Fast-Forward Planning (Pre Lecture)

Dr. Neil T. Dantam

CSCI-534, Colorado School of Mines

Spring 2020

MINES

Introduction

FF

- FF is:
  - non-optimal
  - heuristic
  - forward search
- FF works well (fast) in practice

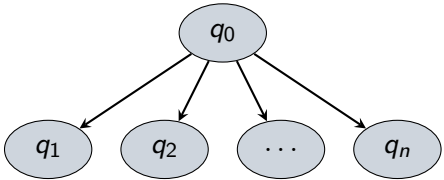
Outcomes

- Understand **hill-climbing** search
- Understand the **Relaxed Planning Graph** heurisitc
- Be able to apply the FF algorithm to planning domains

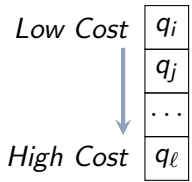
MINES

Outline

Optimal Heuristic Search: A^*



Priority Queue



$$w(q_i) = \underbrace{D(q_0, q_i)}_{\text{distance to } q_i} + \underbrace{h(q_i, q_{\text{goal}})}_{\text{heuristic to goal}}$$

Pro: Optimal (when admissible). Con: Large queues.

Non-optimal Heuristic Search

Greedy Search

- Select best (non-admissible) **frontier** node
- **Pro:** Easier / faster with non-admissible heuristic
- **Con:** Not Optimal

Hill-Climbing

- Select best (non-admissible) **child** node
- **Pro:** Quickly expands nodes / no large queue
- **Con:** Not Optimal / no backtracking

Trade-off vs A^*:
Optimality vs. Efficiency

Gradient Descent vs. Hill-climbing

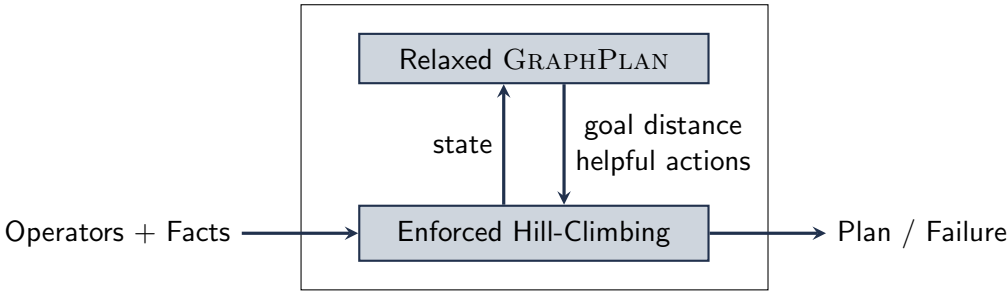
Gradient Descent

- **Given**: gradient function (i.e., a derivative)
- **General Procedure:**
  1. Compute the gradient at the current point
  2. Move some amount in the direction of the gradient
  3. Repeat until gradient is 0
- Necessarily continuous

Hill-Climbing

- **Given**: reward function (i.e., a **not** a derivative)
- **General Procedure:**
  1. Evaluate reward of children near the current point
  2. Move to the best child
  3. Repeat until no better child is found
- Continuous or discrete

Fast-Forward (FF) Outline



Enforced Hill-Climbing: Forward search for nearby child that with better heuristic

Relaxed GraphPlan: Informs search with heuristics: promising successors, helpful actions

Outline

Relaxed Problems

overall idea

Convert complex problem into simpler problem.

Relaxed Actions

Remove negated effects ("delete list")

Example: Relaxed Planning Domain

Domain

```
(define (domain cake-domain)
  (:predicates (have ?x)
               (eaten ?x))
  (:action eat   :parameters (?x)
           :precondition (have ?x)
           :effect (and (not (have ?x))
                        (eaten ?x)))
  (:action bake  :parameters (?x)
           :precondition (not (have ?x))
           :effect (and (have ?x))))
```

Relaxed Domain

```
(define (domain cake-domain)
  (:predicates (have ?x)
               (eaten ?x))
  (:action eat   :parameters (?x)
           :precondition (have ?x)
           :effect (and (eaten ?x)))
  (:action bake  :parameters (?x)
           :precondition (not (have ?x))
           :effect (and (have ?x))))
```

Exercise: Relaxed Planning Domain

Original Domain

```
(define (domain air-cargo)
  (:predicates (plane ?x) (cargo ?x)
               (airport ?x) (at ?x ?y))
  (:action fly  :parameters (?p ?x ?y)
           :precondition
           (and (plane ?p) (airport ?x) (airport ?y)
                (at ?p ?x))
           :effect (and (not (at ?p ?x)) (at ?p ?y)))
  (:action load  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?a) (at ?p ?a))
           :effect (and (not (at ?c ?a)) (at ?c ?p)))
  (:action unload  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?p) (at ?p ?a))
           :effect (and (not (at ?c ?p)) (at ?c ?a))))
```

Exercise: Relaxed Planning Domain

Relaxed Domain

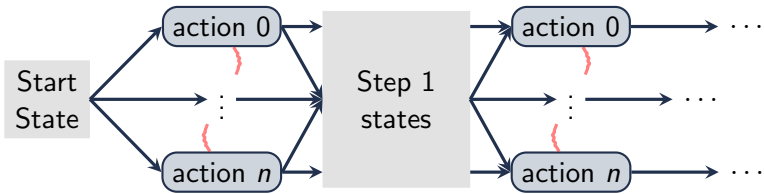
Planning Graph Overview

Nodes: propositions $\cup$ actions $\cup$ $\{\text{nop}\}$

Edges: Transition: connects actions with precondition and effect propositions, $(p \times a) \cup (a \times p)$

Mutex: conflicts (mutual exclusion) between actions and edges, $(p \times p) \cup (a \times a)$

Levels: Sequences of levels: timesteps



Heuristic for the structure of the planning domain

Example: Cake Domain

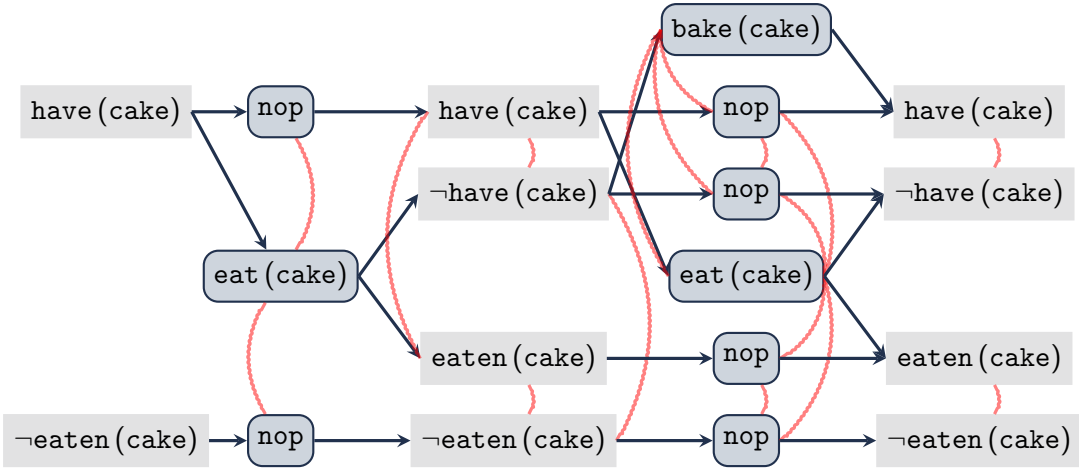
Operators

```
(define (domain cake-domain)
  (:predicates (have ?x)
               (eaten ?x))
  (:action eat  :parameters (?x)
           :precondition (have ?x)
           :effect (and (not (have ?x))
                        (eaten ?x)))
  (:action bake  :parameters (?x)
           :precondition (not (have ?x))
           :effect (and (have ?x))))
```

Facts

```
(define (problem have-and-eat-cake)
  (:domain cake-domain)
  (:objects cake)
  (:init (have cake))
  (:goal (and (have cake)
              (eaten cake))))
```

Example: Cake Planning Graph



Example: Relaxed Cake Domain

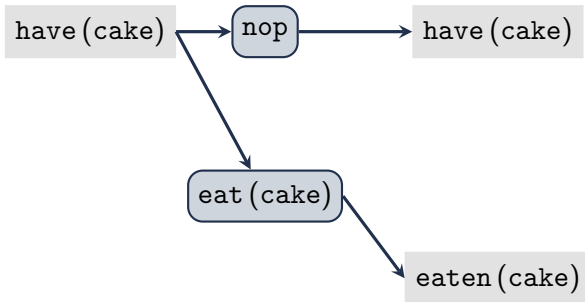
Operators

```
(define (domain cake-domain)
  (:predicates (have ?x)
               (eaten ?x))
  (:action eat  :parameters (?x)
           :precondition (have ?x)
           :effect (and (eaten ?x)))
  (:action bake  :parameters (?x)
           :precondition (not (have ?x))
           :effect (and (have ?x))))
```

Facts

```
(define (problem have-and-eat-cake)
  (:domain cake-domain)
  (:objects cake)
  (:init (have cake))
  (:goal (and (have cake)
              (eaten cake))))
```

Example: Cake Relaxed Planning Graph



Exercise: Air Cargo

Operators

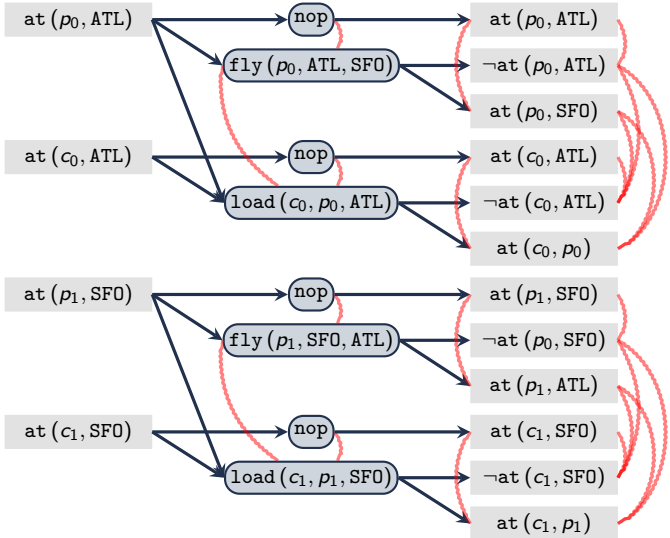
```
(define (domain air-cargo)
  (:predicates (plane ?x) (cargo ?x)
               (airport ?x) (at ?x ?y))
  (:action fly  :parameters (?p ?x ?y)
           :precondition
           (and (plane ?p) (airport ?x) (airport ?y)
                (at ?p ?x))
           :effect (and (not (at ?p ?x)) (at ?p ?y)))
  (:action load  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?a) (at ?p ?a))
           :effect (and (not (at ?c ?a)) (at ?c ?p)))
  (:action unload  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?p) (at ?p ?a))
           :effect (and (not (at ?c ?p)) (at ?c ?a))))
```

Facts

```
(define (problem air)
  (:domain air-cargo)
  (:objects cargo-0 cargo-1
            plane-0 plane-1
            ATL SFO)
  (:init (cargo cargo-0)
         (cargo cargo-1)
         (plane plane-0)
         (plane plane-1)
         (airport ATL)
         (airport SFO)
         (at plane-0 ATL)
         (at plane-1 SFO)
         (at cargo-0 ATL)
         (at cargo-1 SFO))
  (:goal (and (at cargo-0 SFO)
              (at cargo-1 ATL))))
```

MINES

Exercise: Air Cargo



Exercise: Relaxed Air Cargo

Operators

```
(define (domain air-cargo)
  (:predicates (plane ?x) (cargo ?x)
               (airport ?x) (at ?x ?y))
  (:action fly  :parameters (?p ?x ?y)
           :precondition
           (and (plane ?p) (airport ?x) (airport ?y)
                (at ?p ?x))
           :effect (and (at ?p ?y)))
  (:action load  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?a) (at ?p ?a))
           :effect (and (at ?c ?p)))
  (:action unload  :parameters (?c ?p ?a)
           :precondition
           (and (cargo ?c) (plane ?p) (airport ?a)
                (at ?c ?p) (at ?p ?a))
           :effect (and (at ?c ?a))))
```

Facts

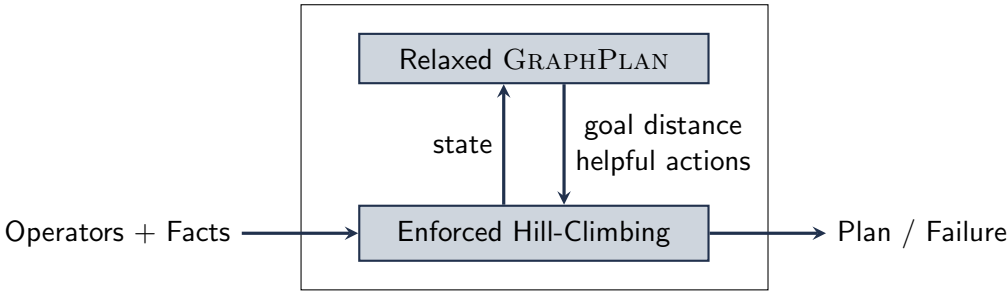
```
(define (problem air)
  (:domain air-cargo)
  (:objects cargo-0 cargo-1
            plane-0 plane-1
            ATL SFO)
  (:init (cargo cargo-0)
         (cargo cargo-1)
         (plane plane-0)
         (plane plane-1)
         (airport ATL)
         (airport SFO)
         (at plane-0 ATL)
         (at plane-1 SFO)
         (at cargo-0 ATL)
         (at cargo-1 SFO))
  (:goal (and (at cargo-0 SFO)
              (at cargo-1 ATL))))
```

MINES

Exercise: Relaxed Air Cargo Planning Graph

Outline

Fast-Forward (FF) Outline



Enforced Hill-Climbing: Forward search for nearby child that with better heuristic

Relaxed GraphPlan: Informs search with heuristics, promising successors, helpful actions

Enforced Hill-Climbing

Procedure FF-enforced-hill-climbing(S)

```
1 plan ← ();
2 while heuristic(S) ≥ 0 do
3     S' ← Breadth-First Search until heuristic(S') < heuristic(S);
4     if no S' found then
5         return failure;
6     Append path from S to S' onto plan;
7     S ← S';
```

Relaxed GraphPlan Heuristic

1. From state S, construct Relaxed Planning Graph to Goal
2. Extract Relaxed Plan:
   - 2.1 Work backwards from last level
   - 2.2 At each level i, if a goal proposition g exists at i, but not $i-1$, select an action that achieves g
3. `heuristic`(S): number of actions in relaxed plan

Can compute relaxed plan in polynomial time

"Helpful Action" Heuristic

- $H(S)$: the most promising ("helpful") actions at state S
- $H(S) \equiv \left\{ a \mid \overbrace{(\texttt{pre}(a) \subseteq S)}^{\text{precondition holds}} \wedge \overbrace{(\texttt{add}(a) \cap G \neq \emptyset)}^{\text{goal progress}} \right\}$
- Usage: In BFS, only take actions in $H(S)$

Breadth-First Search

1. Remove next state S' from queue
2. Evaluate heuristic cost via relaxed GraphPlan
3. If S' is better than S, return S'
4. Else: Add successors of S' in $H(S')$ to queue and repeat

Summary

FF Overview

Relaxed Planning Graphs

FF Details

References

- Hoffmann J. FF: The fast-forward planning system. AI magazine. 2001 Sep 15.