Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning

Patrik Haslum, Adi Botea, Malte Helmert,
Blai Bonet and Sven Koenig

(AAAI 2007)

Severin Gsponer

Problem of Pattern Databases

PDBs are a promising approach but:

- Difficult to select good patterns
- Memory and time limit the usable patterns

Target: Find good pattern collections completely automatic

Background

- Planning problem described in STRIPS
- Multi-valued state variables
- Abstraction of the problem
  - Subset A of state variables
  - Ignore variables not in A
  - $h^A(s)$ heuristic value for state *s*
- Constrained abstraction

Canonical Heuristic Function

Some definitions:

h is said to **dominate** another admissible heuristic *h'*

iff $h(s) \geq h'(s), \forall s$

Given two PDB heuristics h^A and h^B the heuristic function

$$h(s) = max(h^A(s), h^B(s))$$

is admissible and dominates h^A and h^B alone.

Canonical Heuristic Function

If the set of action that affect some variable in *A* is disjoint from the set of actions that affect any variable in *B*, then the **additive heuristic**

$$h(s) = h^A(s) + h^B(s)$$

is also admissible. We say A and B are **additive**.

A set of patterns is additive iff all patterns in the set are pairwise additive.

The additive heuristic dominates the maximum heuristic.

Canonical Heuristic Function

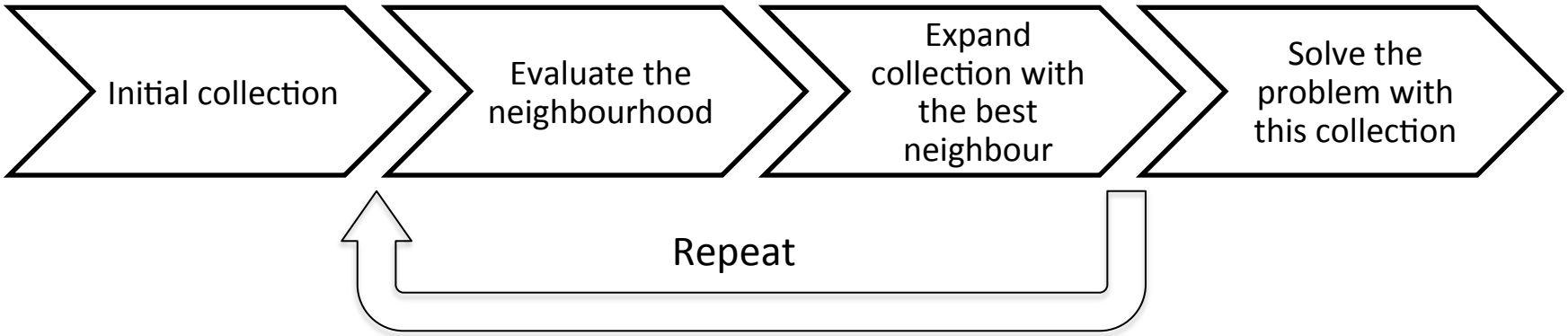
Definition:

Let $C = \{P_1,\ldots,P_k\}$ be a collection of patterns, and let A be the collection of all maximal (w.r.t. set inclusion) additive subsets of C:

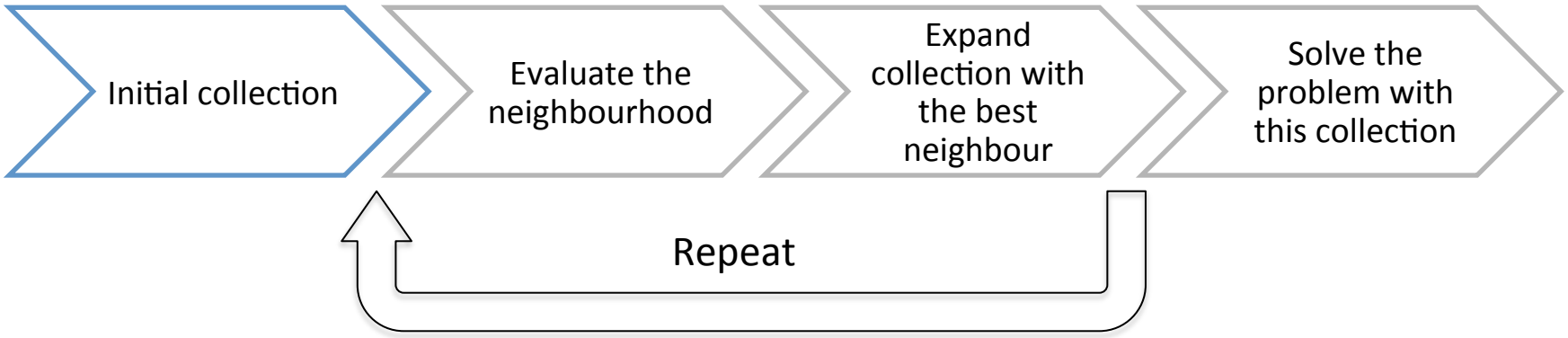
The **canonical heuristic** function of C is

$$h^C(s) = \max_{S \in A} \sum_{P \in S} h^P(s)$$

Finding good Pattern Collections



Finding good Pattern Collections

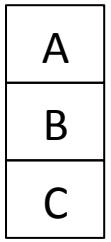


Initial pattern collection

- One pattern for each goal variable
- Each containing only this variable

Example blocks world:

Goal:



$G = \{loc_A=B, loc_B=C\}$

Collection $C = \{P_1, P_2\}$

$P_1 = \{loc_A\}$
$P_2 = \{loc_b\}$

Constructing the Neighbourhood

Given collection $C = \{P_1, \ldots, P_k\}$ a new collection C' can be constructed by:

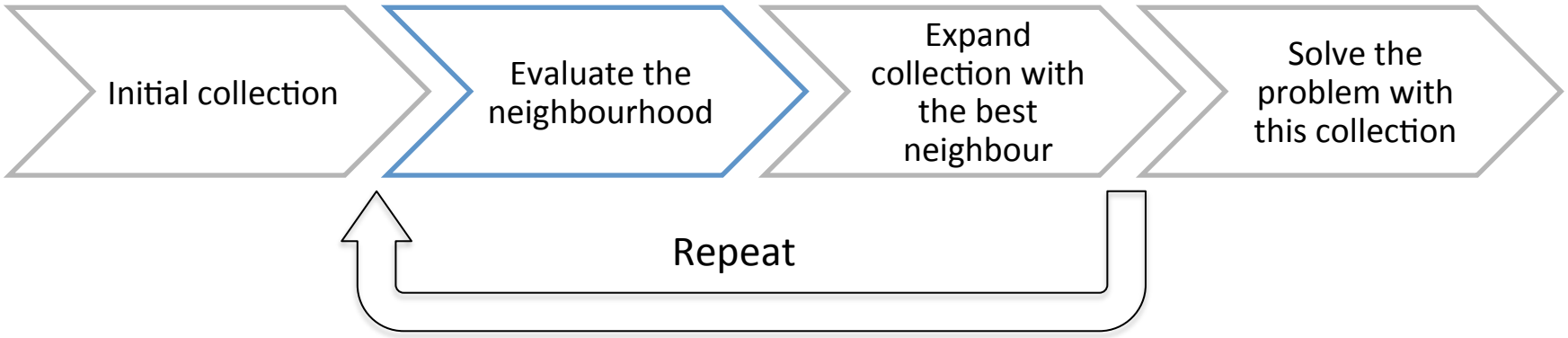
Add one variable *V* to a pattern P_i and add this new pattern to the collection.

$$C' = \{P_1, \ldots, P_k, P_{k+1}\}$$

$$P_{k+1} = P_i \cup \{V\}$$

The **neighbourhood of C** contains all this new collections.

Finding good Pattern Collections



Evaluating the Neighbourhood

Korf, Reid and Edelkamp (Artificial Intelligence 2001) develop a formula for **the number of nodes expanded** by a tree search with cost bound c:

$$\sum_{k=0,\ldots,c} N_{c-k} P(k)$$

where N_i: number of nodes whose accumulated cost equals i
$P(k)$: equilibrium dirstibution of the heuristic function h

It's enough to know for a base heuristic h_0, and two alternative improvements *h* and *h'*, which improves more over h_0.

Evaluating the Neighbourhood

In our case *C* and *C'* only differs in one new pattern.

$$h^{C'}(s) \geq h^{C}(s), \forall s$$

So the **search effort saved** by using *C'* instead of *C* is:

$$\sum_{k=0,\ldots,c} N_{c-k}(P(k) - P'(k))$$

If we draw a sample of m nodes at random from the search tree we get:

$$\frac{1}{m} \sum_{n_i} \sum_{h^{C}(n_i) \leqslant k < h^{C'}(n_i)} N_{c-k}$$

Evaluating the Neighbourhood

Since N_k tends to grow exponentially, we assume that $\boldsymbol{N_k}$ **dominates the other** $\boldsymbol{N_i}$:

$$\frac{1}{m} \sum_{\{n_i | h^C(n_i) < h^{C'}(n_i)\}} N_{c-h^C(n_i)}$$

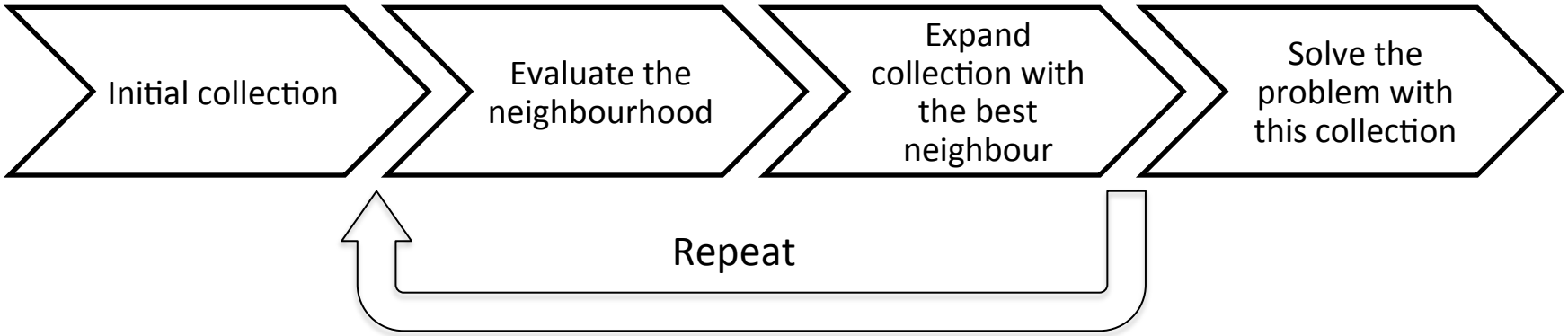
Now we only have to determine if $h^{c'}(n_i) > h^c(n_i)$.

Additional simplifying approximation is to ignore the weight. So we only have to count the number of nodes for which $h^{c'}(n_i) > h^c(n_i)$.

Important Improvements

- Simplify the comparison of $h^{c'}$ and h^{c}
- Sampling the search space
- Avoiding redundant evaluations
- Ending the search

Finding good Pattern Collections



Experiments

Sokoban:

40 Problems

28 solved

679’650 node expanded per instance

600.4 seconds average runtime

508.2 seconds spend in constructing the PDB (84.6%)

Regression search (using other heuristics):

Solve none of these 40 test problems.

Experiments

15-Puzzle:

100 Problems

93 solved

331’220 node expanded per instance

1’439.7 seconds average runtime

1’381.9 seconds spend in constructing the PDB (96%)

Regression search:

Over a set of 24 Problem

- total number of expanded nodes: 2’559’508
- with the method of the paper: 549’147
- this is less than ¼

Experiments

15-Puzzle	Mean value	Counting approximation
Solved problems	66	80
Node expanded	657’380	418’730
Win ratio	1	7

Impact of Counting Approximation

$$\frac{1}{m} \sum_{\{n_i | h^C(n_i) < h^{C'}(n_i)\}} N_{c-h^C(n_i)}$$

Node expanded	Estimated parameters	Counting approximation
Sokoban	551’290	679’650
15-Puzzle	655’530	483’490
Logistics	24’153	23’557

Questions?