Classical Planning

Part 2: Heuristics

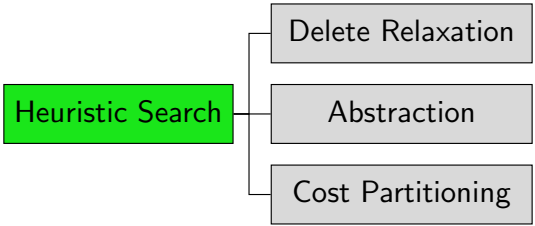
Gabriele Röger

University of Basel

ICAPS 26 Summer School
June 22, 2026

Thanks to Malte Helmert, David Speck, and Silvan Sievers for sharing their slides.

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning

Heuristic Search Planning

Informed forward state space search

- A heuristic estimates the cost from a given state to the closest goal.
- Standard best-first search such as A^* (Hart et al., 1968)
  or greedy best-first search (Pearl, 1984)
- Many variations:
  - helpful actions/preferred operators (Helmert, 2006; Hoffmann and Nebel, 2001)
  - diversification (Imai et al., 2011; Lipovetzky et al., 2017; Xie et al., 2014)
  - partial-order reduction (Haslum and Geffner, 2000; R. C. Holte et al., 2014; Wehrle, Helmert, Alkhazraji, et al., 2013)
  - symmetry-elimination (Domshlak, Katz, et al., 2012, 2015; Wehrle, Helmert, Shleyfman, et al., 2015)
  - exploiting distance-information (Thayer et al., 2011)
  - combination with local search (Hoffmann and Nebel, 2001)

Heuristics

- A heuristic estimates the cost from a given state to the closest goal.
  - heuristic $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$
  - perfect heuristic h^*: $h^*(s)$ cost of optimal solution from s (∞ if unsolvable)
  - properties of heuristics h:
    - safe: $(h(s) = \infty \Rightarrow h^*(s) = \infty)$ for all states s
    - goal-aware: $h(s) = 0$ for all goal states s
    - admissible: $h(s) \leq h^*(s)$ for all states s
    - consistent: $h(s) \leq cost(o) + h(s')$ for all transitions $s \xrightarrow{o} s'$

Coming Up with Heuristics in a Principled Way

General Procedure for Obtaining a Heuristic

- Simplify the problem, for example by removing problem constraints.
- Solve the simplified problem (ideally optimally).
- Use the solution cost for the simplified problem as a heuristic for the real problem.

As heuristic values are computed for every generated search state,
it is important that they can be computed efficiently.

Relaxing a Problem: Example

Example (Route Planning in a Road Network)

The road network is formalized as a weighted graph over points in the Euclidean plane. The weight of an edge is the road distance between two locations.

Example (Relaxation for Route Planning)

Use the Euclidean distance $\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$
as a heuristic for the road distance between $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$
This is a lower bound on the road distance ($\rightsquigarrow$ admissible).

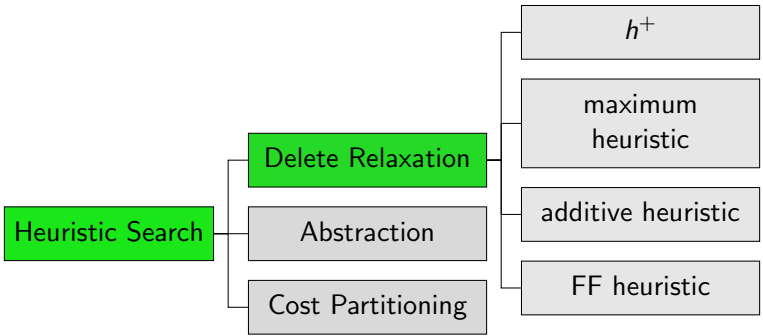
$\rightsquigarrow$ We drop the constraint of having to travel on roads.

Planning Heuristics: Main Concepts

Major ideas for heuristics in the planning literature:

- delete relaxation (Haslum and Geffner, 2000; Hoffmann and Nebel, 2001)
- abstraction (Edelkamp, 2001; Helmert, Haslum, Hoffmann, and Nissim, 2014; Seipp and Helmert, 2018; Sievers and Helmert, 2021)
- critical paths (Haslum, Bonet, et al., 2005)
- landmarks (Büchner et al., 2023; Hoffmann, Porteous, et al., 2004; Richter et al., 2010)
- network flows (Bonet, 2013; Bonet and van den Briel, 2014; Pommerening, Röger, et al., 2014)
- potential heuristics (Fišer et al., 2020; Pommerening, Helmert, et al., 2015; Seipp, Pommerening, et al., 2015)

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
h^+
maximum heuristic
additive heuristic
FF heuristic

Delete Relaxation: Idea

In STRIPS planning tasks, good and bad effects are easy to distinguish*:

- Operator preconditions and the goal are conjunctions of atoms.
- Add effects are good, delete effects are bad for achieving the goal.

Idea: discard all delete effects to simplify the problem.

(*) beyond STRIPS, we can efficiently transform a task into a positive normal form with this property.

Delete-Relaxed Planning Tasks

Definition (Delete Relaxation of STRIPS Tasks)

The delete relaxation Π^+ of a STRIPS planning task $\Pi = \langle V, I, O, \gamma \rangle$ is the task $\Pi^+ := \langle V, I, \{o^+ \mid o \in O\}, \gamma \rangle$, where $pre(o^+) = pre(o)$, $add(o^+) = add(o)$ and $del(o^+) = \emptyset$.

Definition (Delete Relaxation of Operator Sequences)

The delete relaxation of an operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$ is the operator sequence $\pi^+ := \langle o_1^+, \ldots, o_n^+ \rangle$.

Relaxation Lemma

Lemma (Relaxation)

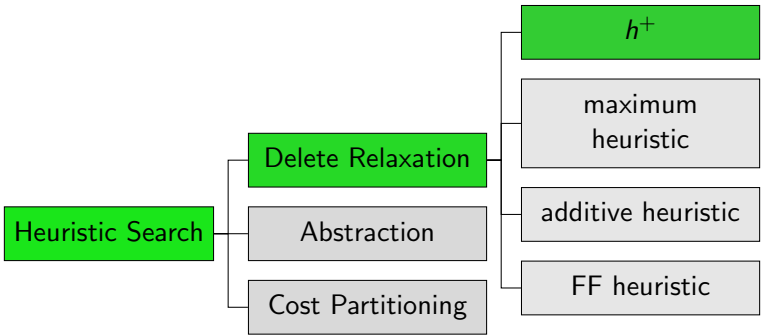
Let s and s' be states with $s \subseteq s'$.

1. *If π is an operator sequence applicable in s, then π^+ is applicable in s' and $s[\![\pi]\!] \subseteq s'[\![\pi^+]\!]$.*
2. *If additionally π leads to a goal state from state s, then π^+ leads to a goal state from state s'.*

Consequences:

- $\rightsquigarrow$ If π is a plan for Π, then π^+ is a plan for Π^+.
- $\rightsquigarrow$ Delete relaxation is no harder to solve than original task.
- $\rightsquigarrow$ Optimal relaxed plans are never more expensive than optimal plans for original tasks.

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
h^+
maximum heuristic
additive heuristic
FF heuristic

Optimal Delete Relaxation Heuristic

- The optimal delete relaxation heuristic h^+ uses the perfect goal distances in Π^+ as heuristic estimates.
- Optimal delete-free planning is still NP-complete.

Optimal Delete Relaxation Heuristic

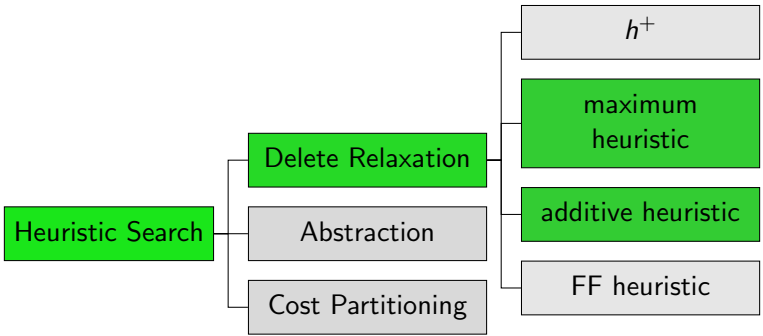
- The optimal delete relaxation heuristic h^+ uses the perfect goal distances in Π^+ as heuristic estimates.
- Optimal delete-free planning is still NP-complete.

How can we use relaxations for heuristic planning in practice?

Different possibilities:

- Do not actually solve the relaxed planning task, but compute an approximation of its solution cost.
  $\rightsquigarrow$ $h^{\max}$ heuristic, h^{add} heuristic
- Compute a solution for relaxed planning tasks which is not necessarily optimal, but "reasonable".
  $\rightsquigarrow$ h^{FF} heuristic

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
h^+
maximum heuristic
additive heuristic
FF heuristic

Relaxed Task Graph

Definition

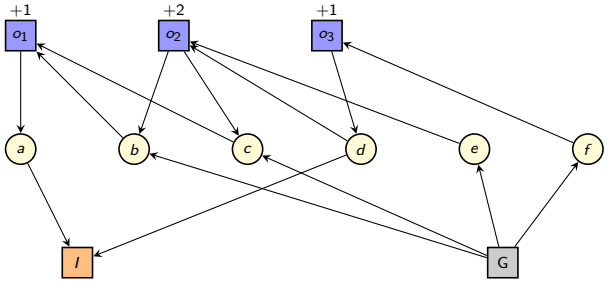
For a STRIPS planning task $\Pi = \langle V, I, O, G \rangle$ (in set representation), the relaxed task graph $RTG(\Pi^+)$ is the AND/OR graph $\langle N_{\text{and}} \cup N_{\text{or}}, A, type \rangle$ with

- $N_{\text{and}} = \{n_o \mid o \in O\} \cup \{v_I, v_G\}$
  with $type(n) = \wedge$ for all $n \in N_{\text{and}}$,
- $N_{\text{or}} = \{n_v \mid v \in V\}$
  with $type(n) = \vee$ for all $n \in N_{\text{or}}$, and
- $A = \{\langle n_a, n_o \rangle \mid o \in O, a \in add(o)\} \cup$
  $\{\langle n_o, n_p \rangle \mid o \in O, p \in pre(o)\} \cup$
  $\{\langle n_v, n_I \rangle \mid v \in I\} \cup$
  $\{\langle n_G, n_v \rangle \mid v \in G\}$

RTG: Example

$$V = \{a, b, c, d, e, f\}, \qquad I = \{a, d\}, \qquad G = \{b, c, e, f\}$$

$$pre(o_1) = \{a\} \qquad pre(o_2) = \{b, c\} \qquad pre(o_3) = \{d\}$$
$$add(o_1) = \{b, c\} \qquad add(o_2) = \{d, e\} \qquad add(o_3) = \{f\}$$
$$cost(o_1) = 1 \qquad cost(o_2) = 2 \qquad cost(o_3) = 1$$



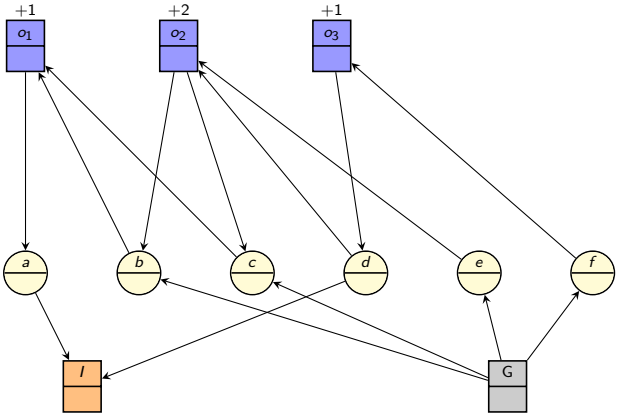
$h^{\max}$ Algorithm

Computing $h^{\max}$ Values

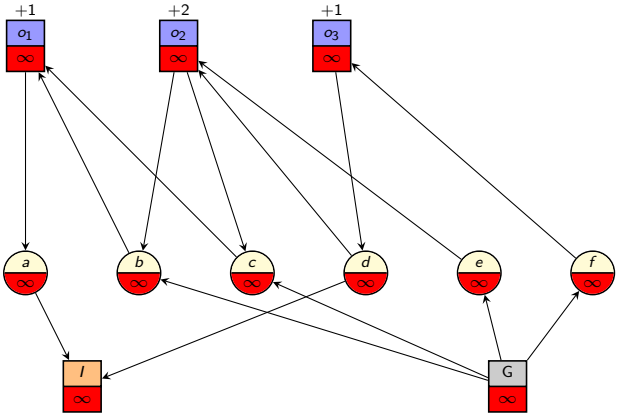
Associate a *cost* attribute with each node.
for all nodes n:
    $n.cost := \infty$
while no fixed point is reached:
    Choose a node n.
    if n is an AND node that is not an operator node:
        $n.cost := \max_{n' \in succ(n)} n'.cost$
    if n is a node for operator o:
        $n.cost := cost(o) + \max_{n' \in succ(n)} n'.cost$
    if n is an OR node:
        $n.cost := \min_{n' \in succ(n)} n'.cost$

The overall heuristic value is the cost of the goal node, $n_G.cost$.

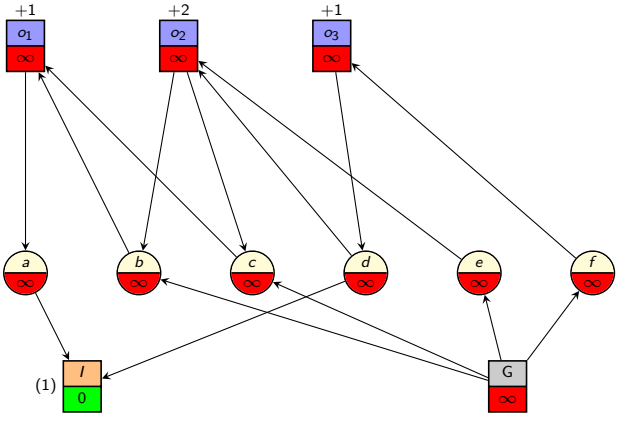
$h^{\max}$: Example of Efficient Computation



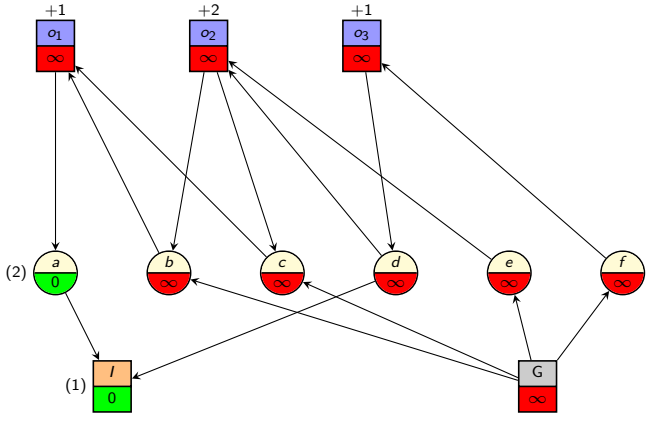
$h^{\max}$: Example of Efficient Computation



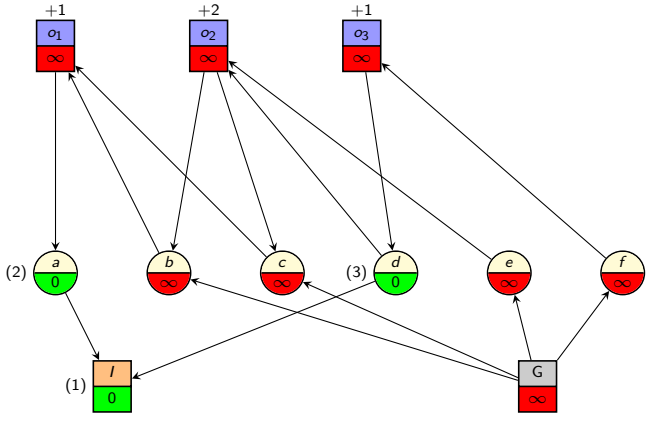
$h^{\max}$: Example of Efficient Computation



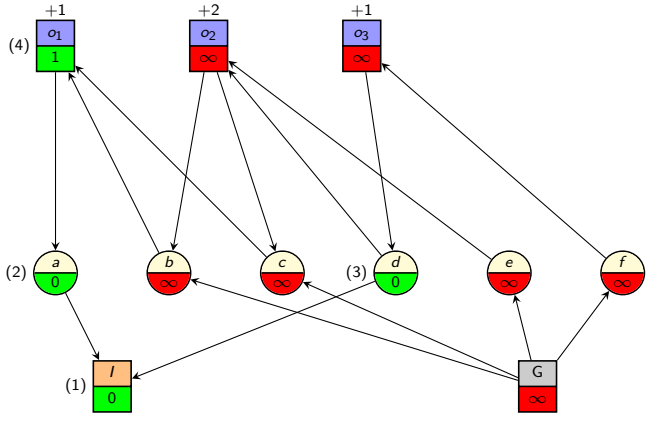
$h^{\max}$: Example of Efficient Computation



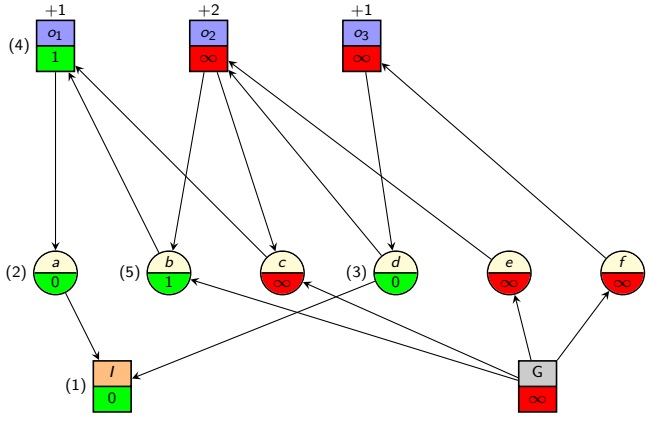
$h^{\max}$: Example of Efficient Computation



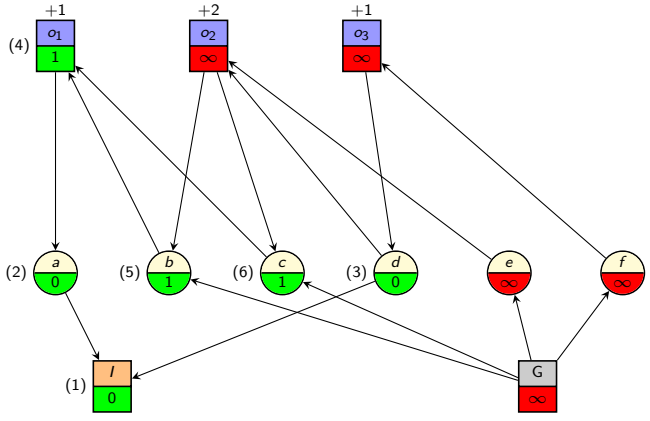
$h^{\max}$: Example of Efficient Computation



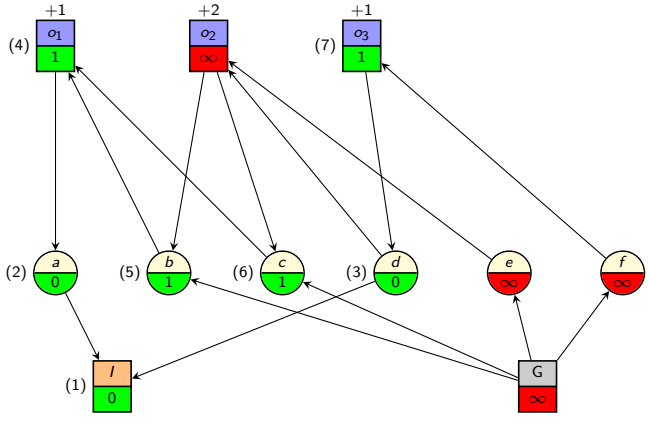
$h^{\max}$: Example of Efficient Computation



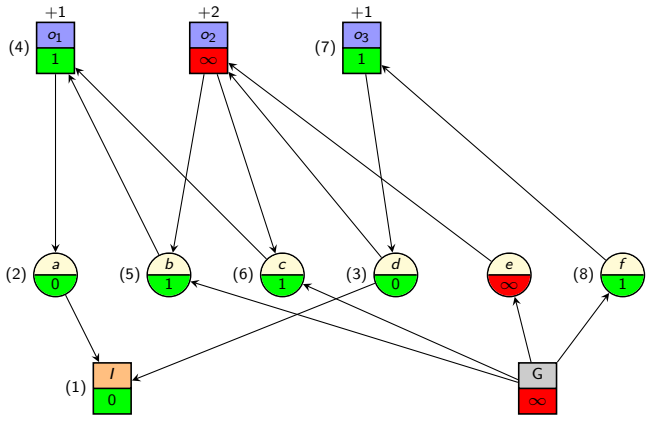
$h^{\max}$: Example of Efficient Computation



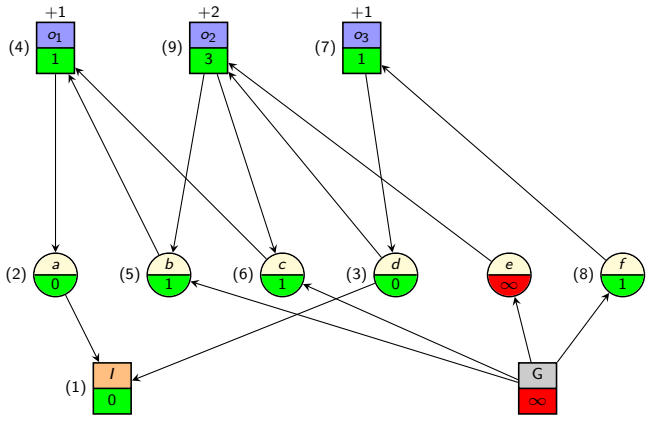
$h^{\max}$: Example of Efficient Computation



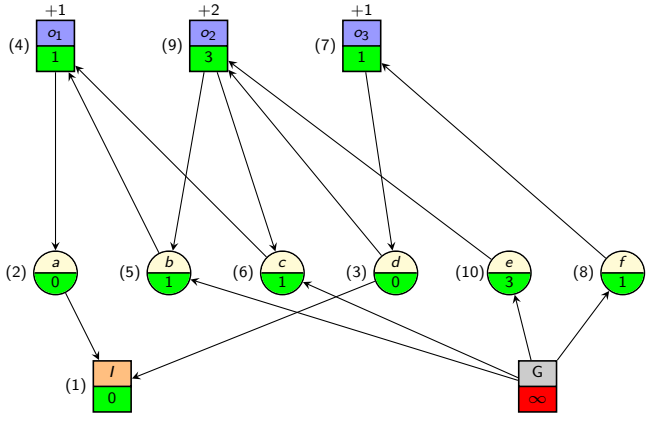
$h^{\max}$: Example of Efficient Computation



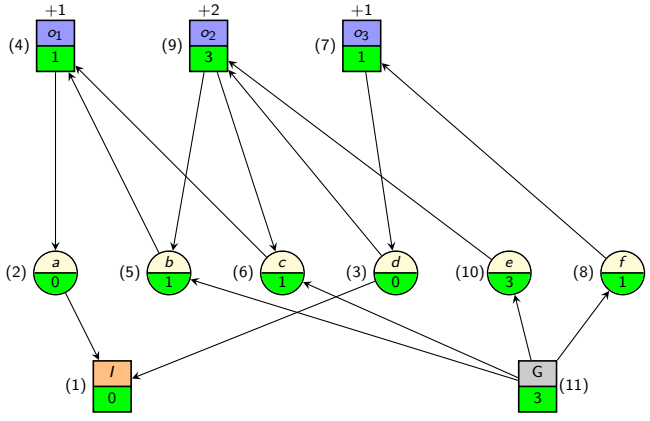
$h^{\max}$: Example of Efficient Computation



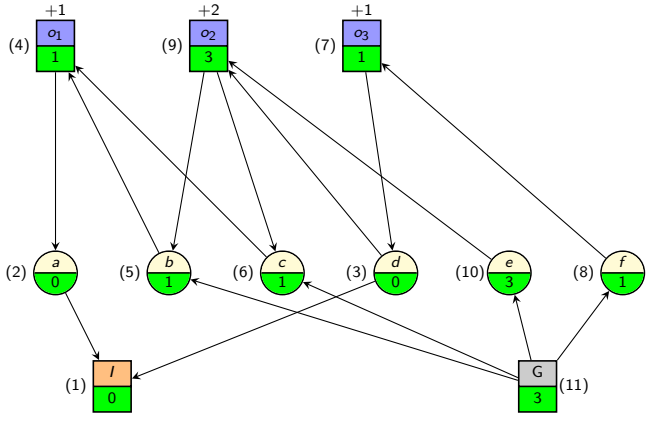
$h^{\max}$: Example of Efficient Computation



$h^{\max}$: Example of Efficient Computation



$h^{\max}$: Example of Efficient Computation



$\rightsquigarrow h^{\max}(I) = 3$

h^{add} Algorithm

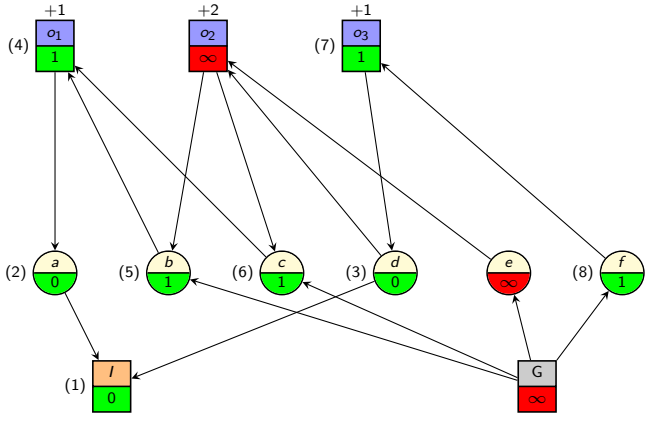
(Differences to $h^{\max}$ algorithm highlighted.)

Computing h^{add} Values

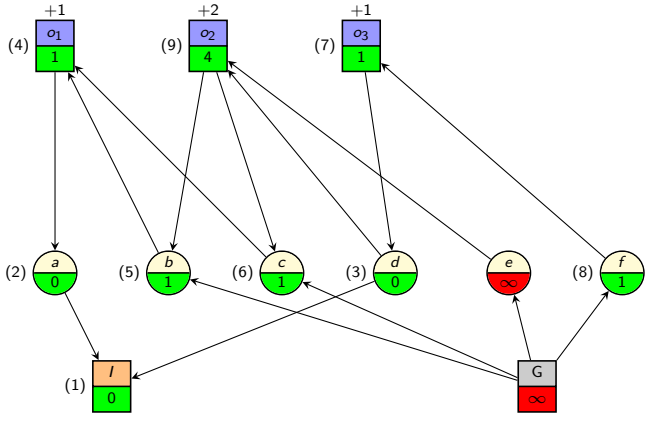
Associate a *cost* attribute with each node.
for all nodes n:
    $n.cost := \infty$
while no fixed point is reached:
    Choose a node n.
    if n is an AND node that is not an operator node:
        $n.cost := \sum_{n' \in succ(n)} n'.cost$
    if n is a node for operator o:
        $n.cost := cost(o) + \sum_{n' \in succ(n)} n'.cost$
    if n is an OR node:
        $n.cost := \min_{n' \in succ(n) n'.cost}$

The overall heuristic value is the cost of the goal node, $n_G.cost$.

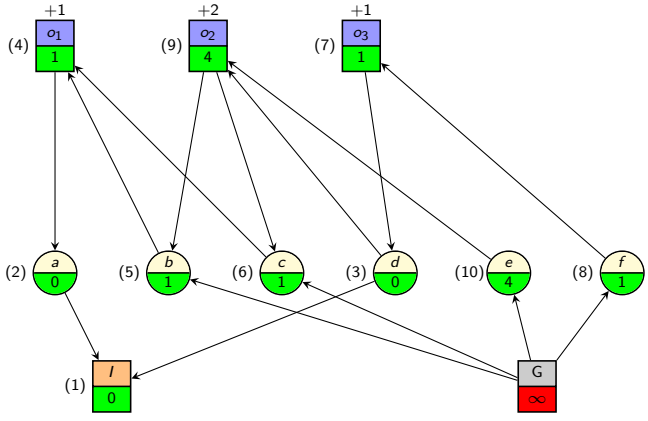
h^{add}: Example of Efficient Computation



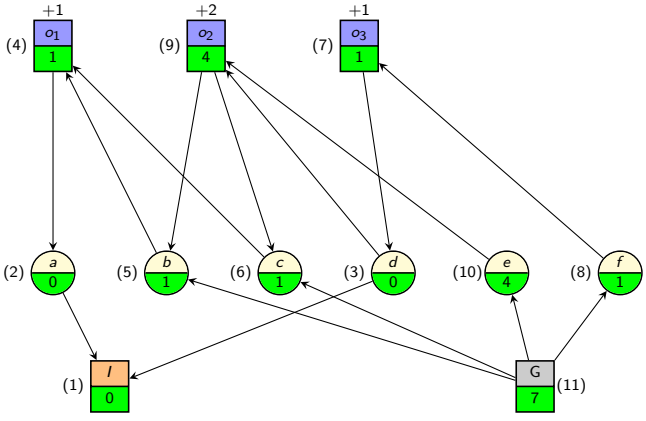
h^{add}: Example of Efficient Computation



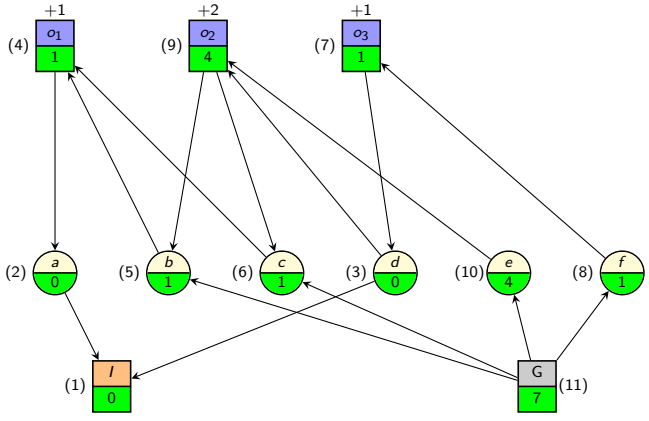
h^{add}: Example of Efficient Computation



h^{add}: Example of Efficient Computation

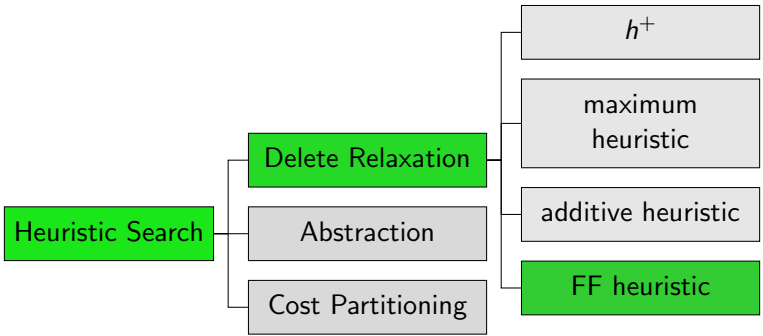


h^{add}: Example of Efficient Computation



$\rightsquigarrow h^{\text{add}}(I) = 7$

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
h^+
maximum heuristic
additive heuristic
FF heuristic

Best Achievers Induced by $h^{\max}$ and h^{add}

Which choices do $h^{\max}$ and h^{add} make?

- At every OR node n, we set the cost of n to the minimum of the costs of the successors of n.
- The motivation for this is to achieve n via the successor that can be achieved most cheaply according to our cost estimates.
- A best achiever of an OR node n is a successor with minimal associated cost among all successors of n (processed before n).

The FF Heuristic

The FF heuristics computes a plan by determining a set of operators that can be sequenced into a (non-optimal) relaxed plan.

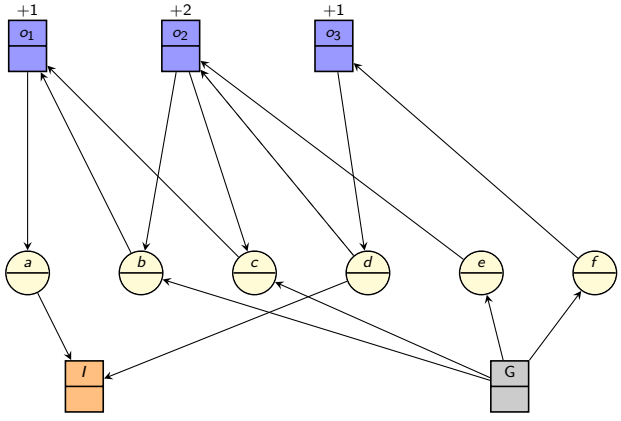
Definition (FF Heuristic)

The FF heuristic $h^{\mathrm{FF}}(s)$ for a state s is computed as follows:

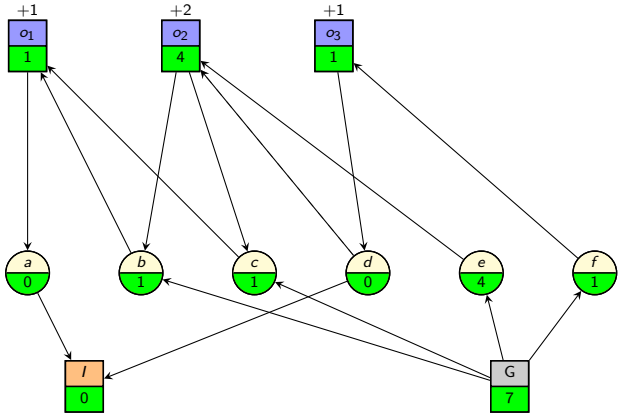
- Construct the relaxed task graph for the task (using s for the initial state).
- Compute h^{add} to determine best achievers.
- For each OR node, discard all outgoing edges except one to a best achiever.
- Compute the set of operator nodes $\{n_{o_1}, \ldots, n_{o_k}\}$ reachable from goal node n_G.
- Return $h^{\mathrm{FF}}(s) = \sum_{i=1}^{k} cost(o_i)$.

Note: h^{FF} is not well-defined; different tie-breaking policies for best achievers can lead to different heuristic values.

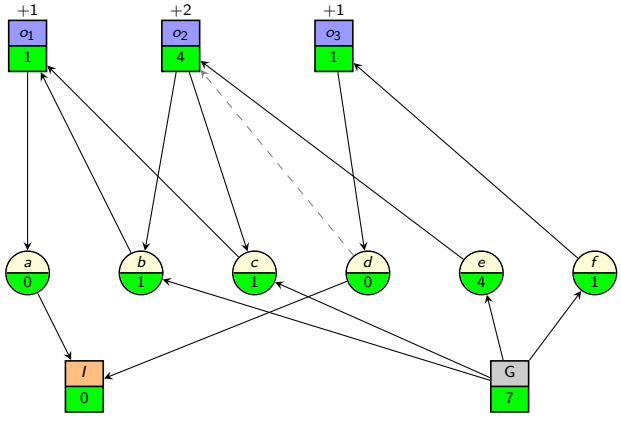
h^{FF}: Example



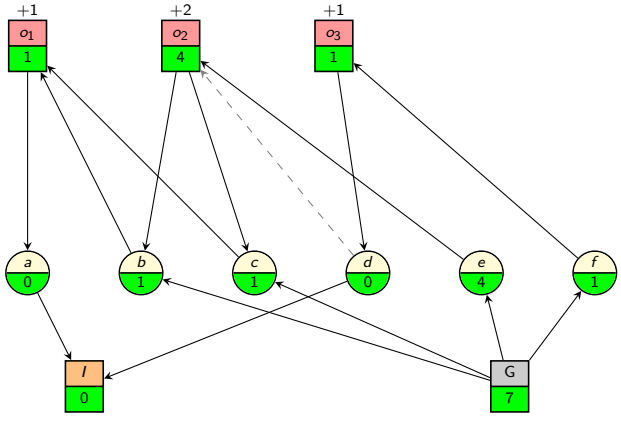
h^{FF}: Example



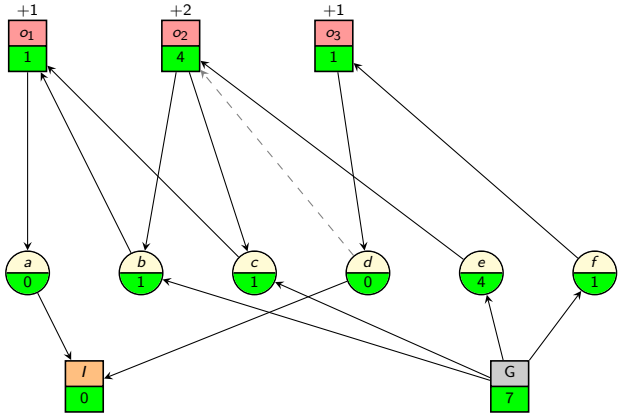
h^{FF}: Example



h^{FF}: Example



h^{FF}: Example



$\rightsquigarrow h^{\mathrm{FF}}(I) = 4$

Relationships between Delete Relaxation Heuristics (1)

Theorem

Let Π be a STRIPS planning task and let s be a state of Π.

Then:

1. $h^{max}(s) \leq h^{+}(s) \leq h^{FF}(s) \leq h^{add}(s)$
2. $h^{max}(s) = \infty$ *iff* $h^{+}(s) = \infty$ *iff* $h^{FF}(s) = \infty$ *iff* $h^{add}(s) = \infty$
3. h^{max} *and* h^{+} *are admissible and consistent.*
4. h^{FF} *and* h^{add} *are neither admissible nor consistent.*

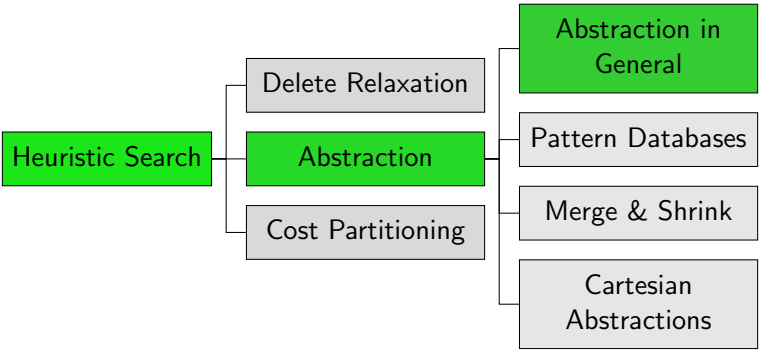
Literature Pointers

(Some) delete-relaxation heuristics in the planning literature:

- additive heuristic h^{add} (Bonet, Loerincs, et al., 1997)
- maximum heuristic $h^{\max}$ (Bonet and Geffner, 1999)
- (original) FF heuristic (Hoffmann and Nebel, 2001)
- cost-sharing heuristic h^{cs} (Mirkis et al., 2007)
- set-additive heuristics h^{sa} (Keyder and Geffner, 2008)
- FF/additive heuristic h^{FF} (Keyder and Geffner, 2008)
- local Steiner tree heuristic h^{lst} (Keyder and Geffner, 2008)

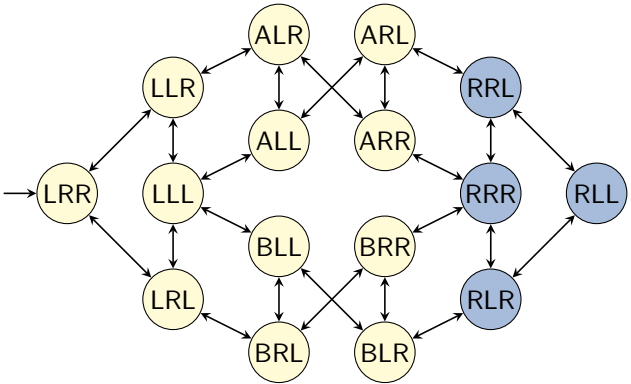
$\rightsquigarrow$ also hybrids such as semi-relaxed heuristics
(Domshlak, Hoffmann, et al., 2015; Keyder, Hoffmann, et al., 2014)
and delete-relaxation landmark heuristics
(Helmert and Domshlak, 2009; Keyder, Richter, et al., 2010)

Content

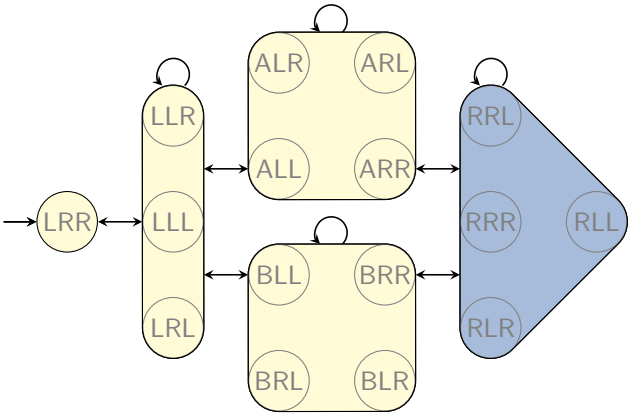

Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
Abstraction in General
Pattern Databases
Merge & Shrink
Cartesian Abstractions

Abstraction: Example

concrete transition system



Abstraction: Example



Note: Most arcs represent many parallel transitions.

Abstractions

Definition (Abstraction)

Let $\mathcal{T}$ be a transition system with state set S.

An abstraction of $\mathcal{T}$ is a function $\alpha : S \to S^\alpha$, where S^α is an arbitrary set.

Without loss of generality, we require that α is surjective.

Intuition: α maps the states of $\mathcal{T}$ to another (usually smaller) abstract state space.

Abstract Transition System

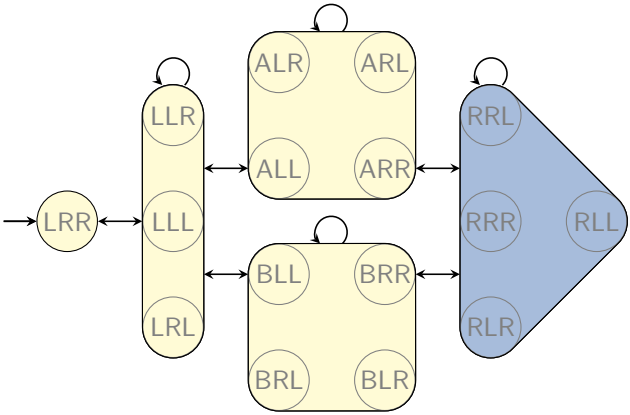
Definition (Abstract Transition System)

Let $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$ be a transition system,
and let $\alpha : S \to S^\alpha$ be an abstraction of $\mathcal{T}$.

The abstract transition system induced by α, in symbols $\mathcal{T}^\alpha$,
is the transition system $\mathcal{T}^\alpha = \langle S^\alpha, L, c, T^\alpha, s_0^\alpha, S_\star^\alpha \rangle$ defined by:

- $T^\alpha = \{\langle \alpha(s), \ell, \alpha(t) \rangle \mid \langle s, \ell, t \rangle \in T\}$
- $s_0^\alpha = \alpha(s_0)$
- $S_\star^\alpha = \{\alpha(s) \mid s \in S_\star\}$

Abstraction Heuristics: Example



$h^{\alpha}(\text{LRL}\}) = 2$

Abstraction Heuristics

Definition (Abstraction Heuristic)

Let $\alpha : S \to S^\alpha$ be an abstraction of a transition system $\mathcal{T}$.

The abstraction heuristic induced by α, written h^α,
is the heuristic function $h^\alpha : S \to \mathbb{R}_0^+ \cup \{\infty\}$ defined as

$$h^\alpha(s) = h^*_{\mathcal{T}^\alpha}(\alpha(s)) \quad \text{for all } s \in S,$$

where $h^*_{\mathcal{T}^\alpha}$ denotes the goal distance function in $\mathcal{T}^\alpha$.

Abstraction Heuristics

Definition (Abstraction Heuristic)

Let $\alpha : S \to S^\alpha$ be an abstraction of a transition system $\mathcal{T}$.

The abstraction heuristic induced by α, written h^α,
is the heuristic function $h^\alpha : S \to \mathbb{R}_0^+ \cup \{\infty\}$ defined as

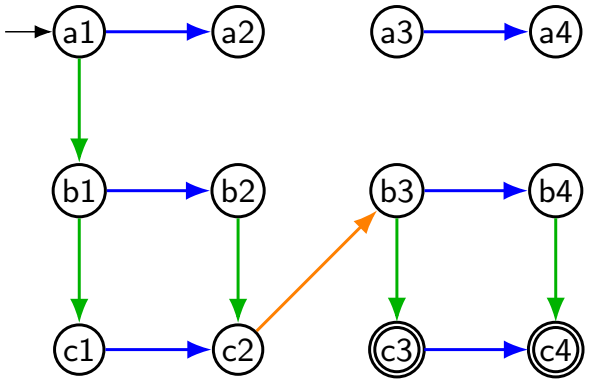
$$h^\alpha(s) = h^*_{\mathcal{T}^\alpha}(\alpha(s)) \quad \text{for all } s \in S,$$

where $h^*_{\mathcal{T}^\alpha}$ denotes the goal distance function in $\mathcal{T}^\alpha$.

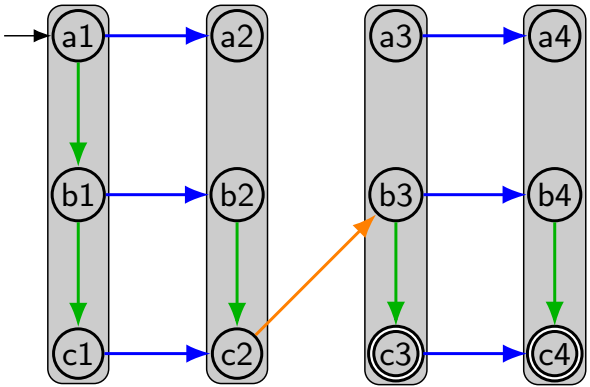
- Precomputation phase with explicit construction of abstract transition system and computation of abstract goal distances.
- For the heuristic evaluation during search,
  only store the abstraction function and the abstract goal distances.

Abstraction heuristics are consistent and admissible.

Four Classes of Abstractions: Concrete Transition System

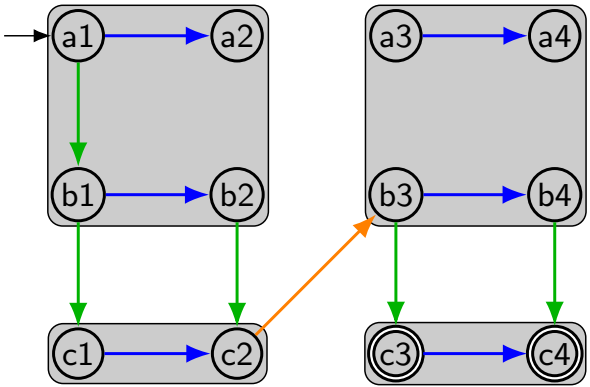


Four Classes of Abstractions: Projection/Pattern Database



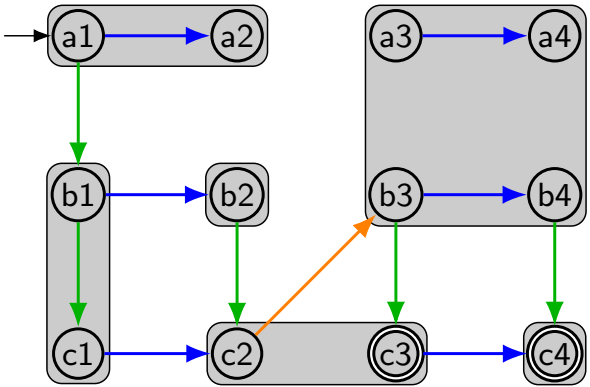
Culberson et al., 1996; Edelkamp, 2001

Four Classes of Abstractions: Domain Abstraction



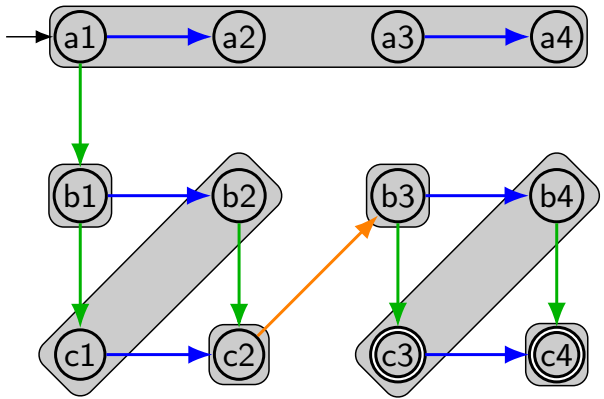
Hernádvölgyi et al., 2000

Four Classes of Abstractions: Cartesian Abstraction



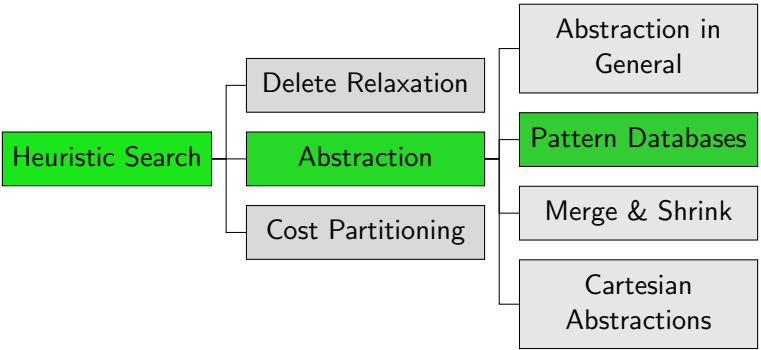
Seipp and Helmert, 2018

Four Classes of Abstractions: Merge-and-Shrink Abstraction



Dräger et al., 2009; Helmert, Haslum, Hoffmann, and Nissim, 2014; Sievers and Helmert, 2021

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
Abstraction in General
Pattern Databases
Merge & Shrink
Cartesian Abstractions

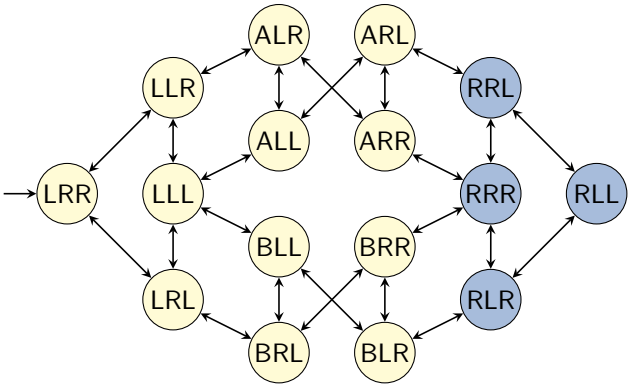
Pattern Database Heuristics

- The oldest commonly used abstraction heuristics in search and planning are pattern database (PDB) heuristics.
- PDB heuristics were originally introduced for the 15-puzzle (Culberson et al., 1996) and for Rubik's cube (Korf, 1997).
- The first use for domain-independent planning is due to Edelkamp, 2001.

Projections

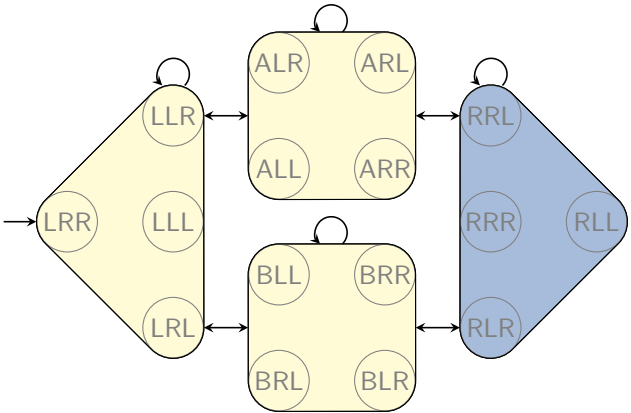
- Pattern database heuristics are abstraction heuristics induced by a particular class of abstractions called projections.
- For a pattern $P \subseteq V$, the projection π_P is defined as $\pi_P(s) := s|_P$.
- This simply discards all variables that are not in the pattern.
- π_P maps two states s_1 and s_2 to the same abstract state iff they agree on all variables in P.
- We can simply discard all variables that are not in P from the task. The transition system of the resulting task is isomorphic to the induced abstract transition system.

Example: Transition System



Example: Projection (1)

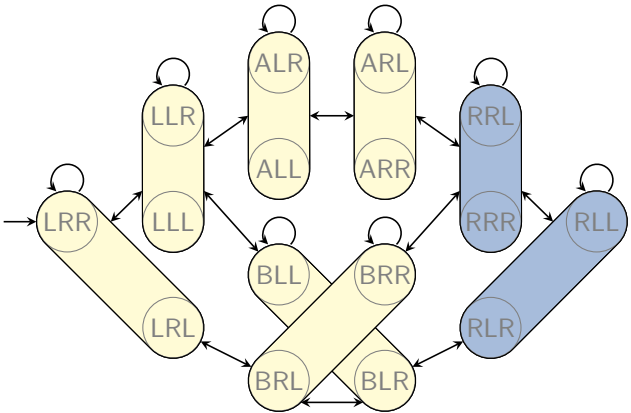
Abstraction induced by projection on 1st variable:



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

Example: Projection (2)

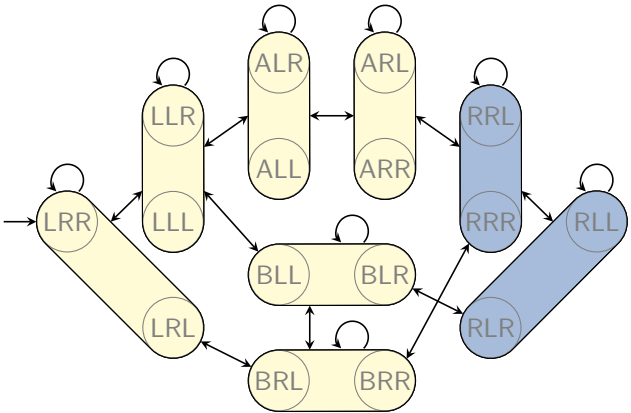
Abstraction induced by projection on 1st and 2nd variable:



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

Example: Projection (2)

Abstraction induced by projection on 1st and 2nd variable:

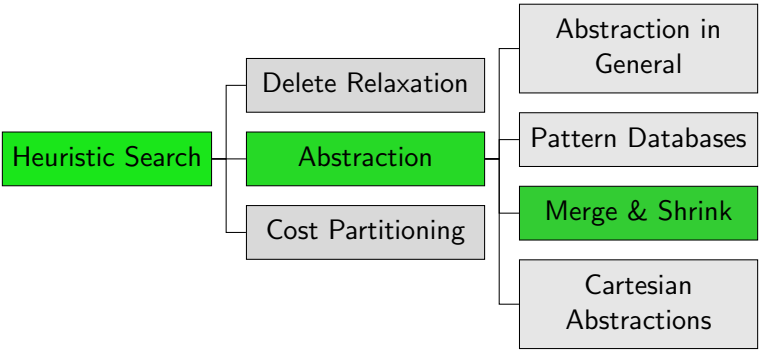


$h(\text{LRR}) = 2$

Major Topics on Pattern Databases

- Additive PDBs (with admissible sums of estimates)
  (Haslum, Botea, et al., 2007; Korf and Felner, 2002)
- Finding good patterns/pattern collections
  (Edelkamp, 2006; Franco et al., 2017; Haslum, Botea, et al., 2007; Rovner et al., 2019)
- Strengthening PDBs (Haslum, Bonet, et al., 2005)
- Compact representation of the pattern database
  (Breyer et al., 2010; Edelkamp, 2002; Felner et al., 2007; Schmidt et al., 2011)

Content

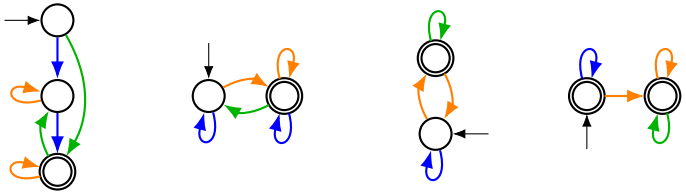

Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
Abstraction in General
Pattern Databases
Merge & Shrink
Cartesian Abstractions

Merge-and-shrink: Idea

Factored transition system: set of small transitions systems ("factors")
representing a large product transition system

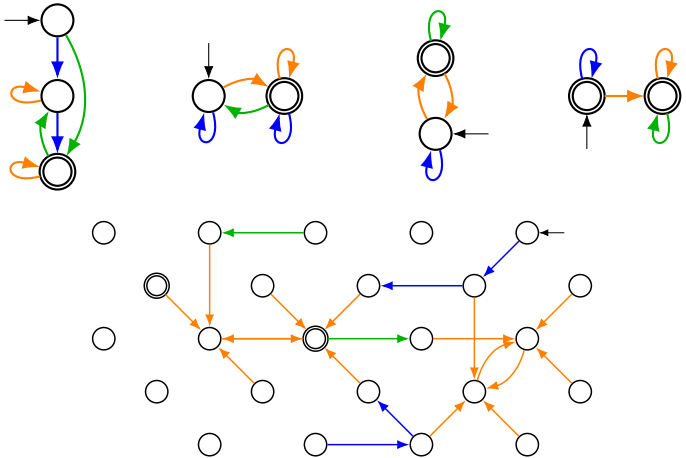
Merge-and-shrink: Idea

Factored transition system: set of small transitions systems ("factors")
representing a large product transition system



Merge-and-shrink: Idea

Factored transition system: set of small transitions systems ("factors")
representing a large product transition system

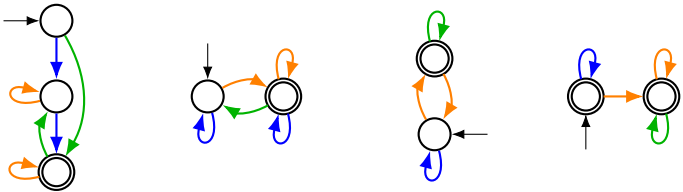


Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product

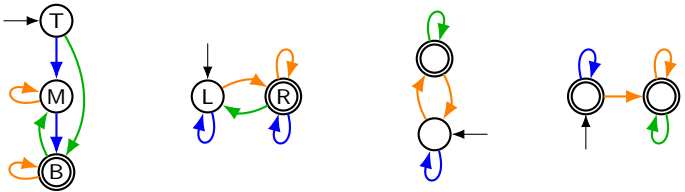
Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product



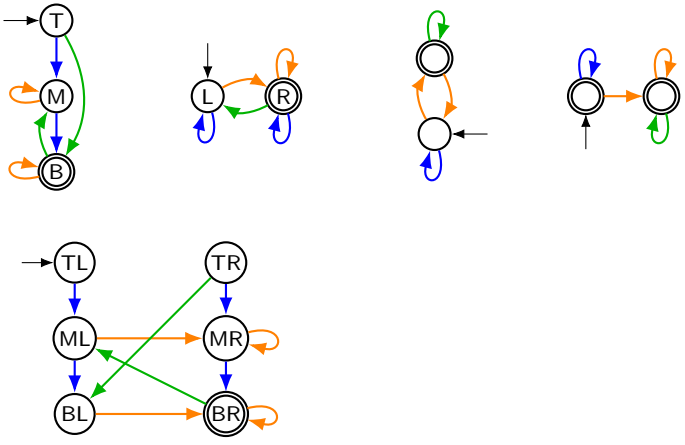
Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product



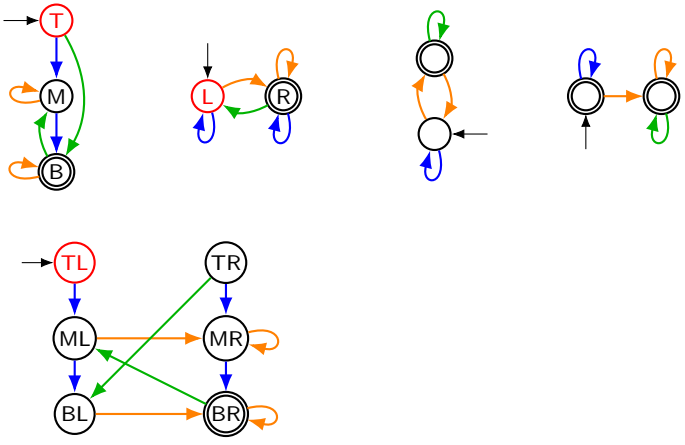
Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product



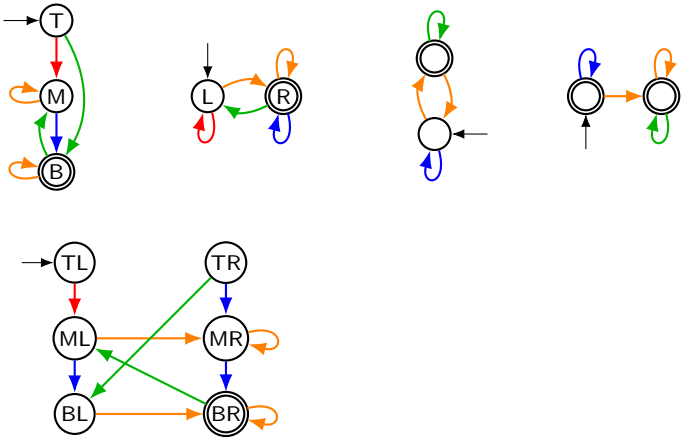
Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product



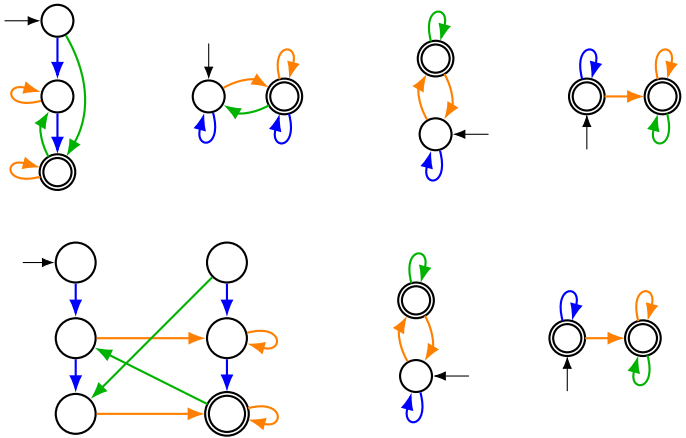
Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product



Merge-and-shrink Transformations: Merging

Replace two factors by their synchronized product

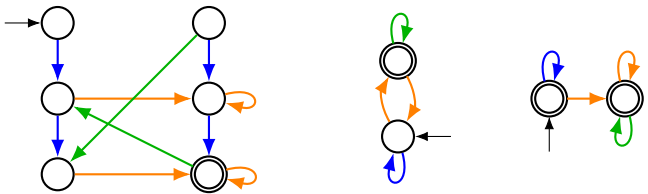


Merge-and-shrink Transformations: Shrinking

Apply abstraction to some factor

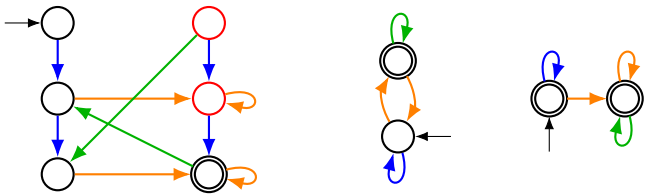
Merge-and-shrink Transformations: Shrinking

Apply abstraction to some factor



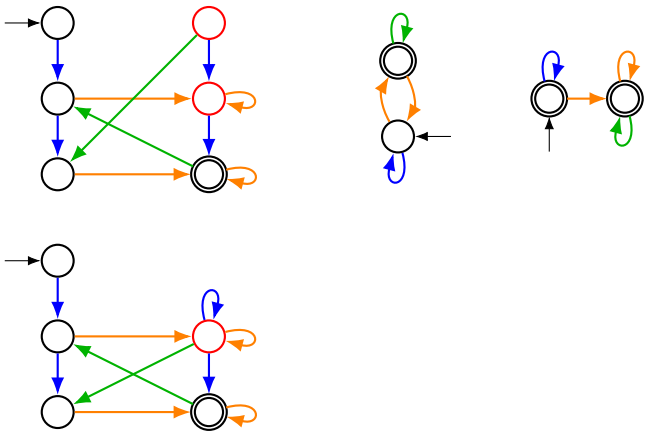
Merge-and-shrink Transformations: Shrinking

Apply abstraction to some factor



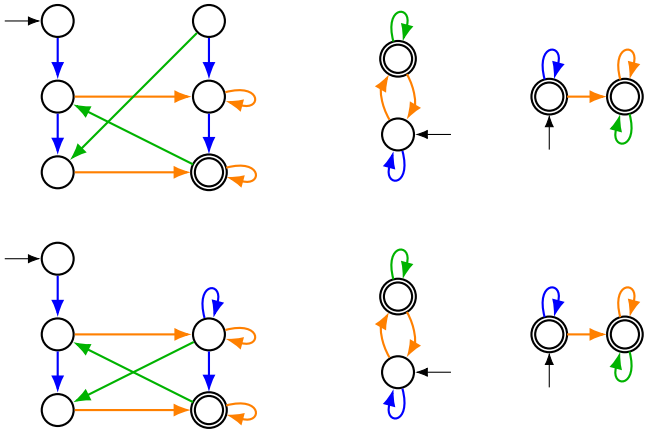
Merge-and-shrink Transformations: Shrinking

Apply abstraction to some factor



Merge-and-shrink Transformations: Shrinking

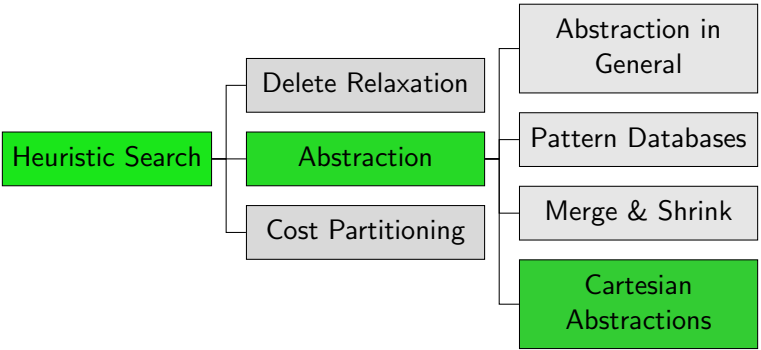
Apply abstraction to some factor



Literature on Merge and Shrink

- Original framework from model checking (Dräger et al., 2006)
- Transferred to planning (Helmert, Haslum, and Hoffmann, 2007; Helmert, Haslum, Hoffmann, and Nissim, 2014).
- Strategies for merging and shrinking (Fan, R. Holte, et al., 2018; Fan, Müller, et al., 2014; Helmert, Haslum, Hoffmann, and Nissim, 2014; Nissim et al., 2011; Sievers, Keller, et al., 2024; Sievers, Wehrle, et al., 2016)
- Label reduction (Sievers, Wehrle, et al., 2014)
- M&S Theory (Helmert, Röger, et al., 2015; Sievers and Helmert, 2021)
- Combination with saturated cost partitioning (Sievers, Pommerening, et al., 2020)

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning
Abstraction in General
Pattern Databases
Merge & Shrink
Cartesian Abstractions

Counterexample-Guided Abstraction Refinement

Counterexample-guided abstraction refinement (CEGAR) is an approach to compute a tailored abstraction for a task (or to solve it).

- Start with a very coarse abstraction.
- Iteratively compute an (optimal) abstract solution and check whether it works for the concrete tasks.
  - If yes, the task is solved.
  - If not, refine the abstraction so that the same flaw will not be encountered in future iterations.

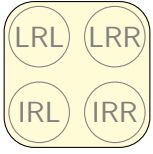
CEGAR is a technique originally introduced for model checking (Clarke et al., 2000), transferred to planning by Seipp and Helmert, 2018.

Cartesian Sets and Abstractions

Definition

A set of states for a planning task with variables $\langle v_1, \ldots, v_n \rangle$ is called Cartesian if it is of the form $A_1 \times \cdots \times A_n$, where $A_i \subseteq \mathrm{dom}(v_i)$ for all $1 \leq i \leq n$.

$$\{L, I\} \times \{R\} \times \{L, R\} = \{(L, R, L), (L, R, R), (I, R, L), (I, R, R)\}$$



Definition

An abstraction α is Cartesian if all abstract states correspond to Cartesian sets of concrete states.

CEGAR Algorithm

Generic CEGAR algorithm for planning task Π

$\mathcal{T} :=$ TrivialAbstractTransitionSystem(Π) $\leftarrow$ one abstract state
while not TerminationCondition(): $\leftarrow$ e.g. time/memory limit
    $\tau :=$ FindOptimalTrace($\mathcal{T}$) $\leftarrow$ abstract solution (path in $\mathcal{T}$)
    if τ is "no trace" then **return** Π unsolvable
    $F :=$ FindFlaw($\tau, \Pi, \mathcal{T}$) $\leftarrow$ reason why τ does not work in the non-abstract task
    if F is "no flaw" then
        return label sequence of τ as plan for Π
    $\mathcal{T} :=$ Refine($\mathcal{T}, F$)
return $\mathcal{T}$

Flaws

A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Flaws

A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Start from the initial state of Π and iteratively apply the next operator (label) o from τ.

Flaws

A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Start from the initial state of Π and iteratively apply the next operator (label) o from τ.

- Precondition flaw: o is not applicable in the current state s.

Flaws

A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Start from the initial state of Π and iteratively apply the next operator (label) o from τ.

- Precondition flaw: o is not applicable in the current state s.
- Goal flaw: the final state is not a goal state.

Flaws

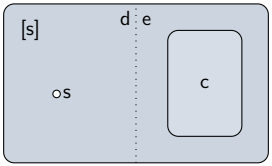
A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Start from the initial state of Π and iteratively apply the next operator (label) o from τ.

- Precondition flaw: o is not applicable in the current state s.
- Goal flaw: the final state is not a goal state.
- Deviation flaw: the next abstract transition is $a \xrightarrow{o} a'$, the current concrete state is s with $\alpha(s) = a$ but for successor state $s' = s[\![o]\!]$ we have $\alpha(s') \neq a'$ (deviating from the abstract path).

Also flaws beyond the first issue (Pozo et al., 2024a) and regression flaws (Pozo et al., 2024b)

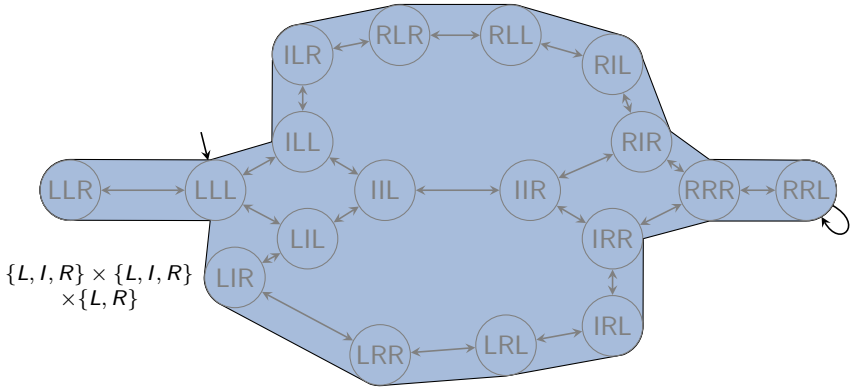
Extracting Flaws for Split



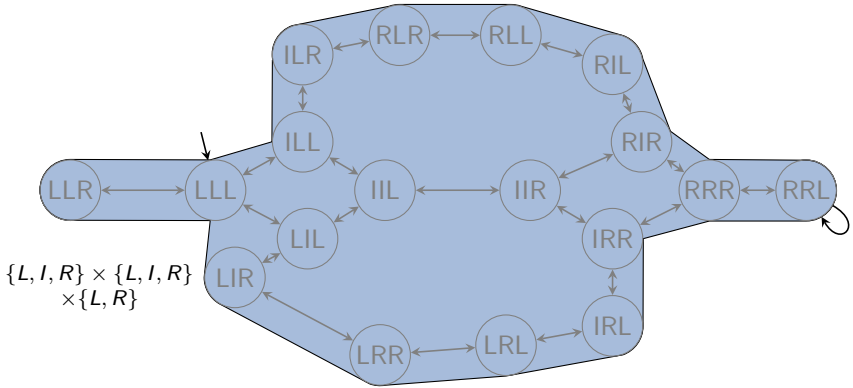
- s is a concrete state, reached by the simulation of the abstract plan on the concrete problem.
- $c \subseteq [s]$ is a non-empty Cartesian set,
- the abstract plan relied on "being in c" but $s \notin c$.
  - Precondition flaw: set of concrete states in $[s]$ in which o is applicable
  - Goal flaw: set of concrete goal states in $[s]$
  - Deviation flaw when applying o in s: set of concrete states in $[s]$ from which o leads into the intended abstract state (c can be determined with regression).

$[s]$ gets split into abstract states d and e.

Example: Two Packages, One Truck

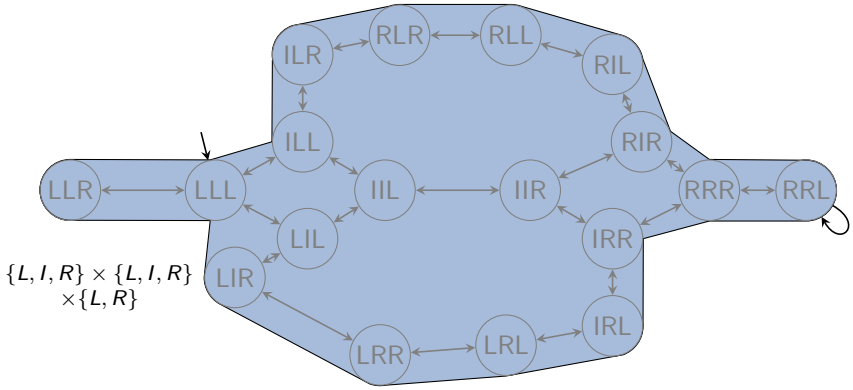


Example: Two Packages, One Truck



Abstract plan $\langle\rangle$ ends in state LLL, which is not a goal.

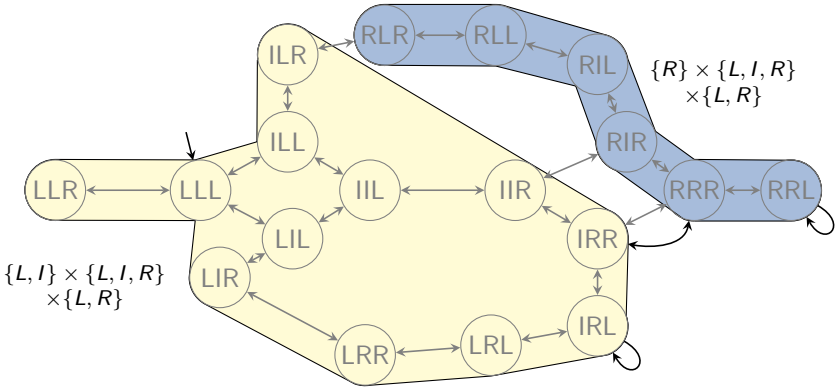
Example: Two Packages, One Truck



Abstract plan $\langle\rangle$ ends in state LLL, which is not a goal.
Refine $\{L, I, R\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{R\} \times \{R\} \times \{L, R\})$.

$\rightsquigarrow$ split on first or second variable;
$\rightsquigarrow \{L, I\} \times \{L, I, R\} \times \{L, R\}$ and $\{R\} \times \{L, I, R\} \times \{L, R\}$

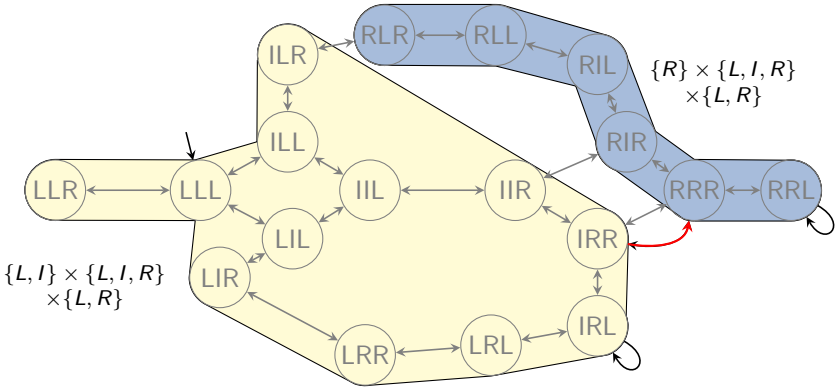
Example: Two Packages, One Truck



Abstract plan $\langle\rangle$ ends in state LLL, which is not a goal.
Refine $\{L, I, R\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{R\} \times \{R\} \times \{L, R\})$.

$\rightsquigarrow$ split on first or second variable;
$\rightsquigarrow \{L, I\} \times \{L, I, R\} \times \{L, R\}$ and $\{R\} \times \{L, I, R\} \times \{L, R\}$

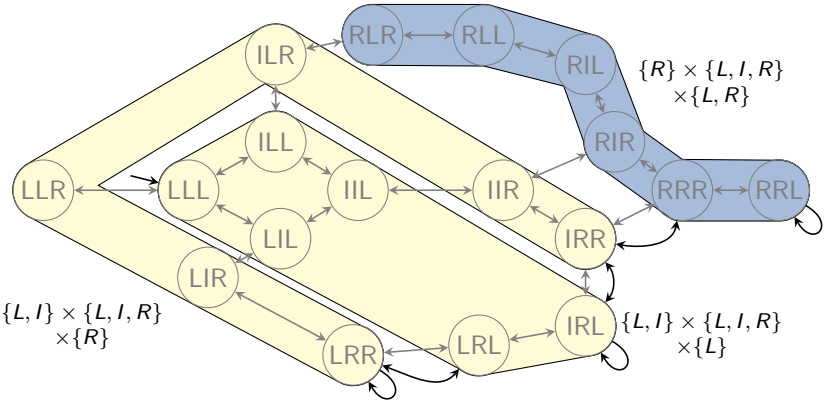
Example: Two Packages, One Truck



Abstract plan $\langle \text{drop}_{A,R} \rangle$; first action inapplicable in LLL.
Refine $\{L, I\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first or third variable;
$\rightsquigarrow \{L, I\} \times \{L, I, R\} \times \{L\}$ and $\{L, I\} \times \{L, I, R\} \times \{R\}$

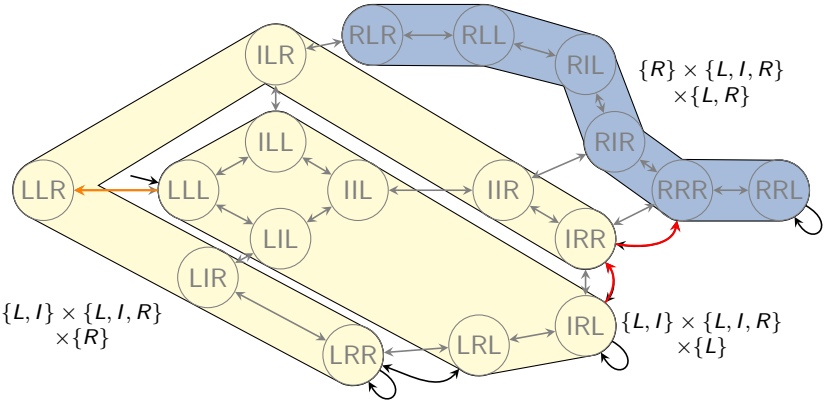
Example: Two Packages, One Truck



Abstract plan $\langle \text{drop}_{A,R} \rangle$; first action inapplicable in LLL.
Refine $\{L, I\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first or third variable;
$\rightsquigarrow \{L, I\} \times \{L, I, R\} \times \{L\}$ and $\{L, I\} \times \{L, I, R\} \times \{R\}$

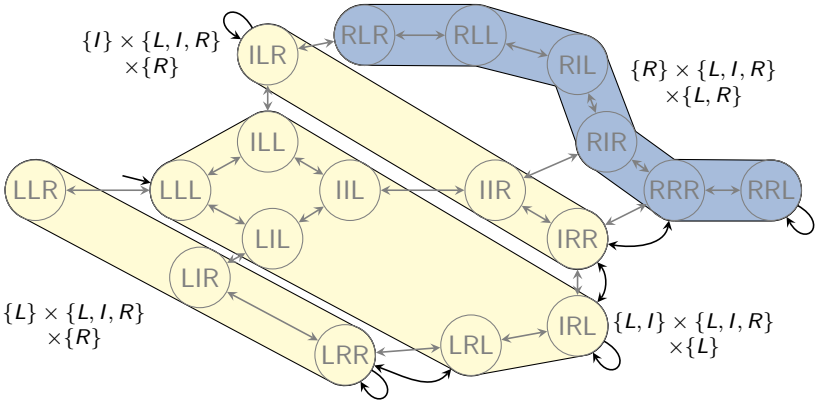
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; second action inapplicable in LLR.
Refine $\{L, I\} \times \{L, I, R\} \times \{R\}$ with split $(LLR, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow \{L\} \times \{L, I, R\} \times \{R\}$ and $\{I\} \times \{L, I, R\} \times \{R\}$

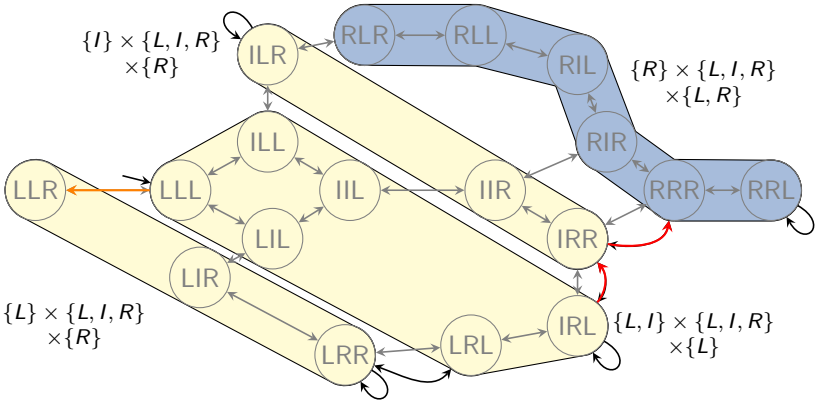
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; second action inapplicable in LLR.
Refine $\{L, I\} \times \{L, I, R\} \times \{R\}$ with split $(LLR, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow \{L\} \times \{L, I, R\} \times \{R\}$ and $\{I\} \times \{L, I, R\} \times \{R\}$

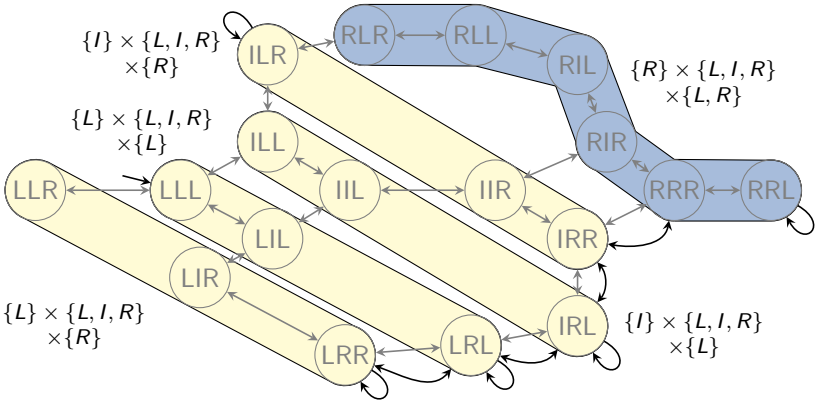
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; deviation flaw at first transition.
Refine $\{L, I\} \times \{L, I, R\} \times \{L\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{L\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow \{L\} \times \{L, I, R\} \times \{L\}$ and $\{I\} \times \{L, I, R\} \times \{L\}$

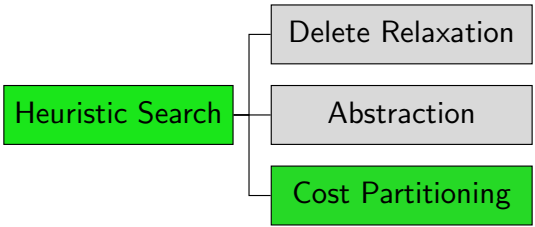
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; deviation flaw at first transition.
Refine $\{L, I\} \times \{L, I, R\} \times \{L\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{L\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow \{L\} \times \{L, I, R\} \times \{L\}$ and $\{I\} \times \{L, I, R\} \times \{L\}$

Content


Heuristic Search
Delete Relaxation
Abstraction
Cost Partitioning

Cost Partitioning

Idea: Cost Partitioning

- create copies $\Pi_1, \ldots, \Pi_n$ of planning task Π
- each has its own operator cost function $cost_i : O \to \mathbb{R}_0^+$ (otherwise identical to Π)
- for all o: require $cost_1(o) + \cdots + cost_n(o) \leq cost(o)$

$\rightsquigarrow$ sum of solution costs in copies is an admissible heuristic:
$h^*_{\Pi_1} + \cdots + h^*_{\Pi_n} \leq h^*_{\Pi}$

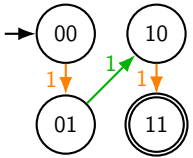
(Katz et al., 2008, 2010; Yang et al., 2008)

Cost Partitioning

- for admissible heuristics $h_1, \ldots, h_n$,
  $h(s) = h_{1,\Pi_1}(s) + \cdots + h_{n,\Pi_n}(s)$
  is an admissible estimate
- $h(s)$ can be better or worse than any $h_{i,\Pi}(s)$
  $\rightarrow$ depending on cost partitioning

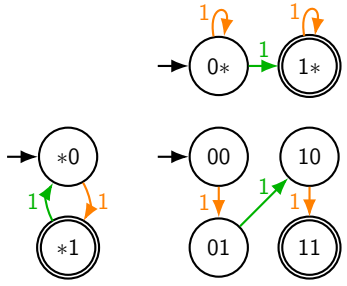
Cost Partitioning

- for admissible heuristics $h_1, \ldots, h_n$,
  $h(s) = h_{1,\Pi_1}(s) + \cdots + h_{n,\Pi_n}(s)$
  is an admissible estimate
- $h(s)$ can be better or worse than any $h_{i,\Pi}(s)$
  $\rightarrow$ depending on cost partitioning



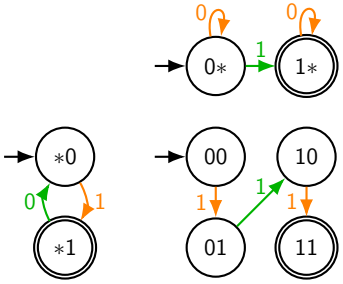
Cost Partitioning

- for admissible heuristics $h_1, \ldots, h_n$,
  $h(s) = h_{1,\Pi_1}(s) + \cdots + h_{n,\Pi_n}(s)$
  is an admissible estimate
- $h(s)$ can be better or worse than any $h_{i,\Pi}(s)$
  $\rightarrow$ depending on cost partitioning



Cost Partitioning

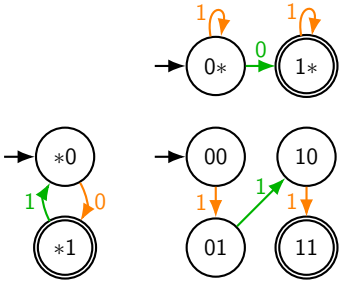
- for admissible heuristics $h_1, \ldots, h_n$,
  $h(s) = h_{1,\Pi_1}(s) + \cdots + h_{n,\Pi_n}(s)$
  is an admissible estimate
- $h(s)$ can be better or worse than any $h_{i,\Pi}(s)$
  $\rightarrow$ depending on cost partitioning



Heuristic value:
$1 + 1 = 2$

Cost Partitioning

- for admissible heuristics $h_1, \ldots, h_n$,
  $h(s) = h_{1,\Pi_1}(s) + \cdots + h_{n,\Pi_n}(s)$
  is an admissible estimate
- $h(s)$ can be better or worse than any $h_{i,\Pi}(s)$
  $\rightarrow$ depending on cost partitioning



Heuristic value:
$0 + 0 = 0$

Strategies for Defining Cost Partitioning

Some strategies for defining cost-functions:

- zero-one: full operator cost in one copy, zero in all others (Edelkamp, 2006)

Strategies for Defining Cost Partitioning

Some strategies for defining cost-functions:

- zero-one: full operator cost in one copy, zero in all others (Edelkamp, 2006)
- uniform: $cost_i(o) = cost(o)/n$

Strategies for Defining Cost Partitioning

Some strategies for defining cost-functions:

- zero-one: full operator cost in one copy, zero in all others (Edelkamp, 2006)
- uniform: $cost_i(o) = cost(o)/n$
- optimal: using linear programming (Katz et al., 2008)

Strategies for Defining Cost Partitioning

Some strategies for defining cost-functions:

- zero-one: full operator cost in one copy, zero in all others (Edelkamp, 2006)
- uniform: $cost_i(o) = cost(o)/n$
- optimal: using linear programming (Katz et al., 2008)
- saturated: consider heuristics one by one;
  later heuristics may use "left-over" operator costs (Seipp, Keller, et al., 2020)

Optimal Cost Partitioning

Optimal Cost Partitioning with Linear Programming

- Use variables for cost of each operator in each task copy
- Express heuristic values with linear constraints
- Maximize sum of heuristic values subject to these constraints

LPs known for

- abstraction heuristics (Katz et al., 2010)
- landmark heuristic (Karpas et al., 2009)

Saturated Cost Function

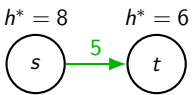
A cost function c is saturated for a heuristic if

- $c(o) \leq cost(o)$ for all operators o and
- replacing the original operator cost function with c does not change any of the heuristic estimates.

Saturated Cost Function

A cost function c is saturated for a heuristic if

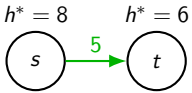
- $c(o) \leq cost(o)$ for all operators o and
- replacing the original operator cost function with c does not change any of the heuristic estimates.



Saturated Cost Function

A cost function c is saturated for a heuristic if

- $c(o) \leq cost(o)$ for all operators o and
- replacing the original operator cost function with c does not change any of the heuristic estimates.



Cost should not become smaller than $h^*(s) - h^*(t)$.

Algorithm

Saturated Cost Partitioning: (Seipp and Helmert, 2014)

Iterate:

1. Pick a heuristic h_i that hasn't been picked before.
   Terminate if none is left.
2. Compute h_i given current *cost*.
3. Compute a saturated cost function c_i for h_i.
4. Decrease $cost(o)$ by $c_i(o)$ for all operators o.

$\langle c_1, \ldots, c_n \rangle$ is a saturated cost partitioning for $\langle h_1, \ldots, h_n \rangle$ (in pick order).

Ordering strategies $\rightarrow$ Seipp, Keller, et al., 2020

General Cost Partitioning

Cost functions usually non-negative

- We tacitly also required this for task copies
- Makes intuitively sense: original costs are non-negative
- But: not necessary for cost-partitioning!

General Cost Partitioning

Cost functions usually non-negative

- We tacitly also required this for task copies
- Makes intuitively sense: original costs are non-negative
- But: not necessary for cost-partitioning!

General Cost Partitioning

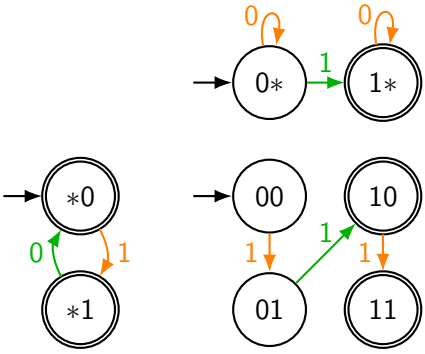
- Create copies $\Pi_1, \ldots, \Pi_n$ of planning task Π
- each has its own operator cost function $cost_i : O \to \mathbb{R}$ (otherwise identical to Π)
- for all o: require $cost_1(o) + \cdots + cost_n(o) \leq cost(o)$

$\rightsquigarrow$ sum of solution costs in copies is admissible heuristic: $h^*_{\Pi_1} + \cdots + h^*_{\Pi_n} \leq h^*_{\Pi}$

(Pommerening, Helmert, et al., 2015)

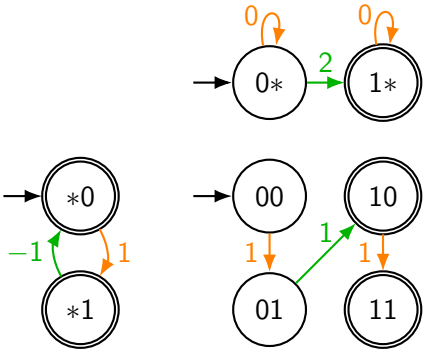
More powerful than non-negative cost partitioning

General Cost Partitioning: Example



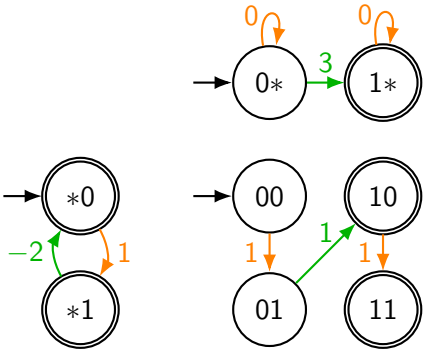
Heuristic value with
best non-negative cost partitioning:
$0 + 1 = 1$

General Cost Partitioning: Example



Heuristic value with general cost partitioning:
$0 + 2 = 2$

General Cost Partitioning: Example



Heuristic value:
$-\infty + 3 = -\infty$

Summary

- Delete-relaxation heuristics simplify the task by ignoring all delete effects.
- Abstraction heuristics simplify the task by not distinguishing all states.
- Cost partitioning permits to admissibly sum up admissible estimates.

References I

- Blai Bonet (2013). "An Admissible Heuristic for SAS$^+$ Planning Obtained from the State Equation". In: *Proc. IJCAI 2013*, pp. 2268–2274.
- Blai Bonet and Héctor Geffner (1999). "Planning as Heuristic Search: New Results". In: *Proc. ECP 1999*, pp. 360–372.
- Blai Bonet, Gábor Loerincs, and Héctor Geffner (1997). "A Robust and Fast Action Selection Mechanism for Planning". In: *Proc. AAAI 1997*, pp. 714–719.
- Blai Bonet and Menkes van den Briel (2014). "Flow-based Heuristics for Optimal Planning: Landmarks and Merges". In: *Proc. ICAPS 2014*, pp. 47–55.
- Teresa Maria Breyer and Richard E. Korf (2010). "1.6-Bit Pattern Databases". In: *Proc. AAAI 2010*, pp. 39–44.
- Clemens Büchner, Salomé Eriksson, Thomas Keller, and Malte Helmert (2023). "Landmark Progression in Heuristic Search". In: *Proc. ICAPS 2023*, pp. 70–79.

References II

- Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith (2000). "Counterexample-Guided Abstraction Refinement". In: *Proc. CAV 2000*, pp. 154–169.
- Joseph C. Culberson and Jonathan Schaeffer (1996). "Searching with Pattern Databases". In: *Proc. CSCSI 1996*, pp. 402–416.
- Carmel Domshlak, Jörg Hoffmann, and Michael Katz (2015). "Red-black planning: A New Systematic Approach to Partial Delete Relaxation". In: *Artificial Intelligence* 221, pp. 73–114.
- Carmel Domshlak, Michael Katz, and Alexander Shleyfman (2012). "Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search". In: *Proc. ICAPS 2012*, pp. 343–347.
- Carmel Domshlak, Michael Katz, and Alexander Shleyfman (2015). *Symmetry Breaking in Deterministic Planning as Forward Search: Orbit Space Search Algorithm*. Tech. rep. IS/IE-2015-03. Technion.

References III

- Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski (2006). "Directed Model Checking with Distance-Preserving Abstractions". In: *Proc. SPIN 2006*, pp. 19–34.
- Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski (2009). "Directed model checking with distance-preserving abstractions". In: *International Journal on Software Tools for Technology Transfer* 11.1, pp. 27–37.
- Stefan Edelkamp (2001). "Planning with Pattern Databases". In: *Proc. ECP 2001*, pp. 84–90.
- Stefan Edelkamp (2002). "Symbolic Pattern Databases in Heuristic Search Planning". In: *Proc. AIPS 2002*, pp. 274–283.
- Stefan Edelkamp (2006). "Automated Creation of Pattern Database Search Heuristics". In: *Proc. MoChArt 2006*, pp. 35–50.
- Gaojian Fan, Robert Holte, and Martin Müller (2018). "MS-lite: A Lightweight, Complementary Merge-and-Shrink Method". In: *Proc. ICAPS 2018*, pp. 74–82.

References IV

- Gaojian Fan, Martin Müller, and Robert Holte (2014). "Non-Linear Merging Strategies for Merge-and-Shrink Based on Variable Interactions". In: *Proc. SoCS 2014*, pp. 53–61.
- Ariel Felner, Richard E. Korf, Ram Meshulam, and Robert Holte (2007). "Compressed Pattern Databases". In: *Journal of Artificial Intelligence Research* 30, pp. 213–247.
- Daniel Fišer, Rostislav Horčík, and Antonín Komenda (2020). "Strengthening Potential Heuristics with Mutexes and Disambiguations". In: *Proc. ICAPS 2020*, pp. 124–133.
- Santiago Franco, Álvaro Torralba, Levi H. S. Lelis, and Mike Barley (2017). "On Creating Complementary Pattern Databases". In: *Proc. IJCAI 2017*, pp. 4302–4309.
- Peter E. Hart, Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

References V

- Patrik Haslum, Blai Bonet, and Héctor Geffner (2005). "New Admissible Heuristics for Domain-Independent Planning". In: *Proc. AAAI 2005*, pp. 1163–1168.
- Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig (2007). "Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning". In: *Proc. AAAI 2007*, pp. 1007–1012.
- Patrik Haslum and Héctor Geffner (2000). "Admissible Heuristics for Optimal Planning". In: *Proc. AIPS 2000*, pp. 140–149.
- Malte Helmert (2006). "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.
- Malte Helmert and Carmel Domshlak (2009). "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Proc. ICAPS 2009*, pp. 162–169.
- Malte Helmert, Patrik Haslum, and Jörg Hoffmann (2007). "Flexible Abstraction Heuristics for Optimal Sequential Planning". In: *Proc. ICAPS 2007*, pp. 176–183.

References VI

- Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim (2014). "Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces". In: *Journal of the ACM* 61.3, 16:1–63.
- Malte Helmert, Gabriele Röger, and Silvan Sievers (2015). "On the Expressive Power of Non-Linear Merge-and-Shrink Representations". In: *Proc. ICAPS 2015*, pp. 106–114.
- István T. Hernádvölgyi and Robert C. Holte (2000). "Experiments with Automatically Created Memory-Based Heuristics". In: *Proc. SARA 2000*, pp. 281–290.
- Jörg Hoffmann and Bernhard Nebel (2001). "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14, pp. 253–302.
- Jörg Hoffmann, Julie Porteous, and Laura Sebastia (2004). "Ordered Landmarks in Planning". In: *Journal of Artificial Intelligence Research* 22, pp. 215–278.

References VII

- Robert C. Holte and Neil Burch (2014). "Automatic Move Pruning for Single-Agent Search". In: *AI Communications* 27.4, pp. 363–383.
- Tatsuya Imai and Akihiro Kishimoto (2011). "A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning". In: *Proc. AAAI 2011*, pp. 985–991.
- Erez Karpas and Carmel Domshlak (2009). "Cost-Optimal Planning with Landmarks". In: *Proc. IJCAI 2009*, pp. 1728–1733.
- Michael Katz and Carmel Domshlak (2008). "Optimal Additive Composition of Abstraction-based Admissible Heuristics". In: *Proc. ICAPS 2008*, pp. 174–181.
- Michael Katz and Carmel Domshlak (2010). "Optimal admissible composition of abstraction heuristics". In: *Artificial Intelligence* 174.12–13, pp. 767–798.
- Emil Keyder and Héctor Geffner (2008). "Heuristics for Planning with Action Costs Revisited". In: *Proc. ECAI 2008*, pp. 588–592.

References VIII

- Emil Keyder, Jörg Hoffmann, and Patrik Haslum (2014). "Improving Delete Relaxation Heuristics Through Explicitly Represented Conjunctions". In: *Journal of Artificial Intelligence Research* 50, pp. 487–533.
- Emil Keyder, Silvia Richter, and Malte Helmert (2010). "Sound and Complete Landmarks for And/Or Graphs". In: *Proc. ECAI 2010*, pp. 335–340.
- Richard E. Korf (1997). "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases". In: *Proc. AAAI 1997*, pp. 700–705.
- Richard E. Korf and Ariel Felner (2002). "Disjoint Pattern Database Heuristics". In: *Artificial Intelligence* 134.1–2, pp. 9–22.
- Nir Lipovetzky and Hector Geffner (2017). "Best-First Width Search: Exploration and Exploitation in Classical Planning". In: *Proc. AAAI 2017*, pp. 3590–3596.
- Vitaly Mirkis and Carmel Domshlak (2007). "Cost-Sharing Approximations for h^+". In: *Proc. ICAPS 2007*, pp. 240–247.

References IX

- Raz Nissim, Jörg Hoffmann, and Malte Helmert (2011). "Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning". In: *Proc. IJCAI 2011*, pp. 1983–1990.
- Judea Pearl (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.
- Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp (2015). "From Non-Negative to General Operator Cost Partitioning". In: *Proc. AAAI 2015*, pp. 3335–3341.
- Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet (2014). "LP-based Heuristics for Cost-optimal Planning". In: *Proc. ICAPS 2014*, pp. 226–234.
- Martín Pozo, Álvaro Torralba, and Carlos Linares López (2024a). "Gotta Catch 'Em All! Sequence Flaws in CEGAR for Classical Planning". In: *Proc. ECAI 2024*, pp. 4287–4294.

References X

- Martín Pozo, Álvaro Torralba, and Carlos Linares López (2024b). "When CEGAR Meets Regression: A Love Story in Optimal Classical Planning". In: *Proc. AAAI 2024*, pp. 20238–20246.
- Silvia Richter and Matthias Westphal (2010). "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39, pp. 127–177.
- Alexander Rovner, Silvan Sievers, and Malte Helmert (2019). "Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning". In: *Proc. ICAPS 2019*, pp. 362–367.
- Tim Schmidt and Rong Zhou (2011). "Representing Pattern Databases with Succinct Data Structures". In: *Proc. SoCS 2011*, pp. 142–149.
- Jendrik Seipp and Malte Helmert (2014). "Diverse and Additive Cartesian Abstraction Heuristics". In: *Proc. ICAPS 2014*, pp. 289–297.

References XI

- Jendrik Seipp and Malte Helmert (2018). "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning". In: *Journal of Artificial Intelligence Research* 62, pp. 535–577.
- Jendrik Seipp, Thomas Keller, and Malte Helmert (2020). "Saturated Cost Partitioning for Optimal Classical Planning". In: *Journal of Artificial Intelligence Research* 67, pp. 129–167.
- Jendrik Seipp, Florian Pommerening, and Malte Helmert (2015). "New Optimization Functions for Potential Heuristics". In: *Proc. ICAPS 2015*, pp. 193–201.
- Silvan Sievers and Malte Helmert (2021). "Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems". In: *Journal of Artificial Intelligence Research* 71, pp. 781–883.

References XII

- Silvan Sievers, Thomas Keller, and Gabriele Röger (2024). "Merging or Computing Saturated Cost Partitionings? A Merge Strategy for the Merge-and-Shrink Framework". In: *Proc. ICAPS 2024*, pp. 541–545.
- Silvan Sievers, Florian Pommerening, Thomas Keller, and Malte Helmert (2020). "Cost-Partitioned Merge-and-Shrink Heuristics for Optimal Classical Planning". In: *Proc. IJCAI 2020*, pp. 4152–4160.
- Silvan Sievers, Martin Wehrle, and Malte Helmert (2014). "Generalized Label Reduction for Merge-and-Shrink Heuristics". In: *Proc. AAAI 2014*, pp. 2358–2366.
- Silvan Sievers, Martin Wehrle, and Malte Helmert (2016). "An Analysis of Merge Strategies for Merge-and-Shrink Heuristics". In: *Proc. ICAPS 2016*, pp. 294–298.
- Jordan T. Thayer and Wheeler Ruml (2011). "Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates". In: *Proc. IJCAI 2011*, pp. 674–679.

References XIII

- Martin Wehrle, Malte Helmert, Yusra Alkhazraji, and Robert Mattmüller (2013). "The Relative Pruning Power of Strong Stubborn Sets and Expansion Core". In: *Proc. ICAPS 2013*, pp. 251–259.
- Martin Wehrle, Malte Helmert, Alexander Shleyfman, and Michael Katz (2015). "Integrating Partial Order Reduction and Symmetry Elimination for Cost-Optimal Classical Planning". In: *Proc. IJCAI 2015*, pp. 1712–1718.
- Fan Xie, Martin Müller andRobert Holte, and Tatsuya Imai (2014). "Type-based exploration with multiple search queues for satisficing planning". In: *Proc. AAAI 2014*, pp. 2395–2401.
- Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner (2008). "A General Theory of Additive State Space Abstractions". In: *Journal of Artificial Intelligence Research* 32, pp. 631–662.