Planning, Execution & Learning: Hierarchical Task Net Planning

Reid Simmons
Manuela Veloso

Hierarchical Task Net (HTN) Planning

- **Basic Ideas**:
  - Complex plans often have identifiable structure
  - That structure can often be captured in the form of hierarchies of abstract subplans
  - Subplans are often (nearly) independent of one another

"To get to conference in ?x, get to the airport, take a plane to ?x, then get to the conference hotel"

"To get to the airport, either drive or take a cab"

"If you have enough money for the fare:
To take a cab to ?y, either call ahead, or flag a cab down, then enter the cab, say "I want to go to ?y", wait until at ?y, pay the fare, then exit the cab"

HTN Plans

- ***Plan = (Tasks, Constraints)***
- ***Tasks***
  - ***Primitive***
    - Standard STRIPS-style operators
    - "Executable"
  - ***Compound***
    - Preconditions and Effects
    - ***Methods*** for decomposing operator into more detailed subplans
      - Details internal structure
      - Parameterized
      - Similar to *macros* or *subroutines*

HTN Plans

- ***Constraints***
  - Precedence (*before, after, between)*
  - Metric temporal
  - Resource
  - Constraints at one level apply to all pairs of tasks at lower level

$$\mathbf{U = \{u_1, u_2\};\ V = \{v_1, v_2\}}$$

$$\mathbf{U < V \Rightarrow u_1 < v_1;\ u_2 < v_1;\ u_1 < v_2;\ u_2 < v_2}$$

- An ***Abstract Plan*** contains compound operators
- An ***Instantiated Plan*** has only primitive operators
- A ***Fully Instantiated Plan*** is a totally-ordered instantiated plan with all variables bound

Search Space

- Problem Reduction Search
  - "Goal state" is an abstract ***task*** to be achieved (***not*** state of the world)
- Search space operators
  - Decompose task into subtask(s)
  - Parameterize task
  - Solve for conflicts

Abstract HTN Algorithm

- HTN-Plan (*tasks*, *constraints*, *methods*)
  - If (*tasks*, *constraints*) has no solution, return {}
  - If (*tasks*, *constraints*) is an instantiated plan
    - **Select** a fully instantiated plan
    - If such a plan exists, return it; otherwise return {}
  - **Select** an abstract task $t \in$ *tasks*
  - **Choose** an applicable $m \in$ *methods*
    - where u is the task-list of m and c is the constraints of m
    - $tasks = tasks - \{t\} \cup u$
    - $constraints = constraints \cup c$
  - (*tasks*, *constraints*) = **apply-critics**(*tasks*, *constraints*)
  - Return HTN-Plan(*tasks*, *constraints*, *methods*)

Where is the Power?

- Methods encode domain knowledge
- Methods encode problem solving knowledge
  - "how" rather than "what"
- Abstractions encapsulate patterns of interaction

+ May be easier to specify domain
- Have to specify all possible goals (and how to achieve them)

Caveats

- HTN planning, in worst case, is still NP-complete
- May not terminate (recursive method expansions – may be hard to detect infinite loops)
- May have to completely expand before finding plan is illegal

Simple Hierarchical Order Planner (SHOP) (Nau, et.al. 1999)

- SHOP Algorithm:
  - Forward search, linear planner
    - Plans in same order as execution
    - Essentially depth-first search
  - Primitive operators have no preconditions
  - No concurrent actions
  - Highly expressive operator representation (numeric calculations)
  - Efficient (but rather inflexible) planning algorithm

SHOP Domain Example

(:operator (!putdown ?block)
  ((holding ?block)) ← **Delete list**
  ((ontable ?block) (handempty))) ← **Add list**

(:method (make-clear ?y)
  ((clear ?y)) ← **Applicability condition**
  nil) ← **Task list**

(:method (make-clear ?y)
  ((on ?x ?y)) ← **Applicability condition**
  ((make-clear ?x)
   (!unstack ?x ?y) (!putdown ?x)) ← **Task list**

Extensions to Simple HTN Planning

1. Threat detection
   - Deal with interacting goals
   - Find ways of reusing operators
2. Methods can include preconditions and effects
   - Find threats earlier in the planning process
3. Methods can indicate resource usage
   - Do some types of scheduling
4. Operators/Methods can include open conditions ("external preconditions")
   - Need to use action-based (refinement) planning
   - Enables planner to find "novel" solutions

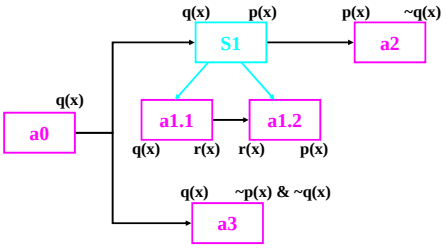
Example: Home Construction (O-Plan)

Method Build (?house)
  Precondition: (and (own land) (have money))
  Effects: (built ?house)
  Applicability: (single-family-home ?house)
  Expansion: S1: Build-Foundation(?house)
    S2: Build-Frame(?house)
    S3: Build-Roof(?house)
    S4: Build-Walls(?house)
    S5: Build-Interior(?house)
    S6: Decorate(?house)
  Orderings: S1<S2, S2<S3, S2<S4, S3<S5, S5<S6
  Links: S1 causes (foundations laid) for S2
    S2 causes (frame erected) for S3 and S4
    S3 causes (roof built) for S5
    S4 causes (walls built) for S5
    S5 causes (interior done) for S6
  TimeWindow: start between 11:30 and 14:30 at S3
  Resources: bricklayers = between 1 and 2 men at S4

Dealing with Subplan Interactions

- Types of Interactions
  - Deleted condition (threat)
  - Resource (e.g., "existing object" bindings)
  - Redundant steps
- HTN Planners Often Use ***Critics*** to Detect Certain Types of Illegal Plans and/or Synergies
  - Time window bounds
  - Resource bounds
  - Interaction of effects between compound tasks
  - Operators that can be ***merged***
    - In contrast, POP planners try to ***share*** operators
    - Merging is more efficient, but may not be complete

Danger of Over-Commitment



Expressivity of HTN Planning

- HTN Planning is More Expressive than Action-Based Planning!!
  - HTN planning can generate a larger class of plans
  - Proof (by Erol) involves reduction to classes of grammars
    - Action-based planning is analogous to ***right-linear*** (***regular***) grammars
    - HTN planning is analogous to ***context-free*** grammars

Example:

Want to create round-trip transportation plans such that the leg of the trip from X to Y always uses the same carrier as the leg of the trip from Y to X.

Metric Resources

- ***Consumable*** (no replenishment)
  - Example: "*available oil reserves*"
  - Representation: **[r_l, r_u]** ranges; Constraint ***propagation***
- ***Renewable*** (complicated by substitution effects)
  - Example: "*money*" "*fuel*"
  - Representation: Algebraic; Constraint ***satisfaction***
- ***Sharable – Single Unit***
  - Example: "*a spacecraft's camera*"
  - Representation: **Mutex**; Simple ***add/delete***
- ***Sharable – Multiple Units***
  - Example: "*drill presses in a factory*"
  - Representation: ***Need explicit scheduling techniques***

Handling Metric Time

- Time Can be Treated as a Metric Resource, but it is Special
  - Non-exclusive resource (multiple actions can occur during same time interval)
  - Non-renewable resource (cannot create more time – ***sigh***)
  - Time resources must be consistent with temporal orderings
- Representations
  - Precise time: map to **reals** or **integers**
  - Time windows: **[min, max]**
  - Algebraic: **(t1 + t2 < duration)**