Principles of AI Planning

11. Planning as search: pattern database heuristics

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel and Robert Mattmüller

December 7th, 2016

1 Pattern databases heuristics

- Projections and pattern database heuristics
- Examples
- Overview

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 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.

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 induced abstractions of a particular class of homomorphisms called projections.

Definition (projections)

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

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

We call P the pattern of the projection π_P.

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

Pattern database heuristics

Abstraction heuristics for projections are called pattern database (PDB) heuristics.

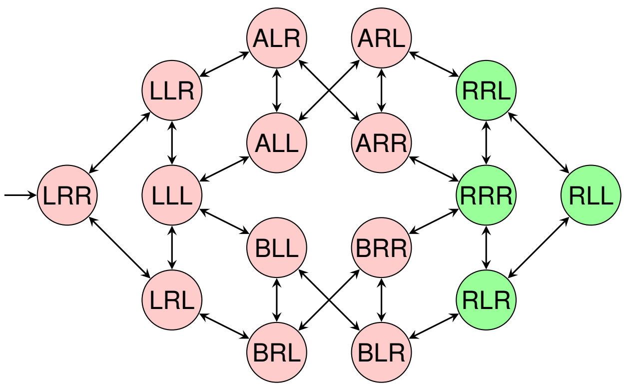
Definition (pattern database heuristic)

The abstraction heuristic induced by π_P is called a pattern database heuristic or PDB heuristic.
We write h^P as a short-hand 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".

Example: transition system

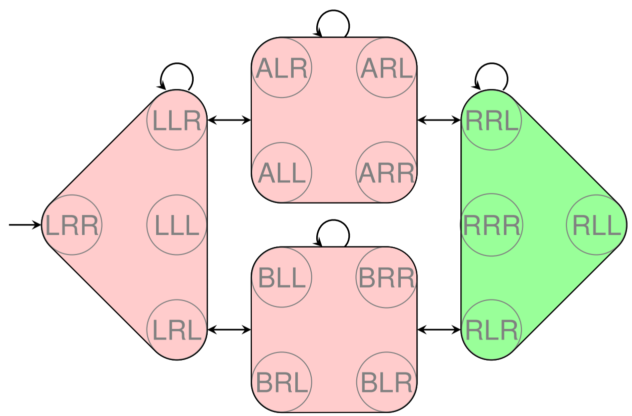


Logistics problem with one package, two trucks, two locations:

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

Example: projection

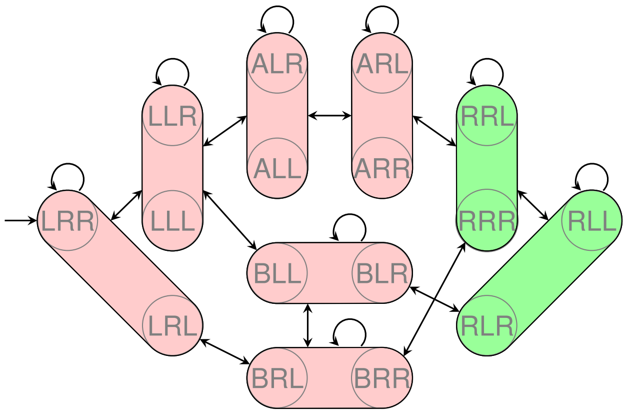
Abstraction induced by $\pi_{\{\text{package}\}}$:



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

Example: projection (2)

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



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

Chapter overview

In the rest of this chapter, we will discuss:

- how to implement PDB heuristics
- how to effectively make use of multiple PDB heuristics
- how to find good patterns for PDB heuristics

2 Implementing pattern database heuristics

- Precomputation step
- Lookup step

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 a homomorphism 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 large (or else we could solve the task using breadth-first 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.

Trivially inapplicable operators

Definition (trivially inapplicable operator)

An operator $\langle \chi, e \rangle$ of a SAS$^+$ task is called trivially inapplicable if

- χ contains the atoms $(v = d)$ and $(v = d')$ for some variable v and values $d \neq d'$, or
- e contains the effects $(v := d)$ and $(v := d')$ for some variable v and values $d \neq d'$.

Notes:

- Trivially inapplicable operators are never applicable and can thus be safely omitted from the task.
- Trivially inapplicable operators can be detected in linear time.

Trivially unsolvable SAS$^+$ tasks

Definition (trivially unsolvable SAS$^+$ tasks)

A SAS$^+$ task $\Pi = \langle V, I, O, \gamma \rangle$ is called trivially unsolvable if γ contains the atoms $(v = d)$ and $(v = d')$ for some variable v and values $d \neq d'$.

Notes:

- Trivially unsolvable SAS$^+$ tasks have no goal states, and are hence unsolvable.
- Trivially unsolvable SAS$^+$ tasks can be detected in linear time.

Equivalence theorem for syntactic projections

Theorem (syntactic projections vs. projections)

Let Π be a SAS⁺ task that is not trivially unsolvable and has no trivially inapplicable operators, and let P be a pattern for Π.

Then $\mathcal{T}(\Pi|_P) \overset{G}{\sim} \mathcal{T}(\Pi)^{\pi_P}$.

Proof.

$\rightsquigarrow$ exercises

PDB computation

Using the equivalence theorem, we can compute pattern databases for (not trivially unsolvable) SAS⁺ tasks Π and patterns P:

Computing pattern databases

def compute-PDB(Π, P):
  Remove trivially inapplicable operators from Π.
  Compute $\Pi' := \Pi|_P$.
  Compute $\mathcal{T}' := \mathcal{T}(\Pi')$.
  Perform a backward breadth-first 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 restrictions to SAS⁺ tasks and to tasks without trivially inapplicable operators are necessary.
- We can slightly generalize the result if we allow general negation-free formulas, but still forbid conditional effects.
  - In that case, the unlabeled graph of $\mathcal{T}(\Pi)^{\pi_P}$ is isomorphic to a subgraph of the unlabeled 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 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.

$\rightsquigarrow$ We will only consider SAS⁺ tasks in this chapter.

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 $\{0,\dots,|S'|-1\}$ using a perfect hash function, then looking up the table entry for that number.

Lookup step: algorithm

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

- We assume that all variable domains are natural numbers counted from 0, i. e., $\mathscr{D}_v = \{0,1,\dots,|\mathscr{D}_v|-1\}$.
- For all $i \in \{1,\dots,k\}$, we precompute $N_i := \prod_{j=1}^{i-1} |\mathscr{D}(v_j)|$.

Then we can look up heuristic values as follows:

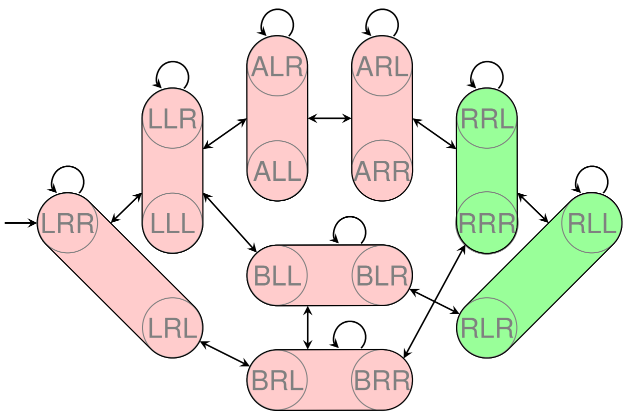
Computing pattern database heuristics

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

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



Lookup step: example (ctd.)

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

$\rightsquigarrow$ $N_1 = \prod_{j=1}^{0} |\mathscr{D}_{v_j}| = 1$, $N_2 = \prod_{j=1}^{1} |\mathscr{D}_{v_j}| = 4$

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

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

3 Additive patterns for planning tasks

- The additivity criterion & the canonical heuristic function
- Algebraic simplification & dominance pruning

Pattern collections

- The space requirements for a pattern database grow exponentially with the number of state variables in the pattern.
- This places severe limits on the usefulness of single PDB heuristics h^P for larger planning task.
- To overcome this limitation, planners using pattern databases work with collections of multiple patterns.
- When using two patterns P_1 and P_2, it is always possible to use the maximum of h^{P_1} and h^{P_2} as an admissible and consistent heuristic estimate.
- However, when possible, it is much preferable to use the sum of h^{P_1} and h^{P_2} as a heuristic estimate, since $h^{P_1} + h^{P_2} \geq \max\{h^{P_1}, h^{P_2}\}$.

Criterion for additive patterns

Theorem (additive pattern sets)

Let $P_1, \dots, P_k$ be patterns for an FDR planning task Π.

If there exists no operator that has an effect on a variable $v_i \in P_i$ and on a variable $v_j \in P_j$ for some $i \neq j$, then $\sum_{i=1}^{k} h^{P_i}$ is an admissible and consistent heuristic for Π.

Proof.

If there exists no such operator, then no label of $\mathscr{T}(\Pi)$ affects both $\mathscr{T}(\Pi)^{\pi_{P_i}}$ and $\mathscr{T}(\Pi)^{\pi_{P_j}}$ for $i \neq j$. By the theorem on affecting transition labels, this means that any two projections π_{P_i} and π_{P_j} are orthogonal. The claim follows with the theorem on additivity for orthogonal abstraction mappings.

A pattern set $\{P_1, \dots, P_k\}$ which satisfies the criterion of the theorem is called an additive pattern set or additive set.

Finding additive pattern sets

The theorem on additive pattern sets gives us a simple criterion to decide which pattern heuristics can be admissibly added.

Given a pattern collection $\mathscr{C}$ (i. e., a set of patterns), we can use this information as follows:

1. Build the compatibility graph for $\mathscr{C}$.
   - Vertices correspond to patterns $P \in \mathscr{C}$.
   - There is an edge between two vertices iff no operator affects both incident patterns.
2. Compute all maximal cliques of the graph. These correspond to maximal additive subsets of $\mathscr{C}$.
   - Computing large cliques is an NP-hard problem, and a graph can have exponentially many maximal cliques.
   - However, there are output-polynomial algorithms for finding all maximal cliques (Tomita, Tanaka & Takahashi, 2004) which have led to good results in practice.

The canonical heuristic function

Definition (canonical heuristic function)

Let Π be an FDR planning task, and let $\mathscr{C}$ be a pattern collection for Π.

The canonical heuristic $h^{\mathscr{C}}$ for pattern collection $\mathscr{C}$ is defined as

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

where $\mathrm{cliques}(\mathscr{C})$ is the set of all maximal cliques in the compatibility graph for $\mathscr{C}$.

For all choices of $\mathscr{C}$, heuristic $h^{\mathscr{C}}$ is admissible and consistent.

How good is the canonical heuristic function?

- The canonical heuristic function is the best possible admissible heuristic we can derive from $\mathscr{C}$ using our additivity criterion.
- In theory, even better heuristic estimates can be obtained from projection heuristics using a more general additivity criterion based on an idea called cost partitioning.
  - Optimal polynomial cost partitioning algorithms exist (Katz & Domshlak, 2008a).

Canonical heuristic function: example

Example

Consider a planning task with state variables $V = \{v_1, v_2, v_3\}$ and the pattern collection $\mathscr{C} = \{P_1, \ldots, P_4\}$ with $P_1 = \{v_1, v_2\}$, $P_2 = \{v_1\}$, $P_3 = \{v_2\}$ and $P_4 = \{v_3\}$.

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

What are the maximal cliques in the compatibility graph for $\mathscr{C}$?

Answer: $\{P_1\}$, $\{P_2, P_3\}$, $\{P_3, P_4\}$

What is the canonical heuristic function $h^{\mathscr{C}}$?

Answer: $h^{\mathscr{C}} = \max\{h^{P_1}, h^{P_2} + h^{P_3}, h^{P_3} + h^{P_4}\}$
$= \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\}$

Computing $h^{\mathscr{C}}$ efficiently: motivation

Consider $h^{\mathscr{C}} = \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\}$.

- We need to evaluate this expression for every search node.
- It is thus worth to spend some effort in precomputations to make the evaluation more efficient.

A naive implementation requires 8 atomic operations:

- 4 heuristic lookups (for $h^{\{v_1,v_2\}}$, $h^{\{v_1\}}$, $h^{\{v_2\}}$ and $h^{\{v_3\}}$),
- 2 binary summations and
- 2 binary maximizations

Can we do better than that?

Algebraic simplifications

One possible simplification is to use algebraic identities to reduce the number of operations:

$$\begin{aligned} &\max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \\ = &\max\{h^{\{v_1,v_2\}}, h^{\{v_2\}} + \max\{h^{\{v_1\}}, h^{\{v_3\}}\}\} \end{aligned}$$

$\rightsquigarrow$ reduces number of operations from 8 to 7

Is there anything else we can do?

Dominated sum theorem

Theorem (dominated sum)

Let $\{P_1, \dots, P_k\}$ be an additive pattern set 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.

Let $\mathcal{T}'$ be the transition system induced by π_P, and for all $i \in \{1, \dots, k\}$, let $\mathcal{T}_i'$ be the transition system induced by π_{P_i}. Because $P_i \subseteq P$, we can write each projection π_{P_i} as a projection onto P followed by a projection onto P_i: $\pi_{P_i} = \pi'_{P_i} \circ \pi_P$. Hence, each $\mathcal{T}_i'$ is a coarsening of $\mathcal{T}'$, and we can consider the h^{P_i} as abstraction heuristics on $\mathcal{T}'$, where they are also additive.

Dominated sum theorem (ctd.)

Proof (ctd.)

We get:

$$\begin{aligned} \sum_{i=1}^k h^{P_i}(s) &= \sum_{i=1}^k h^*_{\mathcal{T}_i'}(\pi_{P_i}(s)) = \sum_{i=1}^k h^*_{\mathcal{T}_i'}(\pi'_{P_i}(\pi_P(s))) \\ &\overset{(1)}{=} \sum_{i=1}^k h^*_{\mathcal{T}_i'}(\pi'_{P_i}(s')) \overset{(2)}{=} \sum_{i=1}^k h^{P_i}(s') \\ &\overset{(3)}{\leq} h^*_{\mathcal{T}'}(s') = h^*_{\mathcal{T}'}(\pi_P(s)) = h^P(s) \end{aligned}$$

where (1) holds for $s' := \pi_P(s)$, (2) holds because we can consider the h^{P_i} as abstraction heuristics on $\mathcal{T}'$, and (3) holds because of the additivity criterion. □

Dominated sum corollary

Corollary (dominated sum)

Let $\{P_1, \dots, P_n\}$ and $\{Q_1, \dots, Q_m\}$ be additive pattern sets of an FDR planning task such that each pattern P_i is a subset of some pattern Q_j (not necessarily proper). Then $\sum_{i=1}^n h^{P_i} \leq \sum_{j=1}^m h^{Q_j}$.

Proof.

$$\sum_{i=1}^n h^{P_i} \overset{(1)}{\leq} \sum_{j=1}^m \sum_{P_i \subseteq Q_j} h^{P_i} \overset{(2)}{\leq} \sum_{j=1}^m h^{Q_j},$$

where (1) holds because each P_i is contained in some Q_j and (2) follows from the dominated sum theorem.

Dominance pruning

- We can use the dominated sum corollary to simplify the representation of $h^{\mathscr{C}}$: sums that are dominated by other sums can be pruned.
- The dominance test can be performed in polynomial time.

Example

$$\begin{aligned} & \max\{h^{\{v_1,v_2\}}, h^{\{v_1\}} + h^{\{v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \\ = & \max\{h^{\{v_1,v_2\}}, h^{\{v_2\}} + h^{\{v_3\}}\} \end{aligned}$$

$\rightsquigarrow$ reduces number of operations from 8 to 5

4 Pattern selection

- Pattern selection as local search
- Search space
- Estimating heuristic quality

Pattern selection as an optimization problem

Only one question remains to be answered now in order to apply PDBs to planning tasks in practice:

How do we automatically find a good pattern collection?

The idea

Pattern selection can be cast as an optimization problem:

- Given: a set of candidate solutions (= pattern collections which fit into a given memory limit)
- Find: a best possible solution, or an approximation (= pattern collection with high heuristic quality)

Pattern selection as local search

How to solve this optimization problem?

- For problems of interesting size, we cannot hope to find (and prove) a globally optimal pattern collection.
  - Question: How many candidates are there?
- Instead, we try to find good solutions by local search.

Two approaches from the literature:

- Edelkamp (2007): using evolutionary algorithm
- Haslum et al. (2007): using hill-climbing

$\rightsquigarrow$ we present the main ideas of the second approach here

Pattern selection as hill-climbing

Reminder: Hill-climbing

σ := make-root-node(init())
forever:
  if is-goal(state(σ)):
    return extract-solution(σ)
  Σ' := { make-node(σ, o, s) | $\langle o, s \rangle \in$ succ(state(σ)) }
  σ := an element of Σ' minimizing h (random tie breaking)

Four questions to answer to use this for pattern selection:

1. init: What is the initial pattern collection?
2. is-goal: When do we terminate?
3. succ: Which collections are neighbours of the current collection?
4. h: How do we rank the quality of pattern collections?

Search space

We first discuss the search space (init, is-goal, succ).

The basic idea is that we

- start from small patterns of only a single variable each,
- grow them by adding slightly larger patterns, and
- stop when heuristic quality no longer improves.

To motivate the precise definition of our search space, we need a little more theory.

Initial pattern collection

Theorem (non-goal patterns are trivial)

Let Π be a SAS$^+$ planning task that is not trivially unsolvable, and let P be a pattern for Π such that no variable in P is mentioned in the goal formula of Π.
Then $h^P(s) = 0$ for all states s.

Proof.

All states in the abstraction are goal states.

This motivates our first answer:

1. What is the initial pattern collection?

The initial pattern collection is
$\{\{v\} \mid v$ is a state variable mentioned in the goal formula$\}$.

Termination criterion

Our second question has a very simple answer:

2. When do we terminate?

We terminate as soon as the current pattern collection has no successors of better quality.

Note that this also covers the case where there are no successors at all because further growth of the current pattern collection would exceed a memory limit.

Search neighborhood: idea

Our search neighbourhood is defined through incremental growth of the current pattern collection.

A successor is obtained by

- starting from the current pattern collection $\mathscr{C}$,
- choosing one of its patterns $P \in \mathscr{C}$ (without removing it from $\mathscr{C}$!),
- generating a new pattern by extending P with a single variable ($P' = P \cup \{v\}$), and
- adding P' to $\mathscr{C}$ to form the new pattern collection $\mathscr{C}'$

However, not all such collections $\mathscr{C}'$ are useful.

Causal graphs

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Definition (causal graph)

Let $\Pi = \langle V, I, O, \gamma\rangle$ be an FDR planning task.
The causal graph of Π, $CG(\Pi)$, is the directed graph with vertex set V and an arc from $u \in V$ to $v \in V$ iff $u \neq v$ and there exists an operator $o \in O$ such that:

- u appears anywhere in o (in precondition, effect conditions or atomic effects), and
- v is modified by an effect of o.

Idea: an arc $\langle u, v\rangle$ in the causal graph indicates that variable u is in some way relevant for modifying the value of v

Causally relevant variables

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Definition (causally relevant variables)

Let $\Pi = \langle V, I, O, \gamma\rangle$ be an FDR planning task and let $P \subseteq V$ be a pattern for Π.
We say that $v \in P$ is causally relevant for P if $CG(\Pi)$ contains a directed path from v to a variable $v' \in P$ that is mentioned in the goal formula γ.

Note: The definition implies that variables in P mentioned in the goal are always causally relevant for P.

Causally irrelevant variables are useless

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Theorem (causally irrelevant variables are useless)

Let $P \subseteq V$ be a pattern for an FDR planning task Π, and let $P' \subseteq P$ consist of all variables that are causally relevant for P. Then $h^{P'}(s) = h^{P}(s)$ for all states s.

Proof.

($\leq$): follows from the dominated sum theorem with $P' \subseteq P$

($\geq$): Obvious if $h^{P'}(s) = \infty$; else, induction over $n = h^{P'}(s)$.

- Base case $n = 0$:
  If $h^{P'}(s) = 0$, then there exists a concrete goal state $\tilde{s}$ that agrees with s on all variables in P'. If we change $\tilde{s}$ so that it agrees with s on all variables in P, it is still a goal state because we only change variables that are not mentioned in the goal.

Causally irrelevant variables are useless (ctd.)

Proof (ctd.)

- Inductive case $n \to n+1$:
  If $h^{P'}(s) = n+1$, then there exist concrete states $\tilde{s}$ and $\tilde{t}$ and an operator o of Π such that:
  - s and $\tilde{s}$ agree on all variables in P',
  - $app_o(\tilde{s}) = \tilde{t}$, and
  - $h^{P'}(\tilde{t}) = n$.

  If we change $\tilde{s}$ and $\tilde{t}$ so that they agree with s on all variables of P, then still $app_o(\tilde{s}) = \tilde{t}$, because
  - o modifies a variable in P' (otherwise $\pi_{P'}(\tilde{s}) = \pi_{P'}(\tilde{t})$ contradicting $h^{P'}(\tilde{s}) = n+1 \neq n = h^{P'}(\tilde{t})$), and hence
  - all variables mentioned in o are causally relevant for P', which implies that
  - o mentions no variable in $P \setminus P'$.

  ...

Causally irrelevant variables are useless (ctd.)

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Proof (ctd.)

- ...
  We get:
  - s and $\tilde{s}$ agree on all variables in P,
  - $app_o(\tilde{s}) = \tilde{t}$, and
  - $h^{P'}(\tilde{t}) = n = h^{P}(\tilde{t})$ (by the induction hypothesis).

  This implies $h^{P}(s) \leq n+1$, concluding the proof.

□

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

Causally connected patterns

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Definition (causally connected patterns)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be an FDR planning task and let $P \subseteq V$ 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, ignoring arc directions).

Disconnected patterns are decomposable

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Theorem (causally disconnected patterns are decomposable)

Let $P \subseteq V$ be a pattern for a SAS$^+$ planning task Π that is not causally connected, and let P_1, P_2 be a partition of P into non-empty subsets 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.

($\leq$): There is no arc between P_1 and P_2 in the causal graph, and thus there is no operator that affects both patterns. Therefore, they are additive, and $h^{P_1} + h^{P_2} \leq h^{P}$ follows from the dominated sum theorem.

Disconnected patterns are decomposable (ctd.)

Proof (ctd.)

($\geq$): If $h^{P_1}(s) + h^{P_2}(s) = \infty$, we are done.
Otherwise, proof by induction over $n = h^{P_1}(s) + h^{P_2}(s)$.

- Base case $n = 0$:
  If $h^{P_1}(s) + h^{P_2}(s) = 0$, then $h^{P_1}(s) = 0$ and hence s satisfies all goal atoms for variables in P_1; similarly for P_2. Hence, it satisfies all goal atoms for variables in $P = P_1 \cup P_2$, and thus $h^P(s) = 0$.

Disconnected patterns are decomposable (ctd.)

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Proof (ctd.)

- Inductive case $n \to n+1$:
  If $h^{P_1}(s) + h^{P_2}(s) = n+1$, then there exist concrete states $\tilde{s}$ and $\tilde{t}$, an operator o of Π and $i \in \{1,2\}$ such that:
  1. s and $\tilde{s}$ agree on all variables in P_i,
  2. $app_o(\tilde{s}) = \tilde{t}$, and
  3. $h^{P_i}(\tilde{t}) = h^{P_i}(\tilde{s}) - 1$.

  Let $j \in \{1,2\}$ with $j \neq i$. Since P_1 and P_2 are causally disconnected, the operator o does not mention any variables in P_j. Therefore, we can change $\tilde{s}$ and $\tilde{t}$ so that they agree with s on all variables of P and still have (2) and (3).

  ...

Disconnected patterns are decomposable (ctd.)

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

Proof (ctd.)

- ...
  We get:
  - $h^{P_j}(\tilde{t}) = h^{P_j}(\tilde{s})$ (because o does not affect variables in P_j)
  - $\rightsquigarrow$ $h^{P_1}(\tilde{t}) + h^{P_2}(\tilde{t}) = n$
  - $\rightsquigarrow$ $h^{P}(\tilde{t}) = n$ (by the induction hypothesis)
  - $\rightsquigarrow$ $h^{P}(\tilde{s}) \leq n+1$ (because $app_o(\tilde{s}) = \tilde{t}$)
  - $\rightsquigarrow$ $h^{P}(s) \leq n+1$ (because s and $\tilde{s}$ agree on P and hence $h^P(s) = h^P(\tilde{s})$).

  This concludes the proof. $\square$

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

Search neighbourhood

PDB heuristics
Implementing PDBs
Additivity
Pattern selection
Local search
Search space
Estimating heuristic quality
Summary

We can now put the pieces together to define our search neighbourhood, obtaining the third answer:

3. Which collections are neighbours of the current collection?

The neighbours of $\mathscr{C}$ are all pattern collections $\mathscr{C} \cup \{P'\}$ where

- $P' = P \cup \{v\}$ for some $P \in \mathscr{C}$,
- $P' \notin \mathscr{C}$,
- all variables of P' are causally relevant in P',
- P' is causally connected, and
- all pattern databases in $\mathscr{C} \cup \{P'\}$ can be represented within some prespecified space limit

Search neighborhood (ctd.)

Remark: For causal relevance and connectivity, there is a sufficient and necessary criterion which is easy to check:

- 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 is mentioned in the goal formula.

What is a good pattern collection?

- The last question we need to answer is how to rank the quality of pattern collections.
- This is perhaps the most critical point: without a good ranking criterion, pattern collections are chosen blindly.

The first search-based approach to pattern selection (Edelkamp, 2007) used the following strategy:

- only additive sets are used as pattern collections
  - no need for something like the canonical heuristic function
- the quality of a single pattern is estimated by its mean heuristic value (the higher, the better)
- the quality of a pattern collection is estimated by the sum of the individual pattern qualities

Discussion of the mean value approach

Pros of the approach:

- mean heuristic values are clearly correlated with search performance $\rightsquigarrow$ the quality measure makes sense
- mean heuristic values are quite easy to calculate

Cons of the approach:

- cannot reasonably deal with infinite heuristic estimates
- difficult to generalize to pattern collections that are not fully additive
- there are better predictors for search performance than mean heuristic values

So what is a good pattern collection, again?

How can be come up with a better quality measure?

- We are chiefly interested in minimizing the number of node expansions for the canonical heuristic function during the actual search phase of the planner.
- There is theoretical work on predicting node expansions of heuristic search algorithms based on parameters of the heuristic (Korf, Reid & Edelkamp, 2001).

$\rightsquigarrow$ Try to estimate these parameters, then use their analysis.

The Korf, Reid & Edelkamp formula

Korf, Reid & Edelkamp (2001)

In the limit of large c, the expected number of node expansions for a failed iteration of IDA* with depth threshold c is

$$E(N, c, P) = \sum_{i=0}^{c} N_i P(c-i)$$

where

- $N = \langle N_0, N_1, \ldots, N_c \rangle$ is the brute-force tree shape
  - ⇝ N_i: number of search nodes in layer i of the brute-force search tree
- P is the equilibrium distribution of the heuristic function
  - ⇝ $P(k)$: probability that node chosen uniformly from layer i of the brute-force search tree has heuristic value at most k, in the limit of large i

Using the formula for quality estimation

Some problems when using the formula for quality estimation:

- only holds in the limit
- applies to IDA*, but most planners use A*
- we do not know N or P

However:

- we do not need absolute node estimates, we only need to know which heuristic among a set of candidates is best
- we would expect heuristics good for IDA* to be good for A* most of the time
- we can use random walks and sampling to get approximate estimates without knowing N and P

Estimating heuristic quality in practice

With some additional assumptions and simplifications, we reduce the problem of ranking the quality of pattern collections to the following criterion:

Measuring degree of improvement

- Generate M states $s_1, \ldots, s_M$ through random walks in the search space from the initial state (according to certain parameters not discussed in detail).
- The degree of improvement of a pattern collection $\mathscr{C}'$ which is generated as a successor of collection $\mathscr{C}$ is the number of sample states s_i for which $h^{\mathscr{C}'}(s_i) > h^{\mathscr{C}}(s_i)$.

Computing $h^{\mathscr{C}'}(s)$

- So we need to compute $h^{\mathscr{C}'}(s)$ for some states s and each candidate successor collection $\mathscr{C}'$.
- We have PDBs for all patterns in $\mathscr{C}$, but not for the new pattern $P' \in \mathscr{C}'$ (of the form $P \cup \{v\}$ for some $P \in \mathscr{C}$).
- If possible, we want to avoid computing the complete pattern database except for the best successor (where we will need it later anyway).

Idea:

- For SAS$^+$ tasks Π, $h^{P'}(s)$ is identical to the optimal solution length for the syntactic projection $\Pi|_{P'}$.
- We can use any optimal planning algorithm for this.
- In particular, we can use A* search using h^P as a heuristic.

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 on 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.

Summary (ctd.)

- When faced with multiple PDB heuristics (a pattern collection), we want to admissibly add their values where possible, and maximize where addition is inadmissible.
- The canonical heuristic function is the best possible additive/maximizing combination for a given pattern collection given our additivity criterion.
- One way to automatically find a good pattern collection is by performing search in the space of pattern collections.
- One such approach uses hill-climbing search guided by the Korf, Reid and Edelkamp formula, which tries to estimate the quality of a heuristic function.