CS 4649/7649
Robot Intelligence: Planning

Efficient Planning, PDDL

Sungmoon Joo

School of Interactive Computing
College of Computing
Georgia Institute of Technology

*Slides based on Dr. Mike Stilman's lecture slides

Course Info.

- **Course Website: joosm.github.io/RIP2014**
- **Course Wiki: github.com/RIP2014/RIP2014Wiki/wiki**
  - add your contact info, start grouping/filling in project ideas, etc.
- HW#1, due Sept. 29

Efficiency in Planning

- Planning Efficiency: **speed** of a planner

Smart robots make **good** decisions.

Smarter robots make **good** decisions **fast**!

We like smart robots

How can we make our robots smarter?

Properties of Heuristics: h(s)

Informed

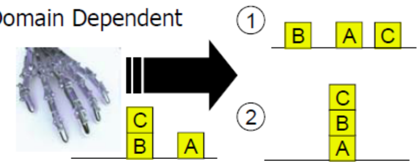
- Does estimate lead to the goal?
- Accuracy of heuristic

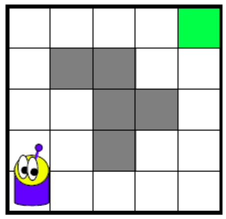
Admissible

- $h(s) \leq$ true "cost to go"
- Is Best-First Search with Admissible h(s) optimal?
- Is A* with Admissible h(s) optimal?

Heading: Heuristics in Planning

- Domain Dependent

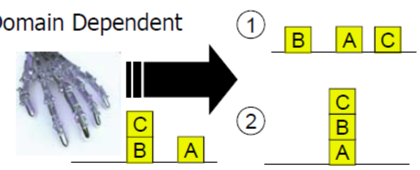


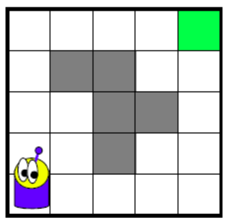


- Domain Independent
  - Automatically Analyze Problem/Domain
  - Derive Heuristic Estimate

Heuristics in Planning

- Domain Dependent





- Domain Independent
  - Automatically Analyze Problem/Domain
  - Derive Heuristic Estimate

Domain Independent Heuristics

Concept

- Solve a relaxed form of the problem
- Use to evaluate states for solving original problem

Approaches

- Assume complete subgoal independence (remember this idea?)
- Assume no negative interactions (new idea)
- Assume limited negative interactions

HSP (Bonet & Geffner 1997)

- Progression or Regression Search
  (We focus on Progression)

- Relax Problem by Eliminating Delete Lists

Expand state in levels by evaluating all valid actions
Continue until no new literals are added

Heuristic cost of literal is number of levels until it first appears

Not quite GraphPlan: level is used as a HEURISTIC – how?

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

HSP Progression Heuristic

HSP_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Heuristic cost of precondition C(p,s) = level of p in graph
- HSP_h(s) = function of C(p,s)

HSP Heuristics

H_0
- Cost of action is sum of precondition costs
- Informed, but not admissible

H_1
- Cost of Action is maximum over costs of preconditions
- Admissible, but not very informed

H_2
- Solve for pairs of literals
- Take maximum cost over all pairs
- Informed, and claimed to be admissible

H_0

- Assume negative effects do not exist
- The cost of achieving a set of preconditions $\{p_0, \ldots, p_n\}$
  = Sum of costs for achieving each precondition

$$C(s) = \sum_i C_i$$

Informed? Somewhat

Admissible? No

HSP_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Heuristic cost of precondition C(p,s) = level of p in graph
- HSP_h(s) = function of C(p,s)

H_1

- Assume negative effects do not exist
- The cost of achieving a set of preconditions $\{p_0, \ldots, p_n\}$
  = Maximum of costs for achieving each precondition

$$C(s) = \max_i C_i$$

Informed? Not very

Admissible? Yes

HSP_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Heuristic cost of precondition C(p,s) = level of p in graph
- HSP_h(s) = function of C(p,s)

H_2

- Assume negative effects do not exist
- The cost of achieving a set of preconditions $\{p_0, \ldots, p_n\}$
  = Maximum of costs for achieving PAIRS of preconditions

$$C(s) = \max_{ij} C_{ij}$$

Informed? More so

Admissible? Yes

HSP_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Heuristic cost of precondition C(p,s) = level of p in graph
- HSP_h(s) = function of C(p,s)

FF (Hoffman 2000)

- Hoffman – Best of both worlds: HSP & GraphPlan
  Refines HSP Heuristics

FF_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Compute a relaxed plan by regression search in the graph.
- FF_h(s) = # actions in plan

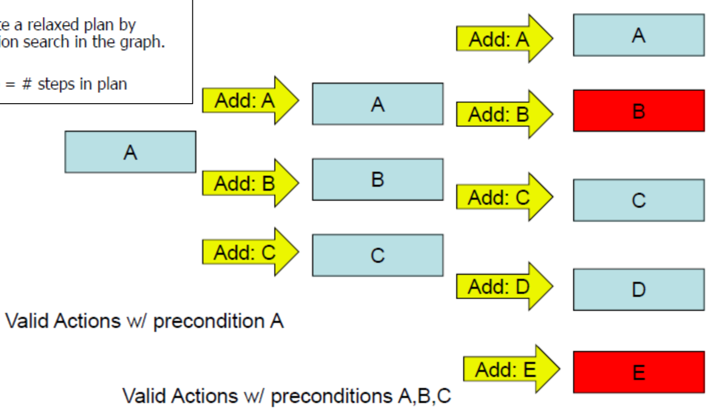
Jorg Hoffmann

FF Heuristic

FF_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Compute a relaxed plan by regression search in the graph.
- FF_h(s) = # steps in plan

Suppose our goal is to achieve B and E

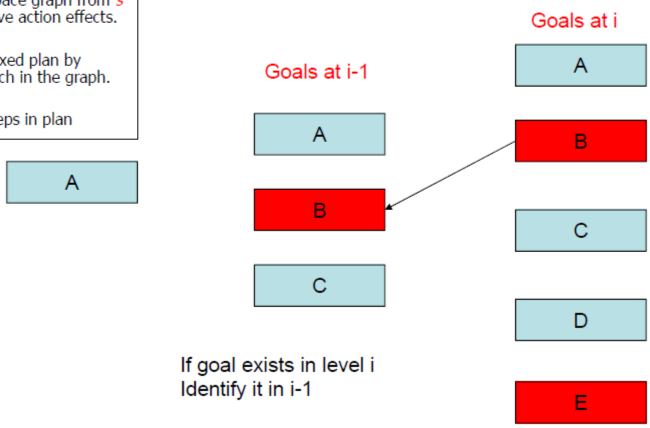


FF Heuristic

FF_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Compute a relaxed plan by regression search in the graph.
- FF_h(s) = # steps in plan

Suppose our goal is to achieve B and E



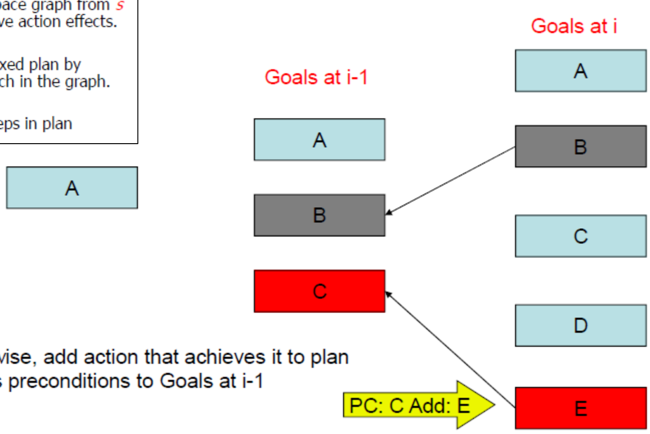
If goal exists in level i
Identify it in i-1

FF Heuristic

FF_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Compute a relaxed plan by regression search in the graph.
- FF_h(s) = # steps in plan

Suppose our goal is to achieve B and E



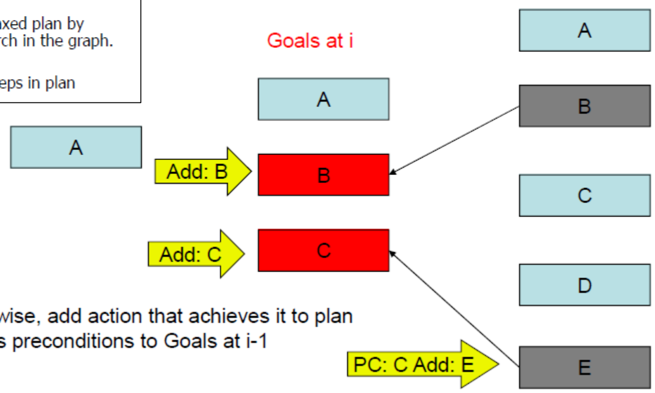
Otherwise, add action that achieves it to plan
Add its preconditions to Goals at i-1

FF Heuristic

FF_h(*s*)

- Grow a state space graph from *s* ignoring negative action effects.
- Compute a relaxed plan by regression search in the graph.
- FF_h(s) = # steps in plan

Suppose our goal is to achieve B and E



Otherwise, add action that achieves it to plan
Add its preconditions to Goals at i-1

FF vs. HSP

- FF's heuristic is more informed and admissible
- Takes into account positive interactions by propagating constraints back through the layers
- FF uses "Enforced Hill Climbing" which appears to work well
  EHC = Hill Climbing + BFS when stuck

Really though – which one is faster?

International Planning Competitions at ICAPS

FF vs. HSP

- FF's heuristic is more informed
- Takes into account positive interactions by propagating constraints back through the layers
- FF uses "Enforced Hill Climbing" which appears to work well
  EHC = Hill Climbing + BFS when stuck

The Hoffman Story:

2000 FF = International Competition Winner

2002 FF is Also very Successful

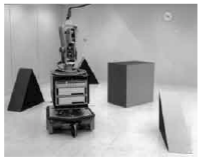
2004 Hoffman becomes International Competition Chair

Plan Space Planning?

- Catching up! Various heuristics have been proposed.
- "Flaw Selection Strategies" (Which subgoal(or threat) should be worked on?)
  - Continue working on a single subproblem
  - Choose easiest subproblems to work on
  - In particular solve ones with forced solutions (reduces branching factor)
  - Put off threats that may be forced by future constraints
- VHPOP (Younes & Simmons)
  - Similar to FFs heuristic for partial order planning (additive)

The Long Road

Shakey – STRIPS 1966

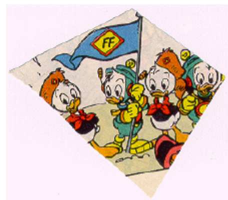


State Space

Plan Space

Graph Space

FF International Planning Winner 2000, 2002



Propositional Satisfiability: Since 1960s

SAT = First PROBLEM shown NP-Complete! 1971 Stephen Cook

Given a boolean formula:

$$(P \lor Q) \land (\neg R \lor S) \land (R \lor Q \lor S)$$

SAT: Exists a model (truth assignment) that makes this true?

Many algorithms for solving it

Especially for achieving average-case polynomial time!

Planning As Satisfiability

- Let P be a planning problem
  - Let (P,k) be a *bounded planning problem*
  - A solution for P of length k is a solution to (P,k)
- For each k = 0, 1 , … , N (some bound)
  - Encode (P,k) as a SAT problem
  - If resulting SAT problem is satisfiable, extract solution plan

Encoding Plans as SAT

Use Only propositional logic

- Turn predicates on(A,Table) into propositions on-A-Table
- Modify action descriptions accordingly

Encoding Plans as SAT

- Initial State: on-A-Table-0 $\wedge$ on-C-A-0 $\wedge$ on-B-Table-0 ...
- Goal State: on-A-B-k $\wedge$ on-B-C-k $\wedge$ on-C-Table-k
- For every action and for every step i (0 < i < k):
  Action-i $\Rightarrow$ Pr_1-i $\wedge$ Pr_2-i... $\wedge$ Ef_1-(i+1) $\wedge$ Ef_2-(i+1)
- Complete Exclusion Axioms (for every action and every step i)
  $\neg$ ActionA-i $\vee$ $\neg$ ActionB-i
  only one action occurs at each time point
- Frame Axioms! (for example: for A, B... that have effect on P)
  describe the predicate that are not affected by the actions
  ...

Encoding Plans as SAT

- Easy, there is only one Action-i true for any given i
- Why? Complete Exclusion Axioms

Solving SAT

- Davis-Putman
  - Transform to CNF (Conjuctive Normal Form)
  - Backtrack Search for Assignments to Literals
- Hillclimbing Local Search!
- Walksat
  - Select random truth assignment
  - With probability p flip a random variable
  - With probability $(1-p)$ flip the variable that minimized unsatisfied clauses

The Last Classical Story

- 1992 Kautz and Selman show first competitive SAT planner

Still takes too much memory and time compared to modern planners

- 1995-1997 Blum and Furst present GraphPlan

  GraphPlan propagates plan constraints

- 1998 **Blackbox**
  Combines Mutex Constraints of GraphPlan with SAT Solvers
  One of the best planners in 1998 Planning Competition!

The Last Classical Story

- 1992 Kautz and Selman show first competitive SAT planner

Still takes too much memory and time compared to modern planners

- 1995-1997 Blum and Furst present GraphPlan

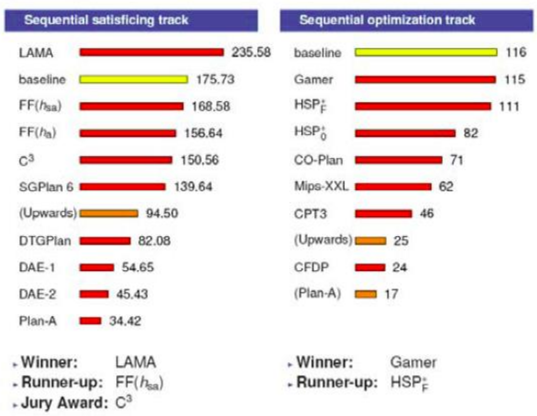
  GraphPlan propagates plan constraints

- 1998 Blackbox
  Combines Mutex Constraints of GraphPlan with SAT Solvers
  One of the best planners in 1998 Planning Competition!

- 2000-2002 FF (Hoffman)

- 2004-2006 **SatPlan extends Blackbox and takes 1st place**!

Who Won in 2008? FF/HSP Strike Back!



Who Won in 2011?

- Optimization Track: Fast Downward Stone Soup-1
- Sequential Track: Winner - LAMA 2011
  Runner up - Stone Soup

55 planners in total from 55 different people from 11 countries: Australia, Canada, China, France, Germany, India, Israel, Italy, Spain, UK and USA

Who Won in 2014?

- Sequential Optimal track: Winner - SymBA*-2, SymBA*-1
  Runner up – cGamer
- Sequential Multi-core track: Winner - ArvandHerd
  Runner up – IBaCoP
- Sequential Satisficing track: Winner – IBaCoP2
  Runner up - Mercury
- Agile track: Winner – YAHSP3
  Runner up – Madagascar-pC
- Temporal track: Winner – YAHSP3-MT
  Runner up – Temporal Fast Downward

67 planners in total from 66 people from 14 countries: Australia, Canada, Czech Republic, Finland, France, Germany, Iran, Israel, New Zealand, Spain, Switzerland, UK, Venezuela, and USA

*China(X), India(X), Italy(X)

Summary

Search
- Important Element of Planning
- Heuristics: Admissible / Informed
- A* is desirable but often too expensive
- Best-First Search & Hill Climbing tend to work well

Heuristic Planning
- Domain Dependent Heuristics
- Domain Independent Heuristics
- Solving Relaxed Problems to Guide Actual Solution
- GraphPlan as a Method for Relaxing Problems!

Satisfiability
- Our Final Approach to Planning
- Most Efficient when combined with GraphPlan!

PDDL

PDDL = Planning Domain Definition Language

- Components of a PDDL planning task:

-Objects: Things in the world
- Predicates: Object properties
- Initial state: The state of the world that we start in
- Goal specification: Things that we want to be true
- Actions/Operators: Ways of changing the state of the world

PDDL

How to use PDDL for planning problems?

- Two files

- A domain file: predicates and actions
- A problem file: objects, initial state and goal specification

```
(define (domain hanoi-domain)
  (:requirements :equality)
  (:predicates (disk ?x) (smaller ?x ?y) (on ?x ?y) (clear ?x))
  (:action move-disk
        :parameters (?disk ?below-disk ?new-below-disk)
        :precondition (and (disk ?disk)
                      (smaller ?disk ?new-below-disk)
                      (not (= ?new-below-disk ?below-disk))
                      (not (= ?new-below-disk ?disk))
                      (not (= ?below-disk ?disk))
                      (on ?disk ?below-disk)
                      (clear ?disk)
                      (clear ?new-below-disk))
        :effect (and (clear ?below-disk)
                  (on ?disk ?new-below-disk)
                  (not (on ?disk ?below-disk))
                  (not (clear ?new-below-disk))))
```


PDDL

How to use PDDL for planning problems?

- Two files

- A domain file: predicates and actions
- A problem file: objects, initial state and goal specification

```
(define (problem hanoi-problem)
  (:domain hanoi-domain)
  (:objects p1 p2 p3 d1 d2 d3)
  (:init (smaller d1 p1) (smaller d2 p1) (smaller d3 p1)
(smaller d1 p2) (smaller d2 p2) (smaller d3 p2) (smaller
d1 p3) (smaller d2 p3) (smaller d3 p3) (smaller d1 d2)
(smaller d1 d3) (smaller d2 d3) (clear p1) (clear p2)
(clear d1) (disk d1) (disk d2) (disk d3) (on d1 d2) (on d2
d3) (on d3 p3))
  (:goal (and (on d1 d2) (on d2 d3) (on d3 p1))))
```

PDDL

- Domain description

Domain name

Requirements

Actions

Predicates

```
(define (domain hanoi-domain)
 (:requirements :equality)
 (:predicates (disk ?x) (smaller ?x ?y) (on ?x ?y) (clear ?x))
 (:action move-disk
      :parameters (?disk ?below-disk ?new-below-disk)
      :precondition (and (disk ?disk)
                    (smaller ?disk ?new-below-disk)
                    (not (= ?new-below-disk ?below-disk))
                    (not (= ?new-below-disk ?disk))
                    (not (= ?below-disk ?disk))
                    (on ?disk ?below-disk)
                    (clear ?disk)
                    (clear ?new-below-disk))
      :effect (and (clear ?below-disk)
                (on ?disk ?new-below-disk)
                (not (on ?disk ?below-disk))
                (not (clear ?new-below-disk)))))
```

*Requirement flags allow a planner to quickly tell if it is likely to be able to hangle the domain
*Action effects can include universal quantifiers(i.e. forall), conditionals (e.g. when)

PDDL

- Problem description

Problem name

Objects

Domain

Initial state

Goal specification

```
(define (problem hanoi-problem)
 (:domain hanoi-domain)
 (:objects p1 p2 p3 d1 d2 d3)
 (:init (smaller d1 p1) (smaller d2 p1) (smaller
d3 p1) (smaller d1 p2) (smaller d2 p2) (smaller
d3 p2) (smaller d1 p3) (smaller d2 p3) (smaller
d3 p3) (smaller d1 d2) (smaller d1 d3) (smaller
d2 d3) (clear p1) (clear p2) (clear d1) (disk d1)
(disk d2) (disk d3) (on d1 d2) (on d2 d3) (on d3
p3))
 (:goal (and (on d1 d2) (on d2 d3) (on d3 p1))))
```

*domain must match the corresponding domain name