Planning and Search

Hierarchical Planning

Outline

- ♢ Hierarchical task networks (HTN)
- ♢ HTN planning
- ♢ Hierarchical search (search for primitive solutions)
- ♢ Angelic search (search for high-level solutions)

Abstract or high-level actions

State-of-the art planning algorithms can generate plans containing thousands of actions

However some planning tasks involve millions of actions

For example, organising a census in a large country

Planning to invade some country

Or (R & N textbook example) for a human to be just moving about, if this is planned at the level of muscle activations (about 10^3 muscles, activation can be modulated 10 times per second, so planning for an hour may involve more than 3 million actions).

Solution: plan at a **higher level** (instead of muscle activations, just an action 'walk to the lecture room'). Then **refine** if necessary.

HTN planners

HTN stands for Hierarchical Task Network.

Similar to classical planning in that states are sets of fluents (ground atomic formulas), and actions correspond to deterministic state transitions

Planning domain description different - includes methods for decomposing tasks into subtasks.

High-level actions

The planning domain description still has a set of actions (now called primitive actions) described as before by preconditions and effects.

In addition, it has high-level actions or HLAs which have the same signature as normal actions, for example, $Go(University, Heathrow)$ (go from the University to Heathrow airport), but instead of preconditions and effects, they have one or more refinements (methods of decomposing them into smaller more concrete steps).

The steps in refinement may also be HLAs.

Example

$Refinement(Go(University, Heathrow),$

STEPS: $[Bus(University, Broadmarsh), Bus(Broadmarsh, Heathrow)])$

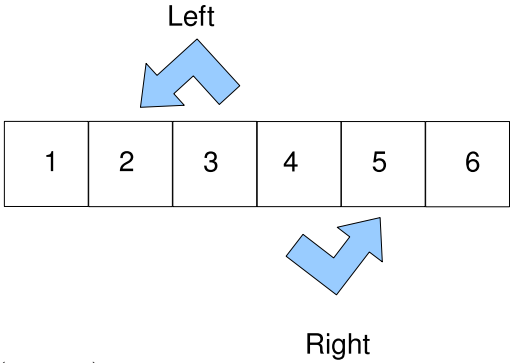
$Refinement(Go(University, Heathrow),$

STEPS: $[Taxi(University, Heathrow)])$

$Refinement(Go(University, Heathrow),$

STEPS: $[Bus(University, Station), Train(Station, StPancras),$
$Underground(StPancras, Heathrow)])$

Another example



$Refinement(Navigate(x, y),$
PRECOND: x=y
STEPS: [])

$Refinement(Navigate(x, y),$
PRECOND: $x > 1$
STEPS: [Left, Navigate(x-1,y)])

$Refinement(Navigate(x, y),$
PRECOND: $x < 6$
STEPS: [Right, Navigate(x+1,y)])

Implementation of an HLA

A refinement of an HLA which consists of only primitive steps is called an implementation of HLA

For example, $[Right, Right]$ is an implementation of $Navigate[1, 3]$

An implementation of a high-level plan (a sequence of HLAs) is the concatenation of implementations of each HLA in the sequence.

Implementations of HLAs have pre- and postconditions. HLAs themselves don't because they may have several different implementations.

Angelic semantics of HLAs: a high-level plan achieves the goal if **at least one** of its implementations achieves the goal

Planning with HLAs

Two possible approaches:

♢ Search for an implementation of a high-level plan that works (search for primitive solutions). This will be presented in detail.

♢ Reason directly about HLAs and search for an abstract plan. This will only be sketched.

Searching for primitive solutions

For each goal, a single top level action Act. It has a refinement which for precondition corresponding to the goal has an empty list of steps.

Start with a set of refinements for Act (simplest case - every primitive action followed by Act is a refinement, but normally will have a more sensible set of refinements)

Find an implementation of Act which achieves the goal; breadth-first replacement of HLAs with refinements

Hierarchical search algorithm

```
function HIERARCHICAL-SEARCH(problem, HTN) returns a solution, or failure
    frontier ← a FIFO queue with [Act] as the only element
    loop do
        if EMPTY?(frontier) then return failure
        plan ← POP(frontier) /* chooses the shallowest plan in frontier */
        hla ← the first HLA in plan, or null if none
        prefix, suffix ← the action subsequences before and after hla in plan
        outcome ← RESULT(problem.INITIAL-STATE, prefix)
        if hla is null then /* so plan is primitive and outcome is its result */
            if outcome satisfies problem.GOAL then return plan
        else if for each sequence in REFINEMENTS(hla, outcome, HTN) do
            frontier ← INSERT(APPEND(prefix, sequence, suffix),frontier)
```

Efficiency gains

Consider a non-hierarchical forward state-space planner

Suppose solution has d steps and there are b actions possible in each state

Complexity is $O(b^d)$

Suppose each HLA has r refinements and k steps in a refinement

For hierarchical planner, complexity is $O(r^{d/k})$ (branching r, depth d/k)

If r is small (at most b), $O(r^{d/k})$ is root k of non-hierarchical cost $O(b^d)$

Moral: keep r small and k large if possible

Question for self-test and revision

As an exercise, please do question 6 from 2010-2011 G52PAS exam

Exam and answers on http://www.cs.nott.ac.uk/ nza/G52PAS/

If you get stuck or find an error in model answers, please email me ...

Searching for abstract solutions

Search with HLAs directly using their descriptions, which are similar to pre-conditions and effects

Need to produce high-level plans with downward refinement property: if the high-level plan claims (according to HLA descriptions) to achieve the goal, then there is an implementation which achieves the goal

How to check preconditions and effects of HLAs: first compute preconditions and effects for each implementation of an HLA, and then say that HLA achieves something if one of its implementations achieves it (intuition - we can **choose** this implementation when making the plan)

Angelic semantics as opposed to demonic semantics (when have to guarantee all implementations work, as with non-deterministic actions)

Reachable states

Given an HLA h and a state s, consider the set of states reachable from s by any of implementationsof h: $\textsc{Reach}(s, h)$

$\textsc{Reach}(h_1, h_2) = \cup_{s' \in \textsc{Reach}(s, h_1)} \textsc{Reach}(s', h_2)$

A high-level plan achieves the goal if its reachable set intersects with the set of goal states

The only snag - how do we compute $\textsc{Reach}(s, h)$ using preconditions and effects?

(Remember we are working with HLAs and they have multiple implementations with different preconditions and effects)

Approximations of reachable states

At the level of fluents, it is hard to predict what an HLA will do with a particular state property

If we started at $x = 3$ and did $Navigate(3, 6)$, is $x > 3$ (hopefully) or $x < 3$ (because we chose the refinement of going left?)

Complicated interactions of conditions, in general

Solution: use **approximations** of REACH

Approximations of reachable states 2

Optimistic description of an HLA $\textsc{Reach}^+$: overstates the set of reachable states

Pessimistic description of an HLA $\textsc{Reach}^-$: understates the set of reachable states

$$\textsc{Reach}^-(s,h) \subseteq \textsc{Reach}(s,h) \subseteq \textsc{Reach}^+(s,h)$$

If $\textsc{Reach}^+$ of a plan does not intersect the goal, the plan does not work

If $\textsc{Reach}^-$ of a plan intersects the goal, it definitely does work

If neither test applies, the plan needs to be refined

Angelic search

Angelic search is called with Act as the $initialPlan$

MAKING-PROGRESS is a check for not being in an infinite loop of recursively applying the same refinement

```
function ANGELIC-SEARCH(problem, HTN, initialPlan) returns a solution, or failure
  frontier ← a FIFO queue with InitialPlan as the only element
  loop do
    if EMPTY?(frontier) then return failure
    plan ← POP(frontier) /* chooses the shallowest node in frontier */
    if REACH+ (problem.INITIAL-STATE,plan) intersects problem.GOAL then
      if plan is primitive then return plan
      guaranteed ← REACH−(problem.INITIAL-STATE,plan) ∩ problem.GOAL
      if guaranteed ≠ { } and MAKING-PROGRESS(plan, initialPlan) then
        finalState ← any element of guaranteed
        return DECOMPOSE(HTN, problem.INITIAL-STATE, plan, finalState)
      hla ← some HLA in plan
      prefix, suffix ← the action subsequences before and after hla in plan
      for each sequence in REFINEMENTS(hla, outcome, HTN) do
        frontier ← INSERT(APPEND(prefix, sequence, suffix),frontier)
```

Decomposition of an abstract plan

Uses regression planning to decompose abstract plan into subproblems, one for each step in the abstract plan

function DECOMPOSE($HTN, s_0, plan, s_f$) **returns** a solution
  $solution \leftarrow$ an empty plan
  while $plan$ is not empty **do**
    $action \leftarrow$ REMOVE-LAST($plan$)
    $s_i \leftarrow$ a state in REACH$^-$(s_0,$plan$) such that $s_f \in$ REACH$^-$(s_i,$action$)
    $problem \leftarrow$ a problem with INITIAL-STATE=s_i and GOAL=s_f
    $solution \leftarrow$ APPEND(ANGELIC-SEARCH($problem, HTN, action$),$solution$)
    $s_f \leftarrow s_i$
  return $solution$

More efficiency gains

Potentially, exponential advantage compared to hierarchical search

If lucky, commit to a good high-level plan and refine it (linear)

Next lecture

Conformant and conditional planning

Monitoring and replanning

Russell and Norvig 3rd edition Chapter 11

Russell and Norvig 2nd edition Chapter 12