Linköping University

Automated Planning

Delete Relaxation:
"Things can only get better!"

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

Re-achieving Conditions

- To **make actions applicable** and **achieve goals**:
  - We often have to **re-achieve** what was **already achieved**

- Example: **Driving**
  - Initial state: **{ at(A), have-fuel }**
  - Goal: **{ at(D), have-fuel }**
  - Actions: **drive(?x,?y)** – must follow roads, must **have-fuel**
  - Solution:
    - **drive(A,B)**
    - **refuel**
    - **drive(B,C)**
    - **refuel**
    - **drive(C,D)**
    - **refuel**

Re-achieving Conditions (2)

jonkv@ida

- *Suppose* conditions always **remained achieved**
  - If **have-fuel** is true, it always remains true
  - New solution:
    - **drive(A,B)**
    - **drive(B,C)**
    - **drive(C,D)**

Can we use this to construct a relaxation?

Positive and Negative Effects (1)

jonkv@ida

- Suppose we use the book's **classical representation**:
  - Precondition = set of **literals** that must be true
  - Goal = set of **literals** that must be true
  - Effects = set of **literals** (making **atoms** true or false)

  - Suppose we have a solution **<A1,A2>**:
    - Initially **have-fuel**
    - Action **drive** → requires **have-fuel**, makes **have-fuel** false
    - Action **refuel** → requires (**not have-fuel**), makes **have-fuel** true

  - Symmetry:
    - **Positive effects** can ***achieve*** positive conditions, ***un-achieve*** negative conditions
    - **Negative effects** can ***achieve*** negative conditions, ***un-achieve*** positive conditions

Positive and Negative Effects (2)

jonkv@ida

- Suppose we use PDDL's plain **:strips** level
  - **Forbids negative preconditions / goals**
    - Precondition = set of **atoms** (no negations!)
    - Goal = set of **atoms** (no negations!)
    - Effects = set of **literals** (making **atoms** true or false)
  - In this setting:
    - **Positive effects** are never ”problematic”:
      Adding more facts to the state can only make *more* preconds/goals satisfied
    - Only **negative effects** can "un-achieve" goals or preconditions
    - And negative effects can **only** ”un-achieve” goals or preconditions:
      We never *need* them

Delete Relaxation (1)

jonkv@ida

- Assuming positive conditions, let's **remove all negative effects**
  - **Example**: (unstack ?x ?y)
    - **Before transformation:**
      :precondition (and (handempty) (clear ?x) (on ?x ?y))
      :effect (and (not (handempty)) (holding ?x) (not (clear ?x)) (clear ?y) (not (on ?x ?y)))
    - **After transformation:**
      :precondition (and (handempty) (clear ?x) (on ?x ?y))
      :effect (and (holding ?x) (clear ?y))
  - A fact that is true **stays** true

Is this a relaxation?

- Positive conditions ➔
  - No solution can ***depend on*** a fact ***being false*** in a visited state
  - No solution can ***disappear*** because we avoid making facts false

Delete Relaxation (2): Example

jonkv@ida

STS for the original problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

ontable(B)
ontable(C)
clear(B)
clear(C)
holding(A)

pickup(B)

on(A,C)
ontable(C)
clear(A)
holding(B)

Delete-relaxed STRIPS problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
clear(C)
holding(A)
handempty

pickup(B)

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
holding(B)
handempty

No physical ”meaning”!

STS for the original problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

ontable(B)
ontable(C)

clear(B)
clear(C)
holding(A)

stack(A,B)

…

Initial state
does not change

=

Same ”origin”,
fewer facts removed

⊆

Different ”origin” but
same *action sequence*,
fewer facts removed

⊆

Delete-relaxed STRIPS problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
clear(C)
holding(A)
handempty

stack(A,B)

…

STS for the original problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

ontable(B)
ontable(C)

clear(B)
clear(C)
holding(A)

Applicable actions: app_1

Delete-relaxed STRIPS problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
clear(C)
holding(A)
handempty

Applicable actions: app_2

$\subseteq$

No **action** requires the *absence* of a fact

$\subseteq$

Delete Relaxation (5): Example

jonkv@ida

STS for the original problem

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

ontable(B)
ontable(C)
clear(B)
clear(C)
holding(A)

Satisfies the goal?

Delete-relaxed STRIPS problem

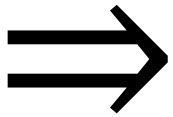
on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
handempty

unstack(A,C)

on(A,C)
ontable(B)
ontable(C)
clear(A)
clear(B)
clear(C)
holding(A)
handempty

Also satisfies the goal

No **goal** requires the *absence* of a fact



Delete Relaxation (6)

jonkv@ida

- **Negative effects** are also called "**delete effects**"
  - They delete facts from the state
- So this is called **delete relaxation**
  - "*Relaxing* the problem by getting rid of the *delete effects*"
- **"Relaxed plan for P"** = plan for the delete-relaxed version of P

Delete relaxation does not mean
that we "delete the relaxation" (anti-relax)!

Delete relaxation is only a relaxation
if preconditions and goals are positive!

Delete Relaxation (7)

jonkv@ida

- Since **solutions are preserved** when **facts are added**:

A state where additional facts are true can never be "worse"!
(Given positive preconds/goals)

$$h^*(\text{ontable(B), ontable(C), clear(B), clear(C), holding(A), } \mathbf{handempty}) \leq h^*(\text{ontable(B), ontable(C), clear(B), clear(C), holding(A)})$$

Given two states (sets of true atoms) s,s':
s ⊃ s' ➔ h*(s) <= h*(s')

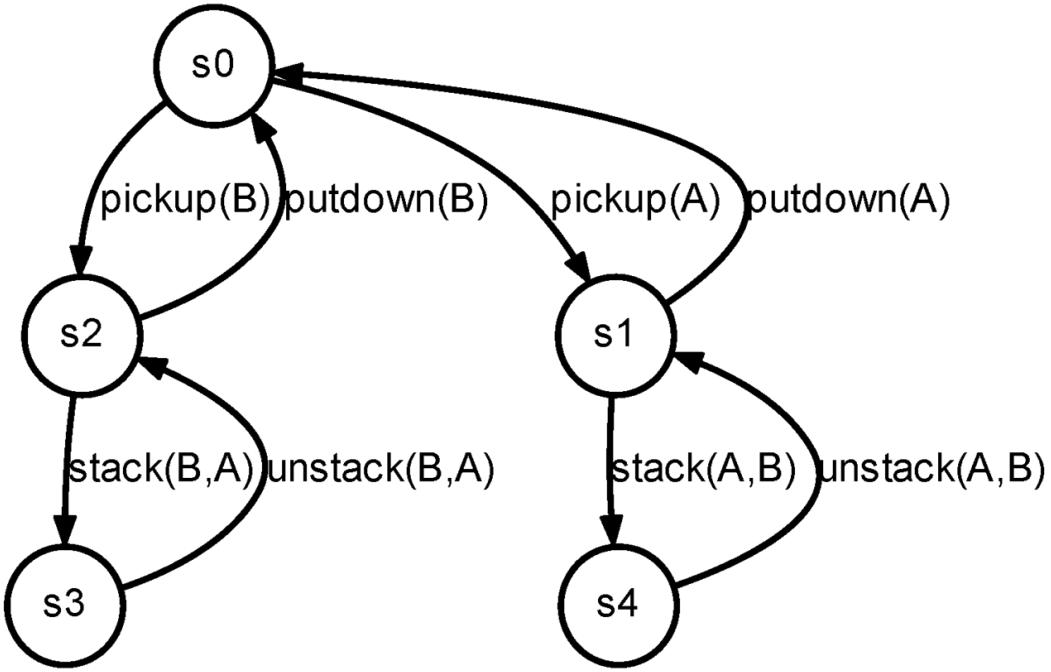
Delete Relaxation:

State Space Examples

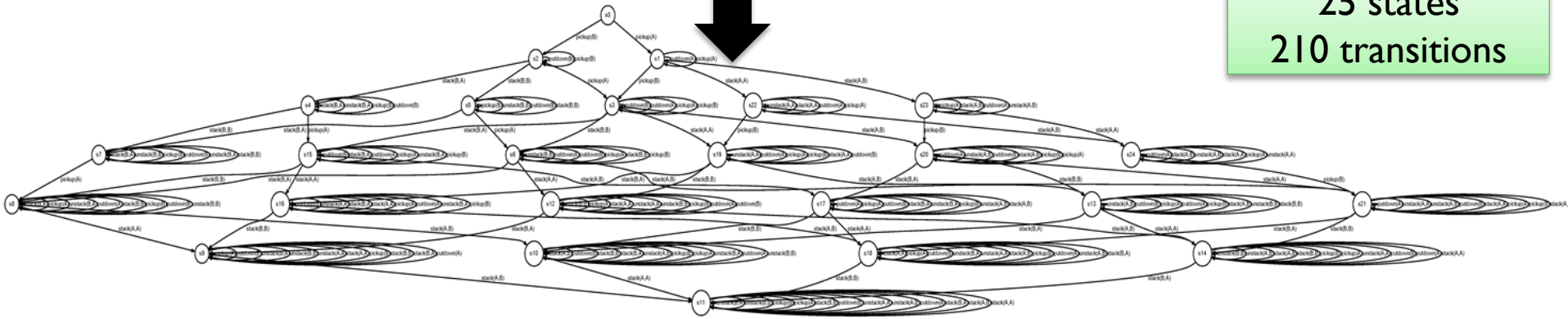
Reachable State Space: BW size 2

jonkv@ida

5 states
8 transitions



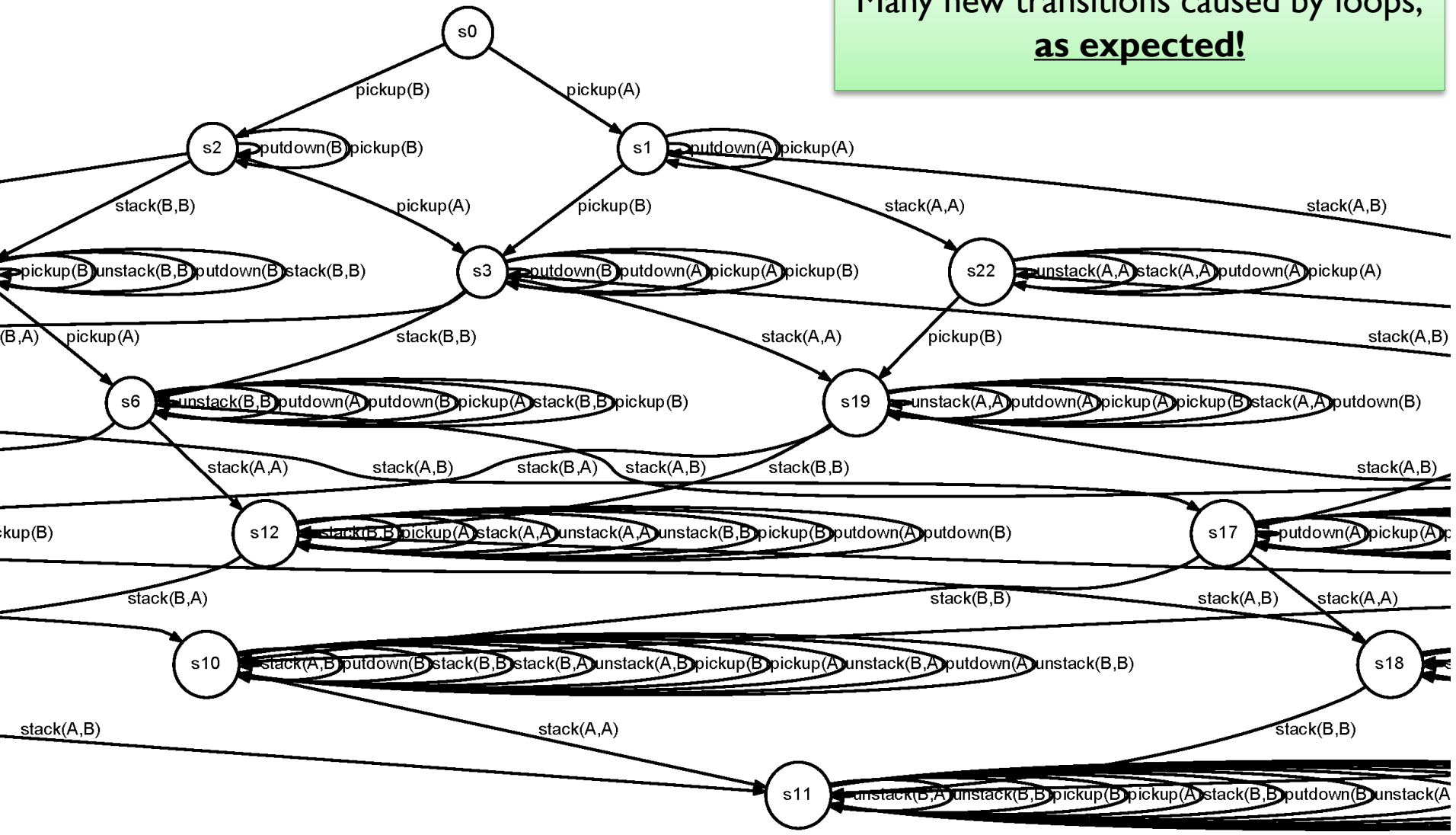
25 states
210 transitions



Delete-Relaxed BW size 2: Detail View

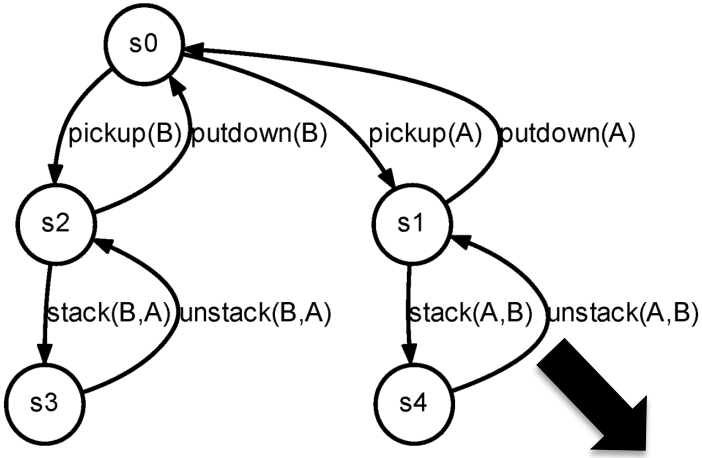
jonkv@ida

Many new transitions caused by loops,
as expected!



Delete-Relaxed: "Loops" Removed

jonkv@ida



5 states
8 transitions

25 states
50 transitions

Insight: **Relaxed** ≠ **smaller**

The Optimal Delete Relaxation Heuristic

Optimal Delete Relaxation Heuristic

jonkv@ida

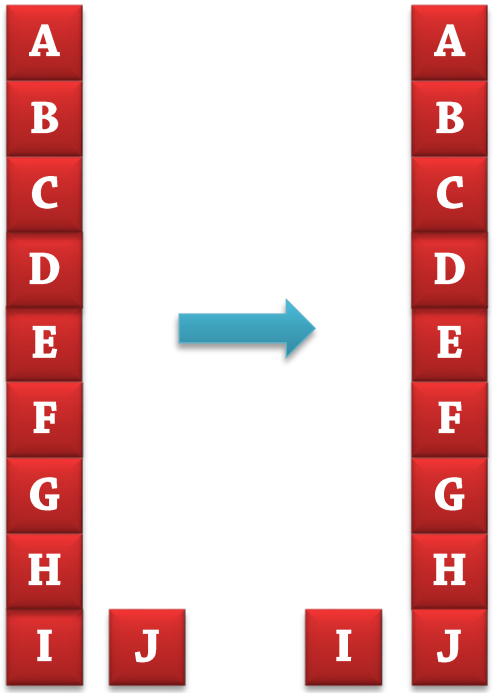
- If **only** delete relaxation is applied:
  - We can calculate the **optimal delete relaxation heuristic**, $h^+(n)$
  - $h^+(n)$ = the cost of an **optimal solution** to a **delete-relaxed** problem starting in node n

Accuracy of h+ in Selected Domains

jonkv@ida

- **How close** is $h^+(n)$ to the true goal distance $h^*(n)$?
  - **Worst case asymptotic accuracy** as problem size approaches infinity:
    - Blocks world: 1/4 ➔ $h^+(n) \geq \frac{1}{4} h^*(n)$

Optimal plans in delete-relaxed Blocks World
can be down to 25% of the length of optimal plans in "real" Blocks World



Standard:

unstack(A,B)	pickup(G)
putdown(A)	stack(G,H)
unstack(B,C)	pickup(F)
putdown(B)	stack(F,G)
unstack(C,D)	pickup(E)
putdown(C)	stack(E,F)
…	pickup(D)
unstack(H,I)	stack(D,E)
stack(H,J)	…

Relaxed:

unstack(A,B)
unstack(B,C)
unstack(C,D)
unstack(D,E)
unstack(E,F)
unstack(F,G)
unstack(G,H)
unstack(H,I)
stack(H,J)
DONE!

Accuracy of h+ in Selected Domains (2)

- **How close** is $h^+(n)$ to the true goal distance $h^*(n)$?
  - **Worst case asymptotic accuracy** as problem size approaches infinity:
    - Blocks world: 1/4 ➔ $h^+(n) \geq \frac{1}{4} h^*(n)$
    - Gripper domain: 2/3 (single robot moving balls)
    - Logistics domain: 3/4 (move packages using trucks, airplanes)
    - Miconic-STRIPS: 6/7 (elevators)
    - Miconic-Simple-ADL: 3/4 (elevators)
    - Schedule: 1/4 (job shop scheduling)
    - Satellite: 1/2 (satellite observations)
  - Details:
    - Malte Helmert and Robert Mattmüller *Accuracy of Admissible Heuristic Functions in Selected Planning Domains*

Accuracy of Admissible Heuristic Functions in Selected Planning Domains

Malte Helmert and Robert Mattmüller
Albert-Ludwigs-Universität Freiburg, Germany
{helmert,mattmuel}@informatik.uni-freiburg.de

Abstract

The efficiency of optimal planning algorithms based on heuristic search crucially depends on the accuracy of the heuristic function used to guide the search. Often, we are interested in domain-independent heuristics for planning. In order to assess the limitations of domain-independent heuristic planning, we analyze the (in)accuracy of common domain-independent planning heuristics in the IPC benchmark domains. For a selection of these domains, we analytically investigate the accuracy of the h^+ heuristic, the h^{m} family of heuristics, and certain (additive) pattern database heuristics, compared to the perfect heuristic h^*. Whereas h^+ and additive pattern database heuristics usually return cost estimates proportional to the true cost, non-additive h^{m} and non-additive pattern-database heuristics can yield results underestimating the true cost by arbitrarily large factors.

Introduction

Heuristic search with A* and similar algorithms remains the most popular method for optimal sequential planning, with significant effort spent on perfecting old heuristic estimators (Haslum et al. 2007) or deriving new ones (Helmert, Haslum, and Hoffmann 2007). While methods not based on state-space search have achieved remarkable success in addressing related problems, such as optimal parallel planning (Kautz and Selman 1996; 1999), the state of the art in optimal sequential planning is still defined by heuristic search almost exclusively. Symbolic state-space exploration (Edelkamp and Helmert 2001) is the only non-classical approach that sometimes outperforms heuristic search.

Considering the important role of admissible heuristics for optimal sequential planning, or search in general, the question arises how to evaluate the quality of a given heuristic. A popular method is to run a search algorithm against some benchmark tasks and count the number of node expansions. The fewer nodes an algorithm expands, the better.

While experiments of this kind are certainly useful, there are some questions they cannot address. In particular, their results can almost exclusively be interpreted with *relative*, i.e., *comparative* statements: "Heuristic h expands fewer nodes than heuristic h' for benchmark suite $\mathfrak{X}$." Unless experiments show polynomial scaling behavior on a family of benchmark tasks of growing size, which they very rarely do, the data usually does not lend itself to *absolute* statements of the type "Heuristic h is well-suited for solving tasks from benchmark suite $\mathfrak{X}$." In this contribution, we address this issue by providing absolute quality results for certain popular planning heuristics on some popular benchmark domains taken from the first four International Planning Competitions (McDermott 2000; Bacchus 2001; Long and Fox 2003; Hoffmann and Edelkamp 2005), in the form of comparisons to the perfect heuristic function h^*.

Planning Domains

We consider the planning domains GRIPPER, LOGISTICS, BLOCKSWORLD, MICONIC-STRIPS, MICONIC-SIMPLE-ADL, SCHEDULE and SATELLITE. Familiarity with the domains is assumed. For an in-depth treatment, we refer to the literature (Helmert 2008).

Heuristics

We compare the accuracy of h^+, h^{m}, non-additive and additive pattern database (PDB) heuristics relative to the perfect heuristic h^*. An admissible heuristic h maps states s to optimistic estimates of the true cost of reaching a goal state from s. Whenever the planning task to which s belongs, including the available operators and the goal description, is clear from the context, we will simply write $h(s)$ without explicitly mentioning operators or goal.

The h^* Heuristic The *perfect heuristic* h^* assigns to each state s the length of a shortest plan from s to a goal state. Computing h^* for the initial state of a planning task is PSPACE-equivalent in general (Bylander 1994), but can be easier for fixed domains. For the domains we consider, the problem is NP-equivalent, with the exception of GRIPPER and SCHEDULE, where it is polynomial (Helmert 2008).

The h^1 Heuristic The h^1 heuristic (McDermott 1996; Bonet and Geffner 2001; Hoffmann 2005) assigns to each state s of a task T the length of a shortest plan leading from s to a goal state in the relaxed task T^+. Evaluating h^+ is NP-equivalent in general (Bylander 1994), but easier for many of the domains we consider.

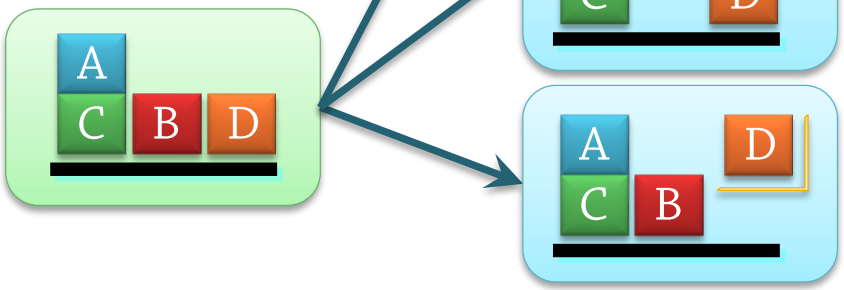
The h^{m} Family of Heuristics The h^{m}, $\mathrm{m} = 1, 2, \ldots$, family of heuristics (Haslum and Geffner 2000) is based on

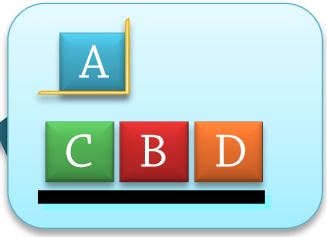
Example of Accuracy

jonkv@ida

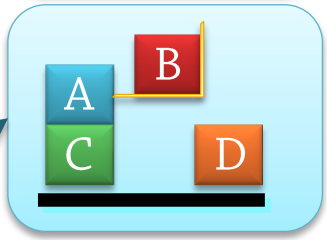
- **How close** is $h^+(n)$ to the true goal distance $h^*(n)$?
  - In practice: Also depends on the **problem instance**!

unstack(A,C)
pickup(B)
stack(B,C)
stack(A,B)
➔ **h+ = 4 [h* = 6]**



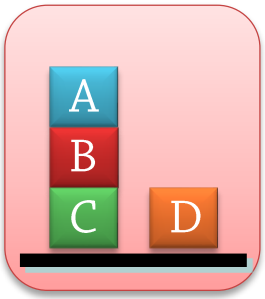


pickup(B); stack(B,C); stack(A,B)
➔ **h+ = 3 [h* = 5]**
Good action!



unstack(A,C); stack(B,C); stack(A,B)
➔ **h+ = 3 [h* = 7]**
Seems equally good as unstack,but *is* worse

unstack(A,C); pickup(B);
stack(B,C); stack(A,B)
➔ **h+ = 4 [h* = 7]**



- **Performance** also depends on the search strategy
  - How sensitive it is to specific types of inaccuracy

Computing the Optimal Delete Relaxation Heuristic

Computing h+

jonkv@ida

- Is $h^+(n)$ **easier to compute** than $h^*(n)$?
  - $h^*(n)$ = length of optimal plan for **arbitrary planning problem**
    - Supports negative effects
    - If we can execute either a1;a2 or a2;a1:
      - a1 removes p, a2 adds p ➔ net result: add p
      - a2 adds p, a1 removes p ➔ net result: remove p
      - **Both orders** must be considered
  - $h^+(n)$ = length of optimal plan after removing negative effects
    - If we can execute either a1;a2 or a2;a1:
      - Must lead to the same state (add p1 before p2, or p2 before p1)
      - Sufficient to consider **one order** – simpler?
  - Incomplete analysis
    - But the worst case for $h^+(n)$ **is** easier than the worst case for $h^*(n)$

Calculating h+

jonkv@ida

- Still **difficult** to calculate in general!
  - NP-equivalent (reduced from PSPACE-equivalent)
    - Since you must find **optimal** solutions to the relaxed problem
  - Even a constant-factor approximation is NP-equivalent to compute!
    - Finding $h(n)$ so that $\forall n.\, h(n) \geq c \cdot h^+(n)$
- Therefore, rarely used "as is"
  - But forms the **basis** of many other heuristics