Planning Summary

Author: **Nicolò Brandizzi**

Contributors:

SAPIENZA
UNIVERSITÀ DI ROMA

DIAG
Sapienza
October 2018

Contents

Abstract

This is **free** material! You should not spend money on it.
Find the complete material on GitHub `https://github.com/nicofirst1/AI_notes`.

This notes are about the *Planning* part taught by professor Daniele Nardi in the Artificial Intelligence class. Everyone is welcome to contribute to this notes in any relevant form, just ask for a pull request and be patient.
Remember to add your name under the contributors list in the title page when submitting some changes (if you feel like it).

1 Classical Planning

A plan is a sequence of actions that allows us to reach a state where the goal condition holds, starting from the initial state.
For a classical planning problem we need to define 3 things:

- States, which are represented by *sets of instantiated literals* with a boolean value
  - Initial State
  - Goal State
- Actions or **action schema** composed of:
  - Action name
  - List of variables used in the schema
  - A precondition : define the state in which the action can be executed
  - An effect : result of an action
- An Environment:
  - Fully observable
  - Deterministic
  - Static
  - Discrete and Finite

Obviously an action can be executed in a state if the state *entails* the preconditions, i.e. the action is **applicable** in the state.

Lets consider the following example:

- Initial state: *Plane 1 in A*
- Goal state: *Plane 1 in B*
- Action: *Fly(Plane,From, To)*
  - Precondition : *Plane in From*
  - Effect: *Plane at To*

If we use the action *Fly(1,A,B)* the following thing happen:

- The plane is not in A anymore, so *At(1,A)* is deleted and added to the **delete list** [DEL(a)]
- The plane is in B, so now *At(1,B)* is true and is added to the **add list** [ADD(a)]

Intuitively we have that the precondition always refer to the time t while the effect refers to the time $t+1$.

Complexity of classical Planning We refer to:

- **PlanSAT** as the question to whenever it exist any plan that solves a planning problem, which is decidable for finite states, but becomes semi-decidable if we add function symbols to the language[1].

[1] The consequence is an infinite space

- **Bounded PlanSAT** as whenever there is a solution of length k,which remains decidable even with infinite states.

Forward state-space search Apply actions whose preconditions are satisfied unitl goal state is found or all states have been explored [2], with the following consequences:

- $Results(s, a) = (s - DEL(a)) \cup ADD(a)$
- The exploring of irrelevant actions
- The issue for the large space this kind of problems have.
- Strong domain-independent heuristics can be derived automatically.

Backward relevant-state search If a domain can be expressed in PDDL then we can do regression on it [3]. If we want to go from the goal g with an action a to the state g' we have:

$$g' = (g - ADD(a)) \cup Precond(a)$$

We can do this as long as an action a is **relevant** to the state g. Regression keeps the branching factor low, but it makes harder to come up with good heuristic, thus forward is preferred.

Heuristic for Planning Remember that an admissible heuristic can be derived from the **relaxed problem** which is easier to solve.
Search problems can be seen as a graph where nodes are stated and edges are actions. There are three ways you can relax a problem with graph:

- **Ignoring preconditions**: drop all precondition of an action. Usually this means that the number of actions to solve the problem is the same as the number of unsolved goal. On the other hand if we may have actions which achieve multiple goals or actions which can undo other actions [4]. .
- **Ignore delete list**: remove all negative literals from effects [5], monotonic progression towards the goal.
- **State abstraction**: many-to-one mapping where we decrease the number of states by ignoring some fluent.
- **Decomposition**: dividing the problems into parts and solving them independently [6], but it is a pessimistic approach [7] when sub-plans contains redundant actions.

Lemma If the effect of P_i leaves all the preconditions of P_j unchanged, then the estimated cost $Cost(P_i) + Cost(P_j)$ is admissible and more accurate of the maximum.

[2] Mapping planning into a search problem
[3] This is because we keep in memory both the DEL(a) and the ADD(a) lists
[4] Can be ignored to relax the problem
[5] No action can undo the progress made by another action.
[6] Assume sub-goal independence
[7] Not admissible

2 Partial Order Planning [POP]

Many sub-problems are independent so we can work on several sub-goals independently, solve them and then combine them to achieve a solution to the original plan.
A Partial Order Planner is a planning algorithm which can place two action into a plan without specifying which comes first, so that the solution is in the form of a *graph* rather than a sequence. POPs are created by a search in the *space of plans* rather than through the state space, where an action is actually a plan [8]. They work by **fixing flaws**[9] in the plan with the principle of **least commitment** [10].

Each plan has four components:

- A **set of actions** that makes up the steps for the plan. An empty plan has only the *Start* and *Finish* actions.
- A set of **ordering constraints** between pairs of actions in the form $A \prec B$ (A happens some time before B).
- **Causal links** $A \xrightarrow{p} B$ (A achieve p for B) [11]. The presence of causal links lead to *early pruning* of portions of state space that, because of not resolvable conflicts, contain no solutions.
- **Open preconditions** for an action not yet causally linked, i.e. a precondition that is not achieved by any action yet.

If a plan has the following proprieties than it is defined as **consistent**:

- All the open preconditions are achieved [12]
- There are no **cycles**, i.e. $A \prec B$ and $B \prec A$ which is also a contradiction.
- There are no **conflicts**, that is when $A \xrightarrow{p} B$ and an effect of another action C is $\neg p$ where $A \prec C$ and $C \prec B$. Note that a conflict can be solved by applying:
  - A **Demotion** $C \prec A$
  - A **Promotion** $B \prec C$

A consistent plan with no open preconditions is a **solution**.

The algorithm works in the following way:

1. The plan only contains *Start* and *Finish* with $Start \prec Finish$
2. The successor function:

[8] The difference is subtle. An action may be *Move(what,where)* where a plan is *Move(ObjA,Pos1)*.
[9] A flaw is anything that keeps the partial plan from being a solution
[10] Delaying a choice during the search until it is strictly necessary.
[11] For example when putting on shoes we may have $RightSock \xrightarrow{RightSockOn} RightShoe$, meaning that putting the right sock on achieves the condition necessary for putting the right shoe on.
[12] A precondition is *achieved* if no action can undo it's effects.

   (a) pick an open condition p of an action B
   (b) pick an action A that achieves p
   (c) add the causal link $A \xrightarrow{\mathrm{p}} B$ and $A \prec B$
   (d) resolves conflicts if possible, otherwise backtrack
3. The goal test succeeds when there are no more open preconditions

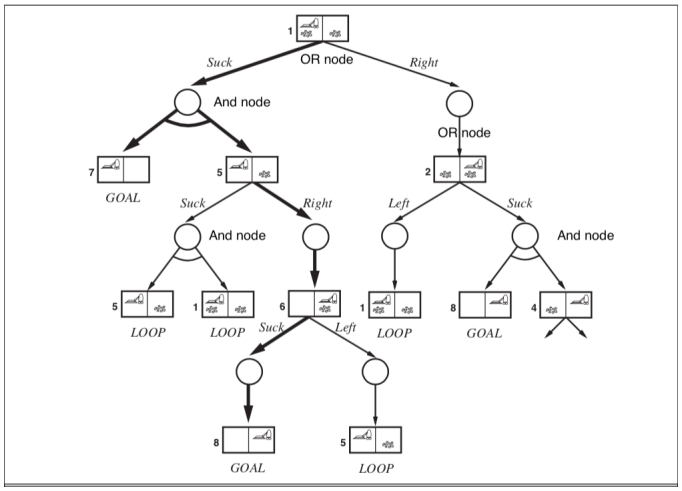
3 Non-Deterministic domains

3.1 Searching with Non-deterministic actions

When the future percepts cannot be determined in advance [13] the solution to a problem is a **contingency plan**. In this case the *result* of a problem is a function that return a **set** of possible outcomes states [14]. That is why the solution of a non-deterministic problem can contain nested **if-then-else** statements so that the form is *tree-like* rather than a sequence.

And-or trees The branching of a tree can depend on:

- The **agent's choices**, which will generate OR nodes
- The **environment's choice**, which will generate an AND node.



Non-cyclic Solutions The issue is that cycles often arise in this kind of problem.We can consider a solution in which: if the current state is identical to a state on the path from the root, than return false. This method guarantee to terminate in every finite space and return a solution which is **non-cyclic**.

Cyclic Solutions When we deal with non-deterministic actions we can get to the point where an action must be repeated some times to make it work [15]. So we want to keep doing an action until some state is reached [16]. This leads to **cyclic solution** which will eventually reach the goal provided that each outcome of a non-deterministic action eventually occurs.

[13] The state is only partially observable.
[14] Rather than a specific one.
[15] Slippery floor for robot movement
[16] Using *while* conditions

3.2 Searching with partial observation

For this kind of searching we need to introduce the concept of **belief state**, representing the agent's current belief about the possible physical states it might be in.

Sensor less belief With a specific sequence of action an agent can **coerce** the world into a state X [17]. Since the agent always know its own belief state we can derive that the latter is *fully observable*, furthermore the solution (if any) is always a sequence of actions.
A sensor-less problem [P] has the following elements:

- **Belief states**: contains every possible set of physical states
- **Initial state**: the set of all states in P
- **Actions**: there are two cases which depends on their effect on the environment
  - *Illegal action have no effect on the environment*: then we can take the **union** of all the actions in the current belief state
  - *Illegal action have effect on the environment*: then we take the **intersection**
- **Transition model**: The model which generate a new belief state b' [18] from a belief b troughs an action a, it depends on the type of action:
  - *Deterministic* action: $b' = Result(b, a) = \{s' : s' = Result_P(s, a), s \in b\}$
  - *Non-Deterministic* action: $b' = Result(b, a) = \{s' : s' = Results_P(s, a), s \in b\} = \cup_{s \in b} Reulsts_P(s, a)$
- **Goal test**: the belief state satisfies the goal only if all the physical states in it satisfy the $GoalTest_P$
- **Path cost**: assumed same for every action

Since The size of each belief state is exponential we can use the **incremental belief-state search** in which we first find a solution that works for state 1; then check if it works for state 2; if not, go back and find a different solution for 1, and so on.

Searching with observations In this case we have the *Percept(s)* function which returns the percept in a given state. When an observation is *partial* then we could have it generated by several states. In this case the difference with the sensor-less problem mentioned above are the following:

- There is a **prediction** stage.
- There is a **observation prediction** stage in which we determine the set of percepts that could be obtained in the predicted belief state
- The **update** stage, for each possible percepts we get the belief state that would result from it [19].

[17] That is the agent is sure to be in X after execution a sequence of actions
[18] Also called the *prediction* step.
[19] Reduces uncertainty.

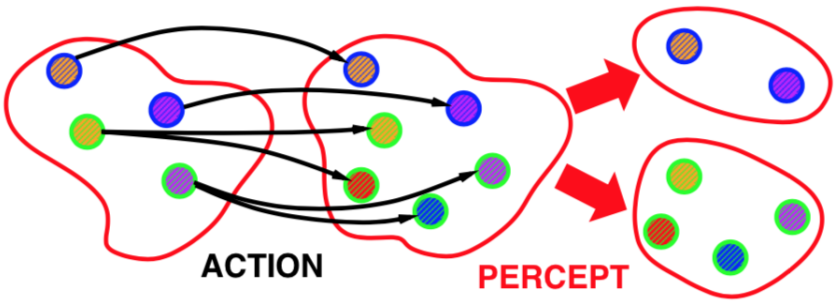
3.3 Planning in Non-Deterministic domains

For planning we augment the PDDL with a **precept schema**.
Online planning can generate a contingent plan with fewer branches at first, and deals with problems as they arise. While **re-planninig** agents would check the result and replan in case of unexpected failure.

Sensorless planning We can convert a sensorless problem to a belief-state planning problem where the transition model is represented by a set of action schemas and the belief model is represented by a logical formula [20].
If an effect depend on the state then we have a **conditional effect** which is denoted like this:

$$\textbf{when} \quad condition : effect$$

This kind of effect brings the belief state to have an exponential size in the worst case.



The different between unsatisfied conditional effect and unsatisfied pre-condition is that the latter do not permit the action to occur, thus the resulting state is **undefined**,while an unsatisfied conditional effect **does not change the state**.
Other kind of solutions can be:

- Perform some actions which keeps the belief state as simple as possible [21] and lower uncertainty.
- Not compute the belief state after some actions, but rather represent it as being followed by those action without computing it.

The final piece is to implement a good heuristic function to guide the search. Since any subset of the original belief is easier to solve we have that any heuristic which is admissible for a subset of the original belief state is also admissible for the entire belief state with the maximum function:

$$H(b) = max(h_1(b), ..., h_n(b))$$

[20] Instead of being all the possible enumeration of the states.
[21] Like when we check the time every once in a while to lower our uncertainty of the current time.

3.4 Online replanning

Execution monitoring Determine the need for a new plan. Some branches of a partially constructed contingent plan can simply say Replan; if such a branch is reached during execution, the agent reverts to planning mode. Replanning may occurs in the following situations:

- **Missing precondition**
- **Missing effect**
- **Missing state variable**
- **Exogenous events**: changes in the goal

Action Monitoring Before executing an action, the agent checks if all the preconditions still hold.

Plan Monitoring Before executing an action, the agent verifies that the remaining problem will still succeed. It can detect failure by checking the preconditions for success of the entire remaining plan. It cut off the execution of a dead plan as soon as possible and it also allows **serendipity** [22].

Goal Monitoring Before executing an action, the agent checks if there is a better set of goals it could try to achieve.

[22] Accidental success.

4 High-level actions

We introduce the concept of **high-level action** [HLA] which is part of the **hierarchical task networks** [HTN] which assumes the full *observability*, *determinism* and *availability* of a set of actions called **primitive actions**.
A **refinement** is a sequence of actions which describes [23] an HLA. If a refinement contains only primitive actions then it is called **implementation**.
A high-level plan achieves the goal from a given states if at least one of its implementations achieves the goal from that state.
A set of possible implementations in HTN planning is not the same as a set of possible outcomes in non-deterministic planning.

4.1 Searching for primitive solutions

It always start with a top level action called **Act**, then, for each primitive action a_i, we provide a refinement of Act with steps $[a_i, Act]$. The algorithm then becomes one that repeatedly chooses an HLA in the current plan and replace it with one of its refinements until the goal is achieved. This is computationally expensive, given:

- d primitive actions
- b allowable actions at each state
- k actions
- r possible refinements

The possible decomposition trees are:

$$r^{\frac{d-1}{k-1}}$$

Which, by taking small r and large k [24] , results in huge savings.
Some algorithms may even save successful plans in memory to later build newer plans on top of the older ones, leading to a more competent planner over time [25].

4.2 Searching for abstract solutions

If we write precondition-effect description for an HLA we can check whenever an HLA achieve goal without decomposing it with the refinements. So an HLP must have at least one implementation that achieves the goal to be considered doable, this is called the **downward refinement property**.
The problem lays in the description of the effect of an action which can be implemented in many different ways. A solution might be to include only the positive effects that are achieved for *every* implementation of the HLA and the negative effects of *any* implementation.
We introduce the notion of **reachable set** for an HLA h [26] in a state s, the

[23] Or decomposes.
[24] Small number of refinements each yielding a large number of actions.
[25] The problem lays in generalizing the methods that are constructed.
[26] Written Reach(s,h)

set of states reachable by any implementation of HLA. We can now say that a sequence of HLAs achieves the goal if its reachable set intersect the set of goal states.

Angelic semantics Given a *fluent* [27], a primitive action can add, delete or leave unchanged a fluent. The HLA under angelic semantics can control the fluent's value depending on which implementation is chosen.
For example to HLA *Go(home,work)* with two possible refinements:

- Drive(home,work)
- Call(taxi),Taxi(home,work)

can have as requirement for the $Taxi$ action to have $Cash$. Under angelic semantics the agent can choose to delete $Cash$.
This results in the derivable descriptions of HLAs from their refinements.
The angelic approach can be extended to find the least-cost solution by introducing the most efficient way to get to a state.

Descriptions There are two kind of descriptions which are kind of a upper/lower bound for the reachable set:

- **Optimistic**: may overstate the reachable set of an HLA
- **Pessimistic** : may understate the reachable set.

We can derive that if an optimistic reachable set does not intersect the goal then the plan does not work, on the other hand if a pessimistic description set intersect the goal then the plan works for sure. Additional refinement is needed when the optimistic intersects but the pessimistic doesn't.

[27] A state varaible