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

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2018/2019

Thanks to Prof. Jörg Hoffmann for slide sources

Agenda

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).

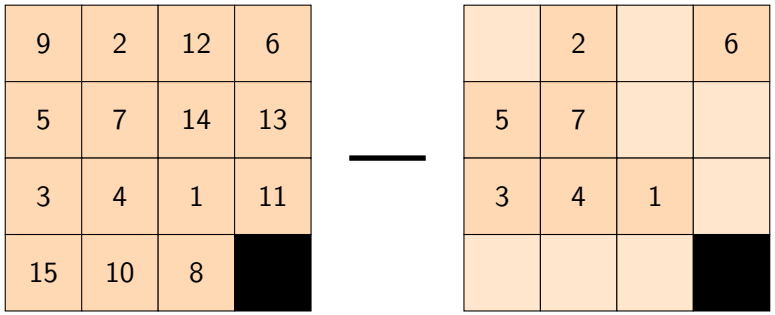
Motivation for Pattern Database Heuristics

→ Pattern databases are a concrete method for designing abstraction functions α, 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:

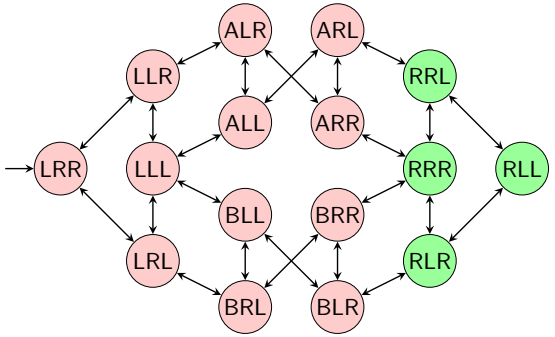
Definition (Projection, PDB Heuristic). *Let $\Pi = (V, A, c, I, G)$ be an FDR planning task with state space $\Theta_\Pi = (S, L, c, T, I, S^G)$, and let $P \subseteq V$. For a partial assignment φ to V, by $\varphi|_P$ we denote the restriction of φ to P. Let S^P be the set of variable assignments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

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

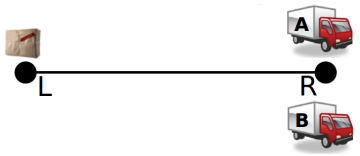
We refer to P as the pattern of π_P. The abstraction heuristic induced by π_P on Θ_Π is called a pattern database heuristic, short PDB heuristic. We write h^P as a short-hand for h^{π_P}, and we write Θ^P_Π or Θ^P as short-hands for $\Theta^{\pi_P}_\Pi$.

- 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



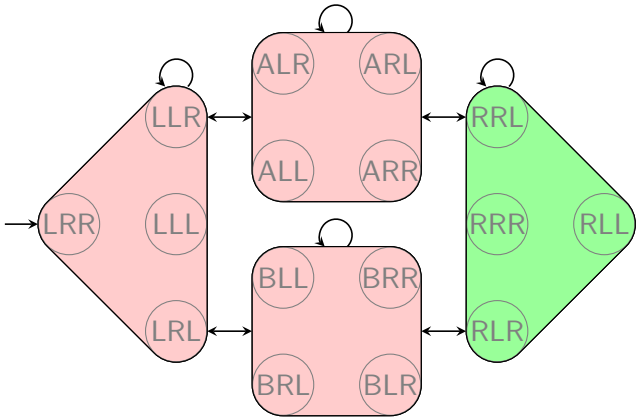
Logistics task with one package, two trucks, two locations:



- State variable package: $\{L, R, A, B\}$.
- State variable truck A: $\{L, R\}$.
- State variable truck B: $\{L, R\}$.

"Logistics mal anders": Projection 1

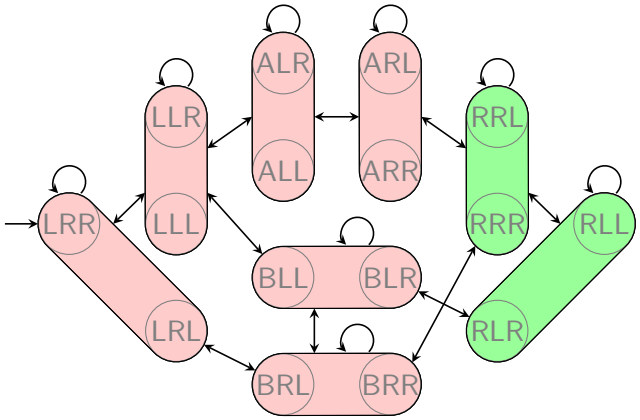
Abstraction induced by $\pi_{\{\text{package}\}}$:



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

"Logistics mal anders": Projection 2

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



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

Larger Pattern = Refinement

Reminder: → Chapter 11

Say α is an abstraction of Θ, and α' is an abstraction of Θ^α. Then α is called a refinement of $\alpha' \circ \alpha$.

Proposition (Larger Patterns are Refinements). *Let Π be an FDR planning task, and let $P' \subseteq P \subseteq V$. Then π_P is a refinement of $\pi_{P'}$.*

Proof. $\pi_{P'}$ can be viewed as an abstraction of Θ^P, i.e., in the above set $\Theta^\alpha := \Theta^P$, and set α' to be the projection of P onto P'.

Corollary (Larger Patterns Yield Better Heuristics). *Let Π be an FDR planning task, and let $P' \subseteq P \subseteq V$. Then h^P dominates $h^{P'}$, i.e., $h^{P'} \leq h^P$.* (From previous proposition and Chapter 11)

- And, of course, $h^V =$
- Pattern size controls the trade-off between accuracy and computational cost.

Questionnaire

Remember (Chapter 11): Optimal abstract plans are not necessarily just abstractions of optimal real plans: Given an optimal real plan $\vec{a}$, skipping the non-α-affecting actions does not necessarily result in an optimal abstract plan. Spurious transitions may lead to "shortcuts" with no real correspondence.

(E.g., if we do not distinguish between the initial state and a state where we have a teleport machine.)

Question!

Can this happen for a projection of "Logistics mal anders"?

I Give You Pattern, You Give Me Database!

Asssume: 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: It's Not That Easy

Let Π 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 Θ'.

So, what's the issue?

- Θ' is defined through a function on Θ:
  - Each concrete transition induces an abstract transition, each concrete goal state induces an abstract goal state.
- In principle, we can we compute Θ' by iterating over all transitions/goal states of Θ. BUT:

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

(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 Π 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 $\mathit{eff}_{a|_P} := (\mathit{eff}_a)|_P$.*

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

Theorem (Syntactic Projection is Equivalent to Projection). *Let Π 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.

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

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

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

Computing Pattern Databases

def compute-PDB(Π, P):
  $\Pi' := \Pi|_P$.
  Compute $\Theta' := \Theta_{\Pi'}$ by a complete forward search (e.g., breadth-first).
  In the explicit graph Θ', 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 Θ'
  PDB := a table containing all remaining costs in Θ'
  return *PDB*

$\rightarrow$ 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.

  $\rightarrow$ We can throw away the abstract state space Θ' once the PDB is computed.

  $\rightarrow$ 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 $\{0, \ldots, |S'| - 1\}$.

  $\rightarrow$ 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: Here's How To

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

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

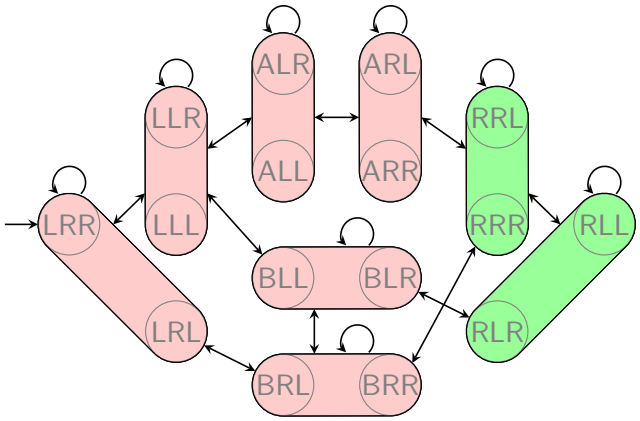
Looking Up a Pattern Database Heuristic Value

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 $O(\|\Pi\|)$ per state.

(II) Lookup Step: "Logistics mal anders"

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



(II) Lookup Step: "Logistics mal anders", ctd.

Pattern variables and domains:

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

$\rightarrow N_1 =$

$\rightarrow N_2 =$

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

$\rightarrow$ **Pattern database:**

abstract state	LL	RL	AL	BL	LR	RR	AR	BR
index								
value								

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?

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: $h^{P_1} + h^{P_2}$ . . . !

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

Let α_1 and α_2 be abstractions of Θ. We say that α_1 and α_2 are orthogonal if no label of Θ affects both α_1 and α_2.

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

Orthogonal Patterns

Lemma (Orthogonal Patterns). *Let P_1 and P_2 be patterns for an FDR planning task Π. Then π_{P_1} and π_{P_2} are orthogonal if and only if there exists no action a in Π 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 π_{P_1} and π_{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?

- If $P_1 \cap P_2 = \emptyset$, are P_1 and P_2 necessarily orthogonal?

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?

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

Questionnaire

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 $\{P_1, \ldots, P_k\}$ be an orthogonal pattern collection for an FDR planning task Π. 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 $\pi_{P_1}, \ldots, \pi_{P_k}$ 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 $P = P_1 \cup \cdots \cup P_k$ does NOT imply $\sum_{i=1}^{k} h^{P_i} = h^P$. (Details see slide 59.)

Large Patterns Dominate Orthogonal Smaller Ones

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

How to Exploit Orthogonal Patterns?

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

1. Build the compatibility graph for $\mathcal{C}$.
   - The vertices are the patterns $P \in \mathcal{C}$.
   - Arcs between pairs of orthogonal patterns.
2. 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 $\mathcal{C} = \{P_1, \ldots, P_4\}$ 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.

$\rightarrow$ Maximal cliques in the compatibility graph for $\mathcal{C}$?

The Canonical Heuristic Function

Definition (Canonical Heuristic). *Let Π be an FDR planning task, let $\mathcal{C}$ be a pattern collection for Π, and let $cliques(\mathcal{C})$ 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*

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

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

Same Abstract Example

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

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 $h^{P_2}(I)$?
- Say $\mathcal{C} = \{P_1, P_2\}$. What is the value of $h^{\mathcal{C}}(I)$?
- Say $P_1' = \{v(Br), at\}$. What is the value of $h^{P_1'}(I)$?
- Say $\mathcal{C} = \{P_1', P_2\}$. What is the value of $h^{\mathcal{C}}(I)$?

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)

ps. On the Power of the Canonical Heuristic

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:
  1. **Non-Goal Patterns.**
  2. **Causally Irrelevant Variables.**
  3. **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 Π 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).

→ 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?

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 Π 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 Π. 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

Causally Irrelevant Variables are Useless

Theorem (Causally Irrelevant Variables are Useless). *Let $P \subseteq V$ be a pattern for an FDR planning task Π, 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 Θ^P.

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

Redundant Case III: Causally Disconnected Patterns

Definition (Causally Connected Patterns). *Let Π be an FDR planning task, and let P be a pattern for Π. 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.*

Causally Disconnected Patterns are Decomposable

Theorem (Causally Disconnected Patterns are Decomposable). *Let Π 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.)

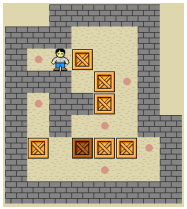
Why Are Causally Disconnected Patterns Decomposable?

. . . 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?

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)

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)

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

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

$\rightarrow$ 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 ← the set of all ChildNodes of n
    n ← an element of N minimizing H /* (random tie breaking) */

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

Introduction PDB Basics Implementation Orthogonal PDBs Redundant PDBs Pattern Selection Conclusion References

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 ← the set of all ChildNodes of n
    n ← an element of N minimizing H /* (random tie breaking) */

→ Overview:

Álvaro Torralba, Cosmina Croitoru AI Planning Chapter 12: Pattern Database Heuristics 51/63

Introduction PDB Basics Implementation Orthogonal PDBs Redundant PDBs Pattern Selection Conclusion References

(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?

Introduction PDB Basics Implementation Orthogonal PDBs Redundant PDBs Pattern Selection Conclusion References

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

(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 $s_1, \ldots, s_M$ 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.

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.

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.

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.