Artificial Intelligence

13. Planning, Part I: Framework

How to Describe Arbitrary Search Problems

Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2014

Agenda

Reminder: Classical Search Problems (Chapters 4 and 5)

- States: Card positions (*position_Jspades=Qhearts*).
- Actions: Card moves (*move_Jspades_Qhearts_freecell4*).
- Initial state: Start configuration.
- Goal states: All cards "home".
- Solution: Card moves solving this game.

Planning

Ambition:

Write one program that can solve all classical search problems.

Reminder: **(Chapter 4)**

- The **blackbox description** of a problem Π is an API (a programming interface) providing functionality allowing to construct the state space: InitialState(), GoalTest(s), ...

  $\rightarrow$ "Specifying the problem" $=$ programming the API.

- The **declarative description** of Π comes in a problem description language. This allows to implement the API, and much more.

  $\rightarrow$ "Specifying the problem" $=$ writing a problem description.

$\rightarrow$ Here, "problem description language" $=$ planning language.

"Planning Language"?

How does a planning language describe a problem?

- A *logical description* of the possible states (vs. Blackbox: data structures). E.g.: predicate $Eq(.,.)$.
- A *logical description* of the initial state I (vs. data structures). E.g.: $Eq(x, 1)$.
- A *logical description* of the goal condition G (vs. a goal-test function). E.g.: $Eq(x, 2)$.
- A *logical description* of the set A of actions in terms of preconditions and effects (vs. functions returning applicable actions and successor states). E.g.: "increment x: pre $Eq(x, 1)$, eff $Eq(x, 2) \wedge \neg Eq(x, 1)$".

$\rightarrow$ Solution (plan) $=$ sequence of actions from A, transforming I into a state that satisfies G. E.g.: "increment x".

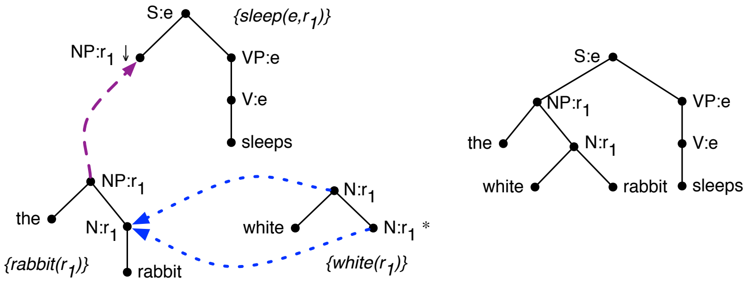
“Planning Language”?

Disclaimer:

→ Planning languages go way beyond classical search problems. There are variants for inaccessible, stochastic, dynamic, continuous, and multi-agent settings.

- We focus on classical search for simplicity (combined with practical relevance).
- For a comprehensive overview, see [Ghallab *et al.* (2004)].

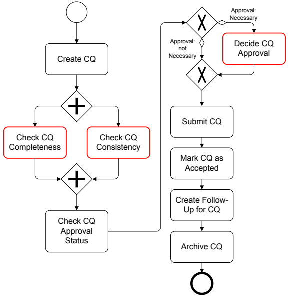
Application: Natural Language Generation



- **Input:** Tree-adjoining grammar, intended meaning.
- **Output:** Sentence expressing that meaning.

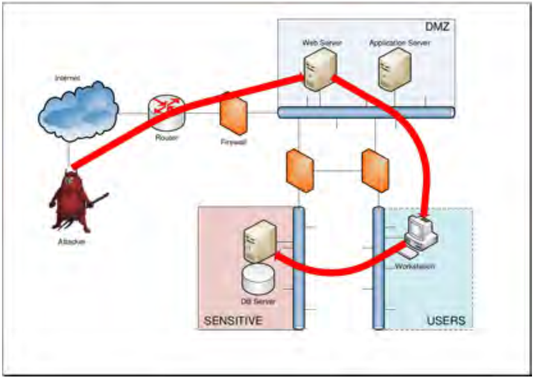
Application: Business Process Templates at SAP

Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived



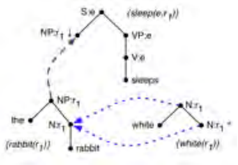
- **Input:** SAP-scale model of behavior of activities on Business Objects, process endpoint.
- **Output:** Process template leading to this point.

Application: Automatic Hacking



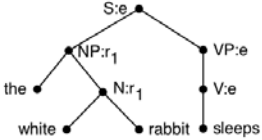
- **Input:** Network configuration, location of sensible data.
- **Output:** Sequence of exploits giving access to that data.

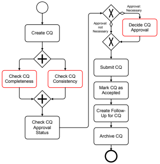
Planning!

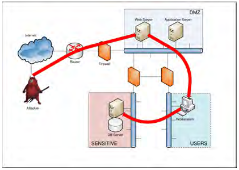


Action name	precondition	effect
Check CQ Completeness	CQ.archiving:notArchived	CQ.completeness:complete OR CQ.completeness:notComplete
Check CQ Consistency	CQ.archiving:notArchived	CQ.consistency:consistent OR CQ.consistency:notConsistent
Check CQ Approval Status	CQ.archiving:notArchived AND CQ.approval:notChecked AND CQ.completeness:complete AND CQ.consistency:consistent	CQ.approval:necessary OR CQ.approval:notNecessary
Decide CQ Approval	CQ.archiving:notArchived AND CQ.approval:necessary	CQ.approval:granted OR CQ.approval:notGranted
Submit CQ	CQ.archiving:notArchived AND (CQ.approval:notNecessary OR CQ.approval:granted)	CQ.submission:submitted
Mark CQ as Accepted	CQ.archiving:notArchived AND CQ.submission:submitted	CQ.acceptance:accepted
Create Follow-Up for CQ	CQ.archiving:notArchived AND CQ.acceptance:accepted	CQ.followUp:documentCreated
Archive CQ	CQ.archiving:notArchived	CQ.archiving:archived

Planning Domain Definition Language (PDDL) $\mapsto$ Planning System







Reminder: General Problem Solving, Pros and Cons

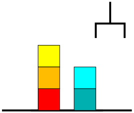
- **Powerful:** In some applications, generality is absolutely necessary. (E.g. SAP)
- **Quick:** Rapid prototyping: 10s lines of problem description vs. 1000s lines of C++ code. (E.g. language generation)
- **Flexible:** Adapt/maintain *the description*. (E.g. network security)
- **Intelligent:** Determines automatically how to solve a complex problem effectively! (The ultimate goal, no?!)

- **Efficiency loss:** Without any domain-specific knowledge about Chess, you don't beat Kasparov . . .

  $\rightarrow$ Trade-off between "automatic and general" vs. "manualwork but effective".

How to make fully automatic algorithms effective?

ps. "Making Fully Automatic Algorithms Effective"



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states	blocks	states
1	1	9	4596553
2	3	10	58941091
3	13	11	824073141
4	73	12	12470162233
5	501	13	202976401213
6	4051	14	3535017524403
7	37633	15	65573803186921
8	394353	16	1290434218669921

→ State spaces typically are huge even for simple problems.

→ In other words: Even solving "simple problems" automatically (without help from a human) requires a form of intelligence. With blind search, even the largest super-computer in the world won't scale beyond 20 blocks!

Our Agenda for This Chapter

- **The History of Planning:** How did this come about?

  → Gives you some background, and motivates our choice to focus on heuristic search.

- **The STRIPS Planning Formalism:** Which concrete planning formalism will we be using?

  → Lays the framework we'll be looking at.

- **The Complexity of (STRIPS) Planning:** How complex are the different decision problems involved?

  → This is needed to distinguish different forms of planning, and serves as the basic tool to design heuristic functions.

- **The PDDL Language:** What do the input files for off-the-shelf planning software look like?

  → So you can actually play around with such software. (Exercises!)

In the Beginning . . .

. . . Man invented Robots:

"Planning" as in "the making of plans by an autonomous robot".

In a little more detail:

- Newell and Simon (1963) introduced general problem solving.
- *. . . not much happened (well not much we still speak of today) . . .*
- Stanford Research Institute developed a robot named "Shakey".
- They needed a "planning" component taking decisions.
- They took inspiration from general problem solving and theorem proving, and called the resulting algorithm "STRIPS".

And then:

History of Planning Algorithms

Compilation into Logics/Theorem Proving:

- **Popular when:** Stone Age – 1990.
- **Approach:** *From planning task description, generate PL1 formula* φ *that is satisfiable iff there exists a plan; use a theorem prover on* φ.
- **Keywords/cites:** Situation calculus, frame problem, . . .

Partial-Order Planning:

- **Popular when:** 1990 – 1995.
- **Approach:** *Starting at goal, extend partially ordered set of actions by inserting achievers for open sub-goals, or by adding ordering constraints to avoid conflicts.*
- **Keywords/cites:** UCPOP [Penberthy and Weld (1992)], causal links, flaw-selection strategies, . . .

History of Planning Algorithms, ctd.

GraphPlan:

- **Popular when:** 1995 – 2000.
- **Approach:** *In a forward phase, build a layered "planning graph" whose "time steps" capture which pairs of actions can achieve which pairs of facts; in a backward phase, search this graph starting at goals and excluding options proved to not be feasible.*
- **Keywords/cites:** [Blum and Furst (1995, 1997); Koehler *et al.* (1997)], action/fact mutexes, step-optimal plans, ...

Planning as SAT:

- **Popular when:** 1996 – today.
- **Approach:** *From planning task description, generate propositional CNF formula φ_k that is satisfiable iff there exists a plan with k steps; use a SAT solver on φ_k, for different values of k.*
- **Keywords/cites:** [Kautz and Selman (1992, 1996); Rintanen *et al.* (2006); Rintanen (2010)], SAT encoding schemes, BlackBox, ...

History of Planning Algorithms, ctd.

Planning as Heuristic Search:

- **Popular when:** 1999 – today.
- **Approach:** Devise a method $\mathcal{R}$ to simplify ("relax") any planning task Π; given Π, solve $\mathcal{R}(\Pi)$ to generate a heuristic function h for informed search.
- **Keywords/cites:** [Bonet and Geffner (1999); Haslum and Geffner (2000); Bonet and Geffner (2001); Hoffmann and Nebel (2001); Edelkamp (2001); Gerevini *et al.* (2003); Helmert (2006); Helmert *et al.* (2007); Helmert and Geffner (2008); Karpas and Domshlak (2009); Helmert and Domshlak (2009); Richter and Westphal (2010); Nissim *et al.* (2011); Katz *et al.* (2012); Keyder *et al.* (2012); Katz *et al.* (2013); Katz and Hoffmann (2013)], critical path heuristics, ignoring delete lists, relaxed plans, landmark heuristics, abstractions, . . .

The International Planning Competition (IPC)

Competition?

"Run competing planners on a set of benchmarks devised by the IPC organizers. Give awards to the most effective planners."

- 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014
- PDDL [McDermott and others (1998); Fox and Long (2003); Hoffmann and Edelkamp (2005); Gerevini *et al.* (2009)]
- ≈ 50 domains, $\gg 1000$ instances, 74 (!!) planners in 2011
- Optimal track vs. satisficing track
- Various others: uncertainty, learning, . . .

`http://ipc.icaps-conference.org/`

"STRIPS" Planning

- STRIPS = Stanford Research Institute Problem Solver.

STRIPS is the simplest possible (reasonably expressive) logics-based planning language.

- STRIPS has only Boolean variables: propositional logic atoms.
- Its preconditions/effects/goals are as canonical as imaginable:
  - Preconditions, goals: conjunctions of positive atoms.
  - Effects: conjunctions of literals (positive or negated atoms).
- We use the common special-case notation for this simple formalism.
- I'll outline some extensions beyond STRIPS later on, when we discuss PDDL.

→ **Historical note:** STRIPS [Fikes and Nilsson (1971)] was originally a planner (cf. Shakey), whose language actually wasn't quite that simple.

STRIPS Planning: Syntax

Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple* $\Pi = (P, A, I, G)$ *where:*

- P *is a finite set of facts (aka propositions).*
- A *is a finite set of actions; each* $a \in A$ *is a triple* $a = (pre_a, add_a, del_a)$ *of subsets of* P *referred to as the action's precondition, add list, and delete list respectively; we require that* $add_a \cap del_a = \emptyset$*.*
- $I \subseteq P$ *is the initial state.*
- $G \subseteq P$ *is the goal.*

We will often give each action $a \in A$ *a name (a string), and identify* a *with that name.*

Note: We assume, for simplicity, that every action has cost 1. (Uniform costs, cf. **Chapter 4**.)

"TSP" in Australia

[→ Strictly speaking, this is not actually a **TSP** problem instance; simplified/adapted for illustration.]

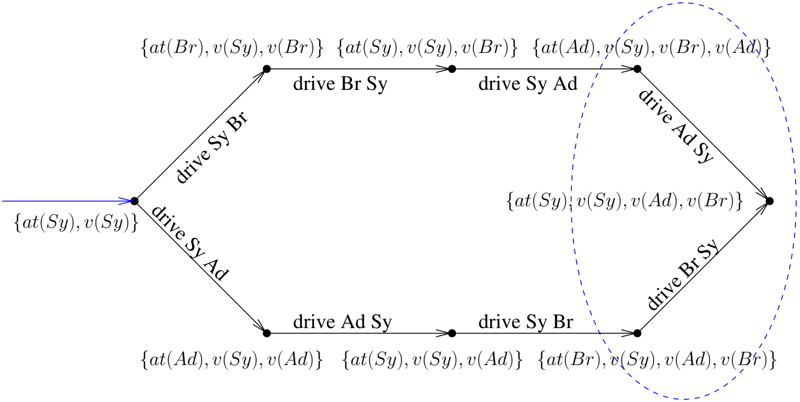
STRIPS Encoding of "TSP"

- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G:
  $\{at(Sydney)\} \cup \{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.
- Plan: $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.

STRIPS Encoding of Simplified "TSP"

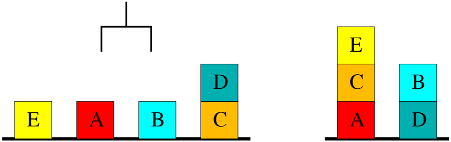
- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a: $\{at(x)\}$.
  Add list add_a: $\{at(y), visited(y)\}$.
  Delete list del_a: $\{at(x)\}$.

STRIPS Encoding of Simplified "TSP": State Space



→ Is this actually the state space? No, only the reachable part. E.g., Θ_Π also includes the states $\{v(Sy)\}$ and $\{at(Sy), at(Br)\}$.

(Oh no it's) The Blocksworld



Initial State Goal State

- Facts: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \dots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- Goal: $\{on(E, C), on(C, A), on(B, D)\}$.
- Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$? $pre : \{holding(x), clear(y)\}$
  $add : \{on(x, y), armEmpty()\}$
  $del : \{holding(x), clear(y)\}$.

Questionnaire

Question!

Which are correct encodings of the STRIPS Blocksworld $pickup(x)$ action schema?

(A): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x)\})$.

(B): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{armEmpty()\})$.

(C): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty(), clear(x)\})$.

(D): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty()\})$.

Questionnaire

Question!

Which are correct encodings of the STRIPS Blocksworld $pickup(x)$ action schema?

(A): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x)\})$.

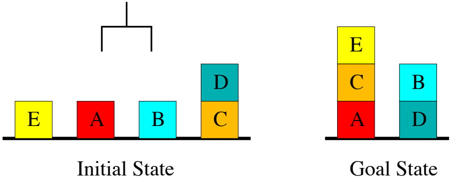
(B): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{armEmpty()\})$.

(C): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty(), clear(x)\})$.

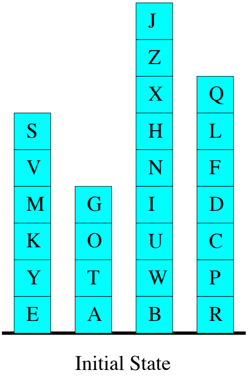
(D): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty()\})$.

→ (A): No, must delete $armEmpty()$. (B): No, must delete $onTable(x)$. (C), (D): Both yes: We can, but don't have to, encode the *single-arm* Blocksworld so that the block currently in the hand is not clear.

The Blocksworld is Hard?



The Blocksworld is Hard!



PDDL Quick Facts

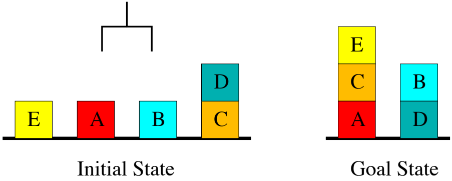
PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

A PDDL planning task comes in two pieces:

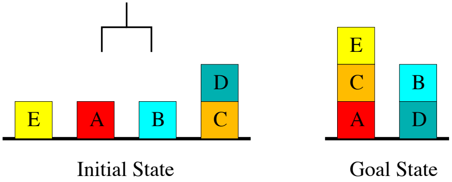
- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each benchmark domain has *one* domain file.

The Blocksworld in PDDL: Domain File




```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
...
```

The Blocksworld in PDDL: Problem File




```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

Summary

- General problem solving attempts to develop solvers that perform well across a large class of problems.
- Planning, as considered here, is a form of general problem solving dedicated to the class of classical search problems. (Actually, we also address inaccessible, stochastic, dynamic, continuous, and multi-agent settings.)
- Heuristic search planning has dominated the International Planning Competition (IPC). We focus on it here.
- STRIPS is the simplest possible, while reasonably expressive, language for our purposes. It uses Boolean variables (facts), and defines actions in terms of precondition, add list, and delete list.
- Plan existence (bounded or not) is **PSPACE**-complete to decide for STRIPS. If we bound plans polynomially, we get down to **NP**-completeness.
- PDDL is the de-facto standard language for describing planning problems.

Artificial Intelligence

14. Planning, Part II: Algorithms

How to *Solve* Arbitrary Search Problems

Jörg Hoffmann

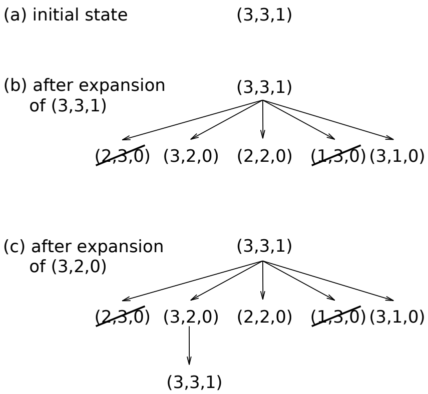
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summer Term 2014

Agenda

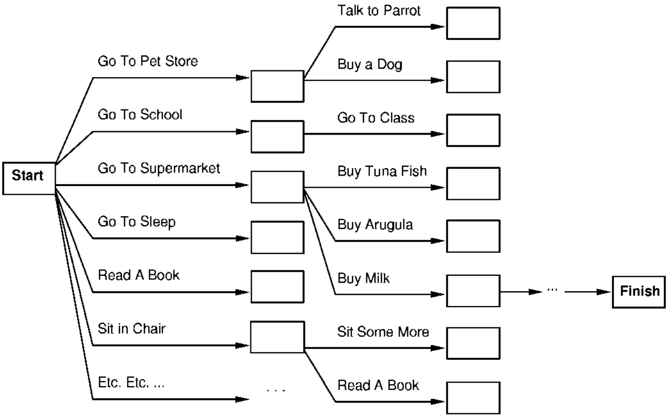
Reminder: Search

→ Starting at initial state, produce all successor states step by step:



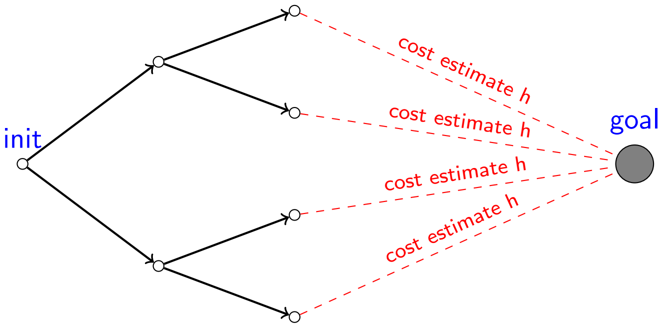
→ In planning, this is referred to as forward search, or forward state-space search.

Search in the State Space?



→ Use heuristic function to guide the search towards the goal!

Reminder: Informed Search



$\rightarrow$ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

Live Demo vs. Breadth-First Search:

`http://qiao.github.io/PathFinding.js/visual/`

Reminder: Heuristic Functions

Definition (Heuristic Function). *Let Π be a planning task with states S. A heuristic function, short heuristic, for Π is a function $h : S \mapsto \mathbb{N}_0^+ \cup \{\infty\}$ so that $h(s) = 0$ whenever s is a goal state.*

$\rightarrow$ Exactly like our definition from **Chapter 5**. Except, because we assume uniform costs here, we use $\mathbb{N}_0^+$ instead of $\mathbb{R}_0^+$.

Definition (h^*, Admissibility). *Let Π be a planning task with states S. The perfect heuristic h^* assigns every $s \in S$ the length of a shortest path from s to a goal state, or ∞ if no such path exists. A heuristic function h for Π is admissible if, for all $s \in S$, we have $h(s) \leq h^*(s)$.*

$\rightarrow$ Exactly like our definition from **Chapter 5**, except for path *length* instead of path *cost* (cf. above).

$\rightarrow$ In all cases, we attempt to approximate $h^*(s)$, the length of an optimal plan for s. Some algorithms guarantee to lower-bound $h^*(s)$.

Reminder: Greedy Best-First Search and A^*

Duplicate elimination omitted for simplicity:

function Greedy Best-First Search $[A^*]$*(problem)* **returns** a solution, or failure
  node $\leftarrow$ a node n with n.*state*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending h $[g+h]$, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n,a*)*
      Insert(n', $h(n')$ $[g(n')+h(n')]$, *frontier*)

→ Is Greedy Best-First Search optimal? No $\Longrightarrow$ *satisficing* planning.

→ Is A^* optimal? Yes, but only if h is admissible $\Longrightarrow$ *optimal* planning, **with such** h.

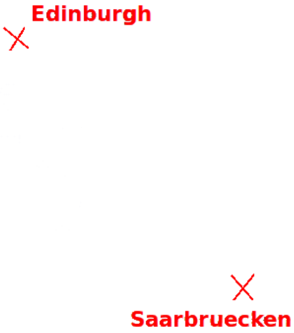
Our Agenda for This Chapter

- **How to Relax:** How to relax a problem?
  → Basic principle for generating heuristic functions.
- **The Delete Relaxation:** How to relax a planning problem?
  → The delete relaxation is the most successful method for the *automatic* generation of heuristic functions. It is a key ingredient to almost all IPC winners of the last decade. It relaxes STRIPS planning tasks by ignoring the delete lists.
- **The h^+ Heuristic:** What is the resulting heuristic function?
  → h^+ is the "ideal" delete relaxation heuristic.
- **Approximating h^+:** How to actually compute a heuristic?
  → Turns out that, in practice, we must approximate h^+.
- **An Overview of Advanced Results:** And what else can we do?
  → This section gives a brief glimpse into the state of the art in the area as a whole.

Reminder: Heuristic Functions from Relaxed Problems

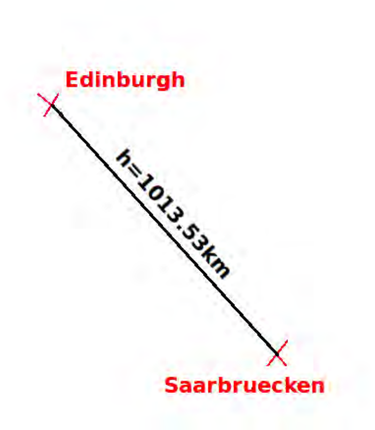
Problem II: Find a route from Saarbruecken To Edinburgh.

Reminder: Heuristic Functions from Relaxed Problems



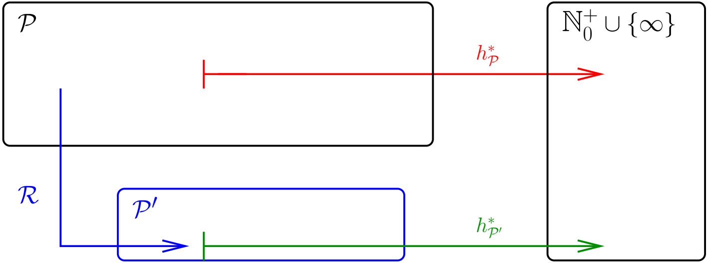
Relaxed Problem Π': Throw away the map.

Reminder: Heuristic Functions from Relaxed Problems



Heuristic function h: Straight line distance.

How to Relax



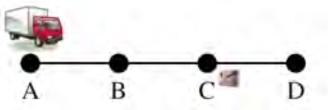
- You have a class $\mathcal{P}$ of problems, whose perfect heuristic $h^*_{\mathcal{P}}$ you wish to estimate.
- You define a class $\mathcal{P}'$ of *simpler problems*, whose perfect heuristic $h^*_{\mathcal{P}'}$ can be used to *estimate* $h^*_{\mathcal{P}}$.
- You define a transformation – the relaxation mapping $\mathcal{R}$ – that maps instances $\Pi \in \mathcal{P}$ into instances $\Pi' \in \mathcal{P}'$.
- Given $\Pi \in \mathcal{P}$, you let $\Pi' := \mathcal{R}(\Pi)$, and estimate $h^*_{\mathcal{P}}(\Pi)$ by $h^*_{\mathcal{P}'}(\Pi')$.

Relaxation in Route-Finding

- Problem class $\mathcal{P}$: Route finding.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length of a shortest route.
- Simpler problem class $\mathcal{P}'$: Route finding on an empty map.
- Perfect heuristic $h^*_{\mathcal{P}'}$ for $\mathcal{P}'$: Straight-line distance.
- Transformation $\mathcal{R}$: Throw away the map.

How to Relax in Planning?

Another Example: "Logistics"



- Facts P: $\{truck(x) \mid x \in \{A, B, C, D\}\} \cup pack(x) \mid x \in \{A, B, C, D, T\}\}$.
- Initial state I: $\{truck(A), pack(C)\}$.
- Goal G: $\{truck(A), pack(D)\}$.
- Actions A: (Notated as "precondition $\Rightarrow$ adds, $\neg$ deletes")
  - $drive(x, y)$, where x, y have a road: "$truck(x) \Rightarrow truck(y), \neg truck(x)$".
  - $load(x)$: "$truck(x), pack(x) \Rightarrow pack(T), \neg pack(x)$".
  - $unload(x)$: "$truck(x), pack(T) \Rightarrow pack(x), \neg pack(T)$".

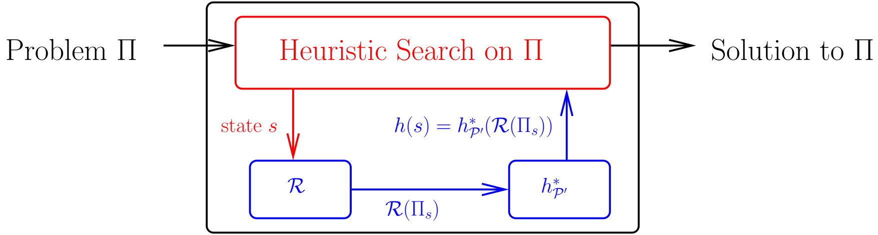
Example "Only-Adds" Relaxation: Drop the preconditions and deletes!

"$drive(x, y)$: $\Rightarrow truck(y)$"; "$load(x)$: $\Rightarrow pack(T)$"; "$unload(x)$: $\Rightarrow pack(x)$".

$\rightarrow$ Heuristic value for I is? 1: A plan for the relaxed task is $\langle unload(D) \rangle$.

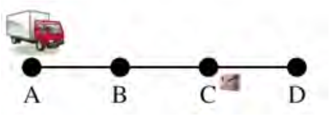
How to Relax During Search: Overview

Attention! Search uses the real (un-relaxed) Π. The relaxation is applied (e.g., in Only-Adds, the simplified actions are used) **only within the call to** $h(s)$**!!!**



- Here, Π_s is Π with initial state replaced by s, i.e., $\Pi = (P, A, I, G)$ changed to (P, A, s, G): The task of finding a plan for search state s.
- A common student mistake is to instead apply the relaxation once to the whole problem, then doing the whole search "within the relaxation".
- The next slide illustrates the correct search process in detail.

How to Relax During Search: Only-Adds



Real problem:

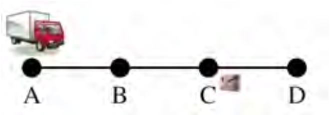
- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here

AC

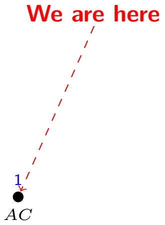
How to Relax During Search: Only-Adds



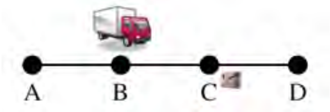
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: *add*.
- $h^{\mathcal{R}}(s) = 1$: $\langle ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds



Real problem:

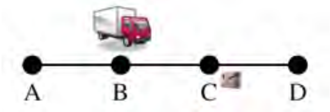
- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here

$$\overset{1}{AC} \xrightarrow{drAB} BC$$

How to Relax During Search: Only-Adds

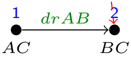


Relaxed problem:

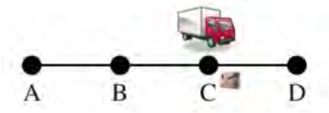
- State s: BC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: e.g. $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



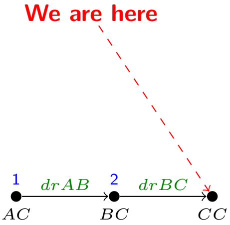
How to Relax During Search: Only-Adds



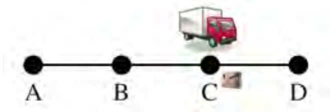
Real problem:

- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search:
(tie-breaking: alphabetic)



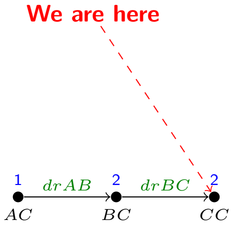
How to Relax During Search: Only-Adds



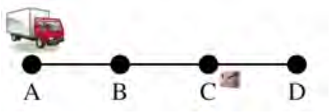
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: e.g. $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

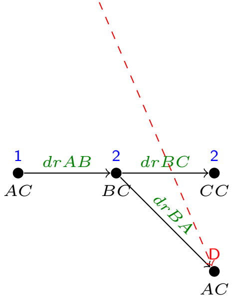


Real problem:

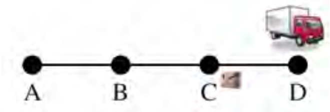
- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



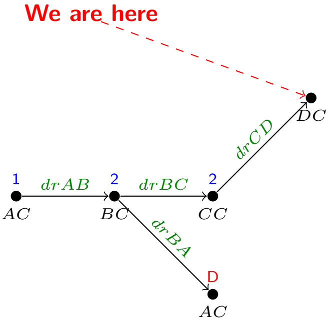
How to Relax During Search: Only-Adds



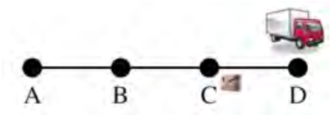
Real problem:

- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search:
(tie-breaking: alphabetic)



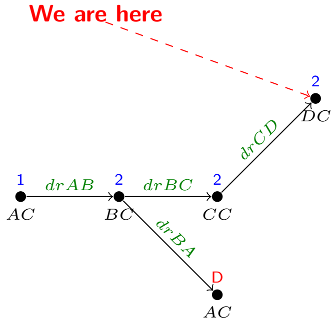
How to Relax During Search: Only-Adds



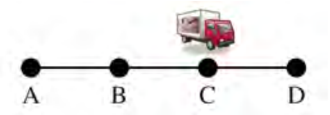
Relaxed problem:

- State s: DC; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: e.g. $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



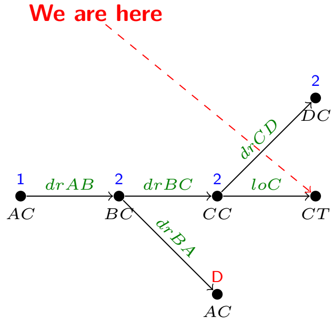
How to Relax During Search: Only-Adds



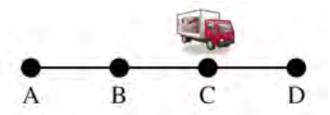
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search:
(tie-breaking: alphabetic)



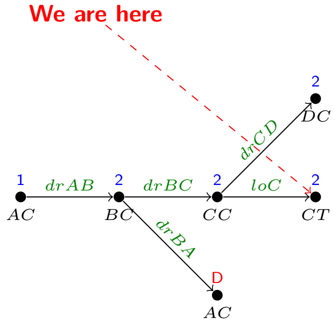
How to Relax During Search: Only-Adds



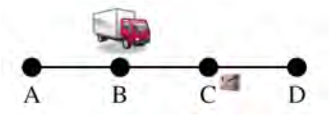
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: add.
- $h^{\mathcal{R}}(s) = 2$: e.g. $\langle drBA, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Only-Adds

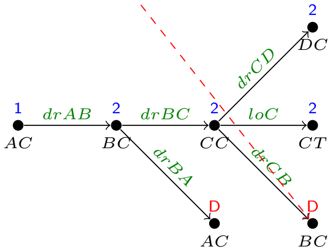


Real problem:

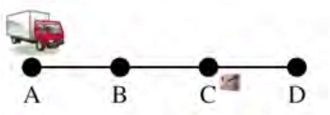
- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



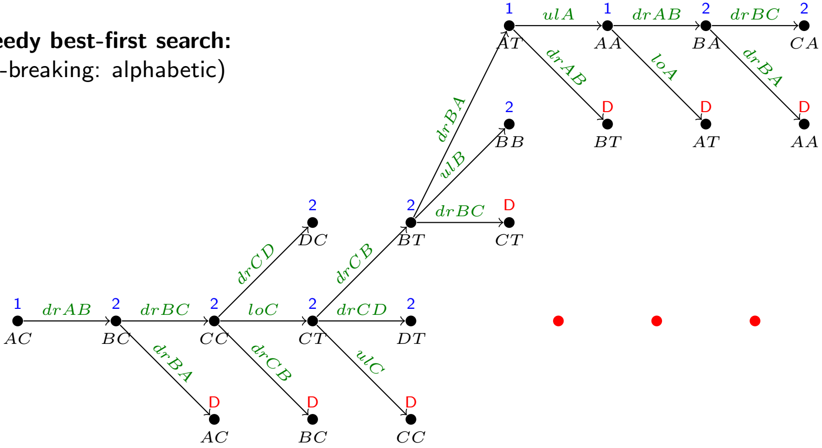
How to Relax During Search: Only-Adds



Real problem:

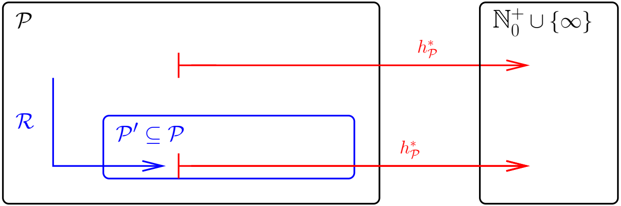
- Initial state I: AC; goal G: AD.
- Actions A: $\color{red}{pre}$, add, $\color{red}{del}$.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



Note: Only-Adds is a "Native" Relaxation

Native Relaxations: Confusing special case where $\mathcal{P}' \subseteq \mathcal{P}$.



- Problem class $\mathcal{P}$: STRIPS planning tasks.
- Perfect heuristic $h^*_{\mathcal{P}}$ for $\mathcal{P}$: Length h^* of a shortest plan.
- Transformation $\mathcal{R}$: Drop the preconditions and delete lists.
- Simpler problem class $\mathcal{P}'$ is a special case of $\mathcal{P}$, $\mathcal{P}' \subseteq P$: STRIPS planning tasks with empty preconditions and delete lists.
- Perfect heuristic for $\mathcal{P}'$: Shortest plan for Only-Adds STRIPS task.

Questionnaire

Question!

Does Only-Adds yield a "good heuristic" (accurate goal distance estimates) in . . .

(A): Freecell?

(B): SAT? (#unsatisfied clauses)

(C): Blocksworld?

(D): Path Planning?

Questionnaire

Question!

Does Only-Adds yield a "good heuristic" (accurate goal distance estimates) in . . .

(A): Freecell?

(B): SAT? (#unsatisfied clauses)

(C): Blocksworld?

(D): Path Planning?

→ (A): No: The heuristic value does take into account how many cards are already "home", but it is completely independent of the placement of all the other cards. In particular, dead-ends are essential in Freecell but the heuristic is completely unable to detect any of them.

→ (B): No: Typically, it is easy to satisfy many clauses, but then satisfying the remaining ones involves re-doing the entire assignment. (Nevertheless, this heuristic is being used in local search for SAT!)

→ (C): No: If we build a goal-tower of size 100 on top of a single block that still needs to move elsewhere, then the heuristic value is 1.

→ (D): No! The heuristic remains constantly 1 until we reach the actual goal state.

How The Delete Relaxation Changes the World

Relaxation mapping $\mathcal{R}$ saying that:

"When the world changes, its previous state remains true as well."

Relaxed world: (before)

How The Delete Relaxation Changes the World

Relaxation mapping $\mathcal{R}$ saying that:

"When the world changes, its previous state remains true as well."

Relaxed world: (after)

The Delete Relaxation

Definition (Delete Relaxation). *Let $\Pi = (P, A, I, G)$ be a planning task. The delete-relaxation of Π is the task $\Pi^+ = (P, A^+, I, G)$ where $A^+ = \{a^+ \mid a \in A\}$ with $pre_{a^+} = pre_a$, $add_{a^+} = add_a$, and $del_{a^+} = \emptyset$.*

→ In other words, the class of simpler problems $\mathcal{P}'$ is the set of all STRIPS planning tasks with empty delete lists, and the relaxation mapping $\mathcal{R}$ drops the delete lists.

Definition (Relaxed Plan). *Let $\Pi = (P, A, I, G)$ be a planning task, and let s be a state. A relaxed plan for s is a plan for $(P, A, s, G)^+$. A relaxed plan for I is called a relaxed plan for Π.*

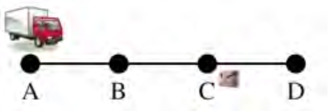
→ A relaxed plan for s is an action sequence that solves s when pretending that all delete lists are empty.

→ Also called delete-relaxed plan; "relaxation" is often used to mean "delete-relaxation" by default.

A Relaxed Plan for "TSP" in Australia

1. **Initial state:** $\{at(Sydney), visited(Sydney)\}$.
2. **Apply** $drive(Sydney, Brisbane)^+$**:** $\{at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
3. **Apply** $drive(Sydney, Adelaide)^+$**:** $\{at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
4. **Apply** $drive(Adelaide, Perth)^+$**:** $\{at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.
5. **Apply** $drive(Adelaide, Darwin)^+$**:** $\{at(Darwin), visited(Darwin), at(Perth), visited(Perth), at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane), at(Sydney), visited(Sydney)\}$.

A Relaxed Plan for "Logistics"



- Facts P: $\{truck(x) \mid x \in \{A,B,C,D\}\} \cup pack(x) \mid x \in \{A,B,C,D,T\}\}$.
- Initial state I: $\{truck(A), pack(C)\}$.
- Goal G: $\{truck(A), pack(D)\}$.
- Relaxed actions A^+: (Notated as "precondition $\Rightarrow$ adds")
  - $drive(x,y)^+$: "$truck(x) \Rightarrow truck(y)$".
  - $load(x)^+$: "$truck(x), pack(x) \Rightarrow pack(T)$".
  - $unload(x)^+$: "$truck(x), pack(T) \Rightarrow pack(x)$".

Relaxed plan:

$$\langle drive(A,B)^+, drive(B,C)^+, load(C)^+, drive(C,D)^+, unload(D)^+\rangle$$

$\rightarrow$ We don't need to drive the truck back, because "it is still at A".

PlanEx$^+$

Definition (Relaxed Plan Existence Problem). *By PlanEx$^+$, we denote the problem of deciding, given a planning task $\Pi = (P, A, I, G)$, whether or not there exists a relaxed plan for Π.*

$\rightarrow$ This is easier than PlanEx for general STRIPS!

Proposition (PlanEx$^+$ is Easy). *PlanEx$^+$ is a member of* **P**.

Proof. The following algorithm decides PlanEx$^+$:

$F := I$
while $G \not\subseteq F$ **do**
    $F' := F \cup \bigcup_{a \in A: pre_a \subseteq F} add_a$
    (*) **if** $F' = F$ **then return** "unsolvable" **endif**
    $F := F'$
endwhile
return "solvable"

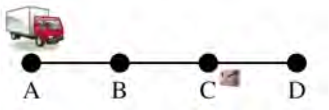
The algorithm terminates after at most $|P|$ iterations, and thus runs in polynomial time. Correctness: See slide 29.

Deciding PlanEx$^+$ in "TSP" in Australia

Iterations on F:

1. $\{at(Sydney), visited(Sydney)\}$
2. $\cup \{at(Adelaide), visited(Adelaide), \; at(Brisbane), visited(Brisbane)\}$
3. $\cup \{at(Darwin), visited(Darwin), \; at(Perth), visited(Perth)\}$

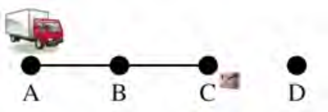
Deciding PlanEx$^+$ in "Logistics"



Iterations on F:

1. $\{truck(A), pack(C)\}$
2. $\cup \{truck(B)\}$
3. $\cup \{truck(C)\}$
4. $\cup \{truck(D), pack(T)\}$
5. $\cup \{pack(A), pack(B), pack(D)\}$

Deciding PlanEx$^+$ in Unsolvable "Logistics"



Iterations on F:

1. $\{truck(A), pack(C)\}$
2. $\cup \{truck(B)\}$
3. $\cup \{truck(C)\}$
4. $\cup \{pack(T)\}$
5. $\cup \{pack(A), pack(B)\}$
6. $\cup \emptyset$

Questionnaire

Question!

How does ignoring delete lists simplify Sokoban?

(A): Free positions remain free.

(B): You can walk through walls.

(C): A single action can push 2 stones at once.

(D): You will never "lock yourself in".

Questionnaire

Question!

How does ignoring delete lists simplify Sokoban?

(A): Free positions remain free.

(B): You can walk through walls.

(C): A single action can push 2 stones at once.

(D): You will never "lock yourself in".

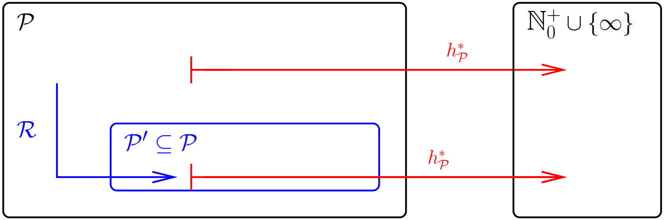
→ (A): Yes, when we move a stone into a free space, that space is still free afterwards.

→ (B): No, we dont get any new moves in the relaxation.

→ (C): Only if we give names to the stones. Within the relaxed problem, it may happen that two stones are in the same position, so in principle we can push them both. However, without distinguishing stone names, it is impossible to separate them again, so the two stones in fact become (behave in all relevant ways exactly like) a single stone.

→ (D): Yes, that's because of (A).

Hold on a Sec – Where are we?



- $\mathcal{P}$: STRIPS planning tasks; $h^*_\mathcal{P}$: Length h^* of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS planning tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: Length of a shortest *relaxed* plan $(h^* \circ \mathcal{R})$.

$\rightarrow$ PlanEx$^+$ is not actually what we're looking for. PlanEx$^+$ $=$ relaxed plan *existence*; we want relaxed plan *length*.

h^+: The Ideal Delete Relaxation Heuristic

Definition (Optimal Relaxed Plan). *Let $\Pi = (P, A, I, G)$ be a planning task, and let s be a state. An optimal relaxed plan for s is an optimal plan for $(P, A, s, G)^+$.*

→ Same as slide 22, just adding the word "optimal".

Here's what we're looking for:

Definition (h^+). *Let $\Pi = (P, A, I, G)$ be a planning task with states S. The ideal delete-relaxation heuristic h^+ for Π is the function $h^+ : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ where $h^+(s)$ is the length of an optimal relaxed plan for s if a relaxed plan for s exists, and $h^+(s) = \infty$ otherwise.*

→ In other words, $h^+ = h^* \circ \mathcal{R}$, cf. previous slide.

h^+ is Admissible

Lemma. *Let* $\Pi = (P, A, I, G)$ *be a planning task, and let* s *be a state. If* $\langle a_1, \dots, a_n \rangle$ *is a plan for* (P, A, s, G)*, then* $\langle a_1^+, \dots, a_n^+ \rangle$ *is a plan for* $(P, A, s, G)^+$*.*

Proof Sketch. Show by induction over $0 \leq i \leq n$ that $appl(s, \langle a_1, \dots, a_i \rangle) \subseteq appl(s, \langle a_1^+, \dots, a_i^+ \rangle)$. (Exercises)

$\rightarrow$ "If we ignore deletes, the states along the plan can only get bigger."

Theorem. h^+ *is Admissible.*

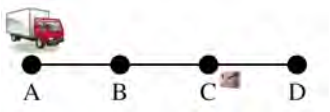
Proof. Let $\Pi = (P, A, I, G)$ be a planning task with states S, and let $s \in S$. $h^+(s)$ is defined as optimal plan length in $(P, A, s, G)^+$. With the above lemma, any plan for (P, A, s, G) also constitutes a plan for $(P, A, s, G)^+$. Thus optimal plan length in $(P, A, s, G)^+$ can only be shorter than that in (P, A, s, G), and the claim follows.

h^+ in “TSP” in Australia

Planning vs. Relaxed Planning:

- Optimal plan: $\langle drive(Sydney, Brisbane), drive(Brisbane, Sydney), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Perth, Adelaide), drive(Adelaide, Darwin), drive(Darwin, Adelaide), drive(Adelaide, Sydney)\rangle$.
- Optimal relaxed plan: $\langle drive(Sydney, Brisbane), drive(Sydney, Adelaide), drive(Adelaide, Perth), drive(Adelaide, Darwin)\rangle$.
- $h^*(I) = 8$; $h^+(I) = 4$.

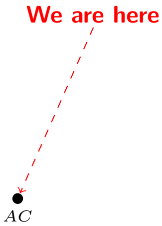
How to Relax During Search: Ignoring Deletes



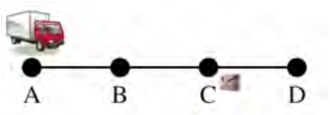
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



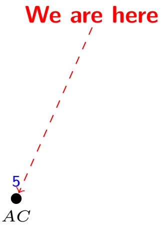
How to Relax During Search: Ignoring Deletes



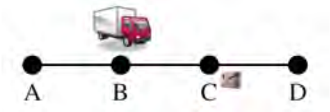
Relaxed problem:

- State s: AC; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 5$: e.g. $\langle drAB, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



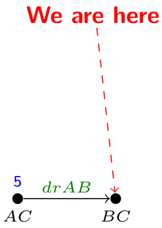
How to Relax During Search: Ignoring Deletes



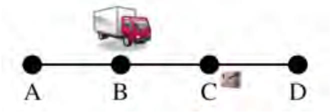
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, $\color{red}{del}$.
- $AC \xrightarrow{drAB} BC$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

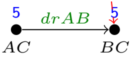


Relaxed problem:

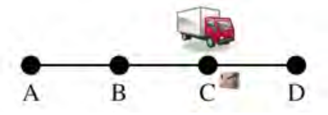
- State s: BC; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 5$: e.g. $\langle drBA, drBC, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



How to Relax During Search: Ignoring Deletes

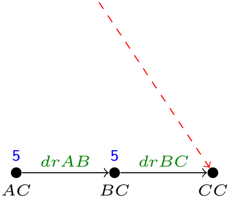


Real problem:

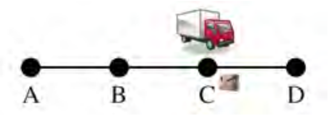
- State s: CC; goal G: AD.
- Actions A: pre, add, del.
- $BC \xrightarrow{drBC} CC$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



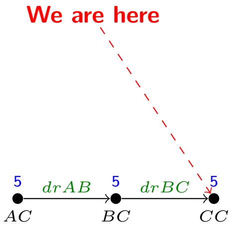
How to Relax During Search: Ignoring Deletes



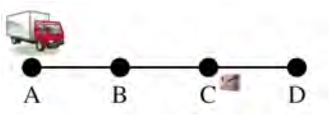
Relaxed problem:

- State s: CC; goal G: AD.
- Actions A: pre, add.
- $h^+(s)$ =5: e.g. $\langle drCB, drBA, drCD, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

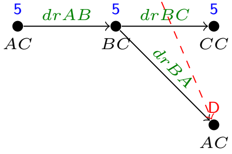


Real problem:

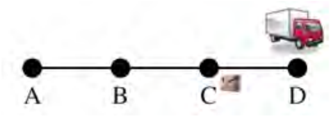
- State s: AC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



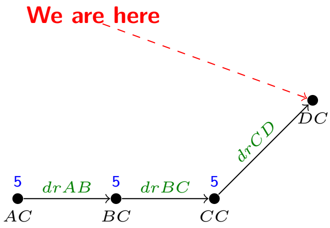
How to Relax During Search: Ignoring Deletes



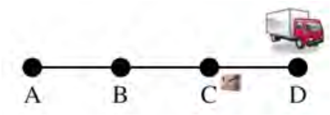
Real problem:

- State s: DC; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{drCD} DC$.

Greedy best-first search:
(tie-breaking: alphabetic)



How to Relax During Search: Ignoring Deletes

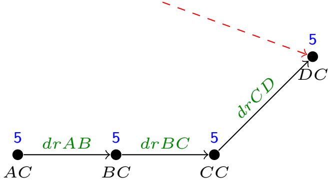


Relaxed problem:

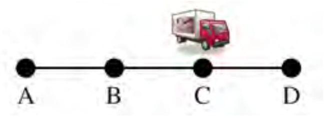
- State s: DC; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 5$: e.g. $\langle drDC, drCB, drBA, loC, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)

We are here



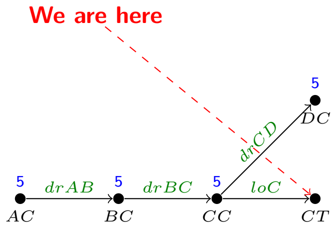
How to Relax During Search: Ignoring Deletes



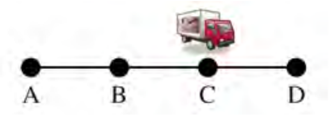
Real problem:

- State s: CT; goal G: AD.
- Actions A: pre, add, del.
- $CC \xrightarrow{loC} CT$.

Greedy best-first search:
(tie-breaking: alphabetic)



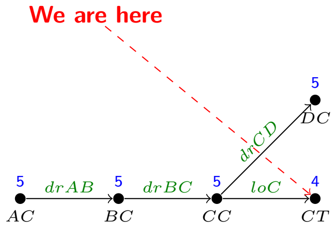
How to Relax During Search: Ignoring Deletes



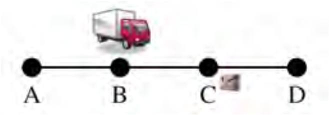
Relaxed problem:

- State s: CT; goal G: AD.
- Actions A: pre, add.
- $h^+(s) = 4$: e.g. $\langle drCB, drBA, drCD, ulD \rangle$.

Greedy best-first search:
(tie-breaking: alphabetic)



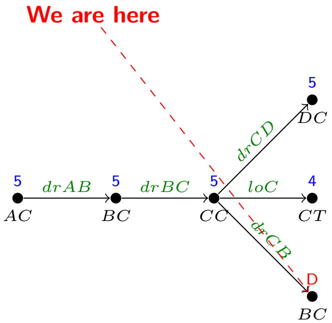
How to Relax During Search: Ignoring Deletes



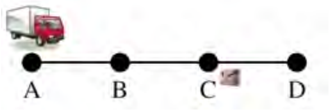
Real problem:

- State s: BC; goal G: AD.
- Actions A: pre, add, del.
- Duplicate state, prune.

Greedy best-first search:
(tie-breaking: alphabetic)



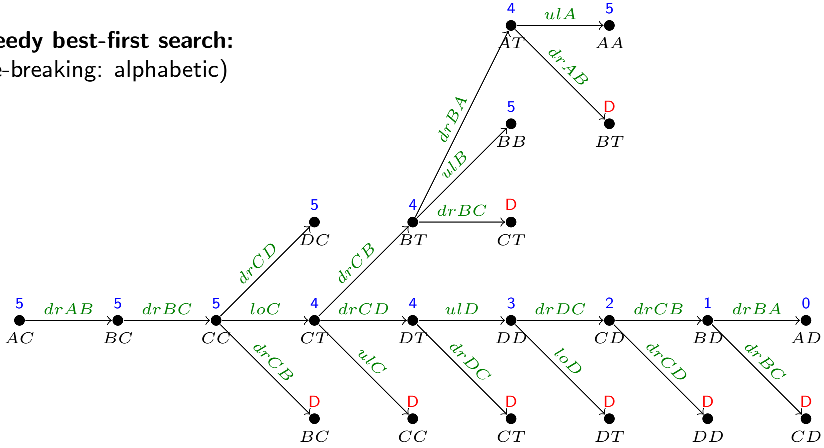
How to Relax During Search: Ignoring Deletes



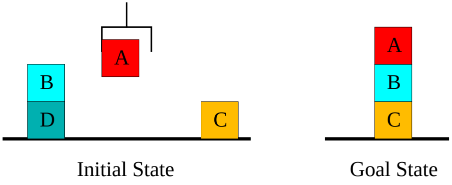
Real problem:

- Initial state I: AC; goal G: AD.
- Actions A: pre, add, del.
- $drXY, loX, ulX$.

Greedy best-first search:
(tie-breaking: alphabetic)



h^+ in the Blocksworld



- Optimal plan: $\langle putdown(A),\ unstack(B,D),\ stack(B,C),\ pickup(A),\ stack(A,B)\rangle$.
- Optimal relaxed plan: $\langle stack(A,B),\ unstack(B,D),\ stack(B,C)\rangle$.

Question: What can we say about the “search space surface” at the initial state here? The initial state lies on a local minimum under h^+: All neighbor states have a stricly higher heuristic value.

On the "Accuracy" of h^+

Reminder: Heuristics based on ignoring deletes are the key ingredient to almost all IPC winners of the last decade.

$\rightarrow$ **Why?**

$\rightarrow$ A heuristic function is useful if its estimates are "accurate".

How to measure this?

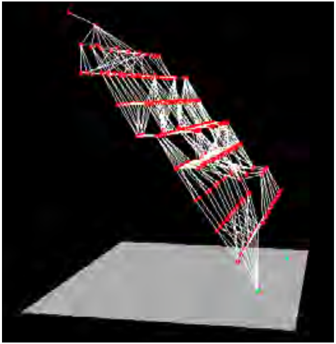
- **Known method 1:** Error relative to h^*, i.e., bounds on $|h^*(s) - h(s)|$.
- **Known method 2:** Properties of the search space surface: Local minima etc.

$\rightarrow$ For h^+, method 2 is the road to success:

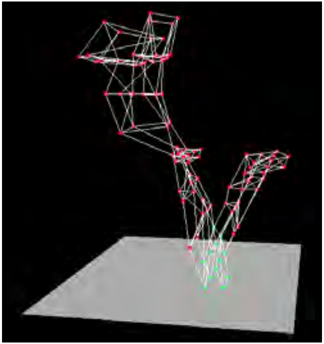
$\rightarrow$ In many benchmarks, under h^+, local minima *provably* do not exist! [Hoffmann (2005)]

A Brief Glimpse of h^+ Search Space Surfaces

→ Graphs = state spaces, vertical height = h^+:



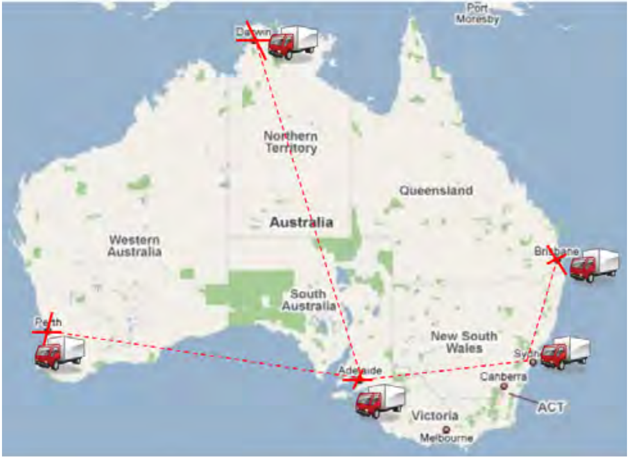
"Gripper"



"Logistics"

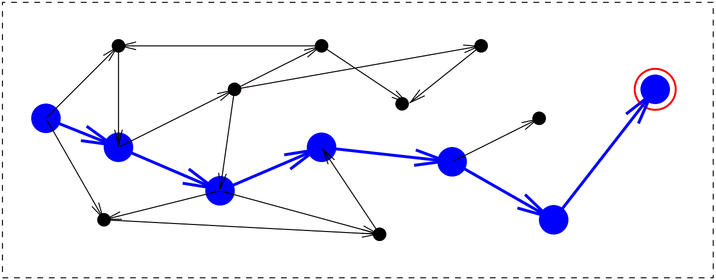
On the side: I *am* happy Russel/Norvig included these pictures. I'm not so happy that the text reads as if these illustrations referred to computing the heuristic, rather than to finding a plan. (And no, the number of states within the relaxed problem is not the motivation for abstractions. And no, FF does not do iterative deepening, nor restarts.)

h^+ in (the Real) TSP



$\rightarrow h =$ **Minimum Spanning Tree**

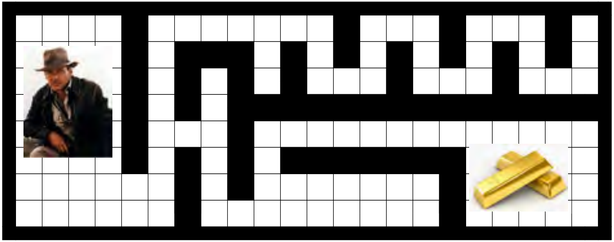
h^+ in Graphs



h^+(Graph-Distance) $=$ real distance

(shortest paths never "walk back")

Questionnaire



Question!

In this domain, h^+ is equal to?

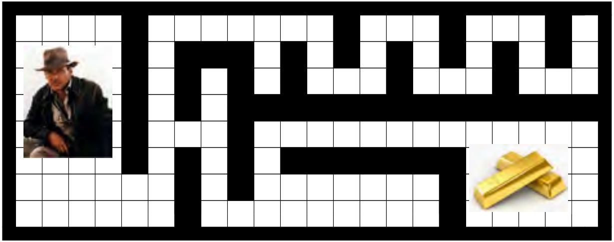
(A): Manhattan Distance.

(B): Horizontal distance.

(C): Vertical distance.

(D): h^*.

Questionnaire



Question!

In this domain, h^+ is equal to?

(A): Manhattan Distance.

(B): Horizontal distance.

(C): Vertical distance.

(D): h^*.

→ (A): No, relaxed plans can't walk through walls. (B), (C): No, relaxed plans must move both horizontally and vertically. (D): Yes, optimal plan = shortest path = optimal relaxed plan (cf. previous slide).

How to Compute h^+?

Definition (PlanLen$^+$). *By PlanLen$^+$, we denote the problem of deciding, given a planning task Π and an integer B, whether or not there exists a relaxed plan for Π of length at most B.*

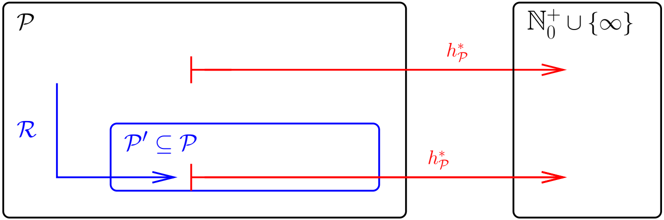
$\rightarrow$ By computing h^+, we would solve PlanLen$^+$.

Theorem. *PlanLen$^+$ is* **NP***-complete.*

Proof. Membership: (Exercises). Hardness: By a polynomial reduction from SAT. Assume a CNF with m variables v_i and n clauses c_j.

- For each variable v_i, three facts v_i, $notv_i$, and $assignedv_i$; for each clause c_j in the CNF, one fact $satisfiedc_j$.
- Actions $assigntrue_i$: $(\emptyset, \{v_i, assigned_i\}, \emptyset)$ and $assignfalse_i$: $(\emptyset, \{notv_i, assigned_i\}, \emptyset)$.
- Actions $makesatisfiedc_j$: $(\{v_i\}, \{satisfiedc_j\}, \emptyset)$ where v_i appears positively in clause c_j; $(\{notv_i\}, \{satisfiedc_j\}, \emptyset)$ where v_i appears negatively in clause c_j.
- Initial state $\emptyset$, goal $\{assigned_1, \dots, assigned_m, satisfiedc_1, \dots, satisfiedc_n\}$; $B := m + n$.

Hold on a Sec – Where are we?



- $\mathcal{P}$: STRIPS planning tasks; $h^*_{\mathcal{P}}$: Length h^* of a shortest plan.
- $\mathcal{P}' \subseteq \mathcal{P}$: STRIPS planning tasks with empty delete lists.
- $\mathcal{R}$: Drop the delete lists.
- Heuristic function: $h^+ = h^* \circ \mathcal{R}$, which is hard to compute.

$\rightarrow$ We can't compute our heuristic h^+ efficiently. So we approximate it instead.

Approximating h^+: h^{FF}

Definition (h^{FF}). *Let $\Pi = (P, A, I, G)$ be a planning task with states S. The relaxed plan heuristic h^{FF} for Π is a function $h^{\mathsf{FF}} : S \mapsto \mathbb{N}_0 \cup \{\infty\}$ returning the length of some, not necessarily optimal, relaxed plan for s if a relaxed plan for s exists, and returning $h^{\mathsf{FF}}(s) = \infty$ otherwise.*

Proposition. *Let $\Pi = (P, A, I, G)$ be a planning task. Then $h^{\mathsf{FF}} \geq h^+$.*
$\rightarrow$ h^{FF} is not admissible, i.e., $h^{\mathsf{FF}} > h^*$ can happen! Thus h^{FF} can be used for satisficing planning only, not for optimal planning.

Note: h^{FF} as per this definition is not unique. How do we find *"some, not necessarily optimal, relaxed plan for (P, A, s, G)"*?

$\rightarrow$ In what follows, we consider the following algorithm computing relaxed plans, and therewith (one variant of) h^{FF}:

1. Chain forward to build a relaxed planning graph (RPG).
2. Chain backward to extract a relaxed plan from the RPG.

Computing h^{FF}: Relaxed Planning Graphs (RPG)

$F_0 := s$, $t := 0$
while $G \not\subseteq F_t$ **do**
  $A_t := \{a \in A \mid pre_a \subseteq F_t\}$
  $F_{t+1} := F_t \cup \bigcup_{a \in A_t} add_a$
  if $F_{t+1} = F_t$ **then** stop **endif**
  $t := t + 1$
endwhile

$\rightarrow$ Does this look familiar to you? Could. It's the same algorithm we used to decide PlanEx$^+$ (slide 25).

"Logistics" example: Blackboard (similar to slide 27).

Informations from the RPG: ($\min$ over an empty set is ∞)

- For $p \in P$: $level(p) := \min\{t \mid p \in F_t\}$.
- For $a \in A$: $level(a) := \min\{t \mid a \in A_t\}$.

Computing h^{FF}: Extracting a Relaxed Plan

$M := \max\{level(p) \mid p \in G\}$
If $M = \infty$ **then** $h^{\mathsf{FF}}(s) := \infty$; stop **endif**
for $t := 0, \dots, M$ **do**
    $G_t := \{g \in G \mid level(g) = t\}$
endfor
for $t := M, \dots, 1$ **do**
    for all $g \in G_t$ **do**
        select a, $level(a) = t - 1$, $g \in add_a$
        for all $p \in pre_a$ **do**
            $G_{level(p)} := G_{level(p)} \cup \{p\}$
        endfor
    endfor
endfor
$h^{\mathsf{FF}}(s) :=$ number of selected actions

“Logistics” example: Blackboard.

Computing h^{FF} in "TSP" in Australia

RPG:

- $F_0 = \{at(Sydney), visited(Sydney)\}$.
- $A_0 = \{drive(Sydney, Adelaide), drive(Sydney, Brisbane)\}$.
- $F_1 = F_0 \cup \{at(Adelaide), visited(Adelaide), at(Brisbane), visited(Brisbane)\}$.
- $A_1 = A_0 \cup \{drive(Adelaide, Darwin), drive(Adelaide, Perth), drive(Adelaide, Sydney), drive(Brisbane, Sydney)\}$.
- $F_2 = F_1 \cup \{at(Darwin), visited(Darwin), at(Perth), visited(Perth)\}$.

Computing h^{FF} in "TSP" in Australia

Inserting the goals:

- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

Computing h^{FF} in "TSP" in Australia

Supporting the goals at $t = 2$:

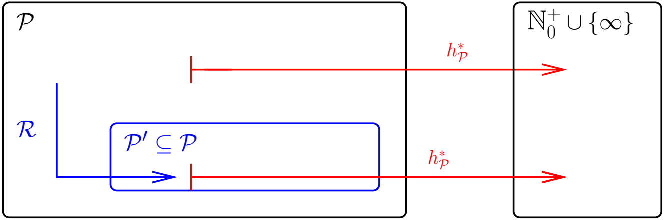
- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

Computing h^{FF} in "TSP" in Australia

Supporting the goals at $t = 1$:

- F_0: $at(Sydney)$, $visited(Sydney)$.
- A_0: $drive(Sydney, Adelaide)$, $drive(Sydney, Brisbane)$.
- F_1: $at(Adelaide)$, $visited(Adelaide)$, $at(Brisbane)$, $visited(Brisbane)$.
- A_1: $drive(Adelaide, Darwin)$, $drive(Adelaide, Perth)$, $drive(Adelaide, Sydney)$, $drive(Brisbane, Sydney)$.
- F_2: $at(Darwin)$, $visited(Darwin)$, $at(Perth)$, $visited(Perth)$.

How Does it All Fit Together?



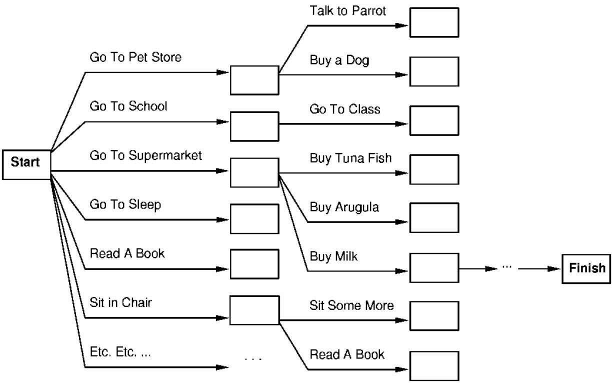
$\mathcal{P}$: STRIPS planning tasks. $h^*_{\mathcal{P}}$: Length h^* of a shortest plan. $\mathcal{P}'$: STRIPS planning tasks with empty delete lists. $\mathcal{R}$: Drop the delete lists. $h^* \circ \mathcal{R}$: Length h^+ of a shortest relaxed plan.

$\rightarrow$ Use h^{FF} to approximate h^+ which itself is hard to compute.

$\rightarrow$ h^+ is admissible; h^{FF} is not.

Helpful Actions Pruning

Idea: In search, expand only those actions contained in the relaxed plan.



$\rightarrow$ Relaxed plan = "Go To Supermarket, Buy Milk, . . . "

$\rightarrow$ Absolutely essential, used in all state-of-the-art satisficing planners.

Other Approximations of h^+

h^{max}: Approximate the cost of fact set g by the most costly single fact $p \in g$

$$h^{\text{max}}(s,g) := \begin{cases} 0 & g \subseteq s \\ min_{a \in A, p \in add_a} 1 + h^{\text{max}}(s, pre_a) & g = \{p\} \\ \max_{p \in g} h^{\text{max}}(s, \{p\}) & |g| > 1 \end{cases}$$

$\rightarrow$ Admissible, but very uninformative (under-estimates vastly).

h^{add}: Use instead the sum of the costs of the single facts $p \in g$

$$h^{\text{add}}(s,g) := \begin{cases} 0 & g \subseteq s \\ min_{a \in A, p \in add_a} 1 + h^{\text{add}}(s, pre_a) & g = \{p\} \\ \sum_{p \in g} h^{\text{add}}(s, \{p\}) & |g| > 1 \end{cases}$$

$\rightarrow$ Not admissible, and prone to over-estimation; h^{FF} works better.

Good lower bounds on h^+: (cf. next section)

- Admissible landmarks [Karpas and Domshlak (2009)]
- LM-cut [Helmert and Domshlak (2009)].

Questionnaire

Question!

In the initial state of the Towers of Hanoi task with 5 discs, what is the value of h^+?

(A): 1 (B): 2

(C): 5 (D): 32

$\rightarrow$ (C): The discs always "remain stacked", so we can just clear the bottom disc and move it over. For n discs, this takes $h^+(I) = n$ steps.