

## AI Planning

### 12. Pattern Database Heuristics

It's a Long Way to the Goal, But How Long Exactly?  
 Part III, How-To (A): *Willfully Ignoring Some of Those Variables*

Álvaro Torralba, Cosmina Croitoru



Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

## Agenda

- 1 Introduction
- 2 Pattern Database Basics
- 3 Pattern Database Implementation
- 4 Orthogonal Patterns, and How to Exploit Them
- 5 Redundant Patterns, and How to Recognize Them
- 6 Pattern Selection
- 7 Conclusion

## Reminder: Our Program for Abstraction Heuristics

**We take a look at abstractions and their use for generating admissible heuristic functions:**

- In **Chapter 11**, we formally introduced abstractions and abstraction heuristics and studied some of their most important properties.
- In **This Chapter**, we discuss a particular class of abstraction heuristics and its practical handling in detail, namely **pattern database heuristics**.
- In **Chapter 13**, we will discuss another particular class of abstraction heuristics and its practical handling in detail, namely **merge-and-shrink abstractions**.

→ We handle all these methods in FDR, where they are most natural. We do not mention STRIPS at all (which is a special case anyway).

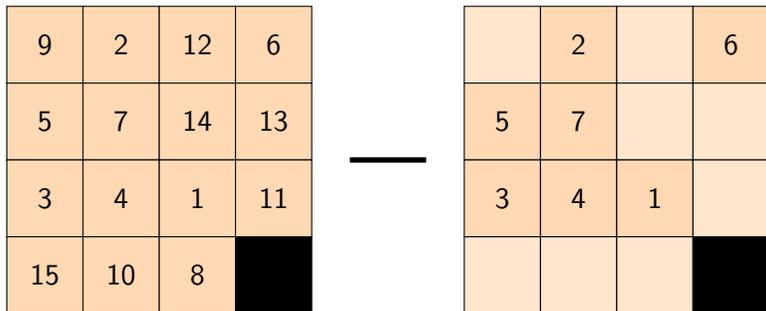
## Motivation for Pattern Database Heuristics

→ Pattern databases are a concrete method for designing abstraction functions  $\alpha$ , and for computing the associated heuristic functions.

**There's many good reasons to be considering PDBs:**

- Pattern database (PDB) heuristics are the most commonly used class of abstraction heuristics outside planning (Games, mostly).
- PDBs are one of the two most commonly used classes of abstraction heuristics in planning (we discuss the other class in **Chapter 13**).
- PDBs have been a very active research area from their inception, and still are a very active research area today. (Theoretical properties, how to implement and use PDBs effectively, how to find good patterns, ...)
- For many search problems, pattern databases are the most effective admissible heuristics currently known.

# Pattern Databases in a Nutshell



“Abstract the planning task by choosing a subset  $P$  of variables (the pattern), and ignoring the values of all other variables.”

# Our Agenda for This Chapter

- 2 **Pattern Database Basics:** Formal definition; illustration.
- 3 **Pattern Database Implementation:** How to implement PDB heuristics (namely, via a “pattern database”).
- 4 **Orthogonal Patterns, and How to Exploit Them:** How to admissibly sum-up multiple PDB heuristics. Important because PDB heuristics are a very restricted class of abstractions, and any single individual PDB is not typically very useful. Much of their power lies in summing up their values, where admissibly possible.
- 5 **Redundant Patterns, and How to Recognize Them:** A redundant pattern is one that wastes time and memory, incurring a larger than necessary PDB without any benefits. We introduce three easily recognizable cases where that happens.
- 6 **Pattern Selection:** How to find good pattern collections. There are many possible choices and it is important to identify good ones (no redundant patterns of course, but there’s much more to it than that).

# Pattern Database Heuristics

“Pattern database heuristics” = Heuristics induced by a particular class of abstraction mappings, namely **projections**:

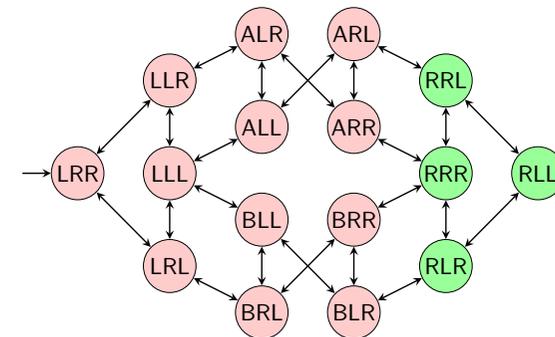
**Definition (Projection, PDB Heuristic).** Let  $\Pi = (V, A, c, I, G)$  be an FDR planning task with state space  $\Theta_{\Pi} = (S, L, c, T, I, S^G)$ , and let  $P \subseteq V$ . For a partial assignment  $\varphi$  to  $V$ , by  $\varphi|_P$  we denote the restriction of  $\varphi$  to  $P$ . Let  $S^P$  be the set of variable assignments to  $P$ . The **projection**  $\pi_P: S \mapsto S^P$  is defined by  $\pi_P(s) := s|_P$ . We say that  $\pi_P$  is **atomic** if  $|P| = 1$ .

→  $\pi_P$  maps two states  $s_1$  and  $s_2$  to the same abstract state iff they agree on all variables in the pattern.

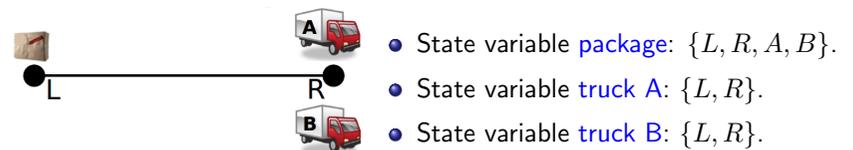
We refer to  $P$  as the **pattern** of  $\pi_P$ . The abstraction heuristic induced by  $\pi_P$  on  $\Theta_{\Pi}$  is called a **pattern database heuristic**, short **PDB heuristic**. We write  $h^P$  as a short-hand for  $h^{\pi_P}$ , and we write  $\Theta_{\Pi}^P$  or  $\Theta^P$  as short-hands for  $\Theta_{\Pi}^{\pi_P}$ .

- $h^P$  is usually stored in a lookup table called a **pattern database** (PDB).
- **Atomic projections** will be heavily used in **Chapter 13**.

# “Logistics mal anders”: State Space



Logistics task with one package, two trucks, two locations:





# I Give You Pattern, You Give Me Database!

**Assume:** You are given a pattern  $P$ .

- How do you compute  $h^P$ ?
- More precisely: How do you compute a data structure that efficiently represents the function  $h^P(s)$ , for all states  $s$ ?

**Here's how:**

- 1 In a **precomputation** step, we compute an explicit graph representation for the abstract state space  $\Theta_{\Pi}^{\pi P}$ , and compute the abstract remaining cost for every abstract state.
- 2 During search, we use the precomputed abstract remaining costs in a **lookup** step.

# (I) Precomputation Step: Here's How To

**Definition (Syntactic Projection).** Let  $\Pi = (V, A, c, I, G)$  be an FDR planning task, and let  $P \subseteq V$ . The **syntactic projection** of  $\Pi$  to  $P$  is the FDR planning task  $\Pi|_P = (P, A|_P, c, I|_P, G|_P)$  where  $A|_P := \{a|_P \mid a \in A\}$  with  $pre_{a|_P} := (pre_a)|_P$  and  $eff_{a|_P} := (eff_a)|_P$ .

→  $\Pi|_P$  removes the variables outside  $P$  from all constructs in the planning task description  $\Pi$ .

**Theorem (Syntactic Projection is Equivalent to Projection).** Let  $\Pi$  be an FDR planning task, and let  $P \subseteq V$ . Then  $\Theta_{(\Pi|_P)}$  is identical to  $\Theta_{\Pi}^{\pi P}$  except that labels  $a$  in the latter become labels  $a|_P$  in the former.

**Proof.** Easy from definition.

→ The state space of the syntactic projection is (modulo label renaming) the same as the abstract state space of the projection.

# (I) Precomputation Step: It's Not That Easy

Let  $\Pi$  be a planning task and  $P$  a pattern. Let  $\Theta = \Theta_{\Pi}$  and  $\Theta' = \Theta_{\Pi}^{\pi P}$ . We want to compute a graph representation of  $\Theta'$ .

**So, what's the issue?**

- $\Theta'$  is defined through a function on  $\Theta$ :
  - Each concrete transition induces an abstract transition, each concrete goal state induces an abstract goal state.
- In principle, we can compute  $\Theta'$  by iterating over all transitions/goal states of  $\Theta$ . BUT:
  - This would take time  $\Omega(\|\Theta\|)$ .
  - Which comes down to solving the original (concrete, not abstract) planning task in the first place, using blind search.

→ We need a way of computing  $\Theta'$  in time polynomial in  $\|\Pi\|$  and  $\|\Theta'\|$ .

# (I) Precomputation Step: Here's How To, ctd.

Using the Theorem on the previous slide, we can compute pattern databases for FDR tasks  $\Pi$  and patterns  $P$ :

## Computing Pattern Databases

```
def compute-PDB( $\Pi, P$ ):
     $\Pi' := \Pi|_P$ .
    Compute  $\Theta' := \Theta_{\Pi'}$  by a complete forward search (e.g., breadth-first).
    In the explicit graph  $\Theta'$ , add a new node  $x$  with a 0-cost incoming edge
    from every goal node
    Run Dijkstra starting from  $x$  and traversing edges backwards, to compute
    all cheapest paths to  $x$  and thus the remaining costs  $h_{\Theta'}^*$  in  $\Theta'$ 
     $PDB :=$  a table containing all remaining costs in  $\Theta'$ 
    return  $PDB$ 
```

→ This algorithm runs in time and space polynomial in  $\|\Pi\| + \|\Theta'\|$ .

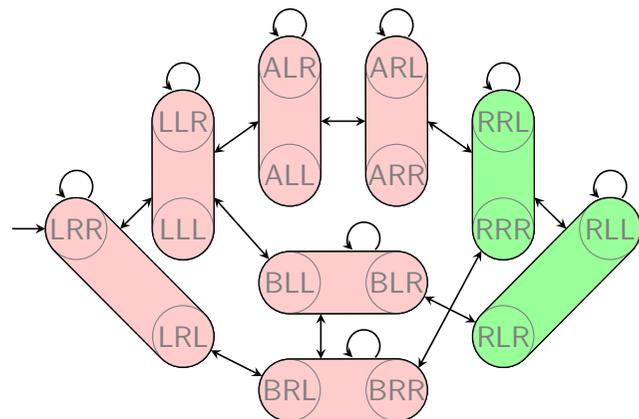
## (II) Lookup Step: Overview

### Basic observations and method:

- During search, we do not need the actual abstract state space (transitions etc): The PDB is the only piece of information necessary to represent  $h^P$ .  
 → We can throw away the abstract state space  $\Theta'$  once the PDB is computed.  
 → Space requirement for the PDB heuristic during search is linear in number of abstract states  $S'$ : PDB has one table entry for each abstract state.
- Design a perfect hash function mapping projected states  $s|_P$  to numbers in the range  $\{0, \dots, |S'| - 1\}$ .  
 → Index PDB by these hash values. Given a state  $s$  during search, to compute  $h^P(s)$ , map  $\pi_P(s) = s|_P$  to its hash value and lookup the table entry of PDB.

## (II) Lookup Step: “Logistics mal anders”

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



## (II) Lookup Step: Here’s How To

### Perfect hash function $\approx$ numeral system over variable domains:

- Let  $P = \{v_1, \dots, v_k\}$  be the pattern.
- Assume wlog that all variable domains are natural numbers counted from 0, i.e.,  $D_v = \{0, 1, \dots, |D_v| - 1\}$ .
- For all  $i \in \{1, \dots, k\}$ , we precompute  $N_i := \prod_{j=1}^{i-1} |D_{v_j}|$ .

### Looking Up a Pattern Database Heuristic Value

```
def PDB-heuristic(s):
    index :=  $\sum_{i=1}^k N_i s(v_i)$ 
    return PDB[index]
```

**Note:** This lookup runs in time and space  $O(k)$ . This is very fast. For comparison, delete-relaxation heuristics need time  $O(\|II\|)$  per state.

## (II) Lookup Step: “Logistics mal anders”, ctd.

### Pattern variables and domains:

- $P = \{v_1, v_2\}$  with  $v_1 = \text{package}$ ,  $v_2 = \text{truck A}$ .
- $D_{v_1} = \{L, R, A, B\} \approx \{0, 1, 2, 3\}$
- $D_{v_2} = \{L, R\} \approx \{0, 1\}$

→  $N_1 = \prod_{j=1}^0 |D_{v_j}| = 1$ .

→  $N_2 = \prod_{j=1}^1 |D_{v_j}| = 4$ .

→  $index(s) = 1 * s(\text{package}) + 4 * s(\text{truck A})$ .

### → Pattern database:

abstract state	LL	RL	AL	BL	LR	RR	AR	BR
index	0	1	2	3	4	5	6	7
value	2	0	2	1	2	0	1	1

## And Now: The Australia Example



- Variables:  $at : \{Sy, Ad, Br, Pe, Ad\}$ ;  
 $v(x) : \{T, F\}$  for  $x \in \{Sy, Ad, Br, Pe, Ad\}$ .
- Actions:  $drive(x, y)$  where  $x, y$  have a road.
- Costs:  $Sy \leftrightarrow Br : 1, Sy \leftrightarrow Ad : 1.5, Ad \leftrightarrow Pe : 3.5, Ad \leftrightarrow Da : 4$ .
- Initial state:  $at = Sy, v(Sy) = T, v(x) = F$  for  $x \neq Sy$ .
- Goal:  $at = Sy, v(x) = T$  for all  $x$ .

**Question:** Say our pattern  $P$  is  $\{v_1 = v(Br), v_2 = v(Pe), v_3 = v(Da)\}$ . What is the PDB?

$\rightarrow D_{v(Br)} = \{F, T\} \approx \{0, 1\}, N_1 = 1; D_{v(Pe)} = \{F, T\} \approx \{0, 1\}, N_2 = 2;$   
 $D_{v(Da)} = \{F, T\} \approx \{0, 1\}, N_3 = 4.$

abstract state	FFF	TFF	FTF	TTF	FFT	TFT	FTT	TTT
index	0	1	2	3	4	5	6	7
value	8.5	7.5	5	4	4.5	3.5	1	0

**Note:** "Value = sum over the Fs", i.e.  $h^P =$  sum of the corresponding single-variable heuristics.  $P$  is "causally disconnected", cf. slides 44–46.

## Pattern Collections

### Pattern Collections? Why and How?

- The space requirements for a pattern database grow exponentially with the number of state variables in the pattern.  
 $\rightarrow$  Hence  $P$  must be small, severely limiting the usefulness of single-PDB heuristics  $h^P$  for large planning tasks.
- To overcome this limitation, planners using pattern databases can work with **collections** of multiple patterns.
- Given heuristics  $h^{P_1}$  and  $h^{P_2}$ , we can always get an admissible heuristic dominating each of  $h^{P_1}$  and  $h^{P_2}$  by  $\max\{h^{P_1}, h^{P_2}\}$ .
- Combination of  $h^{P_1}$  and  $h^{P_2}$  that would be much preferable because it dominates the previous one:  $h^{P_1} + h^{P_2} \dots !$   
 $\rightarrow$  But, **for this to be admissible,  $h^{P_1}$  and  $h^{P_2}$  must be additive.**

## When Does a Label Affect a PDB?

**Reminder:**  $\rightarrow$  **Chapter 11**

Let  $\alpha$  be an abstraction of  $\Theta$ , and let  $l$  be a label in  $\Theta$ . We say that  $l$  **affects**  $\alpha$  if  $\Theta^\alpha$  has at least one non-self-loop transition labeled by  $l$ .

Let  $\alpha_1$  and  $\alpha_2$  be abstractions of  $\Theta$ . We say that  $\alpha_1$  and  $\alpha_2$  are **orthogonal** if **no label of  $\Theta$  affects both  $\alpha_1$  and  $\alpha_2$ .**

**Lemma (Labels Affecting PDBs).** Let  $P$  be a pattern for an FDR planning task  $\Pi$ , and let  $a$  be an action in  $\Pi$ . Then  $a$  **affects**  $\pi_P$  if and only if there exists a variable  $v \in P$  on which  $eff_a$  is defined.

**Proof.** Consider the syntactic projection  $\Pi|_P$ .

$\rightarrow$  **Only if:** If  $a$  has no effect on  $P$  then its effect in  $\Pi|_P$  is empty so it can label only self-loops.

$\rightarrow$  **If:** Design a state  $s$  in  $\Pi|_P$  by projecting  $pre_a$  onto  $P$  and filling up the remaining  $P$ -values arbitrarily but different from  $eff_a$ . Then  $a$  labels the non-self-loop transition  $(s, a, s[[a]])$  in  $\Pi|_P$ .

## Orthogonal Patterns

**Lemma (Orthogonal Patterns).** Let  $P_1$  and  $P_2$  be patterns for an FDR planning task  $\Pi$ . Then  $\pi_{P_1}$  and  $\pi_{P_2}$  are orthogonal if and only if there exists no action  $a$  in  $\Pi$  such that  $eff_a$  is defined for variables  $v, v'$  where  $v \in P_1$  and  $v' \in P_2$ . (Direct from Lemma on previous slide)

**Terminology:** In this situation, we also call the patterns  $P_1$  and  $P_2$  themselves (as opposed to  $\pi_{P_1}$  and  $\pi_{P_2}$ ) **orthogonal**.

### On orthogonality and (non)-intersecting patterns:

- If  $P_1 \cap P_2 \neq \emptyset$ , can  $P_1$  and  $P_2$  be orthogonal? No, except if all  $v \in P_1 \cap P_2$  are not affected by any action, which is pathological.
- If  $P_1 \cap P_2 = \emptyset$ , are  $P_1$  and  $P_2$  necessarily orthogonal? No, there may be an action with a "cross-effect", on one variable from each.

**Note:** Disjoint  $P_1$  and  $P_2$  are orthogonal iff there is no causal graph (ii) arc between them (effect-effect arcs, cf. **Chapter 5** or slide 41).

# The Australia Example (Strikes Again)



- Variables:  $at : \{Sy, Ad, Br, Pe, Ad\}$ ;  
 $v(x) : \{T, F\}$  for  $x \in \{Sy, Ad, Br, Pe, Ad\}$ .
- Actions:  $drive(x, y)$  where  $x, y$  have a road.
- Costs:  $Sy \leftrightarrow Br : 1, Sy \leftrightarrow Ad : 1.5, Ad \leftrightarrow Pe : 3.5, Ad \leftrightarrow Da : 4$ .
- Initial state:  $at = Sy, v(Sy) = T, v(x) = F$  for  $x \neq Sy$ .
- Goal:  $at = Sy, v(x) = T$  for all  $x$ .

**Observe:** Are the patterns  $P_1 = \{v(Br)\}$  and  $P_2 = \{v(Pe), v(Da)\}$  orthogonal?

→ Yes: Every action affects at most one of them.

**Observe:** Are the patterns  $P'_1 = \{v(Br), at\}$ ,  $P_2 = \{v(Pe), v(Da)\}$  orthogonal?

→ No: E.g., driving to Perth affects both of them.

# Questionnaire



- Variables, for boxes  $b_i$  and cells  $c_i$ :  $robotPos : \{c_i\}$ ,  $boxPos(x) : \{c_i\}$  for each  $b_i$ ,  $free(c_i) : \{T, F\}$  for each  $c_i$ .
- Actions:  $move(c, c')$  where  $c, c'$  adjacent:  
 pre  $robotPos = c, free(c') = T$ ; eff  $robotPos = c'$ .  
 $push(b, c, c', c'')$  where  $c, c', c''$  arranged in a line:  
 pre  $robotPos = c, boxPos(b) = c'; free(c'') = T$   
 eff  $robotPos = c', boxPos(b) = c'', free(c'') = F, free(c') = T$ .
- Goal:  $free(c) = F$  for the goal cells  $c$ .

## Question!

What are orthogonal patterns  $P_i$  in this Sokoban task?

- (A):  $robotPos \in P_i$  for  $\leq$  one  $P_i$
- (B): For all  $b_j$ :  $boxPos(b_j) \in P_i$  for  $\leq$  one  $P_i$
- (C):  $free(c) \in P_i \Rightarrow$  f.a.  $j \neq i, c'$  adjacent to  $c$ :  $free(c') \notin P_j$
- (D): Other (choose freely)

→ (C) is needed: Every cell is member of at least one line of 3 cells, so for every pair of adjacent cells  $c$  and  $c'$ , there is a  $push$  action affecting both  $free(c)$  and  $free(c')$ .

→ (A) and (B) are needed (else, the patterns intersect). But they are not enough!  $push$  actions affect  $robotPos$  and  $boxPos(b_j)$ . So if  $robotPos \in P_i$  then no other pattern can contain any  $boxPos(b_j)$ ! → The effectiveness of orthogonality depends on the domain ...

# Orthogonal Patterns and Additivity

**Terminology:** Instead of “collection of pairwise orthogonal patterns”, we also say **orthogonal pattern collection** or **orthogonal collection**.

**Theorem (Orthogonal Patterns are Additive).** Let  $\{P_1, \dots, P_k\}$  be an orthogonal pattern collection for an FDR planning task  $\Pi$ . Then  $\sum_{i=1}^k h^{P_i}$  is consistent and goal-aware, and thus also admissible and safe.

**Proof.** By the slide 28 Lemma, the abstractions  $\pi_{P_1}, \dots, \pi_{P_k}$  are pairwise orthogonal. The claim follows with the Theorem “Orthogonal Abstractions are Additive” from **Chapter 11**.

**Observation:** A single large pattern is more expensive, but is better informativity-wise.

→ **At least as good:** For orthogonal  $P_i$  each of which is contained in  $P$ , we have  $\sum_{i=1}^k h^{P_i} \leq h^P$ . (See next slide)

→ **Potentially better:** Orthogonal  $P_i$  with  $P = P_1 \cup \dots \cup P_k$  does NOT imply  $\sum_{i=1}^k h^{P_i} = h^P$ . (Details see slide 59.)

# Large Patterns Dominate Orthogonal Smaller Ones

**Theorem.** Let  $\{P_1, \dots, P_k\}$  be an orthogonal pattern collection for an FDR planning task, and let  $P$  be a pattern with  $P_i \subseteq P$  for all  $i \in \{1, \dots, k\}$ . Then  $\sum_{i=1}^k h^{P_i} \leq h^P$ .

**Proof.** Consider the syntactic projection onto  $P$  i.e., the FDR planning task  $\Pi|_P$ .

Because  $P_i \subseteq P$ , we can perceive each  $\pi_{P_i}$  as a projection of  $\Pi|_P$ , and hence perceive the corresponding heuristics  $h^{P_i}$  as abstraction heuristics on  $\Pi|_P$ .

Clearly,  $\{P_1, \dots, P_k\}$  on  $\Pi|_P$  is still an orthogonal pattern collection. Hence, as  $h^P$  is defined as  $h^*$  on  $\Pi|_P$ , the claim follows with the theorem on the previous slide.

## How to Exploit Orthogonal Patterns?

Given **pattern collection  $\mathcal{C}$** : How to get the best possible lower bound?

- Build the **compatibility graph** for  $\mathcal{C}$ .
  - The vertices are the patterns  $P \in \mathcal{C}$ .
  - Arcs between pairs of orthogonal patterns.
- Compute all **maximal cliques** (maximal orthogonal sub-collections). (This is actually **NP-hard**, but this graph will be small in practice)

### Abstract Example

FDR task with variables  $V = \{v_1, v_2, v_3\}$ , and pattern collection  $\mathcal{C} = \{P_1, \dots, P_4\}$  with  $P_1 = \{v_1, v_2\}$ ,  $P_2 = \{v_1\}$ ,  $P_3 = \{v_2\}$  and  $P_4 = \{v_3\}$ .

There are actions affecting each individual variable, and the only actions affecting several variables affect  $v_1$  and  $v_3$ .

→ **Maximal cliques in the compatibility graph for  $\mathcal{C}$** ?  $\{P_1\}$ ,  $\{P_2, P_3\}$ ,  $\{P_3, P_4\}$ .

## The Canonical Heuristic Function

**Definition (Canonical Heuristic)**. Let  $\Pi$  be an FDR planning task, let  $\mathcal{C}$  be a pattern collection for  $\Pi$ , and let  $\text{cliques}(\mathcal{C})$  be the set of all maximal cliques in the compatibility graph for  $\mathcal{C}$ . Then the **canonical heuristic  $h^{\mathcal{C}}$**  for  $\mathcal{C}$  is defined as

$$h^{\mathcal{C}}(s) = \max_{\mathcal{D} \in \text{cliques}(\mathcal{C})} \sum_{P \in \mathcal{D}} h^P(s)$$

→ The canonical heuristic maximizes over all largest orthogonal subsets of our pattern collection. It is admissible and consistent.

### Same Abstract Example

Maximal cliques in the compatibility graph:  $\{P_1\}$ ,  $\{P_2, P_3\}$ ,  $\{P_3, P_4\}$ .

→ Canonical heuristic function  $h^{\mathcal{C}}$ :

$$\begin{aligned} h^{\mathcal{C}} &= \max \{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\} \\ &= \max \{h^{\{v_1, v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \end{aligned}$$

## The Australia Example (Yet Again)



- Variables:  $at : \{Sy, Ad, Br, Pe, Ad\}$ ;  
 $v(x) : \{T, F\}$  for  $x \in \{Sy, Ad, Br, Pe, Ad\}$ .
- Actions:  $drive(x, y)$  where  $x, y$  have a road.
- Costs:  $Sy \leftrightarrow Br : 1$ ,  $Sy \leftrightarrow Ad : 1.5$ ,  $Ad \leftrightarrow Pe : 3.5$ ,  $Ad \leftrightarrow Da : 4$ .
- Initial state:  $at = Sy$ ,  $v(Sy) = T$ ,  $v(x) = F$  for  $x \neq Sy$ .
- Goal:  $at = Sy$ ,  $v(x) = T$  for all  $x$ .

### Observations:

- Say  $P_1 = \{v(Br)\}$  and  $P_2 = \{v(Pe), v(Da)\}$ . What are the values of  $h^{P_1}(I)$  and  $h^{P_2}(I)$ ?  $h^{P_1}(I) = 1$ ,  $h^{P_2}(I) = 4 + 3.5 = 7.5$  (cf. slide 24).
- Say  $\mathcal{C} = \{P_1, P_2\}$ . What is the value of  $h^{\mathcal{C}}(I)$ ?  $P_1$  and  $P_2$  are orthogonal (slide 29), so  $h^{\mathcal{C}} = h^{P_1}(I) + h^{P_2}(I) = 1 + 7.5 = 8.5$ .
- Say  $P'_1 = \{v(Br), at\}$ . What is the value of  $h^{P'_1}(I)$ ?  $h^{P'_1}(I) = 2$  because now we have to satisfy also the truck goal.
- Say  $\mathcal{C} = \{P'_1, P_2\}$ . What is the value of  $h^{\mathcal{C}}(I)$ ?  $P'_1$  and  $P_2$  are not orthogonal (slide 29), so  $h^{\mathcal{C}} = \max \{h^{P'_1}(I), h^{P_2}(I)\} = 7.5$ .

**Note:** Even though  $P'_1$  itself yields a better lower bound than  $P_1$ , replacing  $P_1$  with  $P'_1$  is detrimental for the overall lower bound  $h^{\mathcal{C}}(I)$ .

## Questionnaire (The Attack of the Zombie Tomatoes)



- Goal:  $A$  and  $B$  both true.
- Initial state:  $A$  and  $B$  both false.
- Actions:  $carA$  effect  $A$  cost 1;  $carB$  effect  $A$  and  $B$  cost 1.5;  $fancyCar$  effect  $A$  and  $B$  cost 1.5.
- Patterns  $P_1 = \{A\}$  and  $P_2 = \{B\}$ .

### Question!

Are  $P_1$  and  $P_2$  orthogonal? What is the value of  $h^{P_1}(I)$  and  $h^{P_2}(I)$ ?

→ They are not orthogonal as  $fancyCar$  affects both patterns.  $h^{P_1}(I) = h^{P_2}(I) = 1$  due to  $carA$  respectively  $carB$ .

### Question!

So  $h^{\mathcal{C}}(I) =$

→ As  $P_1$  and  $P_2$  not orthogonal,  $h^{\mathcal{C}}(I) = \max \{h^{P_1}(I), h^{P_2}(I)\} = 1$ .

# ps. On the Power of the Canonical Heuristic

→ The canonical heuristic  $h^C$  is the best possible admissible heuristic we can derive from  $C$  based on exploiting only orthogonality.

## In The Attack of the Zombie Tomatoes, that is not enough:



- Goal:  $A$  and  $B$  both true.
- Initial state:  $A$  and  $B$  both false.
- Actions:  $carA$  effect  $A$  cost 1;  $carB$  effect  $B$  cost 1;  $fancyCar$  effect  $A$  and  $B$  cost 1.5.
- Patterns  $P_1 = \{A\}$  and  $P_2 = \{B\}$ .

→  $h^C(I) = 1 < 1.5 = h^*(I)$ .

→ In Chapter 16 we introduce cost partitioning, a more general technique for combining heuristics, which always dominates  $h^C$ , and which yields the perfect lower bound 1.5 in this particular example.

# Redundant Patterns

A pattern  $P$  is redundant if there exist patterns  $P'_1, \dots, P'_k \subsetneq P$  so that  $\sum_{i=1}^k h^{P'_i} = h^P$ .

- We can use  $P'_1, \dots, P'_k$  instead of  $P$ .
- The sum of PDB sizes for  $P'_1, \dots, P'_k$  will (typically) be much smaller than that of the PDB for  $P$ , so building a PDB for  $P$  is a waste of time and memory.
- We identify three “redundant cases” where  $P$  as above occurs:
  - ❶ Non-Goal Patterns.
  - ❷ Causally Irrelevant Variables.
  - ❸ Causally Disconnected Patterns.

# Redundant Case I: Non-Goal Patterns

**Proposition (Non-Goal Patterns are Redundant).** Let  $\Pi = (V, A, c, I, G)$  be an FDR planning task, and let  $P$  be a pattern for  $\Pi$  such that  $G$  is not defined for any variable in  $P$ . Then  $h^P(s) = 0$  for all states  $s$ .

**Proof.** Any PDB heuristic  $h^P$  is equivalent to optimal planning in the syntactic projection  $\Pi|_P$  onto the pattern (cf. slide 18). As  $P$  here contains no goal variable, the goal in  $\Pi|_P$  is empty, from which the claim follows trivially.

→ If a pattern contains no goal variable, then the heuristic returned is constant 0.

→ There is no point in making a pattern without a goal.

→ Which  $P' \subsetneq P$  yields the same heuristic function as  $P$ ?  $P' = \emptyset$ .

# Redundant Case II: Causally Irrelevant Variables

**Reminder:** → Chapter 5

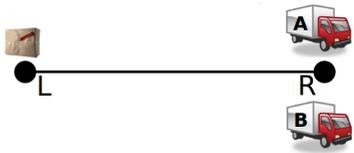
Let  $\Pi = (V, A, c, I, G)$  be an FDR planning task. The causal graph of  $\Pi$  is the directed graph  $CG(\Pi)$  with vertices  $V$  and an arc  $(u, v)$  whenever there exists an action  $a \in A$  so that either (i) there exists  $a \in A$  so that  $pre_a(u)$  and  $eff_a(v)$  are both defined, or (ii) there exists  $a \in A$  so that  $eff_a(u)$  and  $eff_a(v)$  are both defined.

**Definition (Causally Relevant Variables).** Let  $\Pi = (V, A, c, I, G)$  be an FDR planning task, and let  $P$  be a pattern for  $\Pi$ . We say that  $v \in P$  is causally relevant for  $P$  if the sub-graph of  $CG(\Pi)$  induced by  $P$  contains a directed path from  $v$  to a variable  $v' \in P$  for which  $G$  is defined.

→ Note that any goal variable  $v$  is causally relevant for  $P$ , due to the empty path in  $CG(\Pi)$ .

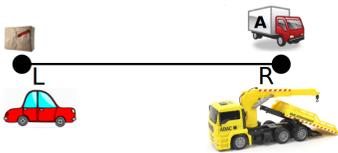
## Causally Irrelevant Variables: Examples

**Example 1:** “Logistics mal anders” ( $P :=$  all variables)



- **Is the package causally relevant?** Yes: It's a goal variable.
- **Are the trucks causally relevant?** Yes: Via (i) arcs because loading/unloading the package has a precondition on the truck.
- **If I added a dog, would that be causally relevant?** No.

**Example 2:** “Logistics mal ganz anders”



- The truck loads/unloads the package, and the ADAC Abschleppwagen loads/unloads either the truck or the red car. The package and red car need to be moved.
- **In  $P = \{\text{redcar, truck, ADAC}\}$ , is the truck causally relevant?** No: Moving it cannot contribute to the red-car goal.
- **And in  $P = \{\text{truck, package}\}$ ?** Yes. Causal relevance depends on the pattern.

## Causally Irrelevant Variables are Useless

**Theorem (Causally Irrelevant Variables are Useless).** Let  $P \subseteq V$  be a pattern for an FDR planning task  $\Pi$ , and let  $P_G \subseteq P$  consist of all variables that are causally relevant for  $P$ . Then  $h^{P_G}(s) = h^P(s)$  for all states  $s$ .

→ If a variable  $v$  has no influence on any goal variable in  $P$ , then  $v$  does not affect remaining cost in  $\Theta^P$ .

→ There is no point in growing a pattern by adding a variable that is causally irrelevant in the resulting pattern.

**Proof.** Any PDB heuristic  $h^P$  is equivalent to optimal planning in the syntactic projection  $\Pi|_P$  onto the pattern (cf. slide 18).

In the FDR task  $\Pi|_P$ , the variables  $P \setminus P_G$  have no path in the causal graph to any goal variable.

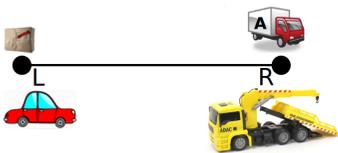
As we showed in **Chapter 5**, if  $v$  is a causal graph leaf without an own goal, then  $v$  can be removed without affecting  $h^*$ . Iterating such removal, all of  $P \setminus P_G$  will be removed, from which the claim follows.

## Redundant Case III: Causally Disconnected Patterns

**Definition (Causally Connected Patterns).** Let  $\Pi$  be an FDR planning task, and let  $P$  be a pattern for  $\Pi$ . We say that  $P$  is **causally connected** if the subgraph of  $CG(\Pi)$  induced by  $P$  is weakly connected, i.e., contains a path from every vertex to every other vertex when ignoring arc directions.

**Example 1:** In “Logistics mal anders” (2 trucks 1 package), is  $\{\text{Truck A, Truck B}\}$  causally connected? No: The trucks are connected only via the package, which is not in the pattern.

**Example 2:** “Logistics mal ganz anders”



- The truck loads/unloads the package, and the ADAC Abschleppwagen loads/unloads either the truck or the red car. The package and red car need to be moved.
- → **Is  $P = \{\text{package, redcar, ADAC}\}$  causally connected?** No: The package is connected to red car and ADAC only via the truck.

## Causally Disconnected Patterns are Decomposable

**Theorem (Causally Disconnected Patterns are Decomposable).** Let  $\Pi$  be an FDR planning task, and let  $P$  be a pattern that is not causally connected. Let  $P_1, P_2$  be a partition of  $P$  such that  $CG(\Pi)$  contains no arc between the two sets. Then  $h^{P_1}(s) + h^{P_2}(s) = h^P(s)$  for all states  $s$ .

→ If  $P_1$  and  $P_2$  don't influence each other at all, then their contributions to remaining cost are independent.

→ There is no point in including a causally disconnected pattern: Using the connected components instead requires less space and gives identical results.

**Note:** “Causally disconnected” is strictly stronger than “orthogonal”. (Basically, “causally disconnected” refers to both (i) and (ii) causal graph arcs, while “orthogonal” refers only to (ii) arcs. For details, see slide 59.)

## Why Are Causally Disconnected Patterns Decomposable?

**Reminder:** Let  $P$  be a pattern that is not causally connected. Let  $P_1, P_2$  be a partition of  $P$  such that  $CG(\Pi)$  contains no arc between the two sets. Then  $h^{P_1}(s) + h^{P_2}(s) = h^P(s)$  for all states  $s$ .

**Proof.** Any PDB heuristic  $h^P$  is equivalent to optimal planning in the syntactic projection  $\Pi|_P$  onto the pattern (cf. slide 18).

The FDR task  $\Pi|_P$  falls apart into two separate components with underlying variable subsets  $P_1$  and  $P_2$ .

As we showed in **Chapter 5**, if two parts of a planning task are completely separate in the causal graph, then optimal plans for the overall task can be obtained simply by sequencing optimal plans for each of the parts.

Hence, for any state  $s$ , we have  $h^P(s) = h^{P_1}(s) + h^{P_2}(s)$  as desired.

## ... and Australia Again



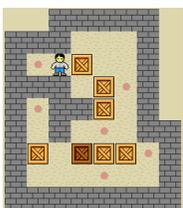
- Variables:  $at : \{Sy, Ad, Br, Pe, Ad\}$ ;  $v(x) : \{T, F\}$  for  $x \in \{Sy, Ad, Br, Pe, Ad\}$ .
- Actions:  $drive(x, y)$  where  $x, y$  have a road.
- Costs:  $Sy \leftrightarrow Br : 1, Sy \leftrightarrow Ad : 1.5, Ad \leftrightarrow Pe : 3.5, Ad \leftrightarrow Da : 4$ .
- Initial state:  $at = Sy, v(Sy) = T, v(x) = F$  for  $x \neq Sy$ .
- Goal:  $at = Sy, v(x) = T$  for all  $x$ .

**Question:** Is the pattern  $P = \{v(Br), v(Pe), v(Da)\}$  causally connected?

→ No. The causal graph  $CG(\Pi)$  arcs in this example are  $(at, v(x))$  and  $(v(x), at)$  for  $x \in \{Sy, Ad, Br, Pe, Ad\}$ . The sub-graph of  $CG(\Pi)$  induced by  $P$  has no arcs at all.

→ Therefore, by the theorem on causally disconnected patterns,  $h^P = h^{\{v(Br)\}} + h^{\{v(Pe)\}} + h^{\{v(Da)\}}$ . In other words, "Value = sum over the Fs", cf. slide 24.

## Questionnaire



- Variables, for boxes  $b_i$  and cells  $c_i$ :  $robotPos : \{c_i\}$ ,  $boxPos(x) : \{c_i\}$  for each  $b_i$ ,  $free(c_i) : \{T, F\}$  for each  $c_i$ .
- Actions: **move**( $c, c'$ ) where  $c, c'$  adjacent:  
pre  $robotPos = c, free(c') = T$ ; eff  $robotPos = c'$ .  
**push**( $b, c, c', c''$ ) where  $c, c', c''$  arranged in a line:  
pre  $robotPos = c, boxPos(b) = c'; free(c'') = T$   
eff  $robotPos = c', boxPos(b) = c'', free(c'') = F, free(c') = T$ .
- Goal:  $free(c) = F$  for the goal cells  $c$ .

### Question!

**What are the non-redundant patterns  $P$  in this Sokoban task?**

- (A):  $robotPos \in P$                       (B):  $boxPos(b) \in P$  for some  $b$   
(C):  $free(c) \in P$  for a goal cell  $c$     (D): Other (choose freely)

→ We definitely need (C), otherwise  $P$  is non-goal.

→ If we have (A) or (B), then all variables in  $P$  are causally relevant, and  $P$  is causally connected: Every cell is member of at least one line of 3 cells, so due to  $push(b, c, c', c'')$  we have causal graph (ii) arcs between every  $free(c)$  and  $robotPos$ /every  $boxPos(b)$ .

→ If we have neither (A) nor (B), consider the set  $C$  of cells  $c$  where  $free(c) \in P$ . Every connected component in  $C$  must contain a goal cell, else  $P$  has causally irrelevant variables. In fact,  $C$  must be connected, because otherwise  $P$  is not causally connected.

## How to Find a Pattern Collection?

### Pattern Selection as an Optimization Problem

- **Within:** A large set of candidate solutions.  
(= pattern collections whose summed-up PDB size is  $\leq$  size bound)
- **Find:** A best possible solution, or an approximation thereof.  
(= pattern collection yielding the most informative heuristic)

→ Number of pattern collections for an FDR task with variables  $V$ :  $2^{2^{|V|}}$ . We can't possibly hope to solve this optimally.

→ We try to find good solutions (pattern collections yielding informative heuristics) by **local search in the space of candidate solutions**.

→ Done by [Edelkamp (2006)] (evolutionary algorithm) and [Haslum et al. (2007)] (hill-climbing). We focus on the latter here.

## Pattern Selection as Hill-Climbing

```
function Hill-Climbing returns a solution
  node ← a node  $n$  with  $n.content = Start$ 
  loop do
    if  $TerminationCondition(n)$  then stop
     $N \leftarrow$  the set of all  $ChildNodes$  of  $n$ 
     $n \leftarrow$  an element of  $N$  minimizing  $H$  /* (random tie breaking) */
```

→ For use in pattern selection, which questions do we need to answer?

- I Start: What is the initial pattern collection?
- II TerminationCondition: When do we terminate?
- III ChildNodes: Which collections are neighbors of the current collection?
- IV  $H$ : How do we rank the quality of pattern collections?

## Pattern Selection as Hill-Climbing

```
function Hill-Climbing returns a solution
  node ← a node  $n$  with  $n.content = Start$ 
  loop do
    if  $TerminationCondition(n)$  then stop
     $N \leftarrow$  the set of all  $ChildNodes$  of  $n$ 
     $n \leftarrow$  an element of  $N$  minimizing  $H$  /* (random tie breaking) */
```

→ Overview:

- I Start: Start from **small patterns** of only a single variable each.
- II TerminationCondition: Stop when **collection quality can no longer be improved**.
- III ChildNodes: **Select a pattern and add one more variable to it**, unless the resulting PDB breaks the size bound.
- IV  $H$ : Use **sample states** to estimate pattern collection quality.

## (Our Simplistic Answers to) Questions (I) and (II)

### (I) What is the initial pattern collection?

The initial pattern collection is  $\{\{v\} \mid v \in V, G \text{ is defined for } v\}$ .

→ We start with **goal variables** because non-goal patterns are redundant.

### (II) When do we terminate?

We terminate as soon as the current pattern collection has no **ChildNodes** of better quality  $H$ .

(Note that the current pattern collection may have no better-quality successors simply because all successors break the size bound.)

→ We stop at the **first local minimum**. Can you guess the motivation? Our search in the space of pattern collections consumes a lot of runtime, and doing *several* iterations of Hill-Climbing wouldn't be cost-effective (often, even a single iteration isn't).

## Question (III), ChildNodes

### (III) Which collections are neighbors of the current collection?

The neighbors of  $\mathcal{C}$  are all pattern collections  $\mathcal{C}'$  where:

- i  $\mathcal{C}' = \mathcal{C} \cup \{P'\}$ .
- ii  $P' = P \cup \{v\}$  for some  $P \in \mathcal{C}$ , and  $P' \notin \mathcal{C}$ .
- iii All variables of  $P'$  are causally relevant in  $P'$ , and  $P'$  is causally connected.
- iv All PDBs in  $\mathcal{C} \cup \{P'\}$  can be represented within the size bound.

### Notes:

- (i) We add  $P'$  to  $\mathcal{C}$  *without* removing  $P$  from  $\mathcal{C}$ :  $P$  is not necessarily useless in  $\mathcal{C}'$  because it may be orthogonal with some patterns  $Q \in \mathcal{C}'$  that are not orthogonal with  $P'$  (cf. slide 35).
- (iii) We can easily select those  $v$  adding which to  $P$  preserves causal relevance and connectivity: Either  $v$  is a predecessor of some  $u \in P$  in the causal graph; or  $v$  is a successor of some  $u \in P$  in the causal graph, and  $G$  is defined for  $v$ .

## Question (IV), $H$ : What is a Good Pattern Collection?

Without a good ranking criterion, pattern collections are chosen blindly!

Given two pattern collections, which one is better?

- The one that “typically” gives the better lower bound, i.e., on the majority of states in the original state space.
- We cannot, of course, actually check this.
- Way out: **sample** the original state space: [Haslum *et al.* (2007)]

### (IV) How do we rank the quality of pattern collections?

- Generate  $M$  concrete states  $s_1, \dots, s_M$  through random walks from the concrete initial state.
- Given a pattern collection  $\mathcal{C}'$ , generated as a successor of collection  $\mathcal{C}$ , the **degree of improvement** is the number of sample states  $s_i$  for which  $h^{\mathcal{C}'}(s_i) > h^{\mathcal{C}}(s_i)$ .
- $H(\mathcal{C}') :=$  its degree of improvement.

## Summary

- **Pattern database (PDB) heuristics** are abstraction heuristics based on **projection** to a subset of variables: the **pattern**. For FDR tasks, they can easily be implemented via **syntactic projection** on the task representation.
- **Pattern databases** are **lookup tables** that store heuristic values, indexed by **perfect hash values** for projected states.
- Two patterns are **orthogonal** if no action affects variables from both. The heuristics are then **additive**, i.e., their sum is admissible (cf. **Chapter 7**).
- Given a **pattern collection**, the **canonical heuristic function** sums heuristic values from all orthogonal pattern subsets, and maximizes over these sums. This is the best one can do based on exploiting only orthogonality (but we will see more powerful methods in **Chapter 16**).
- A pattern makes sense only if it is **causally connected**, and all its variables are **causally relevant**. Both can be identified easily using the causal graph.
- One way to automatically find a good pattern collection is by **hill-climbing in the space of pattern collections**.

## Ok. But How to Compute $h^{\mathcal{C}'}(s)$ ?

What is the problem?

- We have PDBs for all patterns in  $\mathcal{C}$ , but *not* for the new pattern  $P' \in \mathcal{C}'$  (of the form  $P \cup \{v\}$  for some  $P \in \mathcal{C}$ ).
- We need to compute  $h^{\mathcal{C}'}(s)$  for all candidate successors  $\mathcal{C}'$  and all sample states  $s$ .
- We would rather not compute the complete PDB for every candidate new pattern  $P'$ .

But how to compute  $h^{P'}(s)$  effectively? There's a nice trick:

- $h^{P'}(s)$  is identical to the **optimal solution cost for  $s$  in the syntactic projection, i.e., the FDR task  $\Pi|_{P'}$** .
- We can use any optimal planning algorithm for this.
- In particular, we can **use  $A^*$  with  $h^P$  as the heuristic**.

## Historical Remarks

- PDB heuristics were originally introduced for the 15-puzzle [Culberson and Schaeffer (1998)] and for Rubic's Cube [Korf (1997)].
- The first use of PDBs in planning is due to [Edelkamp (2001)]. This spawned various follow-up works [Edelkamp (2002); Haslum *et al.* (2005); Helmert *et al.* (2007); Haslum *et al.* (2007)]. Much of this chapter is based on [Haslum *et al.* (2007)].
- Manually designed PDB heuristics are currently the state of the art admissible heuristics for several search problems (e.g., 15-puzzle & Rubic's Cube).
- Automatically designed PDB heuristics are also very competitive with other admissible heuristics for planning. A major obstacle is the runtime overhead for automatically selecting the pattern collection.

## A Technical Remark

→ “Causally disconnected” implies “orthogonal”:

- Say  $P$  is causally disconnected. For disjoint  $P_1, P_2 \subseteq P$  s.t.  $CG(\Pi)$  contains no arc between the two sets,  $P_1$  and  $P_2$  are orthogonal: otherwise, there would be a  $CG(\Pi)$  (ii) arc between them (cf. note on slide 28).

→ “Causally disconnected” can be strictly stronger than “orthogonal”:

- E.g., in “Logistics mal anders”,  $P_1 = \{\text{package, truck A}\}$  and  $P_2 = \{\text{truckB}\}$  are orthogonal, but not causally disconnected because truckB has a  $CG(\Pi)$  (i) arc to package (cf. note on slide 45).
- While being causally disconnected implies that  $h^{P_1}(s) + h^{P_2}(s) = h^P(s)$ , the same is *not* true if we require only orthogonality.

E.g., in “Logistics mal anders” as above, we have  $h^{P_1}(\text{LRR}) = 2$  via load/unload using truckB, and we have  $h^{P_2}(\text{LRR}) = 0$ .

However,  $h^{P_1 \cup P_2}(\text{LRR}) = 4$ : The package depends on truckB, so when we combine the two patterns, the previous abstract plan “load/unload using truckB” (without actually driving truckB) does not work anymore.

## Reading

- *Planning with Pattern Databases* [Edelkamp (2001)].

Available at:

<http://www.tzi.de/~edelkamp/publications/drafts/patternPlan.pdf>

**Content:** The first paper introducing pattern database heuristics to planning. Formulated in the STRIPS setting, where a PDB is defined as a subset of facts. Contains the corresponding version of orthogonality, and empirical results on the benchmarks at the time.

## Reading, ctd.

- *Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning* [Haslum et al. (2007)].

Available at:

<http://users.cecs.anu.edu.au/~patrik/publik/patterns3.pdf>

**Content:** State of the art method for automatically constructing and using collections of patterns, formulated in FDR (well, in STRIPS as well as its correspondence to “multi-valued state variables”). Main basis of this chapter. Introduces the canonical heuristic, and briefly describes the notions of dominance and causal relevance / connectedness that we elaborate here. Describes in a lot of detail the heuristic evaluation of pattern collection quality, that we only summarized briefly here. Empirical results in 15-puzzle, Sokoban, and Logistics.

## References I

Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Stefan Edelkamp. Planning with pattern databases. In A. Cesta and D. Borrajo, editors, *Proceedings of the 6th European Conference on Planning (ECP'01)*, pages 13–24. Springer-Verlag, 2001.

Stefan Edelkamp. Symbolic pattern databases in heuristic search planning. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, pages 274–283, Toulouse, France, April 2002. Morgan Kaufmann.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, pages 35–50, 2006.

Patrik Haslum, Blai Bonet, and Héctor Geffner. New admissible heuristics for domain-independent planning. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference of the American Association for Artificial Intelligence (AAAI'05)*, pages 1163–1168, Pittsburgh, Pennsylvania, USA, July 2005. AAAI Press.

## References II

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig.  
Domain-independent construction of pattern database heuristics for cost-optimal planning. In Adele Howe and Robert C. Holte, editors, *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, pages 1007–1012, Vancouver, BC, Canada, July 2007. AAAI Press.

Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Richard E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. In Benjamin J. Kuipers and Bonnie Webber, editors, *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI'97)*, pages 700–705, Portland, OR, July 1997. MIT Press.