Planning and Optimization
D4. Pattern Databases: Introduction

Malte Helmert and Gabriele Röger
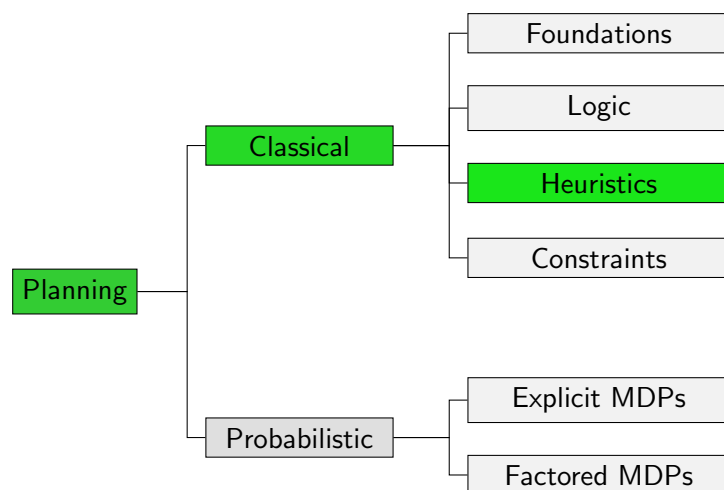
Universität Basel

D4.1 Projections and Pattern Database Heuristics

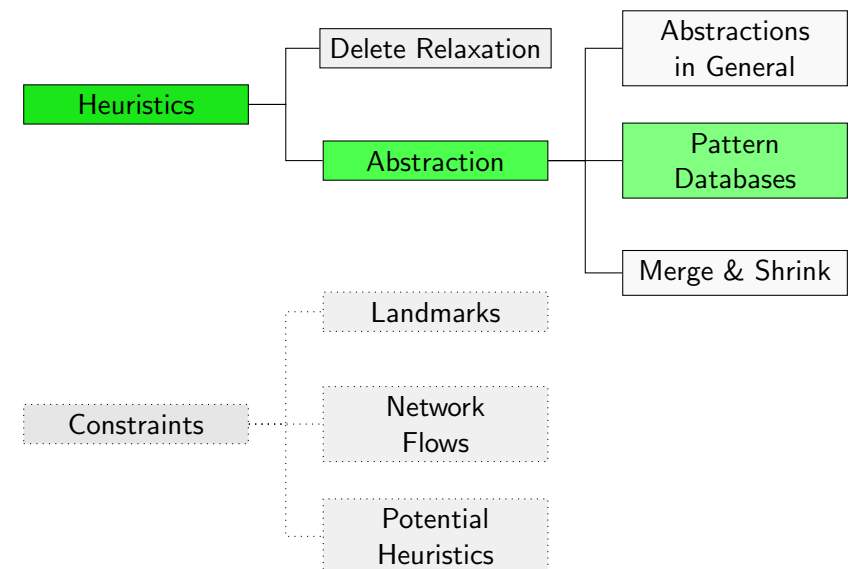## D4.2 Implementing PDBs: Precomputation

D4.3 Implementing PDBs: Lookup

D4.4 Summary

Content of this Course

Content of this Course: Heuristics

D4.1 Projections and Pattern Database Heuristics

Pattern Database Heuristics

▶ The most commonly used abstraction heuristics in search and planning are pattern database (PDB) heuristics.

▶ PDB heuristics were originally introduced for the 15-puzzle (Culberson & Schaeffer, 1996) and for Rubik's cube (Korf, 1997).

▶ The first use for domain-independent planning is due to Edelkamp (2001).

▶ Since then, much research has focused on the theoretical properties of pattern databases, how to use pattern databases more effectively, how to find good patterns, etc.

▶ Pattern databases are a very active research area both in planning and in (domain-specific) heuristic search.

▶ For many search problems, pattern databases are the most effective admissible heuristics currently known.

Pattern Database Heuristics Informally

Pattern Databases: Informally

A pattern database heuristic for a planning task is an abstraction heuristic where

▶ some aspects of the task are represented in the abstraction with perfect precision, while

▶ all other aspects of the task are not represented at all.

This is achieved by projecting the task onto the variables that describe the aspects that are represented.

Example (15-Puzzle)

▶ Choose a subset T of tiles (the pattern).

▶ Faithfully represent the locations of T in the abstraction.

▶ Assume that all other tiles and the blank can be anywhere in the abstraction.

Projections

Formally, pattern database heuristics are abstraction heuristics induced by a particular class of abstractions called projections.

Definition (Projection)

Let Π be an FDR planning task with variables V and states S. Let $P \subseteq V$, and let S' be the set of states over P.

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

We call P the pattern of the projection π_P.

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

Pattern Database Heuristics

Abstraction heuristics based on projections are called pattern database (PDB) heuristics.

> **Definition (Pattern Database Heuristic)**
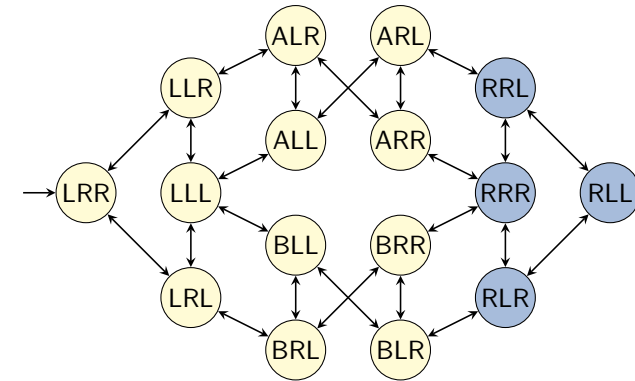> The abstraction heuristic induced by π_P is called
> a pattern database heuristic or PDB heuristic.
> We write h^P as a shorthand for h^{π_P}.

Why are they called pattern database heuristics?

▶ Heuristic values for PDB heuristics are traditionally stored in a 1-dimensional table (array) called a pattern database (PDB). Hence the name "PDB heuristic".

▶ The word pattern database alludes to endgame databases for 2-player games (in particular chess and checkers).

Example: Transition System
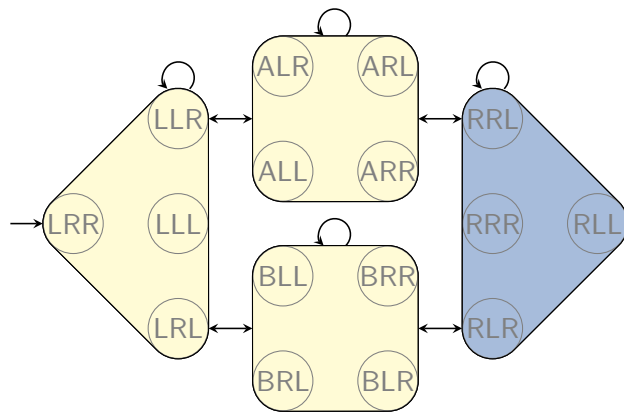

Logistics problem with one package, two trucks, two locations:

▶ state variable package: $\{L, R, A, B\}$

▶ state variable truck A: $\{L, R\}$

▶ state variable truck B: $\{L, R\}$

Example: Projection (1)

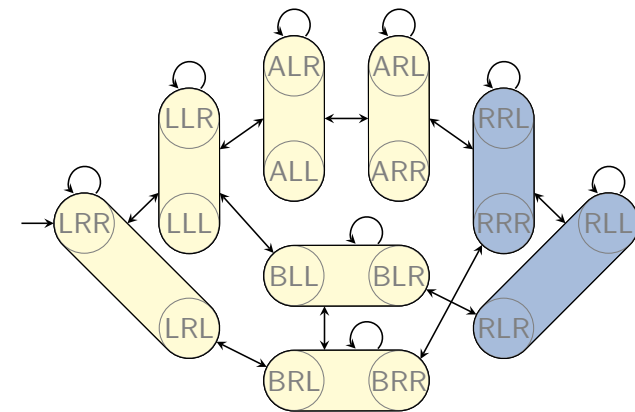Abstraction induced by $\pi_{\{package\}}$:

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

Example: Projection (2)

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

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

Pattern Databases: Chapter Overview

In the following, we will discuss:

- ▶ how to implement PDB heuristics
  ↝ this chapter
- ▶ how to effectively make use of multiple PDB heuristics
  ↝ Chapter D5
- ▶ how to find good patterns for PDB heuristics
  ↝ Chapter D6

D4.2 Implementing PDBs: Precomputation

Pattern Database Implementation

Assume we are given a pattern P for a planning task Π.
How do we implement h^P?

1. In a precomputation step, we compute a graph representation for the abstraction $\mathcal{T}(\Pi)^{\pi_P}$ and compute the abstract goal distance for each abstract state.

2. During search, we use the precomputed abstract goal distances in a lookup step.

Precomputation Step

Let Π be a planning task and P a pattern.
Let $\mathcal{T} = \mathcal{T}(\Pi)$ and $\mathcal{T}' = \mathcal{T}^{\pi_P}$.

- ▶ We want to compute a graph representation of $\mathcal{T}'$.
- ▶ $\mathcal{T}'$ is defined through an abstraction of $\mathcal{T}$.
  - ▶ For example, each concrete transition induces an abstract transition.
- ▶ However, we cannot compute $\mathcal{T}'$ by iterating over all transitions of $\mathcal{T}$.
  - ▶ This would take time $\Omega(\|\mathcal{T}\|)$.
  - ▶ This is prohibitively long (or else we could solve the task using uniform-cost search or similar techniques).
- ▶ Hence, we need a way of computing $\mathcal{T}'$ in time which is polynomial only in $\|\Pi\|$ and $\|\mathcal{T}'\|$.

Syntactic Projections

Definition (Syntactic Projection)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be an FDR planning task,
and let $P \subseteq V$ be a subset of its variables.
The syntactic projection $\Pi|_P$ of Π to P is the FDR planning task
$\langle P, I|_P, \{o|_P \mid o \in O\}, \gamma|_P \rangle$, where

- $\varphi|_P$ for formula φ is defined as the formula obtained from φ by replacing all atoms $(v = d)$ with $v \notin P$ by $\top$, and

- $o|_P$ for operator o is defined by replacing all formulas φ occurring in the precondition or effect conditions of o with $\varphi|_P$ and all atomic effects $(v := d)$ with $v \notin P$ with the empty effect $\top$.

Put simply, $\Pi|_P$ throws away all information not pertaining to variables in P.

Equivalence Theorem for Syntactic Projections

Theorem (Syntactic Projections vs. Projections)

Let Π be a SAS^+ task, and let P be a pattern for Π.
Then $\mathcal{T}(\Pi|_P) \overset{G}{\sim} \mathcal{T}(\Pi)^{\pi_P}$.

Proof.
⇝ exercises □

PDB Computation

Using the equivalence theorem, we can compute pattern databases
for SAS^+ tasks Π and patterns P:

Computing Pattern Databases

def compute-PDB(Π, P):
    Compute $\Pi' := \Pi|_P$.
    Compute $\mathcal{T}' := \mathcal{T}(\Pi')$.
    Perform a backward uniform-cost search from the goal
        states of $\mathcal{T}'$ to compute all abstract goal distances.
    $PDB :=$ a table containing all goal distances in $\mathcal{T}'$
    return PDB

The algorithm runs in polynomial time and space
in terms of $\|\Pi\| + |PDB|$.

Generalizations of the Equivalence Theorem

- The restriction to SAS^+ tasks is necessary.
- We can slightly generalize the result if we allow general negation-free formulas, but still forbid conditional effects.
  - In that case, the weighted graph of $\mathcal{T}(\Pi)^{\pi_P}$ is isomorphic to a subgraph of the weighted graph of $\mathcal{T}(\Pi|_P)$.
  - This means that we can use $\mathcal{T}(\Pi|_P)$ to derive an admissible estimate of h^P.
- With negations in conditions or with conditional effects, not even this weaker result holds.

Going Beyond SAS$^+$ Tasks

- ▶ Most practical implementations of PDB heuristics are limited to SAS$^+$ tasks (or modest generalizations).
- ▶ One way to avoid the issues with general FDR tasks is to convert them to equivalent SAS$^+$ tasks.
- ▶ However, most direct conversions can exponentially increase the task size in the worst case.
- ⤳ We will only consider SAS$^+$ tasks in the chapters dealing with pattern databases.

D4.3 Implementing PDBs: Lookup

Lookup Step: Overview

- ▶ During search, the PDB is the only piece of information necessary to represent h^P. (It is not necessary to store the abstract transition system itself at this point.)
- ▶ Hence, the space requirements for PDBs during search are linear in the number of abstract states S': there is one table entry for each abstract state.
- ▶ During search, $h^P(s)$ is computed by mapping $\pi_P(s)$ to a natural number in the range $\{0, \ldots, |S'| - 1\}$ using a perfect hash function, then looking up the table entry for this number.

Lookup Step: Algorithm

Let $P = \{v_1, \ldots, v_k\}$ be the pattern.

- ▶ We assume that all variable domains are natural numbers counted from 0, i.e., $\mathrm{dom}(v) = \{0, 1, \ldots, |\mathrm{dom}(v)| - 1\}$.
- ▶ For all $i \in \{1, \ldots, k\}$, we precompute $N_i := \prod_{j=1}^{i-1} |\mathrm{dom}(v_j)|$.
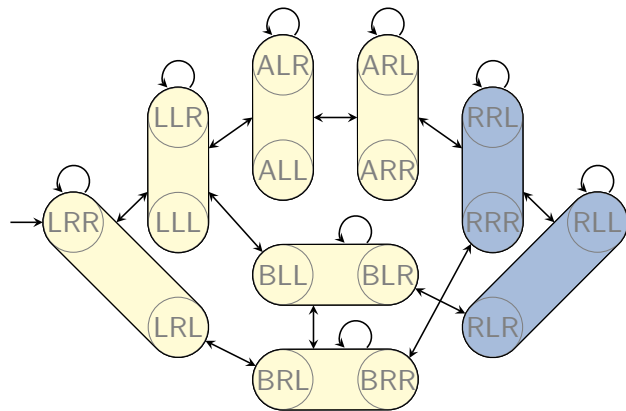
Then we can look up heuristic values as follows:

Computing Pattern Database Heuristics

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

- ▶ This is a very fast operation: it can be performed in $O(k)$.
- ▶ For comparison, most relaxation heuristics need time $O(\|\Pi\|)$ per state.

Lookup Step: Example (1)

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

Lookup Step: Example (2)

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

$\rightsquigarrow N_1 = \prod_{j=1}^{0} |dom(v_j)| = 1$, $N_2 = \prod_{j=1}^{1} |dom(v_j)| = 4$

$\rightsquigarrow index(s) = 1 \cdot s(package) + 4 \cdot s(truck\ A)$

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

D4.4 Summary

Summary

- Pattern database (PDB) heuristics are abstraction heuristics based on projection to a subset of variables.
- For SAS$^+$ tasks, they can easily be implemented via syntactic projections of the task representation.
- PDBs are lookup tables that store heuristic values, indexed by perfect hash values for projected states.
- PDB values can be looked up very fast, in time $O(k)$ for a projection to k variables.