Domain-Abstraction Heuristics for Simple Numeric Planning

CEGAR over infinite integer domains

Markus Fritzsche Mikhail Gruntov Alexander Shleyfman
Daniel Gnad

ICAPS 2026

Outline

- Simple Numeric Planning
- Abstraction Classes
- Numeric Domain Abstractions
- Combining Abstractions
- Experimental Picture

Why numeric planning changes the abstraction story

Classical abstraction heuristics

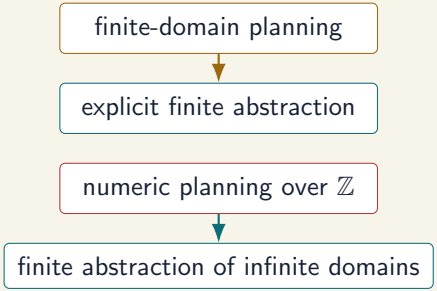
PDBs, Cartesian abstractions, and merge-and-shrink are highly effective because abstract state spaces are finite and can be searched explicitly.

Numeric variables

A variable can range over $\mathbb{Z}$:

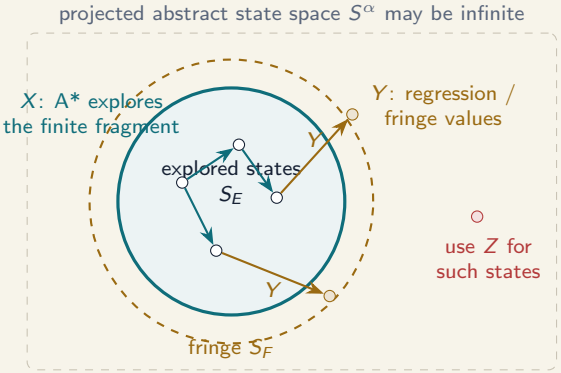
$$s[v] \in \mathbb{Z}, \qquad v+ = c, \qquad v \leq k.$$

Naive abstractions can still leave infinitely many abstract states.



Question: can CEGAR construct useful finite abstractions of infinite numeric transition systems?

Numeric PDBs: partial exploration



XYZ naming scheme

- X: heuristic for abstract A* exploration;
- Y: heuristic for boundary regression;
- Z: heuristic for failed lookup.

B = blind, L = numeric LM-cut

BBB: blind / blind / blind

LLB: LM-cut / LM-cut / blind

Simple numeric planning in one slide

A task is $\Pi = \langle V, A, s_0, G \rangle$ with $V = V_p \cup V_n$.

State variables

$$v \in V_p : D_v \text{ finite}, \qquad v \in V_n : D_v = \mathbb{Z}.$$

A state is a complete assignment to all variables.

Conditions

$$\langle v, d \rangle \quad \text{or} \quad v \bowtie k, \quad \bowtie \in \{\leq, =, \geq\}.$$

Actions

$$a = \langle \mathrm{pre}(a), \mathrm{eff}(a), \mathrm{cost}(a) \rangle.$$

Effects are finite-domain assignments or numeric updates:

$$v := d, \qquad v+ = c, \quad c \in \mathbb{Z} \setminus \{0\}.$$

Search

We still run optimal heuristic search; the hard part is an admissible estimate over infinitely many states.

Example: Sailing with one person to save

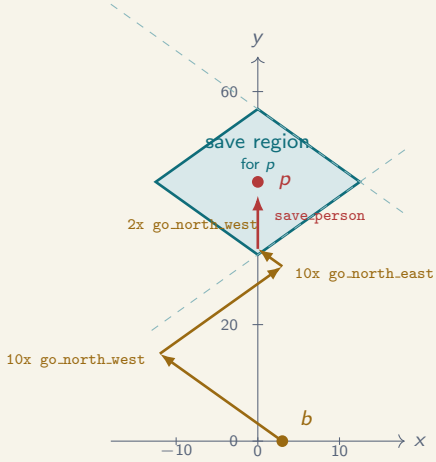
One boat, one person

$$x(b) = 3$$
$$y(b) = 0$$
$$d(p) = 32$$
$$saved(p) = false$$

goal: $saved(p)$

save iff

$$x(b) + y(b) \in [d(p), d(p) + 25]$$
$$y(b) - x(b) \in [d(p), d(p) + 25]$$



Sailing action schemas

Movement actions

All movement actions have precondition $\top$.

action	effect
`go_north_east(`b`)`	$x(b)+=1.5,\ y(b)+=1.5$
`go_north_west(`b`)`	$x(b)-=1.5,\ y(b)+=1.5$
`go_est(`b`)`	$x(b)+=3$
`go_west(`b`)`	$x(b)-=3$
`go_south_west(`b`)`	$x(b)+=2,\ y(b)-=2$
`go_south_east(`b`)`	$x(b)-=2,\ y(b)-=2$
`go_south(`b`)`	$y(b)-=2$

Saving a person

`save_person(`b, p`)`

pre: $d(p) \leq x(b) + y(b) \leq d(p) + 25$
$d(p) \leq y(b) - x(b) \leq d(p) + 25$
eff: $\mathit{saved}(p) := \mathit{true}$

The picture on the previous slide shows these preconditions as the diamond-shaped save region.

The RT formalism used in the paper

Restricted tasks (RT)

We present the construction for Hoffmann's restricted numeric tasks:

- numeric conditions compare one variable to an integer;
- numeric effects add or subtract an integer constant;
- all variables are fully observable state variables.

Example: preconditions may contain $fuel \geq 1$; effects may contain $fuel+ = -1$.

Scope comment

In the code, we support the full simple numeric planning fragment, not only the RT fragment used for the clean presentation here.

Why start with RT?

RT isolates the abstraction issue: finite actions, integer variables, and infinite transition systems.

A common abstraction view

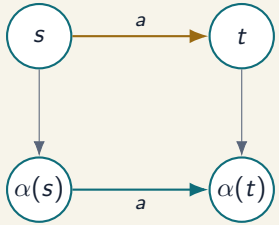
A state abstraction is a map $\alpha : S \to S^\alpha$ inducing an abstract transition system:

$$s \xrightarrow{a} t \quad \Rightarrow \quad \alpha(s) \xrightarrow{a} \alpha(t).$$

Admissible abstraction heuristic

$$h^\alpha(s) = h^*_{\mathcal{T}^\alpha}(\alpha(s)).$$

Because α is a homomorphism, every concrete plan maps to an abstract plan. Thus abstract optimal cost is a lower bound: $h^\alpha(s) \leq h^*(s)$.

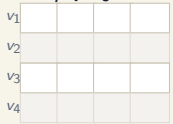


All abstractions in this talk fit this homomorphism view.

PDBs, domain abstractions, and Cartesian sets

Assume FDR and, only for the picture, $D(v_1) = \cdots = D(v_4) = \{0, 1, 2, 3\}$. Rows are variables, columns are values.

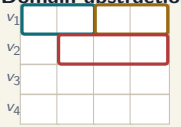
PDB / projection



omit rows

Pattern $P = \{v_1, v_3\}$. Values of omitted variables are ignored.

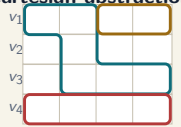
Domain abstraction



group columns per row

Each variable has its own partition of its domain.

Cartesian abstraction



Cartesian sets

Abstract states are products $A_1 \times \cdots \times A_n$, refined locally.

Domain abstractions sit between projections/PDBs and Cartesian abstractions: more expressive than omitting variables, less general than arbitrary Cartesian sets.

Domain abstractions in classical planning

Definition

A domain abstraction is a family

$$\alpha = \{\alpha_v \mid v \in V\},$$

where each $\alpha_v : D_v \to 2^{D_v}$ maps values to blocks and the image of α_v is a partition of D_v.

$$\alpha(s) = \{\langle v, \alpha_v(s[v])\rangle \mid v \in V\}.$$

$D_v = \{0, 1, 2, 3, 4\}$

$$\{0, 1\} \mid \{2\} \mid \{3, 4\}$$

A concrete value 4 becomes the abstract block $\{3, 4\}$.

CEGAR intuition

Start coarse, find an abstract plan, validate it concretely, and split the block responsible for a flaw.

What changes for numeric variables?

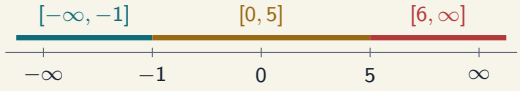
Numeric domain abstraction

For $v \in V_n$, α_v partitions $\mathbb{Z}$ into integer intervals:

$$\ldots \mid [-\infty, -1] \mid [0, 5] \mid [6, \infty] \mid \ldots$$

Blocks can be unbounded above or below.

Finite-domain variables keep ordinary finite partitions. Numeric variables get finite interval partitions that cover every integer.



Core benefit

We do not enumerate finite samples of $\mathbb{Z}$ and we do not store floating point interval relaxations.

Relaxed evaluation of numeric conditions

In an abstract state, $s^\alpha[v]$ is an interval, not one integer.

Definition

$$s^\alpha \vDash v \bowtie k \quad \text{iff} \quad \exists d \in s^\alpha[v] : d \bowtie k.$$

Consequence

Ambiguous conditions are treated optimistically, which preserves admissibility.

$v \geq 3$ on $[0, 5]$

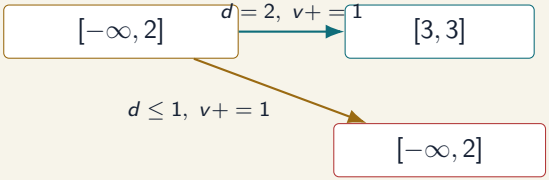
Some values satisfy it and some do not:

$$\{0,1,2\} \nvDash v \geq 3, \qquad \{3,4,5\} \vDash v \geq 3.$$

The abstract condition is true, but ambiguous.

Numeric effects induce nondeterministic abstract transitions

Consider $D(\alpha_v) = \{[-\infty, 2], [3, 3], [4, \infty]\}$ and an action with effect $v+ = 1$.



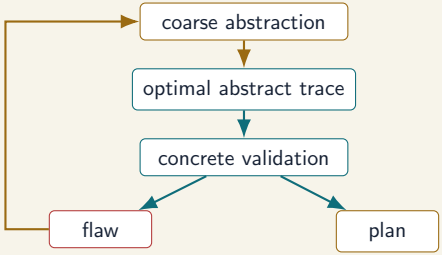
New flaw type

Classical domain abstractions have deterministic effects. Numeric interval effects may branch because different concrete values inside one block land in different target blocks.

We represent the abstract nondeterminism explicitly and validate the selected abstract trace against the concrete execution.

CEGAR loop for numeric domain abstractions

1. Build a coarse interval/domain abstraction.
2. Compute a cheapest abstract plan trace.
3. Execute the action sequence from the concrete state.
4. If it is concrete and reaches the goal: done.
5. Otherwise split the responsible finite-domain block or numeric interval.



Three flaw types

Precondition flaw

An abstract action is applicable because a condition is relaxed, but the concrete state does not satisfy it.

$$s_i \not\models \mathrm{pre}(a_i).$$

Goal flaw

The abstract trace ends in an abstract goal state, but the concrete execution does not satisfy the goal.

$$s_n \not\models G.$$

Deviation flaw

The concrete execution follows the same action but lands in a different abstract state than the trace predicted.

$$\alpha(s_{i+1}) \neq s_{i+1}^{\alpha}.$$

Refinement principle

Split at a concrete witness value, so the same flaw witness cannot recur in the same abstract form.

CEGAR example: car task

Task

Initial state: $loc(c) = 0$, $started(c) = false$

Goal: $G = \{loc(c) = 2\}$

Actions

`start-car`	pre $\top$; eff $started(c) := true$
`drive(c)`	pre $started(c)$; eff $loc(c) := loc(c) + 1$

Initial abstraction

Start with one block for each variable:

$$\alpha(loc(c)) = \{[-\infty, \infty]\}$$

$$\alpha(started(c)) = \{\{false, true\}\}.$$

Counterexamples refine blocks

Each flaw gives a concrete witness value. We keep the current abstraction and split only the block containing that witness.

CEGAR example: abstraction evolution

Current abstract plan	Flaw and refinement	$\alpha(loc(c))$	$\alpha(started(c))$
$\langle\rangle$	**Goal flaw.** Empty plan reaches the abstract goal, but concrete $loc(c) = 0$. Split at the goal boundary.	$\{[-\infty, 1], [2, \infty]\}$	$\{\{false, true\}\}$
$\langle drive \rangle$	**Precondition flaw.** `drive` needs concrete $started(c) = true$. Split the Boolean block.	$\{[-\infty, 1], [2, \infty]\}$	$\{\{false\}, \{true\}\}$
$\langle start, drive \rangle$	**Deviation flaw.** The abstract successor is $[2, \infty]$, but concrete execution reaches $loc(c) = 1$. Split at the witness.	$\{[-\infty, 0], [1, 1], [2, \infty]\}$	$\{\{false\}, \{true\}\}$
$\langle start, drive, drive \rangle$	**Validated.** The abstract plan is executable concretely and reaches the goal.	$\{[-\infty, 0], [1, 1], [2, \infty]\}$	$\{\{false\}, \{true\}\}$

Initially, $\alpha(loc(c)) = \{[-\infty, \infty]\}$ and $\alpha(started(c)) = \{\{false, true\}\}$.

How numeric refinement splits intervals

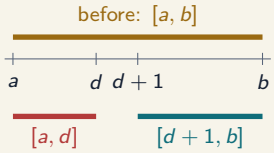
Suppose a condition $v \geq k$ fails for the current concrete value $d < k$, but its abstract interval still made the condition true.

Split at the witness

If $\alpha_v(d) = [a, b]$, replace it by

$$[a, d] \qquad \text{and} \qquad [d + 1, b].$$

The left interval cannot satisfy $v \geq k$; ambiguity is reduced only where the counterexample requires it.



Deviation flaw

For $v+ = c$, split at the concrete source value that caused the wrong abstract successor.

Semi-completeness statement

Theorem

The proposed CEGAR procedure is semi-complete on RT tasks without zero-cost actions.

Proof idea

If a concrete plan of cost L exists and all action costs are positive, then only finitely many concrete states and numeric values are reachable within cost L.

Why refinement terminates

Splits are made only at values witnessed inside this finite fragment. Repeated flaws are eliminated or the abstract plan cost increases until a valid concrete plan is found.

One abstraction is rarely enough

Collection

We generate multiple domain abstractions:

$$H = \{h^{\alpha_1}, \ldots, h^{\alpha_m}\}.$$

Different abstractions focus on different variables, goals, and flaws.

Admissible aggregation

The maximum is admissible but often leaves additive information unused: $h_{\max}(s) = \max_{h \in H} h(s)$.

Canonical heuristic

Build a conflict graph over abstractions. Compatible abstractions can be summed; then maximize over independent sets:

$$h_{\text{can}}(s) = \max_{I \in \mathcal{I}} \sum_{h \in I} h(s).$$

Compatibility: no action induces non-self-loop transitions in both abstract transition systems.

Implementation choices

Planner

Implemented in Numeric Fast Downward and used as an admissible heuristic for A*.

Variants

- **sDA**: a single domain abstraction.
- **cDA**: canonical heuristic over a collection.

Refinement controls

- abstraction-size and collection-size limits;
- blacklisting variables to diversify abstractions;
- flaw selection by minimum abstract state-space growth.

Caveat

The best cDA configuration was tuned on the evaluation benchmarks; PDB baselines were not tuned in the same way.

Coverage summary

Benchmark group	Blind	LMc	BBB	LLB	sDA	cDA
RT domains	86	106	110	109	109	**119**
Compiled to RT	117	**204**	144	163	123	136
Total	203	**310**	254	272	232	255

Where cDA shines

On natively RT-modeled domains, cDA solves the most instances and even beats numeric LM-cut in this aggregate.

Where it falls behind

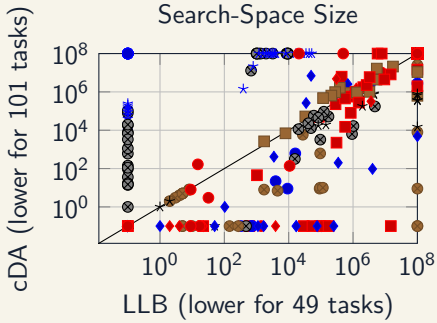
On domains compiled to RT, cDA trails LM-cut and LLB. Understanding the compilation effect remains future work.

Search guidance against numeric PDBs

Empirical pattern

Domain abstractions are complementary to numeric PDBs and often provide better search guidance.

- BBB/LLB explore selected abstract states.
- Failed lookups need a backup heuristic.
- Domain abstractions are total: every concrete state has an abstract state.



Expansions until the last f-layer.

What to remember

- Numeric variables make abstraction construction harder because state spaces can be infinite.
- Domain abstractions generalize cleanly: finite-domain partitions become integer interval partitions.
- Relaxed condition evaluation and nondeterministic numeric effects keep the abstraction admissible.
- CEGAR remains the right construction tool: refine only where an abstract counterexample is wrong.
- Collections combined canonically are competitive on native RT domains.

Why not saturated cost partitioning?

Natural question

We build collections of admissible abstractions.

In classical planning, saturated cost partitioning is the standard way to combine many such heuristics more strongly.

Current answer

For numeric domain abstractions, it is not that simple.

We are currently working on it. We think we have a good understanding of why the straightforward route is problematic, but:

stay tuned.

Thank you.

Questions & discussion

markus.fritzsche@liu.se
Linköping University

Backup slides

appendix

Appendix: abstraction class references

- PDB abstractions: projections onto a subset of variables.
- Cartesian abstractions: abstract states are Cartesian sets over the full variable set; CEGAR refines them locally.
- Domain abstractions: per-variable domain partitions, described as a middle point between PDBs and Cartesian abstractions.

Relevant papers

Seipp and Helmert, "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning", JAIR 62, 2018.
Kreft, Buchner, Sievers, Helmert, "Computing Domain Abstractions for Optimal Classical Planning with Counterexample-Guided Abstraction Refinement", ICAPS 2023.

Appendix: exact experimental setup

- Numeric Fast Downward implementation.
- A* search with admissible heuristics.
- 30 minute / 8 GiB IPC 2023 limits.
- Downward Lab 8.2.

- Baselines: blind, numeric LM-cut, numeric PDB variants BBB and LLB.
- Domain-abstraction variants: sDA and cDA.
- Benchmarks from recent SNP work.