AI Planning for Autonomy

13. Non-Deterministic Planning

What about actions with uncertain outcomes where we don't have probabilities?

Tim Miller

THE UNIVERSITY OF MELBOURNE

Winter Term 2017

Agenda

Agenda

Relevant Reading

- *Improved Non-Deterministic Planning by Exploiting State Relevance.* by Muise, McIlraith, and Beck, ICAPS, 2012. Available from `http://haz.ca/papers/muise-icaps2012-fond.pdf`

Removing Assumptions

Classical Planning

Recall the follow assumptions about classical planning:

- Deterministic events
- Environments change only as the result of an action
- Perfect knowledge (omniscience)
- Single actor (omnipotence)

With MDPs, we dropped the assumption of deterministic events and looked at probabilistic events. In this lecture, we drop one further assumption: that we know the probabilities at all! Why would we not know the probabilities? For many reasons:

1. We don’t have enough data to learn them using reinforcement learning.
2. The probabilities are *non-stationary*; in other words, by the time we learn them, they will change. An example: games, wars, etc. If our opponent figures out we are learning probability distributions, they can exploit this.
3. Our systems are “one shot”; that is, we are operating in an environment we have never encountered before, so we no idea of the probabilities.
4. Actions outcomes are not random: we just don’t know which outcome will occur.

Fully-Observable Non-Deterministic Planning

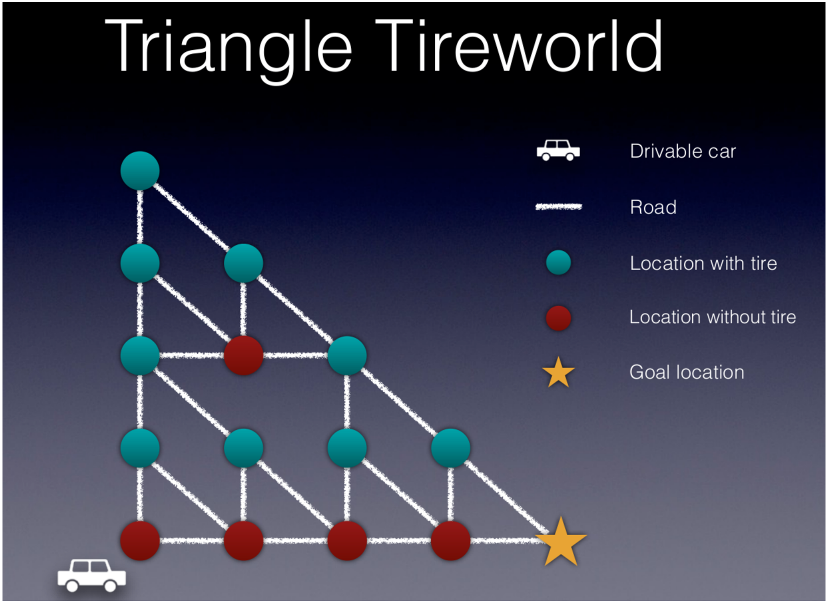
Fully-Observable Non-Deterministic Planning (FOND) removes the assumption the we have a probability distribution over the outcomes of non-deterministic events. As with MDPs, they assume that each action can have multiple outcomes, but they assume nothing about the probabilities between these outcomes.

For example:

- Asking someone to make you a coffee – they can say “yes” or “no”; but with what probability?
- Sending a drone out to scout over a hill for enemy combatants – the enemy can be present or not, but we probably do not have probabilities, and probably do not want them if we have them!
- Sending a query to a web-server for a page – it can send back no response (server is down), 404 Not Found error, or send back the page.

In some cases (e.g. case 1 above), the environment could be entirely deterministic, but we just do not know what outcome will occur, so our *model* of the environment must be non-deterministic.

Canonical example: Triangle Tireworld



Canonical example: Triangle Tireworld (continued)

The Triangle Tireworld is a canonical example of non-deterministic (non-probabilistic) planning. The goal is to move the car from the initial state at the bottom left location, to the goal location at the bottom right.

However, when the car moves, it can get a flat tire.

Some locations have a spare tire, others do not.

The challenge: can we find a solution that guarantees that we will get to the goal when we have non-deterministic outcomes?

Agenda

Planning with Markov Decision Processes: Goal MDPs

MDPs are **fully-observable** non-deterministic state models:

- a state space S
- initial state $s_0 \in S$
- a set $G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each state $s \in S$
- a **non-deterministic** transition function $f(a, s) \subseteq S$ for $a \in A(s)$
- action costs $c(a, s) > 0$

- What is different from classical planning? The transition function is no longer deterministic: if an action is executed as part of a plan, the outcome can be one of several states.
- What is different from MDPs? We don't have the probabilities of transitions.

FOND: Example action

Non-deterministic PDDL is one way to represent a FOND problem. It extends classical PDDL with the `oneof` construct.

```
(define (domain triangle-tire)
  (:requirements :typing :non-deterministic)
  (:predicates (vehicle-at ?loc)
               (spare-in ?loc - location))
               (road ?from ?to))
               (flattire))
  (:action move-car
    :parameters (?from ?to)
    :precondition (and (vehicle-at ?from) (road ?from ?to)
                       (not (flattire)))
    :effect (and (vehicle-at ?to) (not (vehicle-at ?from))
                 (oneof (not (flattire)) (flattire)))
```

Solutions for FOND problems: policies

Like MDPs, FOND planning should produce a policy, rather than a plan.

Recall: a policy, $\pi : S \rightarrow A$, is a *mapping* from states to actions. It specifies which action to choose in every possible state. Thus, if we are in state s, our agent should choose the action defined by $\pi(s)$.

However, in FOND problems, plans are typically classed as one of three types:

1. *Weak plan*: Policy of actions that achieves the goal for *at least one* possible execution. That is, it achieves the goal provided everything "goes our way".
   Example: along the bottom route in the tireworld problem. Provided the tire does not go flat, we reach the goal.
2. *Strong acyclic plan*: Policy of actions that achieves the goal and never visits a state more than once.
   Example: Up to the top location, and then down the diagonal. If the tire goes flat, we can change it at any of the locations on the route.
3. *Strong cyclic plan*: Policy of actions that achieves the goal for all possible executions, possibly revisiting states.
   Example: As for the acyclic definition, but the car goes to a location with a tire more than once.

Solutions for FOND problems: fairness

Not all problems have strong solutions. For example, if the top location in tireworld did not have a tire, there is no way we could guarantee arriving at the goal: we must pass through at least one location without a tire, and our tie can be flat at that point.

Some problems (not tireworld) have strong cyclic solutions but not strong acyclic solutions. They require that the solution revisits some states many times, trying to execute the same action each time and hoping for a different outcome than before.

However, this relies on the assumption of *fairness/*

Def Fairness: If we revisit apply the same action in the same state an infinite amount of times, we will encounter each non-deterministic effect an infinite amount of times.

For example, if we keep going to the same location, we will eventually get a flat tire. Some problems are inherently unfair (e.g. if the non-determinism is caused by an adversary trying to stop you reaching your goal), so this assumption does not hold for all problems.

Agenda

All-outcomes determinisation

The current state-of-the-art approaches to solving FOND problems is to use heuristic search to enumerate weak plans until a strong plan results.

Def All-outcomes Determinisation: The *all-outcomes determinisation* of a FOND problem is a deterministic version of the FOND problem, in which we have the same state space, initial state, and goal state as the original FOND problem, but we have a new set of actions, A', by creating a unique action for each possible outcome of a non-deterministic action, while everything else remains the same.

For example, for the `move-car` action defined above, we create two actions: `move-car-1` and `move-car-2`, in which the parameters and preconditions are the same as the original `move-car`, but the `move-car-1` has the effect:

```
(and (vehicle-at ?to) (not (vehicle-at ?from)) (not (flattire)))
```

and `move-car-2` has the effect

```
(and (vehicle-at ?to) (not (vehicle-at ?from)) (flattire))
```

Finding weak plans

To find a policy that is a weak plan for a given FOND problem, all we need to do is give the all-outcomes determinisation to our favourite classical search algorithm. For example, we can take a FOND model in PDDL, create the all-outcomes determinisation for it, and solve it using a classical planner (and label the all-outcomes actions back to their original labels).

If we ask the classical planner to return the optimal solution, then the result is a weak plan that will be optimal *provided nothing unexpected happens*. In short, it works for at least one set of outcomes for the non-deterministic actions.

The plan is weak however: it will not work if any of the *other* outcomes of any of the non-deterministic actions applied in the plan occur.

Example (from before): The optimal weak plan for the tireworld is to take the route along the bottom of the graph. Provided each time we execute the `move-car` action, the tire is not flat, this plan will work. However, if the car gets a flat tire at any of the locations along the way (the other outcome of `move-car`), then the plan fails.

Finding strong plans

To find a policy that is a strong plan for a given FOND problem, we iterate over many search problems to find a set of solutions that form a strong plan if one exists, or a *stronger* plan (compared to a weak plan) if a strong plan does not exist.

Algorithm:
Start with an initial empty policy P
While P changes do:

1. $Open = \{s_0\}$ (the open states in the search)
2. While $Open$ is not empty do:
    (a) Take an element, s, from $Open$, and if this is not a goal state, find a weak plan from s to the goal using the all-outcomes determinisation.
    (b) For each state-action pair in the weak plan, add this pair to the policy P
    (c) For each state-action pair in the weak plan, take the action from the original FOND problem, and for every outcome state, s', of that action, add s' to $Open$ if s' has not been seen before during the search.

So, intuitively, we generate an initial weak plan for the problem, and then for all outcomes that *could* occur if the plan does not execute as expected, find a weak plan from that outcome to the goal state.

In practice, we terminate the search either when $Open$ is not empty OR when we reach some pre-defined time limit.

Deadends!

But what if we encounter a *deadend*: a state from which the goal cannot be reached by any sequence of actions. That is, at step 2(a), we cannot find a plan from s to the goal. For example, in the Tireworld problem, having a flat tire at a location that has no spare tire is a deadend: the car cannot move and the tire cannot be changed.

In a classical search algorithm, a deadend state becomes *closed*, so we never re-visit it. However, the FOND algorithm re-starts classical search many times, so could re-visit a deadend.

The resolution is straightforward:

1. Exit the loop at line 2(a) if a plan is not found;
2. Record s as a deadend state; and
3. Return to step 1, but never expanding the deadend states during the *classical search*.

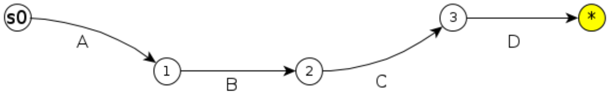
For example, if our first move in Tireworld is to move right and we receive a flat tire, we record that we should not apply the action of moving right from the initial state. In subsequent instances of the classical search, we would not apply that action and would avoid that deadend.

Example

Consider the following abstract example. The blue nodes represent members of the $Open$ set, $s0$ is the initial state, and $*$ (yellow node) is the goal state. Upper-case letters are action names. Arrows represent outcomes of an action. Straight arrows representation outcomes from deterministic actions, while curved arrows represent outcomes from non-deterministic actions. NOTE: this is not a representation of the classical search, but of the construction the *policy*.

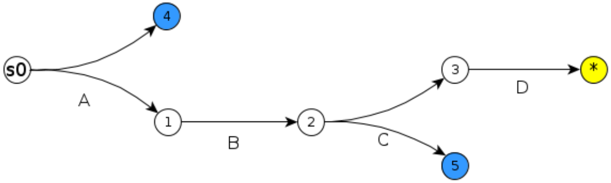
Initially, we have the initial state $s0$ and the goal state $*$:

Next, try to find a weak plan from $s0$ to $*$ (step 2(a)):



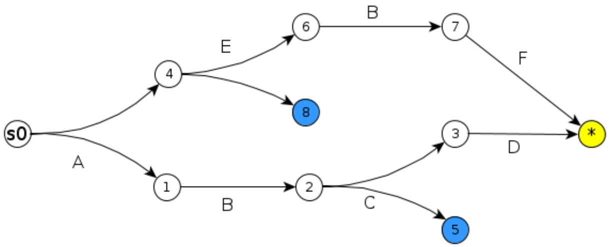
Example (continued)

Then, for every non-deterministic action in the plan, add the other outcomes to the *Open* set (step 2(c)):



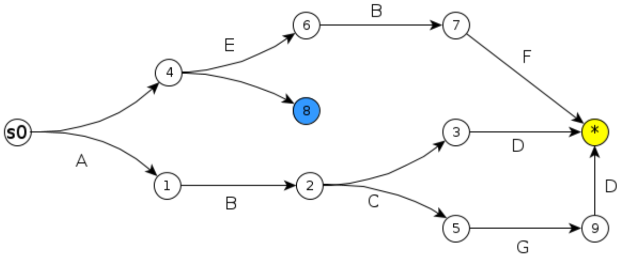
Example (continued)

Then, on the second iteration of the loop at step 2, try to find a weak plan from node 4 to the goal and add the other outcomes (we're combining steps 2(a) and 2(c) here for brevity):



Example (continued)

Then, on the third iteration of the loop, do the same from node 5 to the goal:



Example (continued)

The fourth iteration of the loop would clearly be to try to find a weak plan from node 8 to the goal.

If we successfully found a weak plan from node 8 to the goal and there are non-deterministic actions whose outcomes we have not seen, we add these states to the $Open$ set and continue our search.

If we successfully found a weak plan from node 8 to the goal and there are *no* non-deterministic actions used in the weak plan *or* we have seen all of the outcomes previously in the search, then $Open$ would be empty and the algorithm would terminate.

If we could not find a weak plan from node 8 to the goal, this means that *no strong plan exists* for this problem. In this case, we return the plan that we have so far, which is a weak plan, but is more robust than just the initial weak plan from $s0$ to $*$: it will only fail if we end up at node 8.

Example policy

Assuming that we found a solution from node 8 directly to the goal using action `D`, the policy is created by simply taking each state-action pair in the graph:

```
s0 => A
1 => B
2 => C
3 => D
4 => E
5 => G
6 => B
7 => F
8 => D
```

Question: Can we do better?

Improving the search

There are two simple ways to improve the algorithm outlined above. Both approaches aim to improve the weak-plan search, which is iterated over many times:

1. Using the policy in the weak-plan search
2. Planning locally in the weak-plan search

There are several other improvements that can be made that we will not discuss here — the recommended reading presents one very powerful approach: *relevance*.

Using the policy

This is the most basic and straightforward approach.

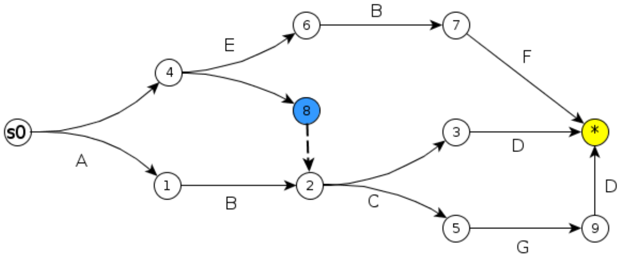
In step 2(a), instead of executing the entire classical search from the open state to the goal, we terminate if we see a state that is already handled by the policy.

If there is a state that is already handled by the policy, then this must represent a way to get to goal from that state.

Therefore, if during the classical search used for weak-plan generation, we see that a state if already handled by the policy, we terminate the search at this point, and update the policy based on the weak-plan search so far. The previous policy contains enough information to reach the goal from there.

Example: Using the policy

For example, consider again the third iteration in our abstract example:



If our classical search algorithm arrives at state 2 during its search, by applying action `B` from state 8, then the link between 8 and 2 connects state 8 with the goal state $*$. We need to just add `8 => B` to the policy, and the weak plan from 8 to the goal is established. Therefore, the aim of finding a weak plan has been achieved, and we can terminate the classical search algorithm there, potentially saving a lot of search.

Planning locally

A weak plan from an open state to the goal is a plan if everything “goes as expected”. We will call states on this path the *expected* states, and the states resulting from the other non-deterministic outcomes the *unexpected* states.

One way to avoid complete classical searches for weak plans to is to search from unexpected states towards the expected states of the action, instead of searching towards the goal.

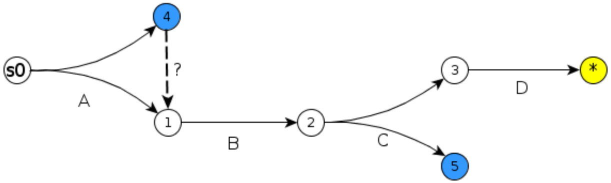
This is effectively the same as previous shortcut of using the policy to determine whether we can terminate the search early, except that we *explicitly* search for a state that in the policy. That is, we run a classical search with the expected state as the “goal”, rather than the final goal state as the goal.

The intuition behind this is that a set of states that are each an outcome of a non-deterministic action will be “closer” to each other in the search graph then they are to a goal state.

However, in many domains, we may spend a lot of time looking for a local plan that ultimately cannot be found.

Example: Planning locally

After the first iteration of the planning loop, we have the following:



Therefore, in the second iteration, rather than searching for a weak plan from node 4 to the goal, we first try for a weak plan from node 4 to node 1 (the expected outcome of action `A`). If we find one, we need not search for the weak plan to the goal. If we do not, we search for a weak plan from node 4 to the goal node as before.

Similarly, in iteration three of the loop, we first search for a plan from node 5 to node 3, and in iteration four, from node 8 to node 6.

Complexity

Solving FOND planning problems is an EXPSPACE problem.

Def EXPSPACE: Decision problems for which there exists a deterministic Turing machine that runs in space exponential in the size of its input.

(It runs in exponential time as well).

Some of the complexity comes from the all-outcomes determinisation: if we have N number of actions, with an average of k outcomes, we have N^k number of actions in our all-outcomes determinisation.

However, the techniques discussed here manage to solve many large problems quickly!

Summary: FOND

We introduced a new class of planning problem: non-deterministic (and non-probabilistic) planning problems; and showed how the state-of-the-art planners solve these problems.

What is interesting is the use of classical search techniques to solve them. In fact, state-of-the-art FOND planners are always built on top of some state-of-the-art classical planner.

To solve a non-deterministic problem, we use classical search to find weak plans, and iteratively build up strong plans (if possible).

However, several techniques, three of which are presented here, are used to avoid solving the entire classical search problem in every iteration.

What's next? Relaxing the assumption of single agents by looking at *perspectives of others*