# *Planning, Execution & Learning: Heuristic Search Planning*

Reid Simmons



## *Search*

- Uninformed: depth-first, breadth-first
- Informed: best-first, hill-climbing, A*, ...
- Heuristics
  - Admissible
    - ***What?***
    - ***Why Important?***
  - Informed
    - ***What?***
    - ***Why Important?***



## *Heuristic Search Strategies*

- **Best-First**
- **Hill-Climbing**
  - Rationale: Heuristics tend to be better discriminators amongst local alternatives than as global (absolute) estimate
  - Random "restarts" when stuck
- **A*** ***Can A* be run "backwards"?***
- **Weighted A***
  - H(s) = cost-so-far(s) + W * estimated-cost(s)
  - Not admissible, but tends to perform much better than A*



## *Heuristic Search Planning*

- Basic Idea
  - ***Automatically Analyze Domain/Problems to Derive Heuristic Estimates to Guide Search***
- Decisions
  - How to evaluate search states
  - How to use the evaluations to guide search
  - How to choose which part of plan to work on next
- ***Resurgence in Total-Order, State-Space Planners***
  - Best such planners (FF, FD, LAMA) dominate other types
  - Still a hot topic for research



## *How to Generate Heuristics*

- Domain-Specific
  - Program in (or ***learn***) heuristics specifically for that domain
- Domain Analysis
  - Preprocess domain to generate meta-control knowledge
- On-Line
  - Solve a ***relaxed*** form of the problem
  - Approaches
    - Assume complete subgoal ***independence***
    - Assume ***no*** negative interactions
    - Assume ***limited*** negative interactions

***Euclidean distance***
***Manhattan distance***
***Spanning tree***



## *HSP (Bonet & Geffner, 1997)*

- Heuristic State-Space Planner
  - Can do either ***progression*** or ***regression***
- Relax Problem by Eliminating "Delete" Lists
  - Essentially compute transitive closure of actions, starting at initial state
    - "Reachability" analysis
  - Cost of literal is stage/level at which first appears
  - Continue until no new literals are added
  - Similar to ***GraphPlan's*** forward search

## Computing Costs of Literals

$^{0}$On(C, A) $^{0}$On(A, Table) $^{0}$On(B, Table) $^{0}$Handempty $^{0}$Clear(C) $^{0}$Clear(B)

**Pick(C, A)** **PickT(B)**

$^{1}$Holding(C) $^{1}$Holding(B) $^{1}$Clear(A)

**PutT(C) Put(C, A) Put(C, B) PutT(B) Put(B, A) Put(B, C) PickT(A)**

$^{2}$On(C, Table) $^{2}$On(C, B) $^{2}$On(B, A) $^{2}$On(B, C) $^{2}$Holding(A)

**PickT(C) Pick (C, B) Pick(B, A) Pick(B, C) Put(A, B) Put(A, C)**

$^{3}$On(A, B) $^{3}$On(A, C)

On(A, B) & On(B, C) **Estimate**: 5 **Actually**: 6

On(A, C) & On(C, B) **Estimate**: 5 **Actually**: 4



## HSP Heuristics

- **Max**
  - Cost of action is *maximum* over costs of preconditions
  - Admissible, but not very informed
- **Sum**
  - Cost of action is *sum* of precondition costs
  - Informed, but not admissible
- **$H^2$**
  - Solve for ***pairs*** of literals
  - Take maximum cost over all pairs
  - Informed, and claimed to be admissible



## FF (Hoffmann, 2000)

- FF (Fast Forward) Refines HSP Heuristic
- Takes ***positive*** interactions into account
  - Avoids double-counting of actions
- Similar to ***GraphPlan's*** forward search combined with a ***relaxed*** version of its backward search
  - Ignores negative interactions
- Admissible and Informed

## FF State Evaluation Heuristic

On(A, C) On(A, Table) On(B, Table) Handempty Clear(C) Clear(B)

**Pick(C, A)** **PickT(B)**

On(C, A) On(A, Table) On(B, Table) Handempty Clear(C) Clear(B)
Holding(C) Holding(B) Clear(A)

**Pick(C, A) PickT(B)**

**PutT(C) Put(C, A) Put(C, B) PutT(B) Put(B, A) Put(B, C) PickT(A)**

On(C, A) On(A, Table) On(B, Table) Handempty Clear(C) Clear(B)
Holding(C) Holding(B) Clear(A)

On(C, Table) On(C, B) On(B, A) On(B, C) Holding(A)

**Pick(C,A) PickT(B)**

**PutT(C) Put(C, A) Put(C, B) PutT(B) Put(B, A) Put(B, C) PickT(A)**

**PickT(C) Pick (C, B) Pick(B, A) Pick(B, C) Put(A, B) Put(A, C)**

On(C, A) On(A, Table) On(B, Table) Handempty Clear(C) Clear(B)
Holding(C) Holding(B) Clear(A)

On(C, Table) On(C, B) On(B, A) On(B, C) Holding(A) On(A, B) On(A, C)

**On(A, C) & On(C, B)**



## Discussion

- Progression: Need to Calculate Heuristic Every Step
- Regression: Just Calculate Heuristic Once
- Heuristic Search Using Progression Generally More Robust
- HSP and FF Heuristics Outperform Partial-Order Planners
  - Ground actions seem to be the big difference
    - Easier to estimate cost without variables
    - Forward search provides reachability analysis
- Similar Techniques Applicable to Partial-Order Planners
  - REPOP (Nguyen & Kambhampati, 2001)
  - VHPOP (Younes & Simmons, 2001)



## "Enforced" Hill Climbing

- Used to Avoid "***Wandering***" on "Plateaus" or in Local Minima
  - Perform breadth-first search until find ***some*** descendant state whose heuristic value is less than the current state
- Shown to be Very Effective
  - Especially when search space is pruned to eliminate actions that are "unlikely" to lead to goal achievement
- Used by FF

## *Landmarks (Richter, Helmert, Westphal)*

- Landmark
  - Proposition that must be true in any plan/solution
  - Proposition A that is precondition of all ways to achieve B
    - Computationally intractable
    - Use planning graph to approximate
    - Restrict attention to operators that achieve B *for the first time*
    - *Look-behind* can expand set of landmarks (although explodes, computationally)
- Using Landmarks
  - Intermediary goals (not optimal, or even complete)
  - As a heuristic: Count landmarks that still need to be achieved from current state



## *Combining Heuristics (Roger & Helmert)*

- No Single Heuristic is Best
  - Heuristics generally better at differentiating local choices
  - Problem exacerbated by ties
- Combining Heuristics
  - Max
  - Tie breaking
  - Pareto
  - Alternation
    - Empirically shown to outperform other methods



## *Flaw Selection*

- Answers the Question:
  - ***Which subgoal (or threat) should be worked on next?***
- **LIFO**
  - \+ Empirically, continuing to work on a given subproblem, all else being equal, tends to perform well ("coherence")
  - \- Uninformed
- **Least-Cost**
  - Use heuristic estimate of subgoal cost to choose "easiest"
  - \+ Prefers forced choices, which reduces branching factor
  - \- Can be myopic



## *Flaw Selection*

- **Delay Separable Threats**
  - Put off handling potential threats that may be solved by adding binding constraints
  - \+ Other choices often force separation to occur naturally (or converts to non-separable threat)
  - \- If separation is forced, could lead to earlier detection of dead end
- **Forced Choice**
  - Choose flaws for which only one possible choice exists (better: "at most one")
  - \+ Adds no additional branching
  - \- May delay the inevitable dead-end detection