Monotonic Variants of Saturated Cost Partitioning

Mauricio Salerno[1] Raquel Fuentetaja[2] Jendrik Seipp[1]

ICAPS 2026

[1]Linköping University

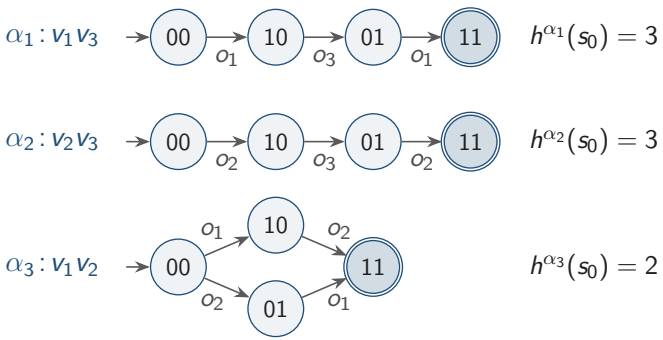
[2]Universidad Carlos III de Madrid

Optimal planning as heuristic search

- Classical planning by heuristic search in the induced state space
- Needs domain-independent heuristics
- We focus on abstraction heuristics: heuristic is the cost of abstract solution
- A *single* heuristic rarely captures *all* aspects of a task — use several
- To use them together, we must combine them admissibly

Abstraction heuristics & cost partitioning

Vars $v_1, v_2, v_3 \in \{0, 1\}$; start 000, goal 111; ops o_1, o_2, o_3 (unit cost).

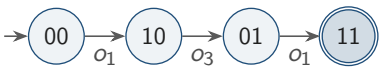


Each α_i projects onto two variables; double circle = goal.

Abstraction heuristics & cost partitioning

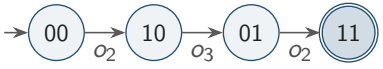
Vars $v_1, v_2, v_3 \in \{0, 1\}$; start 000, goal 111; ops o_1, o_2, o_3 (unit cost).

$\alpha_1 : v_1 v_3$



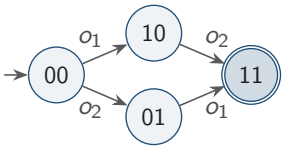
$h^{\alpha_1}(s_0) = 3$

$\alpha_2 : v_2 v_3$



$h^{\alpha_2}(s_0) = 3$

$\alpha_3 : v_1 v_2$



$h^{\alpha_3}(s_0) = 2$

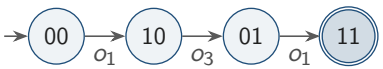
$\sum_i h^{\alpha_i}(s_0)$ is not admissible:
o_3 helps both α_1 and α_2

Each α_i projects onto two variables; double circle = goal.

Abstraction heuristics & cost partitioning

Vars $v_1, v_2, v_3 \in \{0, 1\}$; start 000, goal 111; ops o_1, o_2, o_3 (unit cost).

$\alpha_1 : v_1 v_3$



$h^{\alpha_1}(s_0) = 3$

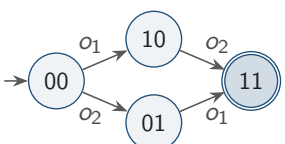
$\sum_i h^{\alpha_i}(s_0)$ is not admissible: o_3 helps both α_1 and α_2

$\alpha_2 : v_2 v_3$

$h^{\alpha_2}(s_0) = 3$

Cost partitioning: split operator costs as $c_1, \ldots, c_n$, $\sum_i c_i \leq cost$; then $\sum_i h^{\alpha_i}(c_i) \leq h^*$.

$\alpha_3 : v_1 v_2$



$h^{\alpha_3}(s_0) = 2$

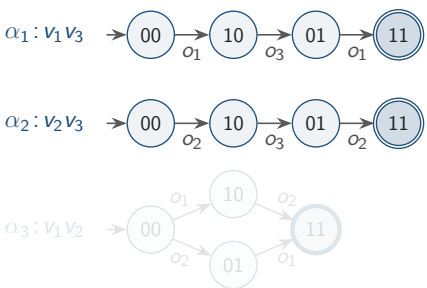
E.g. $c_1 = \langle 1, 0, 1 \rangle$, $c_2 = \langle 0, 1, 0 \rangle$: $3+2 = 5$.

Each α_i projects onto two variables; double circle = goal.

Saturated Cost Partitioning — Running example

Greedily saturate each abstraction in a fixed order ω:

Each α_i takes as much cost as it needs from what *remains*



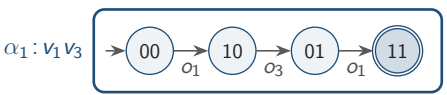
$\mathcal{A} = \{\alpha_1, \alpha_2\}$, order $\langle \alpha_1, \alpha_2 \rangle$

	o_1	o_2	o_3
rem	1	1	1

Saturated Cost Partitioning — Running example

Greedily saturate each abstraction in a fixed order ω:

Each α_i takes as much cost as it needs from what *remains*



$\mathcal{A} = \{\alpha_1, \alpha_2\}$, order $\langle \alpha_1, \alpha_2 \rangle$

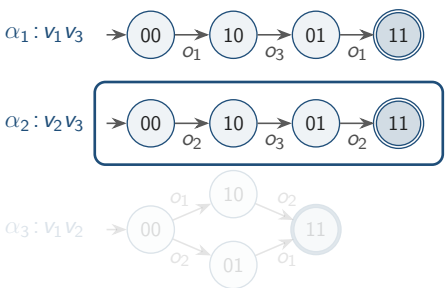
	o_1	o_2	o_3
rem	0	1	0

α_1 takes *scf*=$\langle 1, 0, 1 \rangle$, h^{α_1}=3

Saturated Cost Partitioning — Running example

Greedily saturate each abstraction in a fixed order ω:

Each α_i takes as much cost as it needs from what *remains*



$\mathcal{A} = \{\alpha_1, \alpha_2\}$, order $\langle \alpha_1, \alpha_2 \rangle$

	o_1	o_2	o_3
rem	0	0	0

α_1 takes $scf = \langle 1, 0, 1 \rangle$, $h^{\alpha_1} = 3$

α_2 takes $\langle 0, 1, 0 \rangle$, $h^{\alpha_2} = 2$

$h^{\mathrm{SCP}} = 3 + 2 = \mathbf{5}$ (either order)

More is better, right?

Two monotonic notions we'd *expect* in a cost-partitioning heuristic:

- Adding an abstraction to the set is good
- Refining an abstraction into a finer one is good

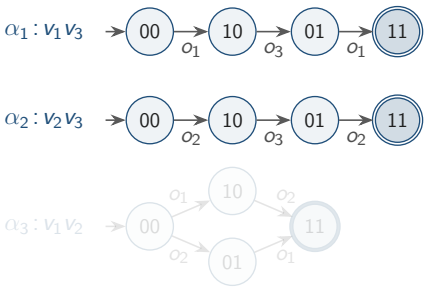
Cost partitioning	Adding	Refining
Optimal	✓	✓
Post-hoc optimization	✓	✗
Saturated (SCP)	✗	✗

SCP is not monotonic for either notion!

Contribution: make SCP monotonic (under certain conditions)

Non-monotonicity under adding abstractions — $h^{SCP}_{\mathcal{A}} > h^{SCP}_{\mathcal{A}\cup\mathcal{B}}$

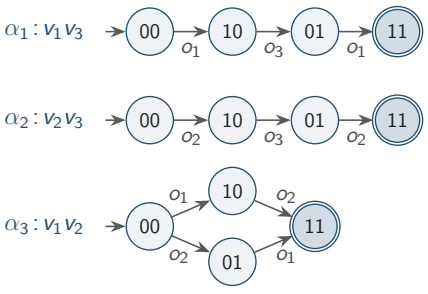
$\mathcal{A} = \{\alpha_1, \alpha_2\}$; $\mathcal{B} = \{\alpha_3\}$



$h^{\mathrm{SCP}}_{\{\alpha_1, \alpha_2\}} = \mathbf{5}$

Non-monotonicity under adding abstractions — $h^{SCP}_{\mathcal{A}} > h^{SCP}_{\mathcal{A}\cup\mathcal{B}}$

$\mathcal{A} = \{\alpha_1, \alpha_2\}$; $\mathcal{B} = \{\alpha_3\}$

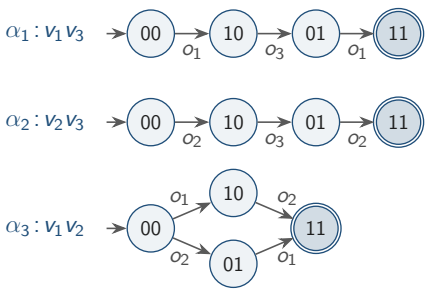


$h^{\mathrm{SCP}}_{\{\alpha_1, \alpha_2\}} = \mathbf{5}$

$h^{\mathrm{SCP}}_{\{\boldsymbol{\alpha_3}, \alpha_1, \alpha_2\}} : \; 2 + 1 + 0 = \mathbf{3}$

Non-monotonicity under adding abstractions — $h^{SCP}_{\mathcal{A}} > h^{SCP}_{\mathcal{A}\cup\mathcal{B}}$

$\mathcal{A} = \{\alpha_1, \alpha_2\}$; $\mathcal{B} = \{\alpha_3\}$



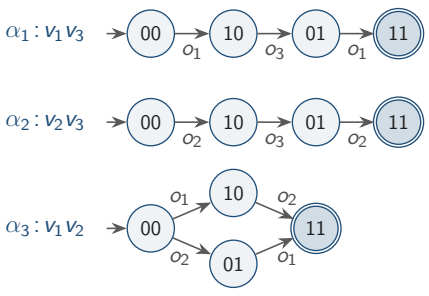
$h^{\mathrm{SCP}}_{\{\alpha_1, \alpha_2\}} = \mathbf{5}$

$h^{\mathrm{SCP}}_{\{\boldsymbol{\alpha_3}, \alpha_1, \alpha_2\}} : \; 2 + 1 + 0 = \mathbf{3}$

$h^{\mathrm{SCP}}_{\{\alpha_1, \boldsymbol{\alpha_3}, \alpha_2\}} : \; 3 + 1 + 0 = \mathbf{4}$

Non-monotonicity under adding abstractions — $h^{SCP}_{\mathcal{A}} > h^{SCP}_{\mathcal{A}\cup\mathcal{B}}$

$\mathcal{A} = \{\alpha_1, \alpha_2\}$; $\mathcal{B} = \{\alpha_3\}$



$h^{\mathrm{SCP}}_{\{\alpha_1, \alpha_2\}} = \mathbf{5}$

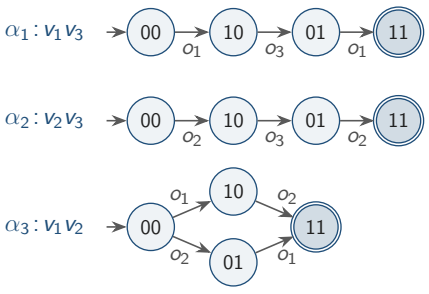
$h^{\mathrm{SCP}}_{\{\boldsymbol{\alpha_3}, \alpha_1, \alpha_2\}} : \; 2 + 1 + 0 = \mathbf{3}$

$h^{\mathrm{SCP}}_{\{\alpha_1, \boldsymbol{\alpha_3}, \alpha_2\}} : \; 3 + 1 + 0 = \mathbf{4}$

Adding α_3 lowered the heuristic value

Monotonicity under adding abstractions — Naive ordering

Place new abstractions $\mathcal{B}$ *after* the old ones $\mathcal{A}$.



$$h^{\mathrm{SCP}}\{\alpha_1, \alpha_2, \alpha_3\} : \; 3 + 2 + 0 = \mathbf{5}$$

α_3 gets *rem*$=\langle 0, 0, 0\rangle$: contributes 0, but *cannot lower* the sum.

Naive — yet sometimes the *only* monotonic ordering

When can we do better? — Refinements and coarsenings

If $\alpha \preceq \beta$, then β refines α (α is a coarsening).

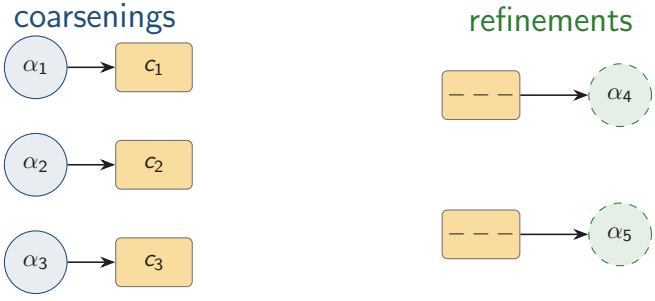
A refinement is always at least as informative:

- $h^\alpha(s) \leq h^\beta(s)$ for every state and *same* cost
- Any cost spent on a coarsening is better spent on its refinement

Greedy SCP may hand β *less* cost than α had — so the value can drop.

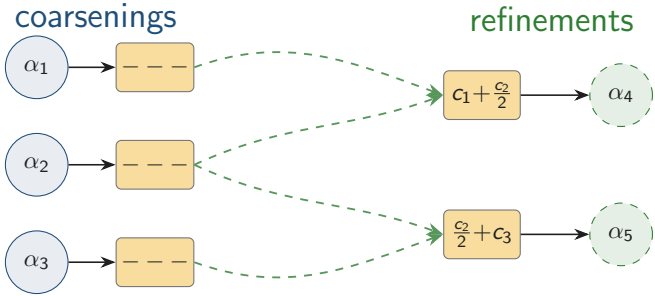
Idea: cost-partition over coarsenings, then distribute those costs to the refinements.

Cost partitioning over refinements



$\alpha_1, \alpha_2 \preceq \alpha_4$ $\alpha_2, \alpha_3 \preceq \alpha_5$

Cost partitioning over refinements



For a refinement α_j, cost: $c'_j = \sum_{\alpha_i \preceq \alpha_j} f_{ij} * c_i$.

Refinements get fractions of the coarsening's cost.

For a coarsening α_i, $\sum_{\alpha_i \preceq \alpha_j} f_{ij} = 1$, guarantees monotonicity.

Experimental setup

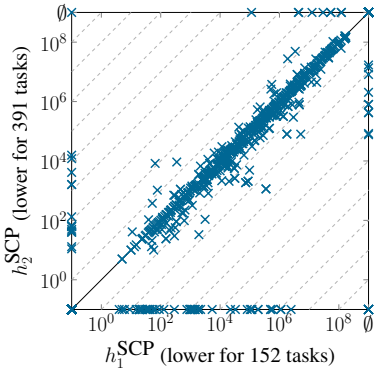
- Implemented in Scorpion [Seipp et al. 2020], an extension of Fast Downward
- Benchmarks: all IPC tasks (1998–2023) without conditional effects
- Limits: 30 min, 8 GiB

Two abstraction families, each with a natural *coarsening* $\preceq$ *refinement* structure:

Family	coarsening $\preceq$ refinement	baselines
Cartesian	1-goal $\preceq$ 2-goal (merged)	h^{SCP}_1, h^{SCP}_2
PDBs	size-1 $\preceq$ size-2 patterns (systematic)	$h^{\mathrm{SCP}}_{\mathrm{PDB}-1}$, $h^{\mathrm{SCP}}_{\mathrm{PDB}-2}$

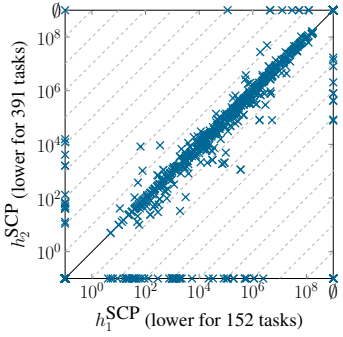
Baselines vs. our monotone variants h^{SCP}_{naive}, h^{SCP}_{unif}, h^{SCP}_{rand}.

Experiments: expansions

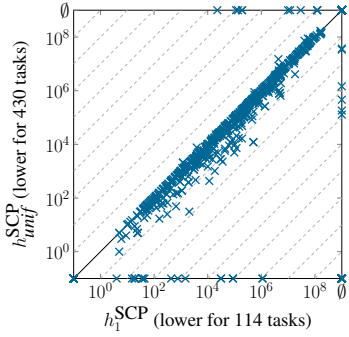


Vanilla SCP, h_1^{SCP} vs. h_2^{SCP}: not monotonic

Experiments: expansions

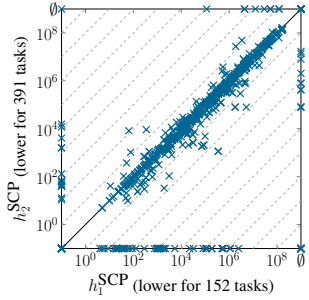


h_1^{SCP} vs. h_2^{SCP}: not monotonic

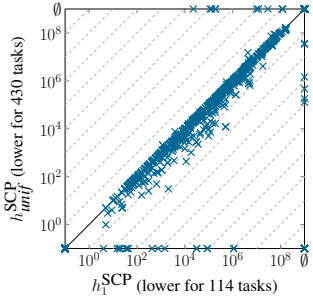


h_{unif}^{SCP} vs. h_1^{SCP}: monotonic*

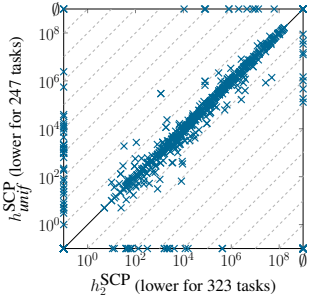
Experiments: expansions



h_1^{SCP} vs. h_2^{SCP}



h_{unif}^{SCP} vs. h_1^{SCP}



h_{unif}^{SCP} vs. h_2^{SCP}: competitive

Conclusion & future work

Introduced two monotonicity notions, with schemes for SCP:

- Monotonicity under Adding $\rightarrow$ naive ordering
- Monotonicity under Refining $\rightarrow$ cost partitioning over refinements

Coverage increase on Cartesian abstractions, competitive on PDBs.

Caveat: on PDBs, SCP already behaves close to monotonically — fewer possible gains.

Future work:

- More monotonic partitions — So far only uniform and random
- Test on merge-and-shrink?

Backup: coverage & heuristic values

Coverage

Configuration	solved
h^{SCP}_{1}	1006
h^{SCP}_{2}	1019
h^{SCP}_{naive}	1020
h^{SCP}_{rand}	1021.3
h^{SCP}_{unif}	**1037**

Initial h: row $>$ column (# tasks)

	h^{SCP}_{1}	h^{SCP}_{2}	h^{SCP}_{naive}	h^{SCP}_{rand}	h^{SCP}_{unif}
h^{SCP}_{1}	–	172	0	0	0
h^{SCP}_{2}	**373**	–	**256**	**237**	214
h^{SCP}_{naive}	**185**	210	–	0	0
h^{SCP}_{rand}	**281**	235	**100**	–	16
h^{SCP}_{unif}	**323**	**254**	**183**	**90**	–

Competitive coverage, better initial heuristic values.

Backup: per-domain coverage (full matrix)

Cell (r, c) = # domains where r solves more tasks than c; last row = total solved.

	h_1^{SCP}	h_2^{SCP}	h_{naive}^{SCP}	h_{rand}^{SCP}	h_{unif}^{SCP}
h_1^{SCP}	–	**6**	3	1	**6**
h_2^{SCP}	**6**	–	5	3	**6**
h_{naive}^{SCP}	**6**	**11**	–	2	**9**
h_{rand}^{SCP}	**6**	**9**	**3**	–	**9**
h_{unif}^{SCP}	5	5	3	3	–
Coverage	1006	1019	1020	1021.3±1.2	**1037**