Heuristics for Domain-Independent Planning
1. Introduction

Emil Keyder Silvia Richter

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Contents

We try to give a general overview with the basic ideas and intuitions for different heuristics

See references for details

Planning as Heuristic Search

Successful and robust

- Top four planners in the satisficing track of IPC6
- Several top planners in the optimal track
- Many well-performing planners from previous competitions

Standardized framework

- Mix and match heuristics and search techniques

Heuristic Search Planning

We characterize planning as a search problem

Given a directed graph $G = \langle V, E \rangle$, where

- V is a finite set of vertices
- E is a set of directed edges $\langle t, h \rangle$, $t, h \in V$

a search problem P is defined by:

- An initial vertex $v_0 \in V$
- goal vertices $V_G \subseteq V$
- A cost function $cost : E \to \mathbb{R}_0^+$

Search Problems

A solution is a sequence of edges $\pi = \langle e_0, \ldots, e_n \rangle$ that forms a path from v_0 to some $v_g \in V_G$

An optimal solution is a path with minimum total cost, where the cost of a path is given by the sum of its edge costs:

$$cost(\pi) = \sum_{e \in \pi} cost(e)$$

Classical planning as a search problem

- Set of states S – The vertices of the graph
- Initial state $s_0 \in S$
- A function $G(s)$ that tells us whether a state is a goal
- Planning operators O – The edges of the graph
- Applicable operators $A(s) \subseteq O$ in state s
- Transition function app: $S \times O \to S$, defined for $s \in S, o \in A(s)$
- Non-negative operator costs $cost(o) \in \mathbb{R}_0^+$

Edges in graph determined by A and app

Solutions are plans

Solutions of minimum cost are optimal plans

Solving Search Problems

Brute-force approach: Systematically explore full graph

Uniform-cost search, Dijkstra

- Starting from v_0, explore reachable vertices until $v_g \in G$ is found

Heuristics help by:

- Delaying or ruling out the exploration of unpromising regions of the graph
- Guiding the search towards promising regions of the graph

Heuristics: What are they?

Heuristics are functions $h : V \mapsto \mathbb{R}_0^+$ that estimate cost of path to a goal node

Definition
$h^*(v)$ is the cost of the lowest-cost path from v to some $v' \in V_G$

$h^*(v) \rightarrow$ optimal solution in linear time
$\rightsquigarrow$ Intractable to compute in general, but useful comparison point

Objective : get as close as possible to h^*

Heuristics: Admissibility

Definition
A heuristic h is admissible if for all $s \in S$: $h(s) \leq h^*(s)$

Intuition: Heuristic is optimistic

- Never overestimates

Among admissible estimates h_1 h_2, the higher the better

- h^* is an upper bound, so larger estimates are more exact
- A* expands fewer nodes when using higher admissible estimates[1]

[1] almost always, caveats apply

Maximum of Admissible Heuristics

Maximum of admissible heuristics
Let h_1, h_2 be two admissible heuristics. Note that

$$h_1(s) \leq h^*(s) \wedge h_2(s) \leq h^*(s) \implies \max(h_1(s), h_2(s)) \leq h^*(s)$$

The maximum of two admissible heuristics is a more informed admissible heuristic

Sum of Admissible Heuristics

Sum of two admissible heuristics not in general admissible
$\rightsquigarrow h^* + h^* > h^*$

But, consider two admissible estimates for length of ICAPS:

- h_1: $\geq$ 2 days, because the workshops last 2 days
- h_2: $\geq$ 3 days, because the main conference lasts 3 days

$\rightsquigarrow$ We can infer that ICAPS lasts $\geq$ 5 days

Why? No "overlap" between the estimates.

Under certain conditions, the sum of two admissible heuristics is an admissible heuristic

We say that a set H of such heuristics is additive

Additive Admissible Heuristics: Action Partitioning

Simple criterion for admissible additivity: Count cost of each action in at most one heuristic.

Given a task Π and a set of operators $O_i \subseteq O$, define the problem Π_{O_i} that is identical except

$$cost'(o) = \begin{cases} cost(o) & \text{if } o \in O_i \\ 0 & \text{if } o \notin O_i \end{cases}$$

Action partitioning

An action partitioning is a set of problems $\Pi_{O_1}, \ldots, \Pi_{O_n}$ resulting from a partitioning of the actions O of a planning task into disjoint sets $O_1, \ldots, O_n$.

Additive Admissible Heuristics: Action Partitioning

Additivity of action partitioning

Let $\Pi_1, \ldots, \Pi_n$ be an action partitioning and let h_{Π_i} be an admissible estimate of the cost of Π_i. Then the sum of the heuristics h_{Π_i} is admissible:

$$\sum_{i=0}^{n} h_{\Pi_i}(s) \leq h^*(s)$$

Proof sketch

Optimal plan for Π is also plan for each Π_{O_i}, and the summed cost of these plans is h^*. Admissible estimates must therefore sum to less than h^*.

Additive Admissible Heuristics: Cost Partitioning

Action partitioning assigns the full cost of each operator to one problem, and zero to the others

Instead, distribute the cost of each action among different problems
$\rightsquigarrow$ Ensure total across problems sums to at most cost of action

Cost partitioning

A cost partitioning of Π is a set of problems $\Pi_0, \ldots, \Pi_n$ that differ from Π only in operator costs, that satisfy

$$\sum_{i=0}^{n} cost_i(o) \leq cost(o)$$

where $cost_i(o)$ is the cost of o in Π_i,

Additive Admissible Heuristics: Cost Partitioning

Additivity of cost partitioning

Let $\Pi_0, \ldots, \Pi_n$ be a cost partitioning of Π and let $h_{\Pi_0}, \ldots, h_{\Pi_n}$ be admissible estimates for the costs of $\Pi_0, \ldots, \Pi_n$. Then the sum of the heuristics h_{Π_i} is admissible:

$$\sum_{i=0}^{n} h_{\Pi_i}(s) \leq h^*(s)$$

Proof sketch

Optimal plan for Π is also plan for each Π_{O_i}, and the summed cost of these plans is h^*. Admissible estimates must therefore sum to less than h^*.

Heuristics: Consistency

Definition
A heuristic h is consistent if $s_g \in G \implies h(s_g) = 0$ and for all $s \in S$ and edge (s, s'):

$$h(s) \leq cost(s, s') + h(s')$$

Intuition: A transition with cost c cannot decrease h by more than c

Some properties

Consistency implies admissibility

- s, o s.t. $app(s, o) = s_g \in G$, $h(s) > h^*(s)$ would violate consistency
- Induction on number of steps to goal

Admissible heuristics with A* and variants compute optimal solutions

- By the time we consider $s_g \in G$, we have already considered all s that might be cheaper

Consistent heuristics with A* guarantee optimal behaviour

- Never expand same state twice

Planning as Heuristic Search

Idea: Search the state space using a heuristic

What kind of heuristic?

- Admissible for optimal solution, no guarantees required otherwise
- Efficiently computable
  $\rightsquigarrow$ In practice, low order polynomial

Tradeoff between informativeness and computational effort

- Evaluate few states at high computational cost, vs.
- Evaluate many states at low computational cost

Hard to tell a priori whether computational effort will pay off

A Logistics Problem Example

Heuristics are obtained from factored representations of states:

- Set V of variables
- Domain D_v of values for each variable v

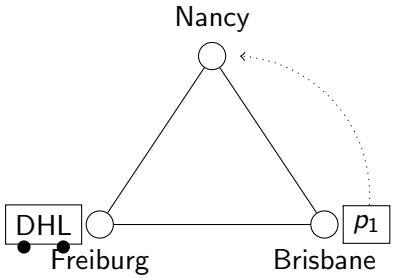


Figure: Deliver p_1 from Brisbane to Nancy.

STRIPS

Boolean variables:

- $D_{\textit{truck-at-Freiburg}} = \{\textit{true}, \textit{false}\}$

Definition (STRIPS task)

A STRIPS task $\Pi = \langle F, s_0, G, O, \textit{cost}\rangle$ is defined by:

- F A set of fluents/facts (boolean variables)
- s_0 The initial state, $s_0 \subseteq F$
- G The goal description, $G \subseteq F$
- O The actions/operators, defined by tuples $\langle \textit{pre}(o), \textit{add}(o), \textit{del}(o)\rangle$, each $\subseteq F$
- $\textit{cost}$ A function $O \to \mathbb{R}_0^+$

- States are sets of fluents that have value *true*
- State space is potentially exponential: $O(2^{|F|})$

SAS$^+$

Finite-domain variables:

- $D_{\textit{loc-truck}} = \{\textit{Freiburg}, \textit{Nancy}, \textit{Brisbane}\}$

Definition (SAS$^+$ task)

An SAS$^+$ task $\Pi = \langle V, s_0, G, O, \textit{cost}\rangle$ is defined by:

- V A set of variables, each with associated domain D_v
- s_0 The initial state, a full variable assignment
- G a partial variable assignment
- O The actions/operators, defined by tuples $\langle \textit{pre}(o), \textit{eff}(o)\rangle$, each a partial variable assignment
- $\textit{cost}$ A function $O \to \mathbb{R}_0^+$

SAS$^+$ continued

- A full variable assignment assigns to each $v \in V$ a value $d \in D_v$ and represents a state
- A partial variable assignment assigns values to a subset $C \subseteq V$
- Sometimes refer to assignments $(v = d)$ for $v \in V$, $d \in D_v$ as fluents/facts
- State space: $\prod_{v \in V} |D_v|$

Translation from STRIPS to SAS$^+$

Easy to translate automatically from STRIPS to SAS$^+$

Idea: Make a graph with node set F, and edges between any two fluents that cannot occur in the same state

- Example: $\langle \textit{truck-at-Freiburg}, \textit{truck-at-Brisbane}, \textit{truck-at-Nancy}\rangle$

Cliques in graph represent finite-domain variables

Replace n boolean variables with a single n-valued variable

- More efficient representation – n vs. 2^n configurations

Heuristics for Domain-Independent Planning
2. The Delete Relaxation

Emil Keyder Silvia Richter

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Relaxations

Relaxations are less constrained versions of problems

- They are therefore easier to solve

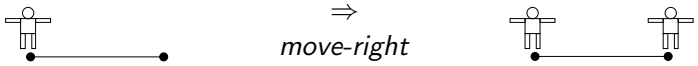
Idea: Solve relaxed problem and use cost of solution as h

- If we solve relaxation optimally, admissibility is guaranteed

The Delete Relaxation Π^+

Assumption: Variables can hold multiple values at the same time, and they retain all values they ever had

- When something is achieved, it stays achieved



$h^+ = h^*(\Pi^+)$ is admissible
- Any solution to Π is a solution to Π^+ as well
- The converse is not true, as in Π, a fluent may have to be made true more than once

The Delete Relaxation Π^+ – Formalization

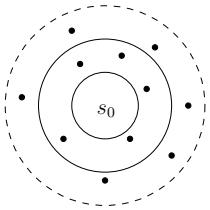
In STRIPS, remove delete lists:

Definition

Given a STRIPS problem $\Pi = \langle F, s_0, G, O, cost \rangle$, its delete relaxation is given by $\Pi^+ = \langle F, s_0, G, O', cost \rangle$, where

$$O' = \{\langle Pre(o), Add(o), \emptyset \rangle \mid o \in O\}$$

- $|s|$ increases with each action
- Stratification of fluents by first level at which they can be achieved



The Delete Relaxation Π^+

Plan existence for Π^+ is easy

- Use all applicable operators until goal achieved or no further fluents can be added
- Naive algorithm: at most $|F|$ iterations, with $O(|O|)$ work per iteration

Optimal solution to P^+ is NP-hard

- Why?

Simplifying Π^+

Ideally, $|s_0| = |Pre(o)| = |Add(o)| = |G| = 1$

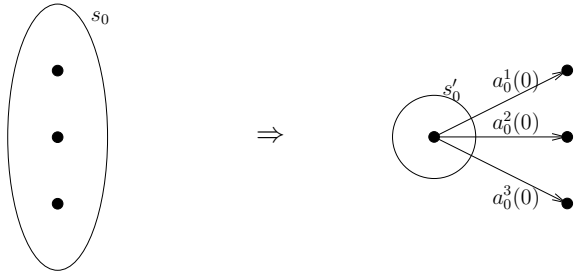
- Shortest path problem on graph with $V = F$, $E = O$
  $\rightsquigarrow$ polynomial in F and E
- $|F|$ is small, so easily solvable

Question: Does there exist a $\Pi^{+\prime}$ equivalent to Π^+ with this property?
Answer: Almost (except for $|Pre(o)|$)

Theorem

For any Π^+, there exists an equivalent problem $\Pi^{+\prime}$ such that $|s'_0| = 1$, $|G'| = 1$, and $|Add(o)| = 1$ for all $o \in O'$

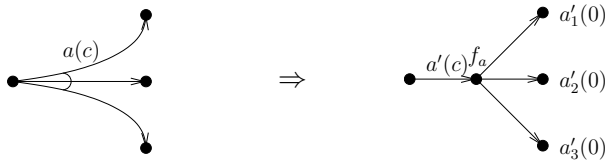
Simplifying Π^+



A start state with size $|s_0| = n$ can be replaced with:

- A new start state containing a single newly-introduced fluent: $s'_0 = \{f\}$
- A set of n zero-cost actions: $O_0 = \{\langle \{f\}, \{s_i\}, \emptyset \rangle \mid s_i \in s_0\}$

Simplifying Π^+

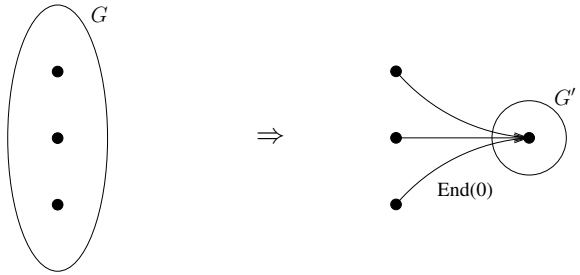


An action o with $|Add(o)| = n$ can be replaced with:

- A new fluent f_o, representing that the action has been executed
- An action $o' = \langle Pre(o), f_o, \emptyset \rangle$ with $cost(o') = cost(o)$
- A set of n zero-cost actions: $O' = \{\langle \{f_o\}, \{o_i\}, \emptyset \rangle \mid o_i \in Add(o)\}$

Simplifying Π^+

And $|G|$?



A goal with size $|G| = n$ can be replaced with:

- A new goal with a single newly-introduced fluent: $G' = \{g\}$
- A single action $End = \langle G, g, \emptyset \rangle$

→ This introduces an action with $|G|$ preconditions

Π^+ as a graph problem

Fundamental difficulty: Estimating costs of sets of fluents

Special cases of Π^+ are well-known graph problems in which nodes → fluents, edges → actions

- $|G| = |Pre(o)| = 1$ → Shortest Path — P
- $|G| > 1$, $|Pre(o)| = 1$ → Directed Steiner Tree — NP-hard
- $|Pre(o)| > 1$ → Optimal Directed Hyperpath — NP-hard

Approximating h^+

Several heuristics defined in terms of method used to estimate the costs of these sets

To be optimal, must take into account interactions between plans for fluents in set
⇝ Intractable in general

Instead, make the independence assumption:

Independence Assumption

The (cost of) the best plan for a set of fluents P can be estimated as a function of the (costs of) the best plans for each fluent $p \in P$

Suboptimal, but makes the problem tractable

Polynomial Approximation Approaches

Theorem
For any set of fluents $P \subseteq F$,

$$\max_{p \in P} h^+(p) \leq h^+(P) \leq \sum_{p \in P} h^+(p)$$

1. Prove a minimum cost for any solution
   - Lower bound, admissible heuristic for optimal planning
2. Find a solution to Π^+ with no guarantees of optimality
   - Upper bound, inadmissible heuristic for satisficing planning

The Max Heuristic $h^{\max}$

An admissible heuristic that gives a lower bound on h^+ (Bonet & Geffner, 2001)

Intuition: Estimates cost of a set as the cost of the most expensive fluent in set:

$$h^{\max}(P; s) = \max_{p \in P} h^{\max}(p; s)$$

Drawbacks:

- Loose lower bound, uninformative

$h^{\max}$ is an instance of a family of heuristics – more on this later

The Max Heuristic $h^{\max}$

$$h^{\max}(s) = h^{\max}(G; s)$$

$$h^{\max}(P; s) = \max_{p \in P} h^{\max}(p; s)$$

$$h^{\max}(p; s) = \begin{cases} 0 & \text{if } p \in s \\ \min_{\{o | p \in add(o)\}} [cost(o) + h^{\max}(pre(o); s)] & \text{otherwise} \end{cases}$$

The Additive Heuristic h^{add}

First practical domain-independent planning heuristic for satisficing planning (Bonet & Geffner, 2001)

Intuition: Estimates cost of a set as the sum of the costs of the elements:

$$h^{\text{add}}(G) = \sum_{g \in G} h^{\text{add}}(g)$$

Assumes no positive interactions

- Inadmissible, so not useful for optimal planning
- Better than $h^{\max}$ for satisficing planning

The Additive Heuristic h^{add}

$$h^{add}(s) = h^{add}(G; s)$$

$$h^{add}(P; s) = \sum_{p \in P} h^{add}(p; s)$$

$$h^{add}(p; s) = \begin{cases} 0 & \text{if } p \in s \\ \min\limits_{\{o | p \in add(o)\}} \left[cost(o) + h^{add}(pre(o); s) \right] & \text{otherwise} \end{cases}$$

Computation

Equations give properties of estimates, not how to calculate them

Basic idea: value iteration

- Start with rough estimates (e.g. $0, \infty$), do updates

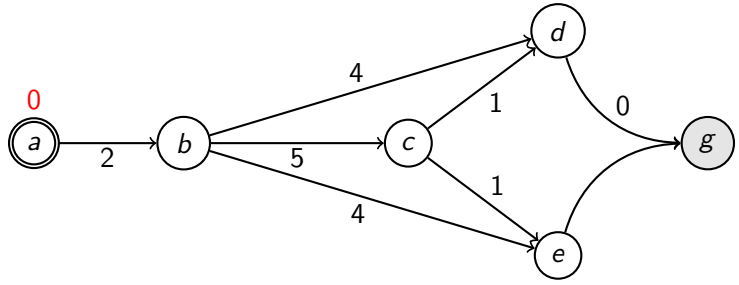
Methods differ principally in choice of update ordering:

1. Label-correcting methods
   - Arbitrary action choice
   - Multiple updates per fluent
2. Dijkstra method
   - Updates placed in priority queue
   - Single update per fluent

Computation

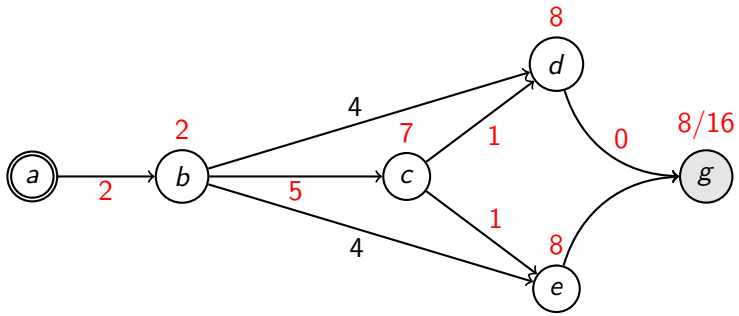
function h($G; s$) **begin**
  for $p \in F$ **do** `/* initialization */`
    if $p \in s$ **then** $h(p) = 0$ **else** $h(p) = \infty$
  end
  repeat
    a = `chooseAction()`
    for $q \in Add(a)$ **do**
      if $h(a; s) < h(q; s)$ **then**
        $h(q; s) = h(a; s)$
      endif
    end
  until *fixpoint*
  return $h(G; s)$
end

Example: Label correcting method



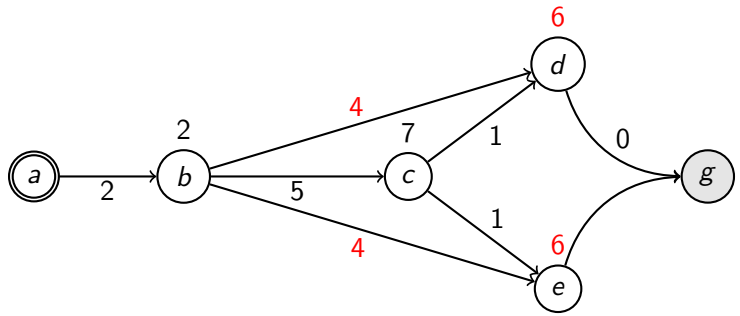
Initialization step

Example: Label correcting method



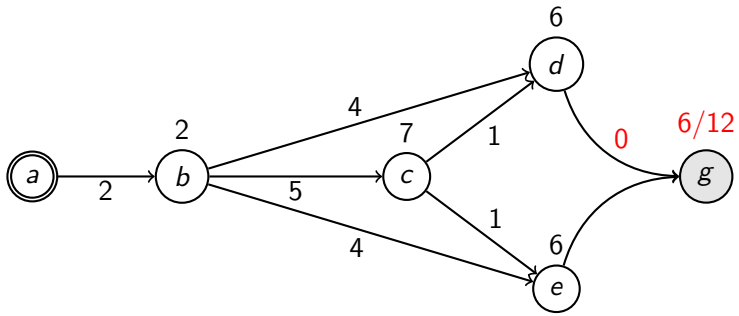
$$0 + \max(h^{max}(d), h^{max}(e)) = 0 + 8 = 8$$
$$0 + (h^{add}(d) + h^{add}(e)) = 0 + 8 + 8 = 16$$

Example: Label correcting method



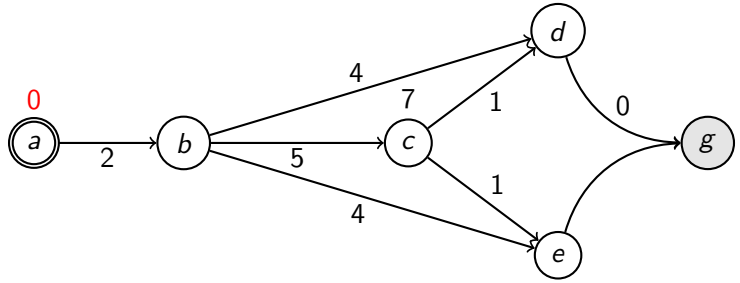
Label correction - cheaper estimates for fluents d and e

Example: Label correcting method



Label correction - cheaper estimates for fluent g

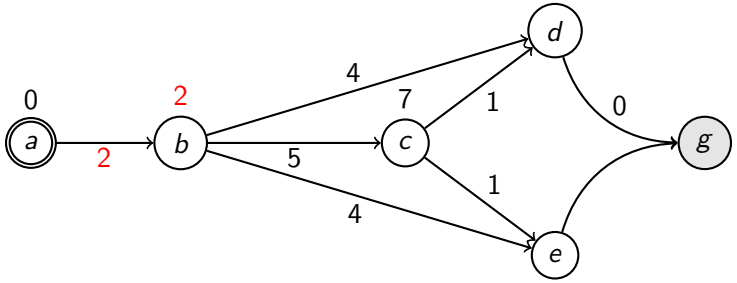
Example: Generalized Dijkstra method



Initialization step

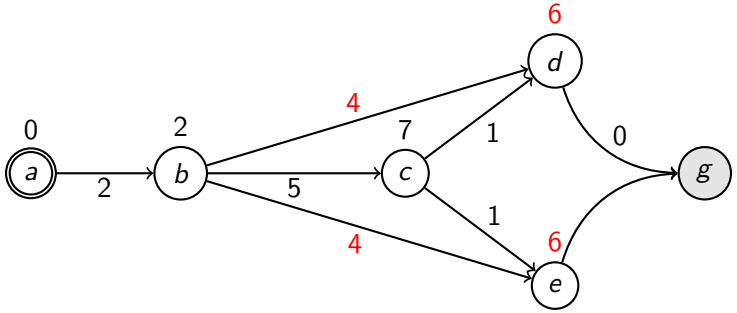
← Priority			
b(2)			

Example: Generalized Dijkstra method



← Priority			
d(6)	e(6)	c(7)	

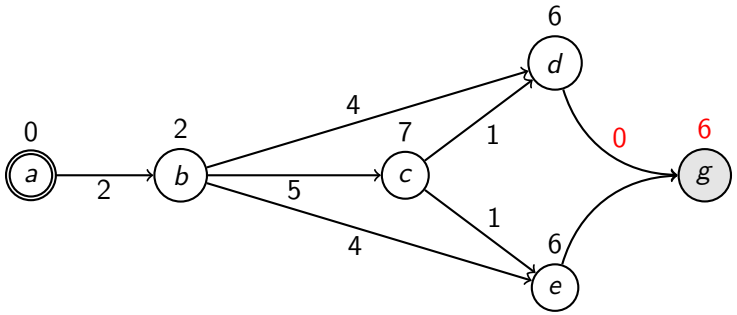
Example: Generalized Dijkstra method



g has priority 6 when computing $h^{\max}$
$\rightsquigarrow$ would have priority 12 when computing h^{add}

← Priority			
g(6)	c(7)		

Example: Generalized Dijkstra method



Can stop when all goals are reached – Generalized Dijkstra guarantees single update, so goal costs won't decrease

← Priority			
c(7)			

Comments

Generalized Dijkstra performs single update per fluent

- Slower due to overhead from priority queue

In practice, Generalized Bellman-Ford often used

- However, GD + incremental computation shown to give speedup (Liu, Koenig, and Furcy, 2002)

Problems with h^{add}

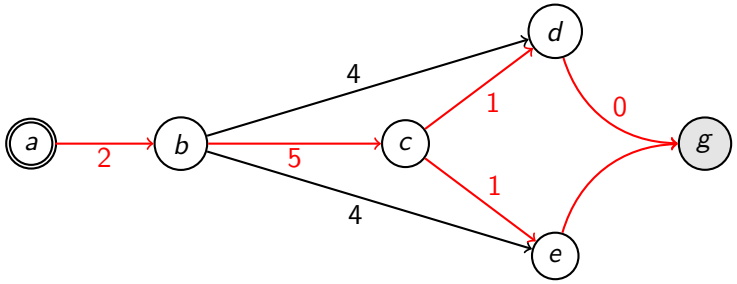


Figure: $h^+(G) = 2 + 5 + 1 + 1 + 0 = 9$

$h^{add}(G) = 12$ – What are the sources of error?

1. Overcounting – Easier to address
   - $a \rightarrow b$ counted twice
2. Independence assumption
   - $h^{add}(d, e) = h^{add}(d) + h^{add}(e)$ – not optimal

Relaxed Plan Heuristics

Idea: Find explicit plan π^+ for Π^+, $h = cost(\pi^+)$

- Incrementally construct plan with no duplicates
- No overcounting
- Suboptimal, not admissible

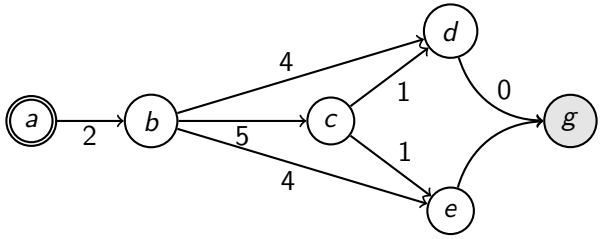
Relies on the idea of best supporters

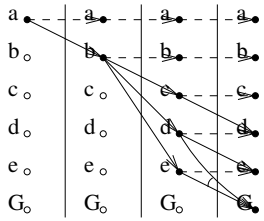
- Best supporter of fluent p is action chosen to make p true

First used in FF with relaxed planning graphs (Hoffmann and Nebel, 2001)

Relaxed Planning Graph

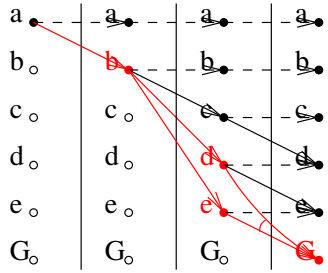
- Tool to graphically represent heuristic computation
- Layer i contains facts achievable with i layer parallel plan





The FF Heuristic

Construct relaxed plan by starting from goal, choose supporter for each fluent at first layer in which it appears



h^{FF} Relaxed Plan

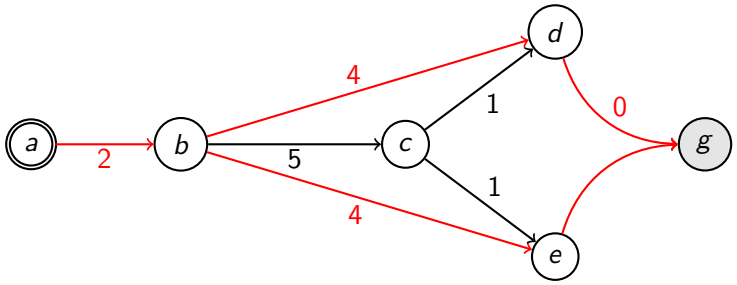


Figure: Plan computed by h^{FF}

$$h^{add}(s) = 12$$

$$h^{FF}(s) = c(\pi) = \{\langle a \to b\rangle, \langle b \to d\rangle, \langle b \to e\rangle, \langle d, e \to g\rangle\} = 10$$

Relaxed Plan Heuristics - Benefits

Generate an explicit plan π whose cost is the heuristic value, rather than just a numeric estimate

Advantages:

- Explicit representation of plan – no overcounting!
- Helpful actions: Operators likely to lead to better state
  $\rightsquigarrow$ Used by search algorithms to generate/evaluate fewer nodes

Relaxed Planning Graphs and h^{max}

When all actions cost 1, the relaxed planning graph level of a fact is equal to its h_{max} estimate

If $p \in s$:

- $h^{max}(p) = 0$
- $rpg\text{-}level(p) = 0$

else:

- $h^{max}(p) = \min_{a \in O(p)} (1 + h^{max}(Pre(a)))$
- $rpg\text{-}level(p) = \min_{a \in O(p)} (1 + rpg\text{-}level(Pre(a)))$

Relaxed Plans from h^{max}, h^{add}, etc.

h^{FF} uses uniform cost h^{max} supporters + plan extraction algorithm

Drawback: Uniform cost h^{max} not cost-sensitive

Solution: Cost-sensitive heuristic to choose best supporter (Keyder & Geffner, 2008, others) ...

$$o_p(s) = \operatorname*{argmin}_{\{o | p \in add(o)\}} cost(o) + h(pre(o); s)$$

... combined with generic plan extraction algorithm

- Construct π^+ by collecting the best supporters recursively backwards from the goal ...

Relaxed Plan Extraction Algorithm

Input: A state s
Input: A *best supporter* function o_p
function $\pi(G; s)$ **begin**
  $\pi^+ \leftarrow \emptyset$
  supported $\leftarrow s$
  goals $\leftarrow G$
  while *to-support* $\neq \emptyset$ **do**
    Remove a fluent p from *goals*
    if $p \notin$ *supported* **then**
      $\pi^+ \leftarrow \pi^+ \cup \{o_p(s)\}$ *supported* $\leftarrow$ *supported* $\cup$ *add*$(o_p(s))$
      goals $\leftarrow$ *goals* $\cup$ (*Pre*$(o_p(s))$ $\setminus$ *supported*)
    endif
  end
  return π^+
end

Relaxed Plans

. . . and estimate the cost of a state s as the cost of $\pi^+(s)$:

$$h(s) = Cost(\pi^+(s)) = \sum_{o \in \pi^+(s)} cost(o)$$

Results in cost-sensitive heuristic with no overcounting

The Set-Additive Heuristic h^{add}_{set}

Different method for computing relaxed plans, sometimes with higher quality (Keyder & Geffner, 2008)

Idea: Instead of costs, propagate the supports themselves

- For each fluent, maintain explicit relaxed plan
- Obtain plan for set as union of plans for each
- Seeds for computation are also sets:

$$\pi(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The Set-Additive Heuristic h^{add}_{set}

$$h^{add}_{set}(s) = Cost(\pi(G; s))$$

$$\pi(P; s) = \bigcup_{p \in P} \pi(p; s)$$

where

$$\pi(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi(o_p(s); s) & \text{otherwise} \end{cases}$$

$$Cost(\pi) = \sum_{a \in \pi} cost(a)$$

$$o_p(s) = \operatorname*{argmin}_{\{o | p \in add(o)\}} \left[Cost(\{o\} \cup \bigcup_{q \in pre(o)} \pi(q; s)) \right]$$

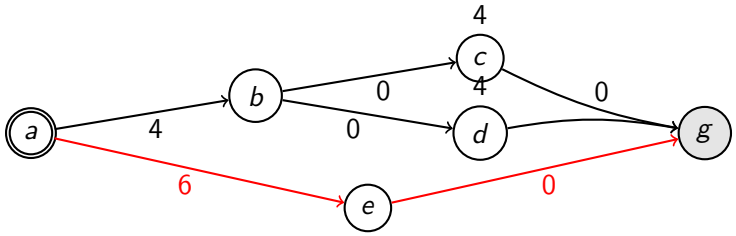
The Set-Additive Heuristic h^{add}_{set}

Intuition: Estimate the cost of sets considering overlap between plans for individual fluents

- Potentially better estimates

Drawback: $\cup$ operation expensive compared to cheaper $\sum$ or max operation

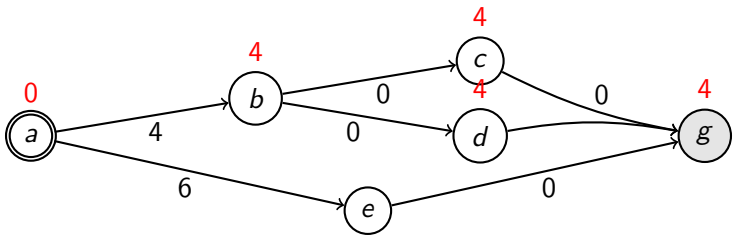
Example: h^{add}



h^{add} counts cost of transition $a \to b$ twice when computing cost of upper path, leading to goal cost of 8

- Therefore chooses suboptimal lower path of cost 6

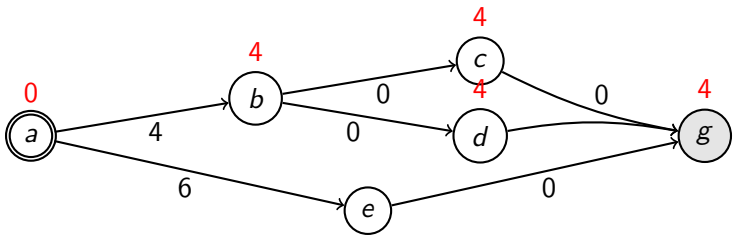
Example: h^{add}_{set} Computation



Initialization

$\pi(a)$	$\{\}$
$\pi(b)$	*undefined*
$\pi(c)$	*undefined*
$\pi(d)$	*undefined*
$\pi(e)$	*undefined*
$\pi(g)$	*undefined*

Example: h^{add}_{set} Computation



$a \to b$ counted only once when taking union of plans

$\pi(a)$	$\{\}$
$\pi(b)$	$\{a \to b\}$
$\pi(c)$	$\{a \to b, b \to c\}$
$\pi(d)$	$\{a \to b, b \to d\}$
$\pi(e)$	*undefined*
$\pi(g)$	$\{a \to b, b \to c, b \to d, cd \to g\}$

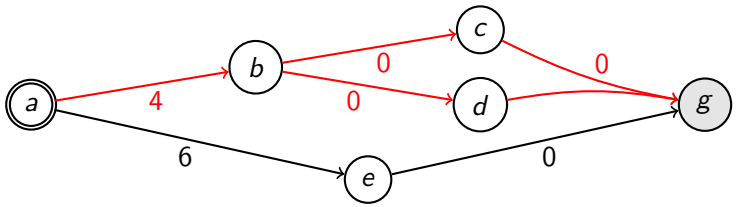


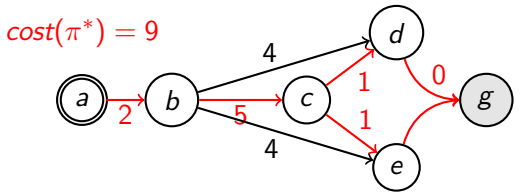
Figure: $cost(\pi^{add}_{set}) = 4$

Computing relaxed plans for each fluent allows detection of shared cost

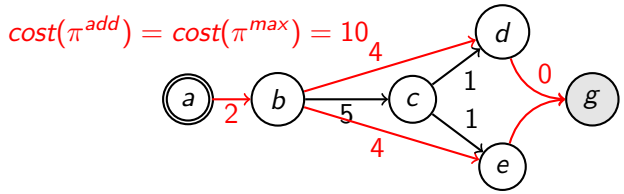
Remaining issues

Relaxed-plan heuristics solve the overcounting issue by computing an explicit relaxed plan with no duplicate actions

Independence assumption issues remain



vs.



The Steiner Tree Problem

The Steiner Tree Problem

Given a graph $G = \langle V, E \rangle$ and a set of *terminal nodes* $T \subseteq V$, find a minimum-cost tree S that spans all $t \in T$

When $T = V$, this is the tractable Minimum Spanning Tree (MST) problem

Otherwise, equivalent to finding best set of non-terminal nodes P to span, known as Steiner Points

- Steiner Tree is then MST over $P \cup T$

Improving Steiner Trees

The Steiner Tree heuristic

Exploits parallels between Steiner trees and relaxed plans, and MST property of Steiner trees, to obtain an improvement algorithm for relaxed plans (Keyder & Geffner, 2009)

Start with a suboptimal relaxed plan, and attempt to get a better one

General idea:

1. Remove a partial plan π' from the relaxed plan
2. Try to find a new partial plan π'' with lower cost that makes it whole again

Intuition: Make relaxed plan closer to MST

Conclusions

- The optimal delete relaxation heuristic h^+ is admissible and informative, but calculating it is NP-hard
  - Sub-optimal solutions have proven to be effective heuristics for satisficing planning
- h^{add} suffers from two problems:
  1. Overcounting of actions when combining estimates
  2. The independence assumption
- Overcounting problem can be solved with relaxed-plan heuristics
  $\rightsquigarrow$ Cost-sensitive best supporters
- Steiner Tree heuristic is an attempt at improving estimates obtained with the independence assumption

Heuristics for Domain-Independent Planning
3. Critical Path Heuristics

Emil Keyder Silvia Richter

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

$h^{\max} = h^1$

$h^{\max}$ estimates cost of set P as max cost of any $p \in P$:

$$h^{\max}(P; s) = \max_{p \in P} h^{\max}(p; s)$$

Alternatively:

$$h^1(P; s) = \max_{P' \subseteq P \wedge |P'| \leq 1} h^1(P'; s)$$

Idea: Generalize to arbitrary m

h^m Heuristics

Consider subsets of P of size m:

$$h^m(P; s) = \max_{P' \subseteq P \wedge |P'| \leq m} h^m(P'; s)$$

Estimate cost of P as most expensive subset of size m or less (Haslum & Geffner, 2000)

Question: How can a set of fluents P' of size m be made true?

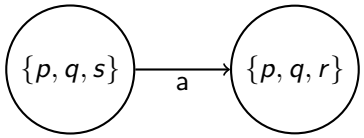
Two ways:

1. $\{a \mid P' \subseteq add(a)\}$
2. $\{a \mid P' \cap add(a) \neq \emptyset \wedge P' \cap del(a) = \emptyset\}$

h^m heuristics take into account delete information

Example

Consider the action $a : \{p, s\} \rightarrow \{r, \neg s\}$



- a always makes $\{p, r\}$ true
- a makes $\{q, r\}$ true if q is true when it is applied
- a never makes any $\{P' \mid s \in P'\}$ true

Formalizing h^m: Regression

For a set of fluents P s.t. $|P| \leq m$, the regression of P through a is:

$$R(P, a) = (P \setminus add(a)) \cup pre(a)$$

and is defined when:

- $P \cap del(a) = \emptyset$
- $P \cap add(a) \neq \emptyset$

Intuition: To make all fluents in P true with a, what would have to be true in the state in which a is applied?

Formalizing h^m: Equations

$$h^m(s) = h^m(G; s)$$

$$h^m(P; s) = \begin{cases} 0 & \text{if } P \subseteq s \\ \min_a(cost(a) + h^m(R(P, a); s) & \text{if } |P| \leq m) \\ \max_{P' \subset P \wedge |P'| \leq m} h^m(P'; s) & \text{otherwise} \end{cases}$$

Computation and Use

Can be computed with label-correcting or generalized Dijkstra method

- Initialization: $h^m(P) = 0$ if $P \subseteq s$, ∞ otherwise

Computation is polynomial for fixed m, exponential in m in general

- Number of subsets of size $\leq m$ is $O(|F|^m)$
- Polynomial can still be very expensive

Too slow to compute in every state

- Used in backwards search
- And to compute mutexes

Mutexes

The h^2 heuristic is commonly used to compute mutexes

- $h^2(\{p, q\}; s_0) = \infty \implies p$ and q are mutex
- Mutexes give finite-domain variables

Mutex detection is sound but not complete (for fixed m)

- Assume $h^3(\{p, q, r\}) = \infty$, but $h^2(\{p, q\})$, $h^2(\{p, r\})$, $h^2(\{q, r\})$ are all finite.
- If the only action making $\{s, t\}$ true has precondition $\{p, q, r\}$, $h^2(\{s, t\})$ is finite, but s and t are mutex.

Properties

For large enough m, $h^m = h^*$

- In the worst case need $m = |F|$

h^+ and h^m are both admissible heuristics, but incomparable

- h^m is not a delete relaxation heuristic

The Π^m Compilation

The Π^m task is a planning task that encodes h^m information (Haslum, 2009)

- Fluents of Π^m are sets of fluents in Π
- One operator in $\Pi \rightarrow$ many operators in Π^m, each with different context
  $\rightsquigarrow$ Fluents that were true in addition to $pre(a)$ when a was applied
- Consider deletes when constructing Π^m actions

Key properties:

- $h^1(\Pi^m) = h^m(\Pi)$
- No deletes in Π^m

Formal Definition

Definition (Π^m)

Given a STRIPS task $\Pi = \langle F, s_0, G, O, cost \rangle$, its Π^m compilation is given by a STRIPS task $\Pi^m = \langle F^m, s_0^m, G^m, O^m, cost \rangle$, where $F^m = \{\pi_C \mid C \subseteq F \wedge |C| \leq m\}$, s_0^m and G^m are defined analogously, and

$$O^m = \{o_C \mid o \in O \wedge C \in contexts(o)\}$$

where

$$\begin{aligned} contexts(o) &= \{C \in F^{m-1} \mid C \cap add(o) = C \cap del(o) = \emptyset\} \\ pre(o_C) &= \{\pi_C \mid C \subseteq (pre(o) \cup C) \wedge |C| \leq m\} \\ add(o_C) &= \{\pi_C \mid C \subseteq (add(o) \cup C) \wedge |C| \leq m\} \\ del(o_C) &= \emptyset \end{aligned}$$

Conclusions

- h^m heuristics are admissible estimates obtained by considering critical paths
- h^m heuristics take deletes into account, and are incomparable to h^+
- Their computation is polynomial for fixed m, but exponential in m in general
- As $m \to |F|$, $h^m(s) \to h^*(s)$

Heuristics for Domain-Independent Planning
4. The Context-Enhanced Additive Heuristic

Emil Keyder Silvia Richter

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Background

h^{cea} (Helmert & Geffner, 2008) is a reformulation of an earlier heuristic, h^{CG} (Helmert, 2004)

- Replaces algorithm with recursive equations, clarifying connection to h^{add}
- Removes limitations on problem structure

h^{cea} is defined in terms of finite-domain variables (SAS$^+$)

Intuition: To achieve values for a set of variables, achieve one of them first
$\rightsquigarrow$ Compute cost of the other variables considering side effects of achieving the first one

h^{add} in SAS$^+$

h^{add} can be written for SAS$^+$ problems as follows:

$$h^{add}(G; s) = \sum_{x \in G} h^{add}(x; x_s)$$

$$h^{add}(x \mid x_s) = \begin{cases} 0 & \text{if } x = x_s \\ \min_{o: z \to x} \left[cost(o) + \sum_{x_i \in z} h^{add}(x_i \mid x_{i_s}) \right] & \text{if } x \neq x_s \end{cases}$$

where $o : z \to x$ is an action that assigns x and has z as a precondition, and x_s is the value of the variable of x in s.

h^{add} in SAS$^+$

h^{add} can be written for SAS$^+$problems as follows:

$$h^{add}(G;s) = \sum_{x \in G} h^{add}(x;x_s)$$

$$h^{add}(x \mid x_s) = \begin{cases} 0 & \text{if } x = x_s \\ \min\limits_{o:z \to x} \left[cost(o) + \sum\limits_{x_i \in z} h^{add}(x_i \mid x_{is'}) \right] & \text{if } x \neq x_s \end{cases}$$

where $o : z \to x$ is an action that assigns x and has z as a precondition, and x_s is the value of the variable of x in s.

Idea of h^{cea}: Take into account side effects by evaluating precondition x_i in a state s' that is different from s

h^{cea}

$$h^{cea}(x \mid x') = \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',z \to x} \left[cost(o) + h^{cea}(x'' \mid x') + \sum\limits_{x_i \in z} h^{cea}(x_i \mid x_i') \right] & \text{if } x \neq x' \end{cases}$$

h^{cea} considers $pre(o)$ in two parts:

- The value x'', defined on the same variable as x
- z, the rest of the precondition

$$x' \longrightarrow \ldots \longrightarrow x'' \xrightarrow{o} x$$

h^{cea}

$$h^{cea}(x \mid x') = \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',z \to x} \left[cost(o) + h^{cea}(x'' \mid x') + \sum\limits_{x_i \in z} h^{cea}(x_i \mid x_i') \right] & \text{if } x \neq x' \end{cases}$$

The cost of achieving the preconditions is expressed as the heuristic cost of achieving x'' ...

$$x' \longrightarrow \ldots \longrightarrow x'' \xrightarrow{o} x$$

h^{cea}

$$h^{cea}(x \mid x') = \begin{cases} 0 & \text{if } x = x' \\ \min\limits_{o:x'',z \to x} \left[cost(o) + h^{cea}(x'' \mid x') + \sum\limits_{x_i \in z} h^{cea}(x_i \mid x_i') \right] & \text{if } x \neq x' \end{cases}$$

... plus the heuristic cost of achieving the other preconditions from their values x_i' that result from achieving x''

The values x_i' are the values in the projected state $s(x'' \mid x')$

Assumption: x'' is achieved first – The pivot condition

$$x' \longrightarrow \ldots \longrightarrow x'' \xrightarrow{o} x$$

How to Compute $s(x'' \mid x')$

Equations defining h^{cea} and $s(x'' \mid x')$ are mutually recursive

Let $o : x''', z' \to x'', y_1, \ldots, y_n$ be the operator that results in minimum h^{cea} value for x''
$\rightsquigarrow$ Similar to best supporters in Π^+

Notation: $s[x']$ denotes s "overridden" with x'

$$s(x'' \mid x') = \begin{cases} s[x'] & \text{if } x'' = x' \\ s(x''' \mid x')[z'][x'', y_1, \ldots, y_n] & \text{if } x'' \neq x' \end{cases}$$

$$\begin{array}{ccccccc} x' & \ldots & \longrightarrow & x''' & \xrightarrow{o} & x'' & \longrightarrow & x \\ s & \ldots & \longrightarrow & s(x''' \mid x') & \xrightarrow{o} & s(x'' \mid x') & \longrightarrow & s(x \mid x') \end{array}$$

How to Compute $s(x'' \mid x')$

The state $s(x'' \mid x')$ is defined recursively beginning from $s(x''' \mid x')$...
... and updated with $pre(o) \setminus \{x'''\} = z'$ ...
$\rightsquigarrow$ known to be true, since precondition of o
... and finally with $eff(o) = x'' \cup \{y_1, \ldots, y_n\}$
$\rightsquigarrow$ known to be true, since effect of o

$$s(x'' \mid x') = \begin{cases} s[x'] & \text{if } x'' = x' \\ s(x''' \mid x')[z'][x'', y_1, \ldots, y_n] & \text{if } x'' \neq x' \end{cases}$$

$$\begin{array}{ccccccc} x' & \ldots & \longrightarrow & x''' & \xrightarrow{o} & x'' & \longrightarrow & x \\ s & \ldots & \longrightarrow & s(x''' \mid x') & \xrightarrow{o} & s(x'' \mid x') & \longrightarrow & s(x \mid x') \end{array}$$

h^{cea}

The value of h^{cea} for the set of goals G is the same as h^{add}:

$$h^{\text{cea}}(s) = \sum_{x \in G} h^{\text{cea}}(x \mid x_s)$$

Sum costs of achieving goal value for each variable in the goal given its value x_s in s

Example: Computation of h^{cea}

Let $\Pi = \langle V, O, s_0, G, cost \rangle$ be an SAS$^+$problem with

- V $X = \{x_0, \ldots, x_n\}$
  $Y = \{true, false\}$
- O $a : \{\neg y\} \to \{y\}$ $b_i : \{y, x_i\} \to \{\neg y, x_{i+1}\}$
- s_0 $\{x_0, y\}$
- G $\{x_n\}$
- $cost$ $c(a) = c(b_i) = 1$

The optimal plan is then

$$\pi^* = \langle b_0, a, \ldots, a, b_{n-1} \rangle$$

containing $n \times b_i + (n-1) \times a = 2n - 1$ actions

Example: Computation of h^{cea}

$$x_0 \xrightarrow{b_0} \ldots \xrightarrow{b_{n-2}} x_{n-1} \xrightarrow{b_{n-1}} x_n$$

What is the value of $s(x_i \mid x_0)$?

$$s(x_0 \mid x_0) = s = \{x_0, y\}$$

$$\begin{aligned} s(x_1 \mid x_0) = & s(x_0 \mid x_0)[pre(b_0)][eff(b_0)] \\ & \{x_0, y\}[pre(b_0)][eff(b_0)] \\ & \{x_0, y\}[eff(b_0)] \\ & \{x_1, \neg y\} \end{aligned}$$

General case: $s(x_i \mid x_0) = \{x_i, \neg y\}$

Example: Computation of h^{cea}

What is the value of $h^{cea}(x_i \mid x_0)$?

$$\begin{aligned} h^{cea}(x_0 \mid x_0) &= 0 \\ h^{cea}(x_1 \mid x_0) &= c(b_0) + h^{cea}(x_0 \mid x_0) + h^{cea}(y \mid y_{s(x_0|x_0)}) \\ &= 1 + 0 + 0 \\ h^{cea}(x_i \mid x_0) &= c(b_{i-1}) + h^{cea}(x_0 \mid x_{i-1}) + h^{cea}(y \mid y_{s(x_{i-1}|x_0)}) \\ &= c(b_{i-1}) + h^{cea}(x_0 \mid x_{i-1}) + h^{cea}(y \mid y_{\{x_0, \neg y\}}) \\ &= c(b_{i-1}) + h^{cea}(x_0 \mid x_{i-1}) + h^{cea}(y \mid \neg y) \\ &= 1 + h^{cea}(x_0 \mid x_{i-1}) + 1 \end{aligned}$$

Since $s(x_{i-1} \mid x_0) = \{x_{i-1}, \neg y\}$, y evaluated from value $\neg y$

Example: Computation of h^{cea}

We have:

$$\begin{aligned} h^{cea}(x_1 \mid x_0) &= 1 \\ h^{cea}(x_n \mid x_0) &= h^{cea}(x_{n-1} \mid x_0) + 2 \end{aligned}$$

h^{cea} gives optimal solution to this problem:

$$\begin{aligned} h^{cea}(x_n \mid x_0) &= 2(n-1) + 1 = 2n - 1 \\ h^{cea}(s) &= h^*(s) \end{aligned}$$

Observations

Depends on finite-domain variables

- With boolean variables, equivalent to h^{add}

$$x' \longrightarrow \ldots \longrightarrow x''(false) \longrightarrow x(true)$$

Must be the case that $x' = x'' = false$, so $s(x''|x') = s$

Preserves information about changes in a single variable from s

$$\ldots + \sum_{x_i \in z} h^{cea}(x_i \mid x_i')$$

Fixed order in which preconditions are assumed to be achieved
$\rightsquigarrow$ Always achieve value of same variable first

Conclusions

Can be more informative than h^+
$\rightsquigarrow$ But not in general comparable

Possible improvements:

- Consider precedences in the preconditions (Cai, Hoffmann, & Helmert, 2009)
- Consider projected states that differ from s in a bounded number of variables rather than a single one

Heuristics for Domain-Independent Planning
5. Landmark Heuristics

Emil Keyder Silvia Richter

Based partially on slides by Erez Karpas

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Landmarks

A landmark is a property of every plan for a planning task

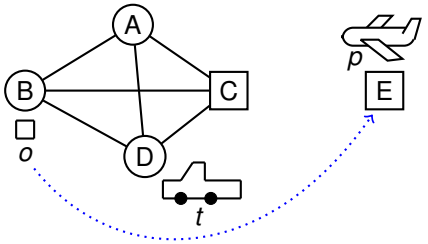
1. Propositional landmark: a formula ϕ over the set of fluents
   $\rightsquigarrow$ ϕ is made true in some state during the execution of any plan
2. Action landmark: a formula ψ over the set of actions
   $\rightsquigarrow$ ψ is made true by any plan interpreted as a truth assignment to its set of actions

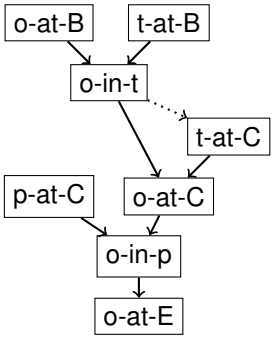
Landmarks can be (partially) ordered

Some landmarks and orderings can be discovered automatically
$\rightsquigarrow$ Mostly restricted to single fact/action landmarks or simple formulas (disjunctions/conjunctions)

Landmarks are used in some state-of-the-art heuristics

Example Planning Problem - Logistics





Partial landmarks graph

Landmarks from Other Landmarks

Propositional landmarks imply action landmarks

- A single fact landmark p implies the disjunctive action landmark

$$\bigvee_{\{a|p\in add(a)\}} a$$

Action landmarks imply propositional landmarks

- A single action landmark a implies the propositional landmarks

$$\bigwedge_{p\in pre(a)} p \text{ and } \bigwedge_{p\in add(a)} p$$

Types of Landmark Orderings

Sound landmark orderings

- Guaranteed to hold
- No pruning of search space since automatically satisfied

Unsound landmark orderings

- Additional constraints on plans
- May rule out valid solutions
- However, likely to hold and can save effort in planning

Sound Landmark Orderings

Natural ordering $A \rightarrow B$

- A always true some time before B becomes true

Necessary ordering $A \rightarrow_n B$

- A always true immediately before B becomes true

Greedy-necessary ordering $A \rightarrow_{gn} B$

- A true immediately before B becomes true for the first time

Note that $A \rightarrow_n B \implies A \rightarrow_{gn} B \implies A \rightarrow B$

Reasonable Orderings

- Not sound - not guaranteed to hold in all plans
- *Reasonable* ordering $A \rightarrow_r B$, iff given B was achieved before A, any plan must delete B on the way to A, and re-achieve B

$$B \rightsquigarrow \neg B \rightsquigarrow A \rightsquigarrow B \implies A \rightarrow_r B$$

- Initial state landmarks can be reasonably ordered after other landmarks (e. g., if they must be made false and true again)
- This can never happen with sound orderings

Landmark Complexity

Basic Result: Everything is PSPACE-complete

- Deciding if a propositional formula is a landmark is PSPACE-complete
  $\rightsquigarrow$ Proof Sketch: Same problem as deciding if the problem without actions that achieve this fact is unsolvable
- Deciding if there is a natural / necessary / greedy-necessary / reasonable ordering between two landmarks is PSPACE-complete

Landmark Discovery in Theory

Theory

$$A \text{ is a fact landmark} \iff \pi_A \text{ is unsolvable}$$

where π_A is a π with all actions that make A true removed

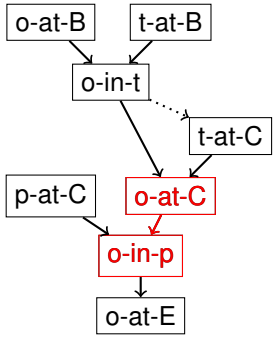
The delete relaxation π_A^+ is unsolvable $\implies \pi_A$ is unsolvable

- This can be used to obtain delete-relaxation landmarks
- But more efficient methods exist

Landmark Discovery I: Backchaining

Alternative: Find landmarks and orderings by backchaining (Hoffmann et al. 2004, Porteous & Cresswell 2002)

- Every goal is a landmark
- If B is a landmark and all actions that achieve B share A as precondition, then
  - A is a landmark
  - $A \rightarrow_n B$

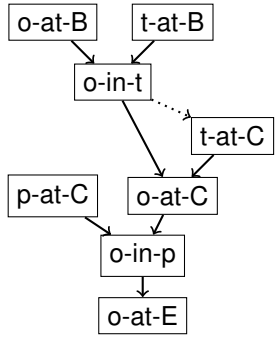


Useful restriction: consider only the case where B is achieved for the first time $\rightsquigarrow$ find more landmarks (and $A \rightarrow_{\text{gn}} B$)

Landmark Discovery I: Backchaining (ctd.)

PSPACE-complete to find first achievers
$\rightsquigarrow$ over-approximation by building relaxed planning graph for π'_B

- This graph contains no actions that add B
- Any action applicable in this graph can possibly be executed before B first becomes true $\rightsquigarrow$ possible first achievers

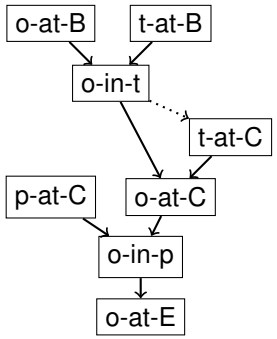


Additionally, if C not in the graph and C later proven to be a landmark, introduce $B \rightarrow C$

Landmark Discovery I: Backchaining (ctd.)

Disjunctive landmarks also possible,
e.g., (o-in-p_1 $\vee$ o-in-p_2):

- If B is a landmark and all actions that (first) achieve B have A or C as a precondition, then $A \vee C$ is a landmark
- Generalises to any number of disjuncts
- Large number of possible disjunctive landmarks $\rightsquigarrow$ must be restricted



Landmark Discovery I: Backchaining (ctd.)

Pro:

- Finds disjunctive landmarks as well as facts

Con:

- No criterion for which backchaining is complete for Π^+
- Requires arbitrary limits e. g. on size/form of disjunctions

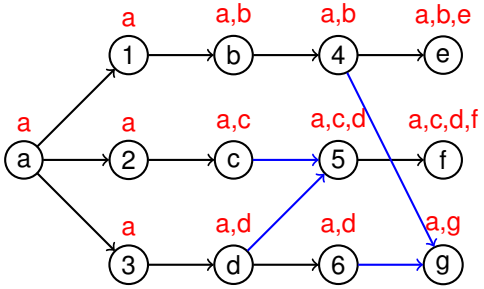
Landmark Discovery II: Forward Propagation

Find landmarks by forward propagation in relaxed planning graph (Zhu & Givan 2003)

- Finds all causal delete-relaxation landmarks in polynomial time (where causal means: appearing in preconditions for actions)
- Propagate information on necessary predecessors
  - Label each fact node with itself
  - Propagate labels along arcs

Landmark Discovery II (ctd.)

Actions (numbers): propagate union over labels on preconditions
— all preconditions are necessary
Facts (letters): propagate intersection over labels on achievers
— only what's necessary for all achievers is necessary for a fact



No-ops and repeated nodes not shown

Landmark Discovery II (ctd.)

- Goal nodes in final layer: labels are landmarks
- $A \rightarrow B$ if A forms part of the label for B in the final layer
- $A \rightarrow_{\text{gn}} B$ if A is precondition for all possible first achievers of B
- Possible first achievers of B are achievers that do not have B in their label (Keyder, Richter & Helmert 2010)

Advanced version of this method counts re-occurrences of landmarks

Landmark Discovery II: Complete Equations

The Zhu & Givan method can be seen as computing the fixpoint solution to a set of equations (Keyder et al.,2010):

$$LM(G) = \bigcup_{g \in G} LM(g)$$

$$LM(p) = \begin{cases} \{p\} & \text{if } p \in I \\ \{p\} \cup \bigcap_{\{a \mid p \in add(a)\}} LM(a) & \text{otherwise} \end{cases}$$

$$LM(a) = \{a\} \cup \bigcup_{p \in pre(a)} LM(p)$$

Advantages:

- Declarative definition
- No reliance on relaxed planning graph

Landmark Discovery IIa: Π^m Landmarks

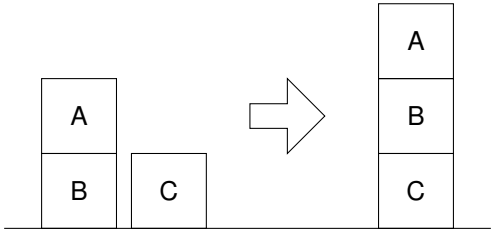
Landmark-detection methods for Π^+ can be applied to any problem without delete effects
$\rightsquigarrow$ This includes the Π^m compilation
Recall that fluents of Π^m are subsets of F of size $\leq m$

Fact landmarks found for Π^m then correspond to conjunctive landmarks for Π that take into account delete information

Any method for Π^+ landmark finding can be applied to Π^m

- Equations guarantee completeness for conjunctive landmarks as $m \rightarrow |F|$

Π^m Landmarks Example



Non-trivial Π^+ fact landmarks:

$$clear\ B \rightarrow_{gn} holding\ B$$

(Some) Π^2 Landmarks:

$$(clear\ B \wedge holding\ A) \rightarrow_{gn} (clear\ B \wedge ontable\ A) \rightarrow_{gn}$$
$$(holding\ B \wedge ontable\ A) \rightarrow_{gn} (on\ B\ C \wedge ontable\ A) \rightarrow_{gn}$$
$$(on\ B\ C \wedge holding\ A)$$

Finding Orderings

Natural and (greedy-)necessary orderings are found along with landmarks

- A is a landmark for B: $A \rightarrow B$
- A is a precondition for all achievers a of B that do not have B as a landmark: $A \rightarrow_{gn} B$

If reasonable orderings used, they are computed in a post-processing step.

- Not discussed here
- See landmark tutorial from ICAPS 2010

Other Methods for Landmark Finding

Find landmarks via domain transition graphs (Richter et al. 2008)

- Not discussed here
- See landmark tutorial from ICAPS 2010

Using Landmarks

- Some landmarks and orderings can be discovered efficiently
- So what can we do once we have these landmarks?

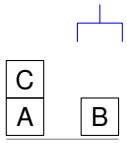
- Previous use as subgoals for search control (Hoffmann et al., 2004)
- Here, we focus on use for deriving heuristic estimates
- Next section assumes landmarks are computed once for initial state
- Recomputing for every state would be more informative but costly

Using Landmarks for Heuristic Estimates

- The number of landmarks that still need to be achieved is a heuristic estimate (Richter and Westphal 2010)
- Used by LAMA - winner of the IPC-2008 sequential satisficing track
- Inadmissible heuristic, because an action may achieve more than one landmark

Path-Dependency

- Suppose we are in state s. Did we achieve landmark A yet?
- Example: did we achieve holding(B)?



- There is no way to tell just by looking at s
- Achieved landmarks are a function of path, not state
  $\rightsquigarrow$ In contrast to previously discussed heuristics, this one is path-dependent.

The LAMA Heuristic

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = (L \setminus \text{Accepted}(s,\pi)) \cup \text{ReqAgain}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks - landmarks that must be achieved again

Accepted Landmarks

- A landmark A is first accepted by path π in state s if
  - all predecessors of A in the landmark graph have been accepted, and
  - A becomes true in s
- Once a landmark has been accepted, it remains accepted

Required Again Landmarks

- A landmark A is required again by path π in state s if:

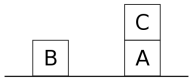
  false-goal A is false in s and is a goal, or

  open-prerequisite A is false in s and is a greedy-necessary predecessor of some landmark B that is not accepted
- It's also possible to use (Buffet and Hoffmann, 2010):

  doomed-goal A is true in s and is a goal, but one of its greedy-necessary successors was not accepted, and is inconsistent with A
- Unsound rule:

  required-ancestor is the transitive closure of *open-prerequisite*

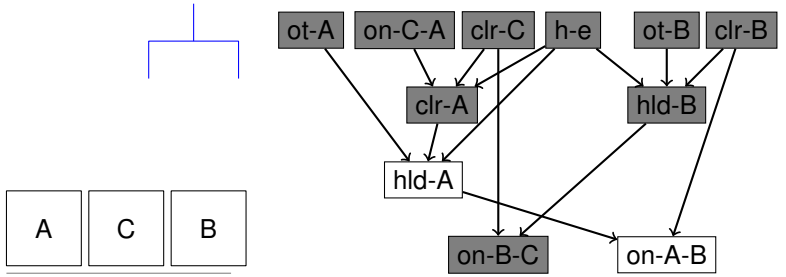
Using Landmarks as Subgoals - Sussman Example

- Consider the following blocks problem ("The Sussman Anomaly")
- Initial State 

- Goal: *on-A-B*, *on-B-C*

Accepted and Required Again Landmarks - Example

- In the Sussman anomaly, after performing: Pickup-B, Stack-B-C, Unstack-B-C, Putdown-B, Unstack-C-A, Putdown-C

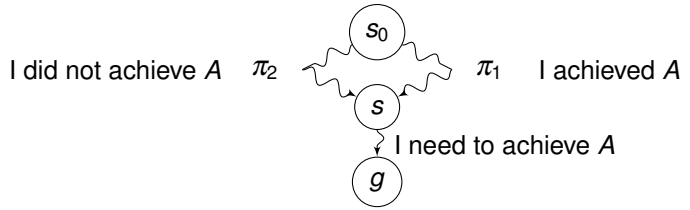


- on-B-C is a *false-goal*, and so it is required again

Problems With Reasonable Orderings

- Heuristic value of a goal state may be non-zero
  (if the plan found does not obey all reasonable orderings, and consequently not all landmarks are accepted).
  Solution: explicitely test states for goal condition
- The definition of reasonable orderings allows an ordering $A \rightarrow_r B$ if A and B become true simultaneously.
  Two solutions:
  - Accept a landmark if it has been made true at the same time as its predecessor (Buffet and Hoffmann, 2010)
  - Modify the definition of reasonable orderings to disallow such orderings

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark A and π_2 did not
- Then A needs to be achieved after state s
- Proof: A is a landmark, therefore it needs to be true in all plans, including plans that start with π_2

Fusing Data from Multiple Paths

- Improvement to path-dependent landmark heuristics (Karpas and Domshlak, 2009)
- Suppose $\mathscr{P}$ is a set of paths from s_0 to a state s. Define

$$L(s,\mathscr{P}) = (L \setminus \text{Accepted}(s,\mathscr{P})) \cup \text{ReqAgain}(s,\mathscr{P})$$

where
  - $\text{Accepted}(s,\mathscr{P}) = \bigcap_{\pi\in\mathscr{P}} \text{Accepted}(s,\pi)$
  - $\text{ReqAgain}(s,\mathscr{P}) \subseteq \text{Accepted}(s,\mathscr{P})$ is specified as before by s and the various rules
- $L(s,\mathscr{P})$ is the set of landmarks that we know still needs to be achieved after reaching state s via the paths in $\mathscr{P}$

Admissible Heuristic Estimates

- LAMA's heuristic h^{LM}: the number of landmarks that still need to be achieved
- h^{LM} is inadmissible - a single action can achieve multiple landmarks
  - Example: *hand-empty* and *on-A-B* can both be achieved by *stack-A-B*
- Admissible heuristic: assign a cost to each landmark, sum over the costs of landmarks (Karpas and Domshlak, 2009)

Ensuring Admissibility

- Actions share their cost between all the landmarks they achieve

$$\forall o \in O: \sum_{B\in L(o|s,\mathscr{P})} cost(o,B) \leq cost(o)$$

$cost(o,B)$: cost "assigned" by action o to B
$L(o|s,\mathscr{P})$: the set of landmarks achieved by o
- The cost of a landmark is the minimum cost assigned to it by any action
- Then

$$h^L(s,\pi) := cost(L(s,\pi)) = \sum_{B\in L(s,\pi)} cost(B)$$

is admissible

Cost Sharing - How?

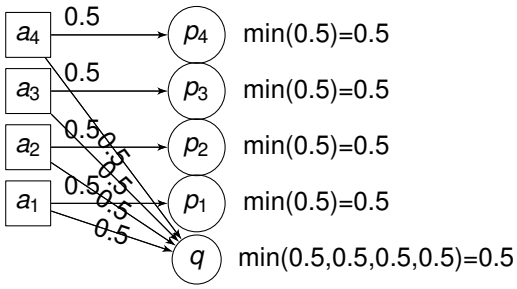
- How to share action costs between landmarks?
- Easy answer: uniform cost sharing - each action shares its cost equally between the landmarks it achieves

$$cost(o,B) = \frac{cost(o)}{|L(o|s,\pi)|}$$

Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example: all actions cost 1 – uniform cost sharing

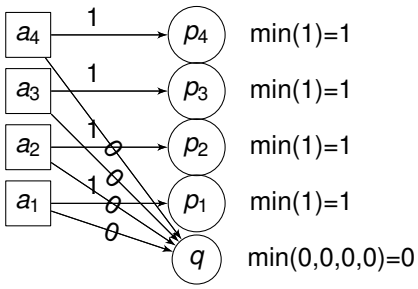
$h^{\mathsf{L}} = 2.5$



Uniform Cost Sharing

- Advantage: Easy and fast to compute
- Disadvantage: can be much worse than the optimal cost partitioning
- Example: all actions cost 1 – optimal cost sharing

$h^{\mathsf{L}} = 4$ uniform $h^{\mathsf{L}} = 2.5$



Optimal Cost Sharing

- The good news: the optimal cost partitioning is poly-time to compute
  - The constraints for admissibility are linear, and can be used in a Linear Program (LP)
  - Objective: maximize the sum of landmark costs
  - The solution to the LP gives us the optimal cost partitioning
- The bad news: poly-time can still take a long time

How can we do better?

So far:

- Uniform cost sharing is easy to compute, but suboptimal
- Optimal cost sharing takes a long time to compute

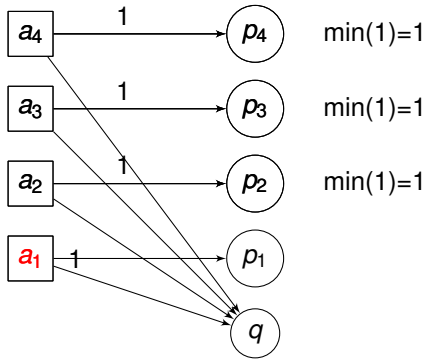
Q: How can we get better heuristic estimates that don't take a long time to compute?

A: Exploit additional information - action landmarks

Using Action Landmarks - by Example

Uniform Cost Sharing

$h^{LA} = 4$



Summary

- Landmarks describe the implicit structure of a planning task
- Can be used to derive admissible and inadmissible landmark-counting heuristics
- These heuristics are path-dependent

The Landmark Cut Heuristic

A landmark heuristic that is not path-dependent

- Computes landmarks for each state rather than once
- New way of finding landmarks
- Very accurate admissible approximation of h^+

The Landmark Cut Heuristic (cont.)

Idea (Helmert & Domshlak, 2009):

- Use critical paths (from h^{max} computation) to find disjunctive action landmarks
- Extract landmarks iteratively, subtracting their cost from h^{max} estimates
- Cost of each landmark := cost of cheapest action in landmark
- Heuristic value := sum over all landmark costs

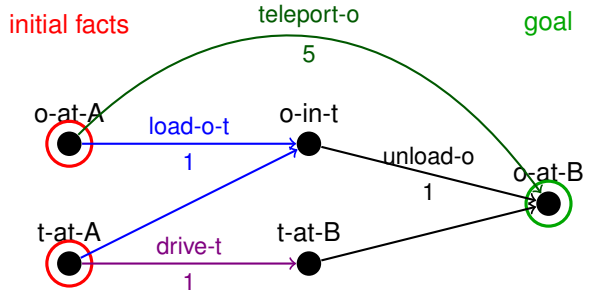
The Landmark Cut Heuristic (cont.)

Computation in a nutshell:

1. Compute $h^{\max}$ costs for all facts
2. Reduce goals and preconditions to singleton sets in a way that preserves $h^{\max}$
3. Replace all multi-effect operators by a set of unary operators in a way that preserves $h^{\max}$
4. Compute the justification graph for the resulting task, where $h^{\max}$ values are shortest distances
5. Compute a cut in the justification graph that separates before-goal zone from goal zone.
6. The cut corresponds to one disjunctive action landmark in the result Extract it through cost partitioning
7. Start from the beginning until $h^{\max}$ values are 0

$h^{\text{LM-cut}}$ Example

Transport object from A to B, by truck or by teleportation
(Partial) RPG:

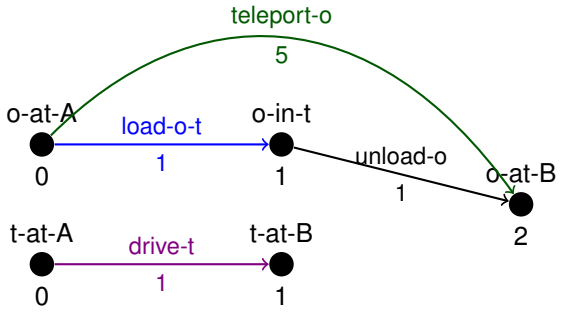


Optimal: load, drive, unload $\rightsquigarrow$ cost $= 3$
Suboptimal: teleport $\rightsquigarrow$ cost $= 5$

$h^{\text{LM}} = h^{\text{LA}} = 1$
$h^{\max} = 2$
$h^{+} = h^{\text{LM-cut}} = 3$

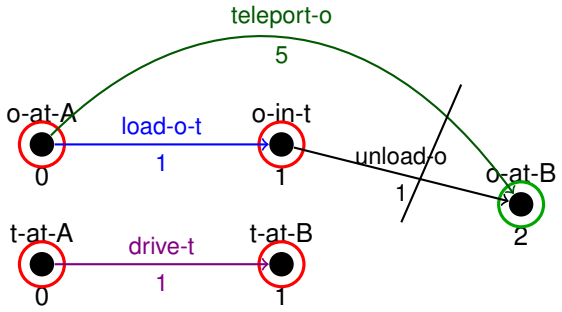
$h^{\text{LM-cut}}$ Example (cont.)

1. Compute $h^{\max}$ costs for all facts
2. Reduce goals and preconditions to singleton sets in a way that preserves $h^{\max}$
3. Replace all multi-effect operators by a set of unary operators in a way that preserves $h^{\max}$



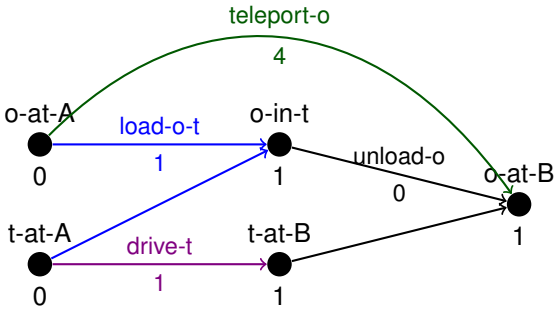
$h^{\text{LM-cut}}$ Example (cont.)

4. Compute the justification graph for the resulting task, where $h^{\max}$ values are shortest distances
5. Compute a cut in the justification graph that separates before-goal zone from goal zone.



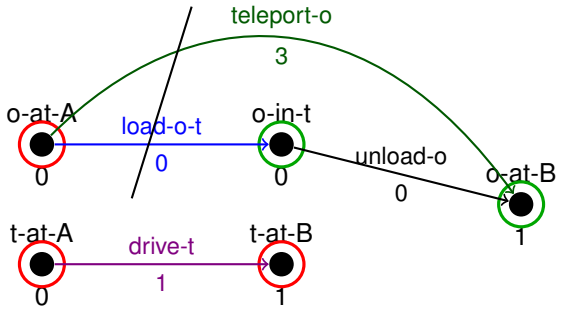
$h^{\text{LM-cut}}$ Example (cont.)

7. The cut corresponds to a disjunctive action landmark in the result
   $\rightsquigarrow$ teleport-o $\vee$ unload-o
   Extract it through cost partitioning
8. Start from the beginning until $h^{\max}$ values are 0



$h^{\text{LM-cut}}$ Example (cont.)

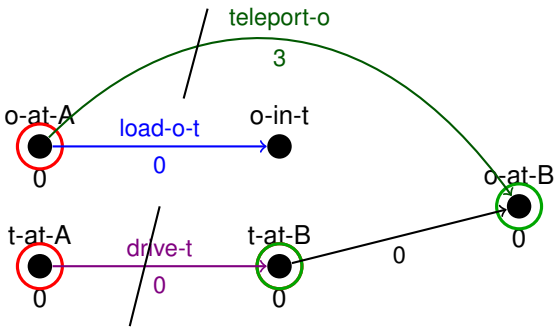
Landmarks $= \{$teleport-o $\vee$ unload-o,
teleport-o $\vee$ load-o-t$\}$



$h^{\text{LM-cut}}$ Example (cont.)

Landmarks $= \{$teleport-o $\vee$ unload-o,
teleport-o $\vee$ load-o-t,
teleport-o $\vee$ drive-t$\}$

$h^{\text{LM-cut}} = 3$



Conclusion

- Clever, heuristically guided way of discovering landmarks
- $h^{\text{LM-cut}}$ approximates h^+ very closely
- Landmarks are computed per state, not per task
  $\rightsquigarrow$ more expensive than other landmark heuristics
- Accuracy can be improved (at further computational expense) using hitting sets (Bonet & Helmert, 2010)
  1. Run procedure multiple times to get different (possibly overlapping) disjunctive landmarks
  2. Compute a set of actions that hits each disjunction

  $\rightsquigarrow$ Optimal hitting set NP-hard, approximate

Heuristics for Domain-Independent Planning
6. Abstraction Heuristics

Emil Keyder Silvia Richter

Based on slides by Carmel Domshlak and Malte Helmert

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Motivation

Like delete-relaxation heuristics, abstraction heuristics are derived from a simplification of the task.

But here, we simplify the search space directly, rather than via the operators.

Transition Systems

Definition (transition system)

A transition system is a 5-tuple $\mathcal{T} = \langle S, L, T, I, G \rangle$ where

- S is a finite set of states (the state space),
- L is a finite set of (transition) labels,
- $T \subseteq S \times L \times S$ is the transition relation,
- $I \in S$ is the initial state, and
- $G \subseteq S$ is the set of goal states.

We say that $\mathcal{T}$ has the transition $\langle s, l, s' \rangle$ if $\langle s, l, s' \rangle \in T$.

Transition Systems of SAS$^+$ Planning Tasks

Definition (transition system of an SAS$^+$ planning task)

Let $\Pi = \langle V, s_0, G, O, cost \rangle$ be an SAS$^+$ planning task. The transition system of Π, in symbols $\mathcal{T}(\Pi)$, is the transition system $\mathcal{T}(\Pi) = \langle S', L', T', I', G' \rangle$, where

- S' is the set of states over V,
- $L' = O$,
- $T' = \{\langle s', o', t' \rangle \in S' \times L' \times S' \mid app(s', o') = t'\}$,
- $I' = s_0$, and
- $G' = \{s' \in S' \mid s' \models G\}$.

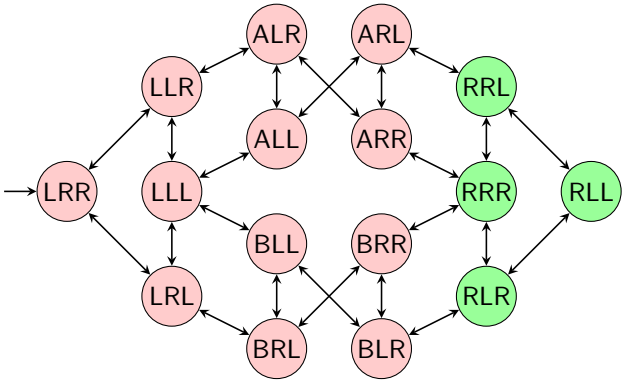
Example Task: One Package, Two Trucks

Example (one package, two trucks)

Consider a Logistics task with

- Two locations: L, R; two trucks: A, B; one package p
- Package can be at locations or in trucks: $\mathcal{D}_p = \{\mathrm{L}, \mathrm{R}, \mathrm{A}, \mathrm{B}\}$
  Trucks can be at the locations: $\mathcal{D}_{t_\mathrm{A}} = \mathcal{D}_{t_\mathrm{B}} = \{\mathrm{L}, \mathrm{R}\}$
- Init state: package at L, both trucks at R
- Goal: package at R
- Operators for pickup, drop and move
  - $\mathrm{pickup}_{i,j}$: pickup the package with truck i at location j
  - $\mathrm{drop}_{i,j}$ analogously
  - $\mathrm{move}_{i,j,j'}$: move truck i from location j to j'

Transition System of Example Task



LRR: package is at L, truck A is at R, and truck B is at R
Transition labels not shown.

Abstracting a Transition System

Abstracting a transition system means dropping some distinctions between states, while preserving the transition behaviour as much as possible.

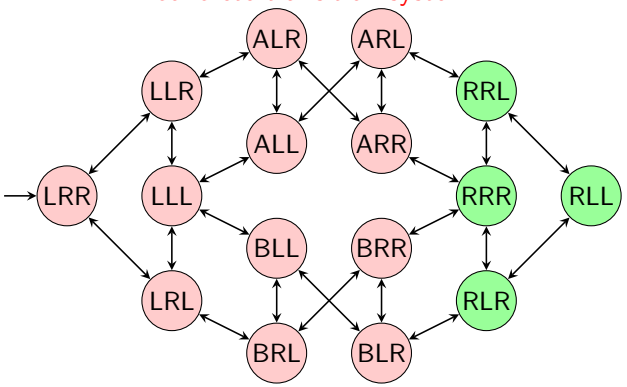
- An abstraction of a transition system $\mathcal{T}$ is defined by an abstraction mapping α that defines which states of $\mathcal{T}$ should be distinguished and which ones should not.
  $\rightsquigarrow$ s, t not distinguished if $\alpha(s) = \alpha(t)$
- From $\mathcal{T}$ and α, we compute an abstract transition system $\mathcal{T}'$ which is similar to $\mathcal{T}$, but smaller.

The abstract goal distances (goal distances in $\mathcal{T}'$) are used as heuristic estimates for goal distances in $\mathcal{T}$:

$$h^\alpha(s, T) = h^*(\alpha(s), T')$$

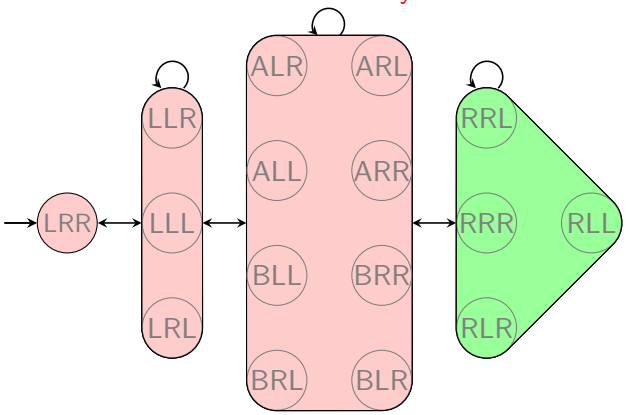
Abstraction: Example

concrete transition system



Abstraction: Example

abstract transition system



Note: Most arcs represent many parallel transitions. $h^\alpha(LRR) = 3$

Consistency of Abstraction Heuristics

Theorem (consistency and admissibility of $h^{\mathcal{A},\alpha}$)

Let Π be an SAS$^+$ planning task, and let $\mathcal{T}'$ be an abstraction of $\mathcal{T}(\Pi)$ with abstraction mapping α.
Then h^α is admissible and consistent.

Informativeness vs. Efficiency Tradeoff

- How to come up with a suitable abstraction mapping α?
- Conflicting goals for abstractions:
  - we want to obtain an informative heuristic, but
  - want it to be efficiently computable (both abstract state $\alpha(s)$ and $h^*(\alpha(s))$)
- Combinations of several abstractions possible

Automatically Deriving Good Abstractions

Abstraction heuristics for planning: main research problem

Automatically derive effective abstraction heuristics
for planning tasks.

So far, success in three directions:

- pattern database heuristics (Culberson & Schaeffer, 1996)
- merge-and-shrink abstractions (Dräger, Finkbeiner & Podelski, 2006)
- structural patterns (Katz & Domshlak, 2008)

Pattern Database Heuristics Informally

Pattern databases: informally

A pattern database (PDB) heuristic is an abstraction heuristic where

- some aspects of the task are represented in the abstraction with perfect precision, while
- all other aspects of the task are not represented at all.

Projections

PDBs: abstraction mappings are projections to certain state variables.

Definition (projections)

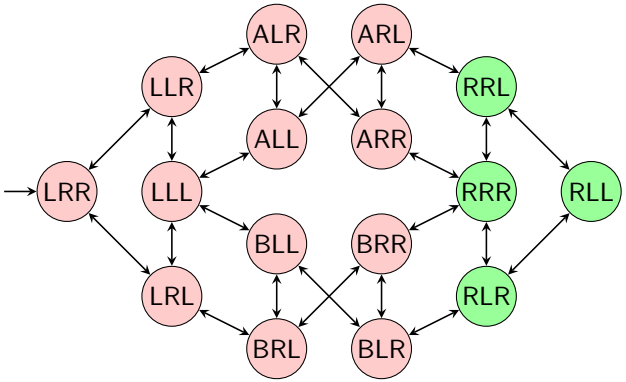
Let Π be an SAS$^+$ planning task with variable set V and state set S. Let $P \subseteq V$, and let S' be the set of states over P.

The projection $\pi_P : S \to S'$ is defined as $\pi_P(s) := s|_P$
(with $s|_P(v) := s(v)$ for all $v \in P$).

We call P the pattern of the projection π_P.

In other words, π_P maps two states s_1 and s_2 to the same abstract state iff they agree on all variables in P.

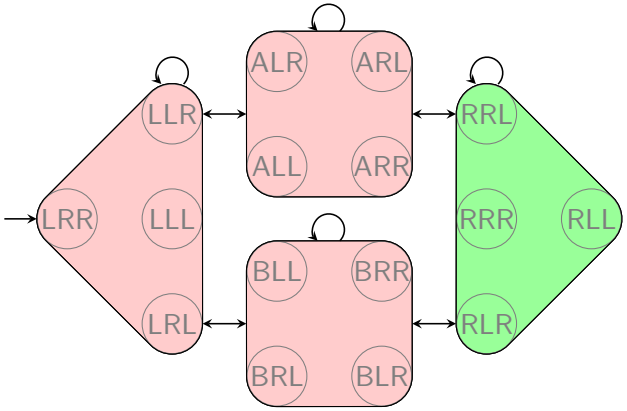
Example: Transition System



Logistics problem with one package, two trucks, two locations as before

Example: Projection

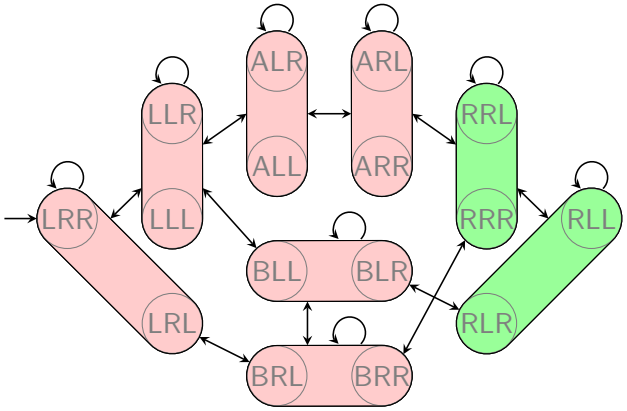
Abstraction induced by $\pi_{\{\text{package}\}}$:



$h^{\{\text{package}\}}(\text{LRR}) = 2$

Example: Projection (2)

Abstraction induced by $\pi_{\{\text{package,truck A}\}}$:



$h^{\{\text{package,truck A}\}}(\text{LRR}) = 2$

Pattern Collections

Space requirements for pattern databases grow exponentially with number of state variables in pattern.
$\rightsquigarrow$ Not practical for larger planning tasks.

Instead use collections of smaller patterns.

- Max of PDB heuristics is admissible (general property)
- Preferable to use sum when possible

Criterion for Additive Patterns

Simple criterion for admissible additivity: Count cost of each action in at most one heuristic.

$\rightsquigarrow$ Can add heuristic estimates from two patterns if no operator affects variables in both

Can find maximal additive subsets of a set of patterns

For a collection of patterns $\mathcal{P}$, the max of these sums is the canonical heuristic function for $\mathcal{P}$
$\rightarrow$ Most informative admissible heuristic we can derive from $\mathcal{P}$

Can do better with cost partitioning

How Do We Come Up with Good Patterns?

- The biggest practical problem when applying pattern databases to planning is to find a good pattern collection in a domain-independent way.
- Most promising approach: search in the space of possible pattern collections (prior to actual planning)

Algorithm of Haslum, Helmert, Bonet, Botea & Koenig (2007)

- Hill-climbing search in space of possible pattern collections
- Start from all singleton patterns $\{\{v\} \mid v \in V\}$
- Neighbors of collection $\mathcal{P}$: $\mathcal{P} \cup \{P \cup \{v\}\}$ for some $P \in \mathcal{P}$ and $v \notin P$
- Evaluation of a pattern collection $\mathcal{P}$:
  - Randomly sample heuristic for some states
  - Use search effort formula (Korf, Reid & Edelkamp, 2001)

Beyond Pattern Databases

Major limitation of PDBs: patterns must be small

Consider Logistics example with N trucks, M locations (still one package):

- If package $\notin P$, $h = 0$
- If any truck $\notin P$, $h \leq 2$

P must include all variables to improve over $P' = \{\text{package}\}$

Merge-and-shrink (M&S) abstractions are a proper generalization of pattern databases.

- They can do everything that pattern databases can do (modulo polynomial extra effort)
- They can do some things that pattern databases cannot

Merge-and-Shrink Abstractions: Main Idea

Main idea of M&S abstractions
(due to Dräger, Finkbeiner & Podelski, 2006):

Instead of perfectly reflecting a few state variables,
reflect all state variables, but in a potentially lossy way.

Do this with a sequence of

1. Merge steps – add a new variable to the abstraction
2. Shrink steps – reduce abstraction size by fusing abstract states

M&S Abstractions: Key Insights

Key insights:

1. Information of two transition systems $\mathcal{A}$ and $\mathcal{A}'$ can be combined (without loss) by a simple graph-theoretic operation, synchronized product $\mathcal{A} \otimes \mathcal{A}'$
2. The complete state space of a planning task can be recovered using only atomic projections:

$$\bigotimes_{v \in \mathcal{V}} \pi_v \text{ is isomorphic to } \pi_{\mathcal{V}}.$$

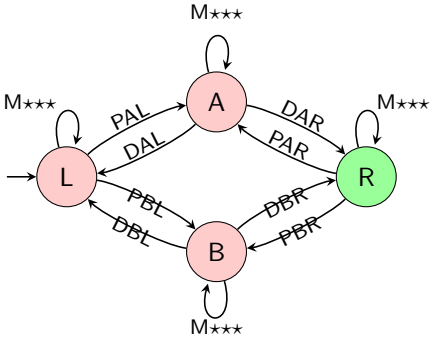
$\rightsquigarrow$ build fine-grained abstractions from coarse ones

Running Example: Explanations

- We abbreviate operator names as in these examples:
  - MALR: move truck A from left to right
  - DAR: drop package from truck A at right location
  - PBL: pick up package with truck B at left location
- We abbreviate parallel arcs with commas and wildcards ($\star$) in the labels as in these examples:
  - PAL, DAL: two parallel arcs labeled PAL and DAL
  - MA$\star\star$: two parallel arcs labeled MALR and MARL

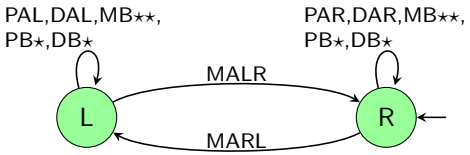
Running Example: Atomic Projection for Package

$\mathcal{T}^{\pi\{\text{package}\}}$:



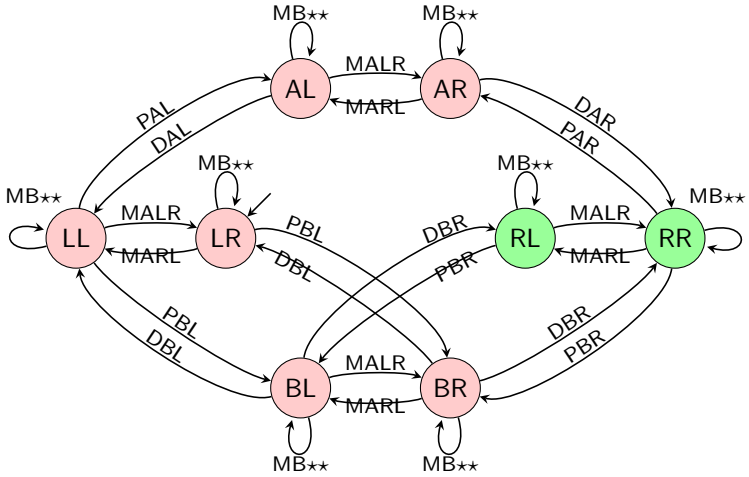
Running Example: Atomic Projection for Truck A

$\mathcal{T}^{\pi\{\text{truck A}\}}$:



Merging: Synchronized Product

$\mathcal{T}^{\pi\{\text{package}\}} \otimes \mathcal{T}^{\pi\{\text{truck A}\}}$:



M&S Abstractions: Key Insights

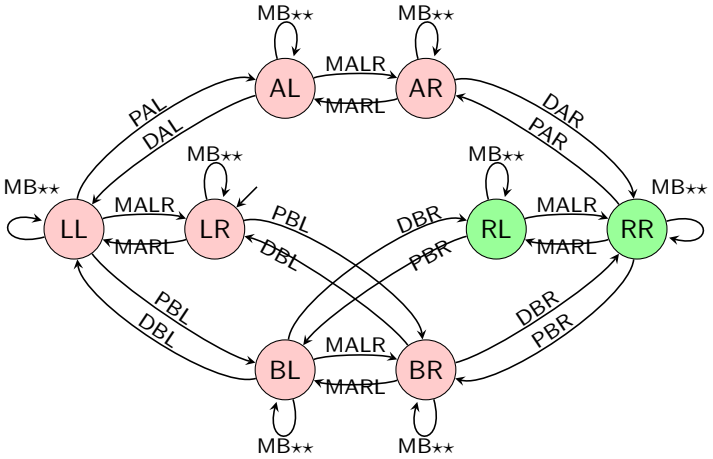
Key insights:

1. Information of two abstractions $\mathcal{A}$ and $\mathcal{A}'$ of the same transition system can be combined by a simple graph-theoretic operation (synchronized product $\mathcal{A} \otimes \mathcal{A}'$).
2. The complete state space of a planning task can be recovered using only atomic projections:

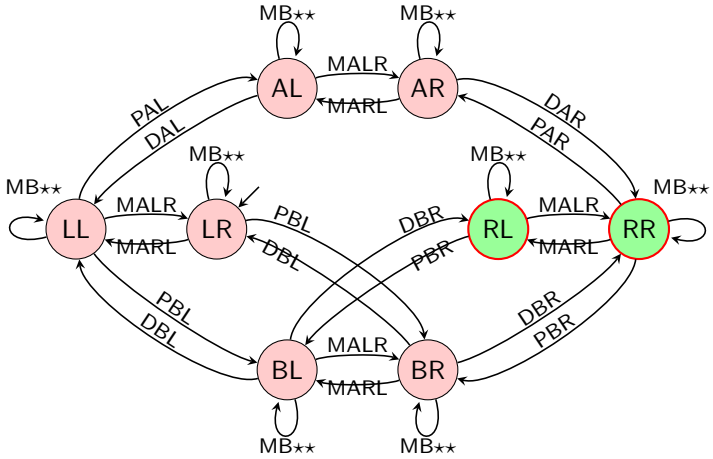
$$\bigotimes_{v \in \mathcal{V}} \pi_v \quad \text{is isomorphic to} \quad \pi_{\mathcal{V}}.$$

   $\rightsquigarrow$ build fine-grained abstractions from coarse ones
3. When intermediate results become too big, we can shrink them by fusing some abstract states.

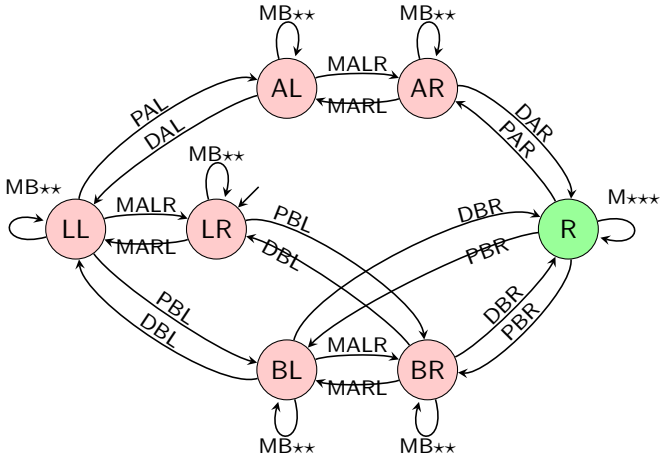
Shrinking



Shrinking



Shrinking



Computing M&S Abstractions

Generic abstraction computation algorithm

abs := all atomic projections π_v ($v \in \mathcal{V}$).
while abs contains more than one abstraction:
    select $\mathcal{A}_1$, $\mathcal{A}_2$ from abs
    shrink $\mathcal{A}_1$ and/or $\mathcal{A}_2$ until $size(\mathcal{A}_1) \cdot size(\mathcal{A}_2) \leq N$
    abs := abs $\setminus \{\mathcal{A}_1, \mathcal{A}_2\} \cup \{\mathcal{A}_1 \otimes \mathcal{A}_2\}$
return the remaining abstraction

N: parameter bounding number of abstract states

Questions for practical implementation:

- Which abstractions to select? $\rightsquigarrow$ merging strategy
- How to shrink an abstraction? $\rightsquigarrow$ shrinking strategy
- How to choose N?

Theoretical Properties

- PDB heuristics are merge-and-shrink abstractions without shrink steps

$\rightsquigarrow$ representational power is at least as large as that of PDB heuristics

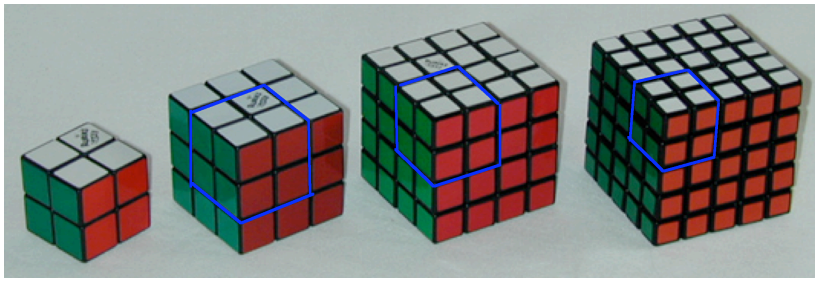
- In fact, representational power is strictly greater than that of PDB heuristics (demonstrated on some planning domains)
- However, specialized PDB algorithms are faster than the generic M&S algorithm
- And we do not generally know merging/shrinking strategies that will ensure superiority of M&S heuristics in practice

Limitation of Explicit Abstractions

No tricks: abstract spaces are searched exhaustively

$\rightsquigarrow$ must keep abstract spaces of fixed size

$\rightsquigarrow$ (often) price in heuristic accuracy in long-run



Structural Abstraction Heuristics: Main Idea

Instead of perfectly reflecting a few state variables,
reflect many (up to $\Theta(|V|)$) state variables, but

- ♠ guarantee abstract space can be searched (implicitly) in poly-time

How

Abstracting Π by an instance of a tractable fragment of cost-optimal planning

$\rightsquigarrow$ Fork Decomposition (Katz & Domshlak, 2008): decompose causal graph of task into fragments with single root/sink variable (and use further tricks)

Details: Carmel Domshlak's lecture in this summer school

Heuristics for Domain-Independent Planning
7. Final Comments

Emil Keyder Silvia Richter

Based partially on slides by Carmel Domshlak and Malte Helmert

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

Connecting the Pieces

- We have seen a number of heuristics in this tutorial, for satisficing and optimal planning
- Now the big question is:

Which ones work best?

And what's there beyond heuristics to improve planning-by-search?

Satisficing vs. Optimal Planning

Satisficing and optimal planning are much more different from each other than it appears at a superficial glance.

- Optimal planning entails proving that there is no better solution
- As a consequence, A*-based planning requires exponential effort in most domains even with almost perfect heuristics (Helmert & Röger, 2008).
- Not so for satisficing planning, where just finding a solution is enough

Satisficing Planning: Beyond Heuristics

- Because satisficing planners do not need to prove optimality, they can use drastic search control strategies
- These often make a larger difference in performance than the choice of heuristic function (Richter & Helmert, 2009)

Examples:

- helpful actions (FF: Hoffmann & Nebel, 2001)
- relaxed plan macros (YAHSP: Vidal, 2004)
- preferred operators (Fast Downward, LAMA: Helmert, 2006)
- alternating multi-queue search (Fast Downward, LAMA: Helmert & Röger, 2010)

Satisficing Planning: Conclusion

- Satisficing heuristics with good coverage (given limited time) are e. g. the FF heuristic and the context-enhanced additive heuristic, whereas the additive heuristic and inadmissible landmark-counting (by themselves) are less strong
- Picture less clear when measuring quality in addition to coverage ($\rightsquigarrow$ question of trade-off), in particular when planning with action costs

  E. g. cost-insensitive variant of FF heuristic dominates cost-sensitive one in terms of the IPC 2008 criterion (Richter & Westphal, 2010)

Optimal Planning: Conclusion

- Landmark heuristics like h^{LA} and $h^{\text{LM-cut}}$ shown to outperform additive $h^{\max}$ and abstraction heuristics like PDBs and M&S (under typical competition conditions)
- Theoretical results show M&S has the power to be more informative than PDBs and landmark heuristics, while additive $h^{\max} \sim$ landmarks (Helmert & Domshlak, 2009)
- But worth keeping in mind that higher accuracy does not necessarily pay off when time is limited

Heuristics for Domain-Independent Planning References

Emil Keyder Silvia Richter

Based partially on slides by Carmel Domshlak and Malte Helmert

ICAPS 2011 Summer School on Automated Planning and Scheduling

June 2011

About This List

- This list is meant to be focused, not comprehensive
- It is a somewhat subjective mix of papers we consider important and relevant to the tutorial topic
- Relevant papers might be missing because we forgot about them or do not know them

References: Past Tutorials


Carmel Domshlak and Malte Helmert
Planning as Heuristic Search
ICAPS 2009 Summer School.
Focus on admissible heuristics, basis of several parts of this tutorial.


Emil Keyder and Blai Bonet.
Heuristics for Classical Planning (With Costs).
ICAPS 2009 tutorial.
Emphasis on inadmissible heuristics based on delete relaxation.


Erez Karpas and Silvia Richter.
Landmarks - Definitions, Discovery Methods and Uses.
ICAPS 2010 tutorial.
Basis of the first landmark part of this tutorial.

References: Delete Relaxation


Blai Bonet and Héctor Geffner.
Planning as Heuristic Search.
Artificial Intelligence 129(1):5–33, 2001.
A foundational paper on planning as heuristic search.
Introduces max heuristic and additive heuristic.


Jörg Hoffmann and Bernhard Nebel.
The FF Planning System: Fast Plan Generation Through Heuristic Search.
JAIR 14:253–302, 2001.
Introduces FF and the FF heuristic.

References: Delete Relaxation (ctd.)


Emil Keyder and Héctor Geffner.
Heuristics for Planning with Action Costs Revisited.
Proc. ECAI 2008, pp. 588–592, 2008.
Introduces cost-sensitive FF/additive and set-additive heuristics.


Emil Keyder and Héctor Geffner.
Trees of Shortest Paths vs. Steiner Trees: Understanding and Improving Delete Relaxation Heuristics.
Proc. IJCAI 2009, pp. 1734–1739, 2009.
Introduces Steiner tree heuristic, improving the accuracy of inadmissible delete relaxation heuristics.

References: Delete Relaxation (ctd.)


Yaxin Liu, Sven Koenig and David Furcy.
Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning.
Proc. AAAI 2002, pp. 484–491, 2002.
Discussion of different methods for computing delete relaxation heuristics.

References: Landmarks


Jörg Hoffmann, Julie Porteous and Laura Sebastia.
Ordered Landmarks in Planning.
JAIR 22:215–278, 2004.
Journal version of the first landmarks paper.


Silvia Richter and Matthias Westphal.
The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks.
JAIR 2010, pp. 127–177, 2010.
Describes the landmark-counting heuristic and compares cost-sensitive vs. cost-insensitive variants of the FF heuristic.


Erez Karpas and Carmel Domshlak.
Cost-Optimal Planning with Landmarks.
Proc. IJCAI 2009, pp. 1728–1733, 2009.
Introduces admissible landmark heuristics based on cost partitioning.

References: Landmarks (ctd.)


Emil Keyder, Silvia Richter, and Malte Helmert.
Sound and Complete Landmarks for AND/OR Graphs.
Proc. ECAI 2010, pp. 335–340, 2010.
Complete landmark finding and landmarks beyond the delete relaxation.


Malte Helmert and Carmel Domshlak.
Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?
Proc. ICAPS 2009, pp. 162–169, 2009
Introduces the landmark cut heuristic.


Blai Bonet and Malte Helmert.
Strengthening Landmark Heuristics via Hitting Sets.
Proc. ECAI 2010, pp. 329–334, 2010.
Improving the accuracy of the landmark cut heuristic.

References: Critical Path

Patrik Haslum and Hector Geffner
Admissible Heuristics for Optimal Planning.
Proc. AIPS 2000, pp. 140–149, 2000.
Introduces critical path heuristics.

Patrik Haslum
$h^m(P) = h^1(P^m)$: Alternative Characterisations of the Generalisation From h^{max} to h^m.
Proc. ICAPS 2009, pp. 354–357, 2009.
Introduces P^m compilation.

References: Pattern Databases

Stefan Edelkamp.
Planning with Pattern Databases.
Proc. ECP 2001, pp. 13–24, 2001.
First paper on planning with pattern databases.

Stefan Edelkamp.
Symbolic Pattern Databases in Heuristic Search Planning.
Proc. AIPS 2002, pp. 274–283, 2002.
Uses BDDs to store pattern databases more compactly.

References: Pattern Databases (ctd.)

Patrik Haslum, Blai Bonet and Héctor Geffner.
New Admissible Heuristics for Domain-Independent Planning.
Proc. AAAI 2005, pp. 1163–1168, 2005.
Introduces constrained PDBs.
First pattern selection methods based on heuristic quality.

Stefan Edelkamp.
Automated Creation of Pattern Database Search Heuristics.
Proc. MoChArt 2006, pp. 121–135, 2007.
First search-based pattern selection method.

References: Pattern Databases (ctd.)

Patrik Haslum, Malte Helmert, Adi Botea, Blai Bonet and Sven Koenig.
Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning.
Proc. AAAI 2007, pp. 1007–1012, 2007.
Introduces canonical heuristic for pattern collections.
Search-based pattern selection based on Korf, Reid & Edelkamp's theory.

Marcel Ball and Robert C. Holte.
The Compression Power of Symbolic Pattern Databases.
Proc. ICAPS 2008, pp. 2–11, 2008.
Detailed empirical analysis showing benefits of symbolic BDDs over explicit-state BDDs.

References: Merge & Shrink


Klaus Dräger, Bernd Finkbeiner and Andreas Podelski.
Directed Model Checking with Distance-Preserving Abstractions.
Proc. SPIN 2006, pp. 19–34, 2006.
Introduces merge-and-shrink abstractions (for model-checking).


Malte Helmert, Patrik Haslum and Jörg Hoffmann.
Flexible Abstraction Heuristics for Optimal Sequential Planning.
Proc. ICAPS 2007, pp. 176–183, 2007.
Introduces merge-and-shrink abstractions for planning.

References: Merge & Shrink (ctd.)


Raz Nissim, Jörg Hoffmann, and Malte Helmert.
Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning.
Proc. IJCAI 2011, 2011.
New stragies for computing merge-and-shrink abstractions.

References: Structural Patterns


Michael Katz and Carmel Domshlak.
Structural Patterns Heuristics via Fork Decomposition.
Proc. ICAPS 2008, pp. 182–189, 2008.
Introduces structural-pattern abstractions, and studies fork-decomposition structural patterns.


Michael Katz and Carmel Domshlak.
Structural-Pattern Databases.
Proc. ICAPS 2009, pp. 186–193, 2009.
Introduces structural-pattern databases, making the idea of fork-decomposition practical.


Carmel Domshlak, Michael Katz and Sagi Lefler
When Abstractions Met Landmarks
Proc. ICAPS 2010, pp. 50–56, 2010.
Improving abstraction heuristics with landmark information.

References: Causal Graph Heuristics


Malte Helmert.
A Planning Heuristic Based on Causal Graph Analysis.
Proc. ICAPS 2004, pp. 161–170, 2004.
Introduces causal graph heuristic.


Malte Helmert and Héctor Geffner.
Unifying the Causal Graph and Additive Heuristics.
Proc. ICAPS 2008, pp. 140–147, 2008.
Introduces context-enhanced additive heuristic, combining the features of the causal graph and additive heuristics.


Dunbo Cai, Jörg Hoffmann and Malte Helmert
Enhancing the Context-Enhanced Additive Heuristic with Precedence Constraints
Proc. ICAPS 2009, 2009.
Explores precedence relations in the context enhanced additive heuristic.

References: Heuristic Performance


Malte Helmert and Gabriele Röger.
How Good is Almost Perfect?
Proc. AAAI 2008, pp. 944–949, 2008.
Shows that optimal planning is hard even with almost perfect heuristics differing from h^* by an additive constant.


Malte Helmert and Robert Mattmüller.
Accuracy of Admissible Heuristic Functions in Selected Planning Domains.
Proc. AAAI 2008, pp. 938–943, 2008.
Introduces asymptotic accuracy of admissible heuristics.

References: Search Enhancements


Jörg Hoffmann and Bernhard Nebel.
The FF Planning System: Fast Plan Generation Through Heuristic Search.
JAIR 14:253–302, 2001.
Introduces relaxed plans and helpful action pruning.


Vincent Vidal.
A Lookahead Strategy for Heuristic Search Planning.
In *Proc. ICAPS 2004*, pp. 150–159, 2004.
Takes helpful actions further with relaxed plan macros.


Malte Helmert.
The Fast Downward Planning System.
JAIR 26:191–246, 2006.
Introduces preferred operators, alternating multi-queue search, and delayed evaluation for systematic best-first search algorithms.

References: Search Enhancements (ctd.)


Silvia Richter and Malte Helmert.
Preferred Operators and Deferred Evaluation in Satisficing Planning.
Proc. ICAPS 2009, pp. 273–280, 2009.
Empirical evaluation of various strategies for preferred operators.


Gabriele Röger and Malte Helmert.
The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning.
Proc. ICAPS 2010, pp. 246–249, 2010.
Empirical evaluation of various strategies for combining several heuristics.

References: Heuristic Analysis & Comparison (ctd.)


Michael Katz and Carmel Domshlak.
Optimal Additive Composition of Abstraction-based Admissible Heuristics.
Proc. ICAPS 2008, pp. 174–181, 2008.
Introduces action cost partitioning, and proves tractability of optimal partitions for (numerous types of) abstractions.


Malte Helmert and Carmel Domshlak.
Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?
Proc. ICAPS 2009, pp. 162–169, 2009.
Uses cost partitioning to prove formal compilability results between various classes of heuristics.

References: Heuristic Analysis & Comparison


Jörg Hoffmann.
Where Ignoring Delete Lists Works, Part II: Causal Graphs
Proc. ICAPS 2011, 2011.
Automated analysis of delete-relaxation heuristics on different domains.