History Of Planning

Planning is a PSPACE problem and hence in quest for a solution which is practical and acceptable, this field has seen lot of interesting developement. STRIPS is the ancestor for planning as a domain as it formalized the study of planning.

STRIPS/ PDDL

Initial generation of planning problems used logic based languages; STRIPS being the first one. It was developed at Stanford Research Institute to act as planning component taking decision in the robot developed by them named “Shakey”. It is based on boolean variables with propositional logic atoms. Its preconditions and goals are conjunctions of positive atoms whereas effects are conjunctions of literals (positive or negative atoms).

Formal Definition

A STRIPS planning task, short planning task, is a quadruple $\Pi = (P, A, I, G)$ where:

1. P is a finite set of facts (aka propositions).
2. A is a finite set of actions; each $a \in A$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of P referred to as the action’s precondition, add list, and delete list respectively; we require that $add_a \cap del_a = \varnothing$.
3. $I \subseteq P$ is the initial state. $G \subseteq P$ is the goal.
4. We will often give each action $a \in A$ a name (a string), and identify a with that name.

Planning Domain Description Language (PDDL) is a description language for planning in the STRIPS formalism and various extensions. It is not a propositional language. Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic) Action schemas parameterized by objects. Predicates to be instantiated with objects.

A PDDL planning task comes in two pieces namely the domain file and the problem file. The problem file gives the objects, the initial state, and the goal state. The domain file gives the predicates and the operators; each benchmark domain has one domain file.

Other contemporary developments include Partial-Order Planning. It starts at goal, extend partially ordered set of actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.

GraphPlan

Graphplan algorithm was developed Blum and Furst in 1995 and was popular during 1995 – 2000. It took away the focus of AI from non-linear or partial-order planning algorithms. This

algorithm had two characteristics that separated it from earlier ones: it finds plans of a fixed length (that is incrementally increased until a plan is found), and it uses reachability information for pruning the search tree. These differences brought the performance of Graphplan to a level not seen in connection with earlier planners

Approach: In a forward phase, build a layered “planning graph” whose “time steps” capture which pairs of actions can achieve which pairs of facts; in a backward phase, search this graph starting at goals and excluding options proved to not be feasible.

Soon after the introduction of Graphplan Kautz and Selman demonstrated that a general purpose satisfiability algorithm can in many cases outperform Graphplan and other algorithms specifically designed for AI planning [Kautz and Selman, 1996]. This motivated researchers to investigate translating planning to other computational problems and using solvers for them in finding plans. More recently, Bonet and Geffner's HSP planners[Bonet and Geffner, 2001], which use plain forward or backward chaining, have showed very good performance on many benchmark problems, and increased interest in heuristic search as a planning technique

SAT

The constraint-based model of classical planning allows plan search in several alternative ways. The main approaches have been backward chaining, as in the Graphplan algorithm, and general constraint solving, as exemplified by the satisfiability planning approach. The latter algorithms often need far less search than backward chaining, but there is a cost topay: the amount of computation per search tree node is often much higher than in a backward chaining planner. Depending on the type of problem instance at hand, the reduction in the search tree size may dramatically outweigh the more expensive computation, and in other types of problems, not much affect the search tree size and lead to an inferior performance.

Approach: From planning task description, generate propositional CNF formula ϕk that is satisfiable iff there exists a plan with k steps; use a SAT solver on ϕk, for different values of k.

Heuristic Search

Planning as Heuristic Search is the latest trend in Planning. It has been ruling the IPC challenges since 2000.
Approach: Devise a method R to simplify (“relax”) any planning task Π; given Π, solve R(Π) to generate a heuristic function h for informed search.

The basic idea of heuristic search is to derive an estimation function, a heuristic, that assigns to each search state a value indicating how good that state is. Then, during search, favor those states that seem to be best. The difficulty in applying this to domain independent planning lies in the derivation of the heuristic.

A straightforward way to relax a planning problem is to ignore the negative effects of all operators. This relaxation has been proposed by Bonet et al. [1997]. Computing the optimal relaxed solution length, which would yield an admissible heuristic, is NP-hard [Bylander, 1994], so Bonet et al. introduced a method for approximating that length. Facing a search state S, initialize the weights of all fluents in S to 0, and that of all other fluents to 1. Then apply all operators. For each operator with preconditions p that adds a fluent f , update the weight of f to weight(f) := min(weight(f); weight(p) + 1): Iterate the updates until the weight values of all fluents have reached their fixpoint. It is assumed here that operators have only positive preconditions. The weight of a set of fluents is defined as the sum of the individual weights. The difficulty of the state can be estimated as h(S) := weight(G): Here, G denotes the goal state of the problem at hand. Bonet and Geffner use this estimation process in all versions of the HSP system [Bonet and Geffner, 2001]. A variation of the process, taking into account positive interactions between the fluents, has recently been proposed by Hoffmann [2000].

References

1. https://kwarc.info/teaching/AI/notes-part4.pdf
2. https://medium.com/towards-data-science/ai-planning-historical-developments-edcd9f24c991
3. http://www.cs.toronto.edu/~sheila/2542/w06/readings/RintanenHoffmann01.pdf
4. https://homes.cs.washington.edu/~weld/papers/pi2.pdf