CSC 191/291 Notes on Planning

The ability to plan sequences of actions so as to achieve desired goals is arguably the most distinctive manifestation of intelligence. It's what has given us our advantage over other creatures; as Steven Pinker has quipped, it is why we study lions, rather than lions studying us, despite our much weaker bodies and senses. Pinker has also written about early humans interpreting an antelope's tracks, predicting where it was headed, anticipating its resting place in the shade of yonder baobab tree in the mid-day heat, approaching it silently from downwind and out of its field of view, etc., eventually hurling their spears. The spears are themselves the product of a planned process, and likewise fishing and cooking gear, and so on.

We'd like to get some computational understanding of how planning can work. The truth is, we don't yet have ways to do that in human-like fashion, allowing for the diverse kinds of knowledge and inference that the antelope-tracking example calls for. However, we do have computational planning methods for limited applications, such as route planning for cars, planning in adversarial games, and other special domains. We can get some ideas about how planning works by looking at ***"closed worlds",*** *where all the relevant participating entities are known to the planner, and the only changes that occur are those brought about by the action of the planner* (once the agent decides to execute a plan).

Closed-world planning

The general idea in closed-world planning is that we have

- a well-defined initial state (specified with logical formulas, or some domain-specific notation);
- a well-defined goal (also specified with a logical formula, or some domain-specific notation);
- a well-defined set of ***operators*** that specify how any give state is changed (what formulas become true, and what formulas become false) when the operator is applied under specified preconditions.

Then we conduct a search over possible sequences of operator applications (actions) that starts at the initial state and leads to the goal state. A sequence of such operator applications that leads to the goal is considered a viable ***plan.***

There are several well-studied ways of performing searches that yield plans. Four of the traditional ones are

- ***State-space planning***: This is possible if all possible states are easily represented in some simple way that fully specifies each state, and there aren't huge numbers of states. For example, in the "Towers of Hanoi" puzzle, we start with a big disk D3, a medium disk D2, and a small disk D1 stacked in that order on the first of three pegs; the goal is to move the stack to the third peg, where the only operator is one that allows moving a disk at the top of a stack to an empty peg or to a bigger disk at the top of another stack. Here we could use an initial state representation like ((D3 D2 D1) ( ) ( )) and a goal state representation (( ) ( ) (D3 D2 D1). Playing around with this, you find that there are only 27 distinct states, which we can draw as a kind of triangular graph whose edges are operator applications. This state space is easily searched for a plan by trial and error, moving "forward" from the start state, and being careful not to reenter the same state twice. Speed can be improved in state-space search by using heuristic estimates of how many steps remain to get to the goal state.

- ***Deductive planning:*** Here we represent states and the effects of operators (under given conditions) logically, and search for a proof that a future state exists (given the current initial state) where the goal is true. It turns out that a plan can be extracted from a successful proof of this type. (This is covered in CSC 244/444 but not here.)
- ***Regression planning:*** We start with the goal (in general using a logical representation of states and the effects of operators, under given conditions); we ask, "What actions directly lead to that goal?" We pick such an action, and ask, "What are the preconditions under which that action has the desired effect?" We then continue backward, picking actions that achieve the preconditions in question, etc., till we have "chained back" (regressed) to the current initial state. (NB: In general, multiple actions may be needed to achieve a complex goal, and multiple actions may be required to achieve the preconditions of an action.) The most famous regression planner is STRIPS (dating back to 1971 or so), and the most important thing about it is its "precondition + effect" representation of operators, which is used in virtually all planning systems.
- ***Forward planning:*** This is a lot like state-space search, but without the assumption that we can simply represent any possible state, and there aren't hugely many states. Again in general we use logical state representations, and we do some sort of systematic forward search trying to get closer to a state where the goals is true, starting from the initial state. The fastest current methods are forward planning methods, but that's probably because in a "closed world" things are a lot simpler than in the real world.

Examples of STRIPS-like operators

Given the ubiquity of the STRIPS operator representation, let's look at some examples.

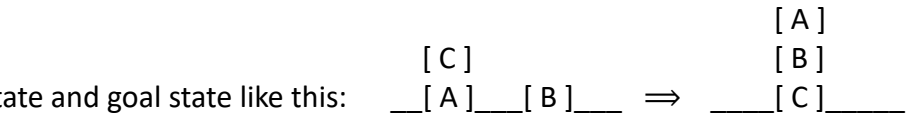
From the "Blocks World"

The "Blocks World" typically consists of some children's blocks on a table, where these can be stacked up or unstacked. This has been a workhorse for studies in planning. Here is one possible way of formulating the operators:

```
(defop stack (?x ?y ?z)
  ; take block ?x from (table or block) ?y and put it on block ?z
  :preconds ((block ?x)  (block ?z)
          (not (?x = ?z))  (not (?y = ?z))
          (clear ?x) ; nothing on ?x
          (on ?x ?y) ; ?x starts out on ?y
          (clear ?z))
  :effects ((on ?x ?z)
         (not (on ?x ?y))
         (not (clear ?z)); ?z now has ?x on it
         (clear ?y)); y is now clear (NB: regard table1 as always clear)
 )
```

```
(defop unstack (?x ?y)
  ; take block ?x off ?y and put it on the table
  :preconds ((block ?x)
             (block ?y)
             (clear ?x) ; nothing on ?x
             (on ?x ?y)); ?x starts out on ?y
  :effects ((on ?x table1)
            (not (on ?x ?y))
            (clear ?y))
 )
```

Consider an initial state and goal state like this:



Then you can see that the plan ((unstack C A) (stack B table1 C) (stack A table1 B)) will achieve the goal. Apart from the deductive approach, one can see intuitively how any of the above approaches would work to find this solution.

From the Gridworld environment

The following operators would be natural in the Gridworld setting that the CSC 291 students will use – this is not a fixed world, but a Lisp code base where you can define a simulated world with a robot (or other agent(s)), various places, various paths connecting places, and various entities that the robot can use or get to.

Actually the syntax here is a bit simplified from what you would actually write in Gridworld – some quotation marks have been omitted that are needed in Gridworld, and the 'defop' needs to be expanded to something slightly more verbose; but that's insignificant. What's more significant is that the syntax allows for computable functions (e.g., arithmetic functions like '*', '+', …, and lisp functions signalled by their final exclamation mark, like 'dist-in-miles!'). ME ("Motivated Explorer") refers to the agent itself, and the predicate names are chosen to make translation into English easy. E.g.,

```
(is_a_footpath_from+to Maple-Road Home School)
```

becomes *"Maple-Road is a footpath from Home to School"* ('+' is used for argument insertion points).

```
(defop walk (?x ?y ?f)
  :preconds ((is_at ME ?x)
             (is_tired_to_degree ME 0)
             (is_a_footpath_from+to ?f ?x ?y))
  :effects ((is_at ME ?y)
            (not (is_at ME ?x))
            (is_tired_to_degree ME (* 0.3 (dist-in-miles! ?x ?y)))
            (not (is_tired_to_degree ME 0)))
  :time-required (* 100 (dist-in-miles! ?x ?y))
  :value 0
)
```

```
(defop drink (?h ?x)
  :preconds ((is_thirsty_to_degree ME ?h)
             (> ?h 0)
             (is_potable ?x)
             (has ME ?x))
  :effects ((is_thirsty_to_degree ME 0)
            (not (is_thirsty_to_degree ME ?h))
            (not (has ME ?x)))
  :time-required 1
  :value (* 4 ?h)
)
```

These examples are sufficient to indicate how goal-directed planning of behavior might be implemented computationally. A point to note about the 'drink' operator above is that it includes a ':value' field, where the quantity computed is proportional to how thirsty the agent was. So this is a reward quantity for ME. In fact, the ME agent is motivated entirely by its reward functions – it behaves so as to optimize the total reward it estimates will accrue, when it conducts a forward search.

Much as in forward-planning as described above, ME considers what actions it can take in the current state, creating forward branches in a search tree. It computes the state descriptions resulting from each of those actions, and these become the starting points for further forward search. In this way a tree of alternative action sequences and resulting states is created. But in contrast with forward planning, the point is not to achieve some goal, but rather to find a sequence of actions for which the cumulative rewards are as high as possible. It then starts to execute the "plan" thus found, and then replans (because it may learn new facts when it has started to execute the plan, enabling it to build a more rewarding plan).

Food for thought

So in a sense, this agent "lives for gratification" (though a difficult question is whether it could actually be "feeling gratified" – one tends to say, "Surely not!" – but how do we know this, either for a machine, or for animals, or for people ??? This is a question about *phenomenal consciousness*). Is such an agent necessarily selfish? Well, in one sense yes, but in an important other sense, no: Suppose it "computes" high rewards for doing altruistic actions, e.g., bringing food to a hungry being. Then it will tend to act altruistically towards other beings! In other words, Gridworld agents can in fact gain "vicarious satsifaction" from benefitting other beings, and hence be motivated to do so. That's also something to think about …

It seems that an agent that plans and acts to maximize its own rewards (and minimize costs or distress) could reasonably be considered ***self-motivated*** (hence the name "*motivated* explorer" ME). Could we then also say that it has ***free will***? If not, what do we mean by "free will"??