности там) 2) Работа метрик для оценки качества плана 3) Доменные функции - позволяют работать с числовыми единицами(переливание воды) 4) Параллелизм действий - самолёт летит, и теряет топливо одновременно 5) Убраны аксиомы | 1) Выводимые литералы(derived predicates) (по сути аксиомы, т.е. если некий p имеет key и key подходит к safe,y то p может открыть safe.) 2) Отложеные начальные литералы (timed initial literals) - средство для декларации внешних событий, которые начнутся по истечении некоторого времени, известного планировщику. | 1) Появились средства для описания дополнительных ограничений (допустимость или присутствие тех или иных действие в плане) 2)Описание предпочтений - желательные цели и ограничения. Они не обязательны в исполнении но с помощью них появляется возможность качественно оценить результат. |

Наиболее точное описание синтаксиса и особенностей лежит здесь(честно взято) тык.

## Другие способы описания.

Первым стоит вспомнить обычный STRIPS язык, который не совсем является языком. Формально это просто вошло в следующие года как обозначение формальных языков, но у него так же имеется некоторое определение и описание задач планирования.
Второй это ADL, который является фактически предком PDDL. Особенность является в том, что по сути был внедрён принцип открытости мира, в отличии от того же STRIPS'a. Также появляется поддержка отрицательных предикатов, незивестные предикаты имеют неизвестное состояние, а не отрицательное и тд. Всё в итоге ведёт к тому, что это переосмысление проблем STRIPS.

## Сравнение алгоритмов поиска и их описание

Описание алгоритмов библиотеки pyperplan:
1) A* это фактически Дейкстра,улучшенная эвристической функцией. Эвристика состоит в том, что штраф за проход к вершине высчитывается не как расстояние до вершины, а расстояние до вершины + расстояние от данной врешины до конечной вершины(на самом деле дополнительный штраф может быть и дргуим, всё зависит от выбранной нами функции $h(x)$).
2) WASTAR - это A* только с заданным весом.
3) BFS - алгоритм поиска в ширину, переходим в соседнии вершины по очереди и записываем их в очередь. Делаем так, пока не придём в нашу целевую вершину.
4) Enforced hill climbing search - жадный алгоритм выбирающий наиболее перспективную вершину по соседству.
5) Iterative deeping search - итеративный алгоритм DFS, немного эффективнее, чем обычный DFS. Он также опитмален как и DFS, но различие в том, что выбор следующей вершины происходит как и в BFS.
6) Greedy best first search - алгоритм поиска первого лучшего. Ищем наиболее лучшего соседа в нашей эвристике, и идём просто к нему.
Будем смотреть на следующеие факты: время работы, количество узлов, длинна плана. (Далее Wall-clock search time, Nodes expanded, Plan length соответственно). Наиболее приоритетный длинна плана, после время работы, после кол-во узлов.

| Алгоритм поиска | Результаты на blocks |
|---|---|
| A* | 23 Nodes expanded<br>Wall-clock search time: 0.061<br>Plan length: 12 |
| WASTAR | 26 Nodes expanded<br>Wall-clock search time: 0.038<br>Plan length: 12 |
| BFS | 3353 Nodes expanded<br>Wall-clock search time: 0.16<br>Plan length: 12 |
| Enforced hillclimbing search | 24 Nodes expanded<br>Wall-clock search time: 0.035<br>Plan length: 16 |
| Iterative deeping search | 14306 Nodes expanded<br>Wall-clock search time: 0.56<br>Plan length: 12 |
| Greedy best first search | 25 Nodes expanded<br>Wall-clock search time: 0.03<br>Plan length: 12 |

Вывод:
Наиболее GBFS, что довольно необычно, учитывая как работает алгоритм. EFS составил самый длинный план и притом с наибольшей затратой времени, что на самом деле нормально, учитывая, что это жадный алгоритм. Дольше всех работал IDS, при том больше всего вершин создано.

## Сравнение эвристическиих алгоритмов и их описание.

В данной части рассмотрим эвристики в библиотеке pyperplan и сравним их на эффективность. Для большей точности я рассматривал две задачи: первая airport(задача планирования вылета самолётов из аэропортов) и вторая sokoban(игра сокобан). Задачи выглядят максимально разными в выборке. Параметры для эвристик: . Будем смотреть на следующеие факты: время работы, количество узлов, длинна плана. (Далее Wall-clock search time, Nodes expanded, Plan length соответственно). Наиболее приоритетный длинна плана, после время работы, после кол-во узлов. Используется стандартный алгоритм поиска в данной библиотеке hff.

| Эвристика | Описание | Резльтат на airport | Результат на sokoban |
|---|---|---|---|
| Blind (вслепую) | Суть простая, если достигли цели - 1, иначе 0. | 11 Nodes expanded<br>Wall-clock search time: 0.00058<br>Plan length: 8 | 2551 Nodes expanded<br>Wall-clock search time: 0.15<br>Plan length: 49 |
| Landmarks (ориентиры) | Данная эвристика основана на разделении ситуаций без которых не достигнуть цели. Эти ориентиры, они имеют больший вес. | 9 Nodes expanded<br>Wall-clock search time: 0.00072<br>Plan length: 8 | 2941 Nodes expanded<br>Wall-clock search time: 0.2<br>Plan length: 49 |
| LM-cut (разрез ориентир.) | Идея схожа с прошлой, но различие в том, что теперь ориентиры отрезают от начала. | 9 Nodes expanded<br>Wall-clock search time: 0.011<br>Plan length: 8 | 95 Nodes expanded<br>Wall-clock search time: 1.1<br>Plan length: 49 |
| Relaxation (Релаксация) | Данный метод содержит в себе несколько доп.алго. Я выбрал лучший результат из каждой категории, ибо все релаксации основаны на том, что все кратчайшые подпути - есть участки кратчайшего пути. | 9 Nodes expanded<br>Wall-clock search time: 0.0032<br>Plan length: 8 | 103 Nodes expanded<br>Wall-clock search time: 0.14<br>Plan length: 49 |

**Выводы:** Были выбраны две задачи, с разными размерами. Размер плана во всех ситуациях получился одинаковым(возможно тогда оптимальным). На маленьких тестах лучшим получился обычный алгортим вслепую, но при этом он заэкспандил больше всех узлов. В больших же тестах ситуация получилась немного другая. Лидером по времени оказалась релаксация, меньше всего заэкспанжено узлов у LM-cut. Что действительно удивительно в данной ситуации, что Blind дал такой быстрый результат, при там кол-ве узлов.