Planning, Execution & Learning
1. Heuristic Search Planning

Reid Simmons

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 generate successor states
- ***Resurgence in Total-Order, State-Space Planners***
  - Best such planner (FF) dominates other types
  - Still a hot topic for research

Search Heuristics

- Admissible
  - ***What?***
  - ***Why Important?***

- Informed
  - ***What?***
  - ***Why Important?***

Evaluating Search States

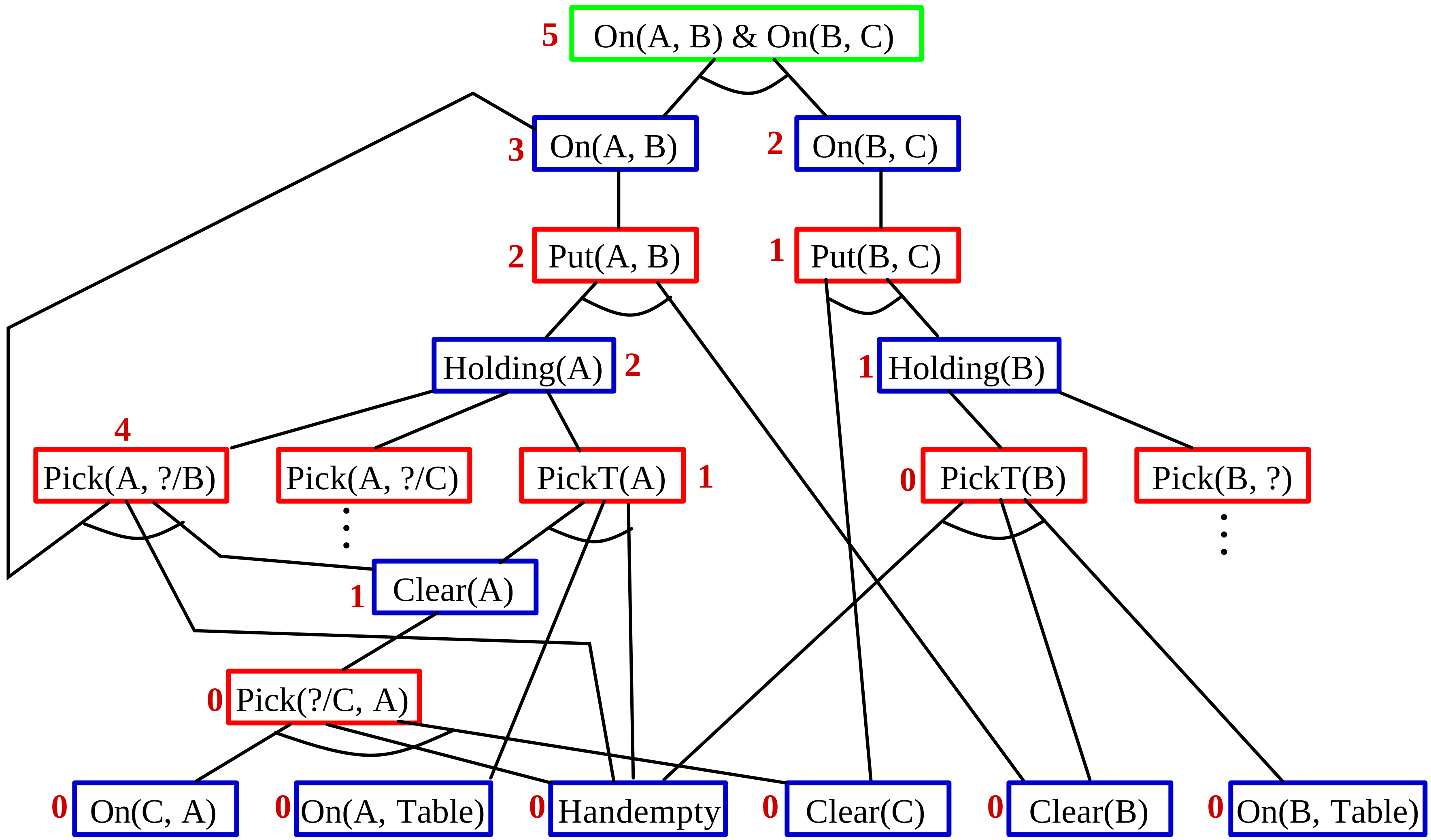
- Basic Idea
  - ***Solve a Relaxed Form of the Problem; Use as Estimate for Original Problem***

- Approaches
  - Assume ***complete*** subgoal independence
  - Assume no ***negative*** interactions
  - Assume ***limited*** negative interactions

UNPOP (McDermott, 1996)

- Non-Linear Planner
  - Modeled after PRODIGY
  - Maintained explicit “tail plan”
  - Completely recomputed “head plan” each step
- Regression Match Graph
  - Create ***and/or*** graph
    - **And** branches: Conjunctions of goals/preconditions
    - **Or** branches: Different ways of achieving subgoal
    - Can contain cycles
  - Cost in initial state = 0; Cost of action = 1
  - ***Sum*** over **and** branches; ***Min*** over **or** branches
- Difficulty in dealing with unbound variables
  - Find substitutions that maximize subgoal satisfaction (***heuristic!***)

Regression Match Graph Example



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
  - 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)**

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

–TPOP (Younes & Simmons, 2001)

Heuristic Search Strategies

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

"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

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

Deriving Domain Properties

- STAN (Fox & Long, 1999)
  - Perform **ST**atic **AN**alysis of domains/problems to find structural properties
    - Mobility
    - Resources
    - Symmetry
  - Apply specialized planning algorithms to extracted subproblems
    - Route planner
    - Scheduler

Generic Types

- Higher Order Types Whose Elements are Themselves Types
- Structural Combinations of Preconditions/Effects that Signal Particular "Categories" of Objects