Automated Planning – Research Review

Safdar Kureishy

Overview

This document is concerned with the development of Automated Planning, specifically a sub-domain called Classical Planning. As Wikipedia puts it: "*Automated planning and scheduling, sometimes denoted as simply AI Planning, is a branch of artificial intelligence that concerns the realization of strategies or action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles*". It is essentially a logical reasoning problem, which uses *search* to explore possible sequences of actions to reach a goal state from an initial state, and *formal reasoning methods* to determine the relations and equivalences between those states.

However, that search may end up with a monstrously large search space, called "combinatorial explosion" and the "curse of dimensionality". An important aspect of planning has therefore been to reduce the search space by using heuristics and state representations/languages to eliminate or prune the possible paths and/or dimensions that would either be suboptimal or unlikely to yield a solution. This review will discuss 3 such developments in automated planning research, specifically in planning representations/languages – STRIPS/Shakey, ADL and PDDL – that have developed over the last few decades to tackle such challenges, their relation to each other, and their impact on the field of AI as a whole.

Developments

Automated planning, as a subfield, was born in the 1970s, and has had its own challenges, particularly due to the high dimensionality and combinatorial explosion it subjects the search algorithms to, on account of the sheer number of variables involved in its representation, and due to numerous environmental characteristics it applies to, such as whether the environment is known or unknown, static or dynamic, deterministic or stochastic, fully or partially observable, episodic or sequential, single-agent or multi-agent, discrete or continuous, etc. However, alongside these challenges has been gradual progress as well – both in the representation of planning problems, and the algorithms to solve them.

Shakey & STRIPS (***S***tanford ***R***esearch ***I***nstitute ***P***roblem ***S***olver)

Though STRIPS was originally the name of the automated planner developed by Richard Fikes and Nils Nilsson in 1971[1], the same name was later also used to refer to the formal language of the *inputs* to the same planner.

Shakey, the robot that benefited from the STRIPS automated planner, was the first general-purpose, logical, goal-based mobile robot to be able to reason about its own actions, and plan all the steps required to fulfill larger tasks that would otherwise have had to be broken down into individual steps for prior robots to perform, thanks to the STRIPS planner it utilized. Built from 1966 – 1972, it was also the first project that melded logical reasoning and physical action, combining research in robotics, computer vision, and natural language processing, and could therefore be considered the first ancestor of UAVs, the Mars rovers and Autonomous Vehicles.

The STRIPS representation language used by the STRIPS planner formed the basis for most of the subsequent planning representations– called action languages – for expressing automated planning problems presently in use. In fact, the development of Shakey had a far-reaching impact on the fields robotics and artificial intelligence, including the development of the A* search algorithm, the Hough transform, and the visibility graph method for finding Euclidean shortest paths among obstacles in a plane.

A STRIPS representation of a problem instance is comprised of:

- An initial state: Conjunction of positive and ground (no variables and no functions) literals. State literals not specified are expected to be False.
- A goal state: Conjunction of positive and ground literals. Again, state literals not specified are expected to be false.
- A set of named actions:
    - Preconditions:
        - Positive conditions: Conjunction of function-free *positive* literals
        - Negative conditions: Conjunction of function-free *positive* literals
    - Post-conditions:
        - Positive effects: Conjunction of function-free *positive* literals
        - Negative effects: Conjunction of function-free *positive* literals

A propositional STRIP instance's decidability algorithm is PSPACE-complete, but restrictions can be enforced to achieve polynomial time decidability, or at least to make it NP-complete.

Example

```
Initial State:
        Init (At (C1, SFO) ∧ At (C2, JFK) ∧ At (P1, SFO) ∧ At (P2, JFK) ∧ Cargo (C1) ∧ Cargo (C2)
        ∧ Plane (P1) ∧ Plane (P2) ∧ Airport (JFK) ∧ Airport (SFO))
Goal State:
        Goal (At (C1, JFK) ∧ At (C2, SFO))
Action (Load (c, p, a),
        PRECOND: At (c, a) ∧ At (p, a) ∧ Cargo (c) ∧ Plane (p) ∧ airport (a)
        EFFECT: ¬At (c, a) ∧ In (c, p))
Action (Unload (c, p, a),
        PRECOND: In (c, p) ∧ At (p, a) ∧ Cargo (c) ∧ Plane (p) ∧ Airport (a)
        EFFECT: At (c, a) ∧ ¬In (c, p))
Action (Fly (p, from, to),
```

PRECOND: At (p, from) ∧ Plane (p) ∧ Airport (from) ∧ Airport (to)
EFFECT: ¬At (p, from) ∧ At (p, to))

ADL (Action Definition Language)

ADL is an improvement (one among others) over STRIP, providing more powerful expressiveness to handle realistic problems, such as the following that are all absent from STRIPS but supported in ADL[6]:

- Allowing both positive and negative literals. For example, expressing Rich ∧ Beautiful in ADL as ¬Poor ∧ ¬Ugly
- Assuming that unmentioned literals are unknown. This is known as the Open World Assumption.
- Allowing quantified variables in goals (not just ground literals). For example, ∃ x At (P1, x) ∧ At(P2, x) is the goal of having P1 and P2 in the same place in the example of the blocks.
- Allowing goals to involve conjunctions and disjunctions (Eg., Rich ∧ (Beautiful ∨ Smart)).
- Allowing conditional effects for actions (Eg., *When P:E* means E is an effect only if P is satisfied).
- Allowing the equality predicate (x = y).
- Allowing support for types (for example, the variable p : Person).

ADL expressiveness and complexity lies between that of the STRIPS language and Situation Calculus, by being sufficient to allow expression of more complex real-world scenarios while, at the same time, being restrictive enough to allow efficient reasoning algorithms to be developed.

Example

Initial State:
Init (At (C1, SFO) ∧ At (C2, JFK) ∧ At (P1, SFO) ∧ At (P2, JFK) ∧ (C1:Cargo) ∧ (C2:Cargo)
∧ (P1:Plane) ∧ (P2:Plane) ∧ (JFK:Airport) ∧ (SFO:Airport) ∧ (SFO ≠ JFK))
Goal State:
Goal (At (C1, JFK) ∧ At (C2, SFO))
Action (Load (c: cargo, p: plane, a: airport),
PRECOND: At (c, a) ∧ At (p, a)
EFFECT: ¬At (c, a) ∧ In (c, p))
Action (Unload (c: cargo, p: plane, a: airport),
PRECOND: In (c, p) ∧ At (p, a)
EFFECT: At (c, a) ∧ ¬In (c, p))
Action (Fly (p: plane, from: airport, to: airport),
PRECOND: At (p, from) ∧ (from ≠ to)
EFFECT: ¬At (p, from) ∧ At (p, to))

PDDL (Problem Domain Definition Language)

STRIPS and ADL (among others) served as inspiration for another extension of representational languages called PDDL (Planning Domain Definition Language), developed by Drew McDermott et al in 1998[3]. Not only was it aimed at addressing numerous limitations in ADL and PDDL, it was also aimed at standardizing planning languages and made the International Planning Competitions (IPC) possible, by allowing comparison of the performance of planning systems against each other using a set of benchmark problems, which in turn facilitated greater cross-pollination of ideas, feeding numerous subsequent advancements in the field.

Between 1998 and 2011, subsequent versions of PDDL (ranging from PDDL 1.2 through PDDL 3.1) were developed, each adding new features to the language, as follows[7]:

- PDDL 1.2:
  Had the following structural composition:
  - Domain Description:
    - Domain name
    - Requirements
    - Object-type hierarchy
    - Constant objects
    - Predicates
    - Actions
      - Parameters
      - Preconditions
      - Effects (Including conditional effects)
  - Problem Description:
    - Problem name
    - Domain name reference
    - Objects in the logical universe
    - Initial conditions
    - Goal states
- PDDL 2.1: Official language of the 3rd IPC in 2002. Added:
  - Numerical fluents (to model non-binary resources such as fuel-level, time, energy, distance, weight…)
  - Plan-metrics (to allow quantitative evaluation of plans, and not just goal-driven, but utility-driven mapping)
  - Durative/continuous actions (which could have variable, non-discrete length, conditions and effects)
  - Allowed expression of many more real-world problems than the original PDDL 1.0. Was a substantial and radical evolution of PDDL 1.0
- PDDL 2.2: Official language of the 4th PIC in 2004. Added:
  - Derived predicates (to model the dependency of given facts from other facts, such as transitive relations)
  - Timed initial literals (to model exogenous events occurring at given time independently from plan-execution).
- PDDL 3.0: Official language of the 5th IPC in 2006. Added:
  - State-trajectory constraints (hard-constraints in the form of modal-logic expressions, which should be true for the state-trajectory produced during the execution of a plan, which is a solution to the given planning problem)

- Preferences (soft-constraints in the form of logical expressions, similar to hard-constraints, but their satisfaction wasn't necessary, although it could be incorporated ito the plan-metric. Eg. To maximize the number of satisfied preferences, or to just measure the quality of a plan.

- PDDL 3.1 (Current): Official language of the 6th and 7th IPCs in 2008 and 2011. Added:
  - Object-fluents (functions' range now could be not only numerical (integer or real), but it could be any object-type also)
  - Adapted the language substantially to modern expectations with a syntactically seemingly small, but semantically quite significant change in expressiveness.

There have also been numerous variants of PDDL, such as PDDL+ (from PDDL 2.1), NDDL (from PDDL 1.0), MAPL (from PDDL 2.1), OPT (from PDDL 2.1), Probabilistic PDDL (from PDDL 2.1, for the probabilistic track of the 4th and 5th IPCs in 2004 and 2006), APPL (from NDDL), RDDL (from PPDDL1.0 and PDDL3.0, but totally different language, for the uncertainty track of the 7th IPC in 2011), and MA-PDDL (from PDDL 3.1, allowing planning for multi-agents). More detailed specifics of these variants are outside the scope of this review.

Example

```
(define (domain air-cargo)
        (:requirements :typing :adl)
        (:types cargo plane airport)
        (:predicates (at ?t - (either cargo plane) ?a - airport)
        (in ?c - cargo ?p - plane))
        (:action load
                :parameters (?c - cargo ?p - plane ?a - airport)
                :precondition (and (at ?c ?a) (at ?p ?a))
                :effect (and (not (at ?c ?a)) (in ?c ?p)))
        (:action unload
                :parameters (?c -cargo ?p - plane ?a - airport)
                :precondition (and (in ?c ?p) (at ?p ?a))
                :effect (and (at ?c ?a) (not (in ?c ?p))))
        (:action fly
                :parameters (?p - plane ?a1 ?a2 - airport)
                :precondition (and (at ?p ?a1) (not (= ?a1 ?a2)))
                :effect (and (not (at ?p ?a1)) (at ?p ?a2))))
(define (problem sfo-jfk)
        (:domain air-cargo)
        (:objects c1 c2 - cargo sfo jfk - airport p1 p2 - plane)
        (:init
                (at c1 sfo)
                (at p1 sfo)
                (at c2 jfk)
                (at p2 jfk)
        (:goal
                (and
                        (at c1 jfk)
                        (at c2 sfo))))
```

Conclusion

This review summarized 3 major advances in planning problem representation languages. Though other branches such as Machine Learning, Computer Vision, Natural Language Processing and Speech Recognition have been recently talked about more, Planning is finding greater applicability through applications in Self Driving Cars, UAVs, etc.

Sources:

[1] Richard E. Fikes, Nils J. Nilsson (Winter 1971). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.
[2] Edwing P.D. Pednault (1989). "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus".
[3] McDermott et. al (1998). ""PDDL---The Planning Domain Definition Language".
[4] Wikipedia: STRIPS (https://en.wikipedia.org/wiki/STRIPS#Extensions)
[5] Wikipedia: Shakey the robot (https://en.wikipedia.org/wiki/Shakey_the_robot)
[6] Wikipedia: Action Description Language (https://en.wikipedia.org/wiki/Action_description_language)
[7] Wikipedia: Planning Domain Definition Language (https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language)