Automated Planning with Modern Heuristics

Raquel Fuentetaja

Escuela de Verano de Inteligencia Artificial – EVIA 2025

July 11, 2025

Planning and Learning Group (PLG)
Universidad Carlos III de Madrid

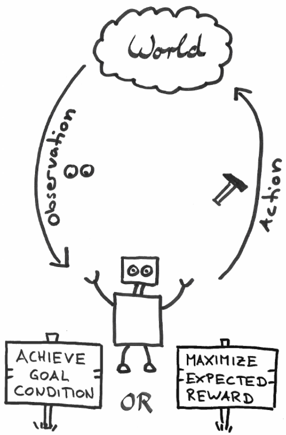
Some slides based on slides from the AI groups at the Universities of Basel and Linköping

Overview

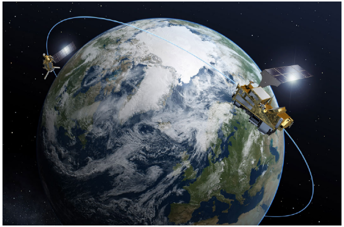
- Very short introduction to Automated Planning
- Intuition behind one of the most relevant heuristics for Classical Planning

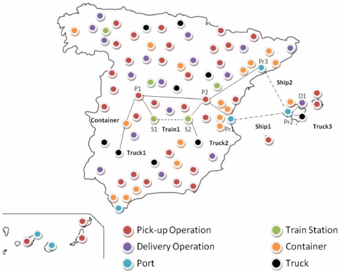
General perspective on planning

"Planning is the art of thinking before acting" —P. Haslum

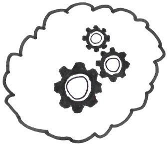


Examples

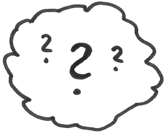



Pr3
Ship2
P1
P2
D1
Container
S1
Train1
S2
Pr1
Ship1
Pr2
Truck3
Truck2
Truck1
Pick-up Operation
Delivery Operation
Port
Train Station
Container
Truck

Model-based vs. data-driven approaches



Model-based approaches now the “inners working” of the world $\rightsquigarrow$ Reasoning



Data-driven approaches rely on collected data from a black-box world $\rightsquigarrow$ Learning

General Problem Solving

Wikipedia: General Problem Solver

General Problem Solver (GPS) was a computer program created in 1959 by Herbert Simon, J.C. Shaw, and Allen Newell intended to work as a universal problem solver machine.

Any formalized symbolic problem can be solved, in principle, by GPS. [...]

GPS was the first computer program which separated its knowledge of problems ... from its strategy of how to solve problems (a generic solver engine).

H. Simon & A. Newell. Carnegie Mellon University Libraries

Now we call this *Domain-Independent Automated Planning*

Research on Automated Planning

- One of the major subfields of Artificial Intelligence
- Represented at major AI conferences (IJCAI, AAAI, ECAI, etc.)
- Annual specialized conference ICAPS
- International Planning Competition (IPC)
- Major journals: general AI journals
  - Artificial Intelligence Journal (AIJ)
  - Journal of Artificial Intelligence Research (JAIR)

Classical Planning

Environment

- sequential
- fully observable
- deterministic
- static
- discrete

Problem solving method

- problem-specific vs. general vs. learning

Classical Planning tasks

Input to a planning algorithm: planning task

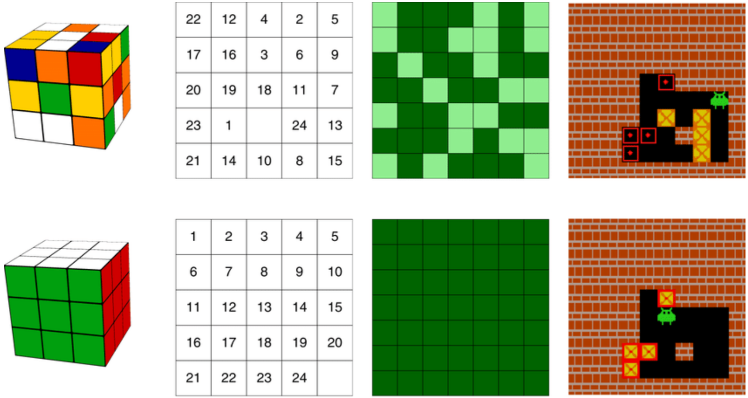
- initial state of the world
- actions that change the state
- goal to be achieved

Output of a planning algorithm: plan

- sequence of actions taking initial state to a goal state or confirmation that no plan exists
- satisficing vs. optimal: in optimal planning the output is a plan with minimal cost

Looks familiar?

Description fit (state space) search



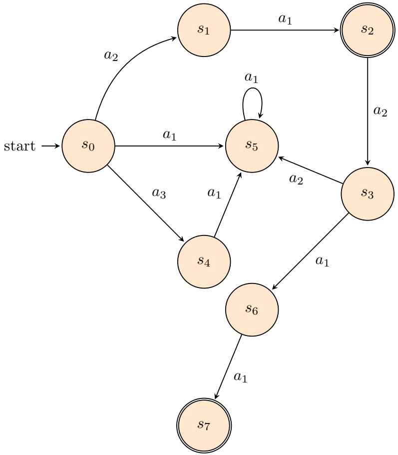
Rubik's cube | 24 puzzle | Lights Out (7×7) | Sokoban

Formal model

Definition (transition system)

A transition system or state space is a tuple $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$

- finite set of states S
- finite set of actions A
- action costs $cost : A \to \mathbb{R}_0^+$
- deterministic transitions $T \subseteq S \times A \times S$
- initial state s_0
- set of goal states $S_\star \subseteq S$



Heuristic search algorithms

We still use heuristic search algorithms like A* (Hart, Nilsson and Raphael, 1968)

⇝ search guided by a heuristic

```
BestFirstSearch(S, A, cost, T, s_0, S*):
    open ← priority queue ordered by f(n) = g(n) + h(n)
    open.insert(make_root_node(s_0))
    while open is not empty do
        n ← open.pop_min()
        if n.state ∈ S* then
            return extract_path(n)
        foreach ⟨a, s'⟩ such that ⟨n.state, a, s'⟩ ∈ T do
            h ← compute_heuristic(s')
            open.insert(make_node(n, a, s', h))
    return unsolvable
```

General algorithms

The developer does not know the tasks the algorithm needs to solve!

$\rightsquigarrow$ problem description language, problem independent heuristic

```
BestFirstSearch(S, A, cost, T, s_0, S*):
    open ← priority queue ordered by f(n) = g(n) + h(n)
    open.insert(make_root_node(s_0))
    while open is not empty do
        n ← open.pop_min()
        if n.state ∈ S* then
            return extract_path(n)
        foreach ⟨a, s'⟩ such that ⟨n.state, a, s'⟩ ∈ T do
            h ← compute_heuristic(s')
            open.insert(make_node(n, a, s', h))
    return unsolvable
```

General algorithms

1. Declarative description language to define the problem (planning formalism)
   - compact description of state space as input to algorithms
   - state spaces exponentially larger than the input
   - computationally challenging (PSPACE-complete)

2. Problem independent heuristic!

$\rightsquigarrow$ allows automatic reasoning about the problem: reformulation, simplification, abstraction, etc.

Description language

Planning formalisms

PDDL (Planning Domain Definition Language)

- input language used in practice
- based on predicate logic

STRIPS (Standford Research Institute Problem Solver): binary state variables

SAS^+ (Simplified Action Structures): state variables with arbitrary finite domains

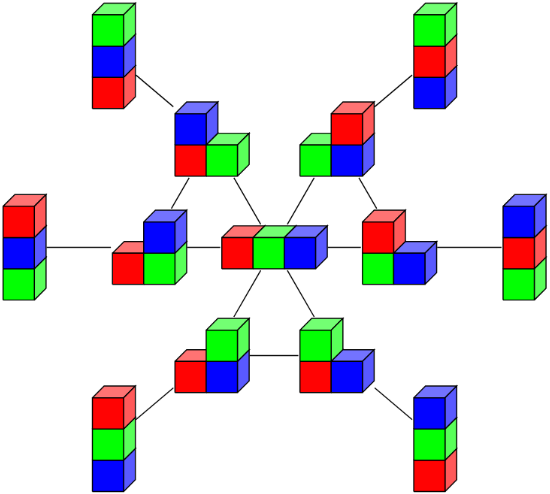
Planners convert automatically from PDDL to STRIPS or SAS^+

SAS$^+$ formalism

- finite set of state variables, each with a finite and non-empty domain
- finite set of actions with
  - preconditions: a partial assignment of variables
  - effects: a partial assignment of variables
  - cost
- initial state: total assignment of variables
- goal: partial assignment of variables

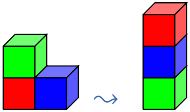
$\leadsto$ induces a transition system

Example – Blocksworld state space



n blocks: more than n! states (50 blocks $\approx 10^{84}$ states)

Example – Blocksworld in SAS$^+$



$\text{Var} = \{\text{posR}, \text{posB}, \text{posG}, \text{clearR}, \text{clearB}, \text{clearG}\}$

$$
\begin{aligned}
\text{domain}(\text{posR}) &= \{onB, onG, onT\} \\
\text{domain}(\text{posB}) &= \{onR, onG, onT\} \\
\text{domain}(\text{posG}) &= \{onR, onB, onT\} \\
\text{domain}(\text{clearR}) &= \{0, 1\} \\
\text{domain}(\text{clearB}) &= \{0, 1\} \\
\text{domain}(\text{clearG}) &= \{0, 1\}
\end{aligned}
$$

$\text{Actions} = \{\text{moveRBG}, \text{moveRGB}, \text{moveBRG}, \text{moveBGR}, \ldots\}$

$$
\begin{aligned}
\text{pre}(\text{moveRBG}) &: \{\text{posR} = onB, \text{clearR} = 1, \text{clearG} = 1\} \\
\text{eff}(\text{moveRBG}) &: \{\text{posR} = onG, \text{clearR} = 1, \text{clearG} = 0\} \\
\text{cost}(\text{moveRBG}) &= 1
\end{aligned}
$$

$$
\begin{aligned}
\text{Initial state} = \{&\text{posR} = onT, \text{posB} = onT, \text{posG} = onR, \\
&\text{clearR} = 0, \text{clearB} = 1, \text{clearG} = 1\}
\end{aligned}
$$

$Goal = \{\text{posR} = onB,\ \text{posB} = onG\}$

Heuristics

Heuristics

- Heuristic: function mapping each state to a non-negative number (or ∞)

$$h : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$$

- Perfect heuristic h^*: map each state s to the cost of an optimal solution for s
- A heuristic is admissible if $h(s) \leq h^*(s)$ for all states

$$h(s) = \textit{number of blocks not in their final position in } s$$

One of the main focus areas in Classical Planning

How do we find good heuristics in a domain independent way?

Planning heuristics

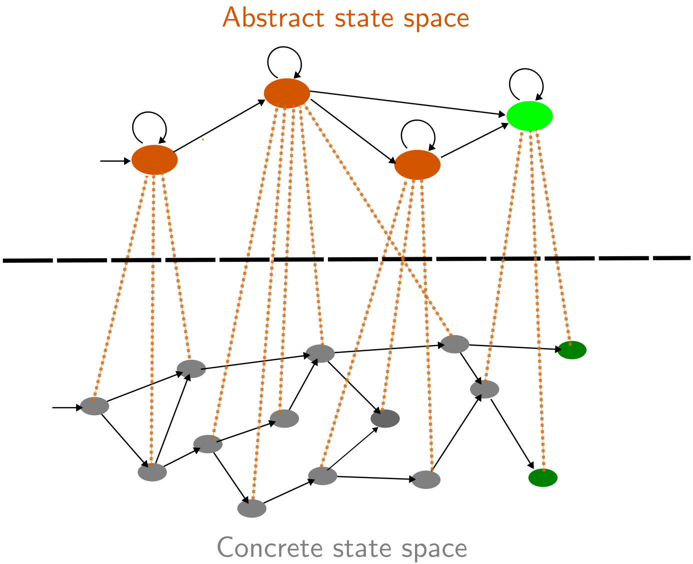
General Procedure for Obtaining a Heuristic

Solve a simplified version of the problem

Many ideas for computing domain independent planning heuristics

- **abstraction**
- delete relaxation
- landmarks
- critical paths
- network flows
- potential heuristics

Abstraction



Abstraction heuristics

- Paths are preserved in abstractions $\rightsquigarrow$ abstraction heuristics are admissible
- Competing objectives
  - informative heuristic, and
  - efficiently computable (small and succinctly encoded abstractions)

How we can find good abstractions?

Automatic computation of suitable abstractions

How we can find good abstractions?

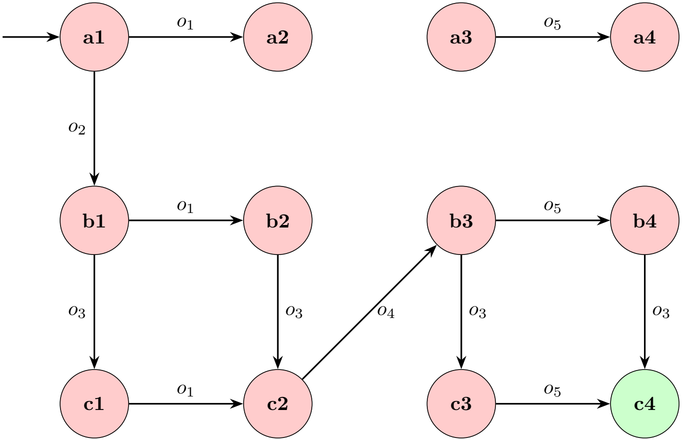
Several succesful methods

- Pattern databases (PDBs) or projections
  (Culberson and Schaeffer, 1996; Edelkamp, 2001; Haslum 2007)
- Domain abstractions (Hernádvölgyi and Holte, 2000)
- **Cartesian abstractions** (Seipp and Helmert, 2013)
- Merge & Srink abstractions (Dräger et al., 2006; Helmert, 2007; Sievers, 2014)

Example – concrete state space

Two variables v_1 and v_2

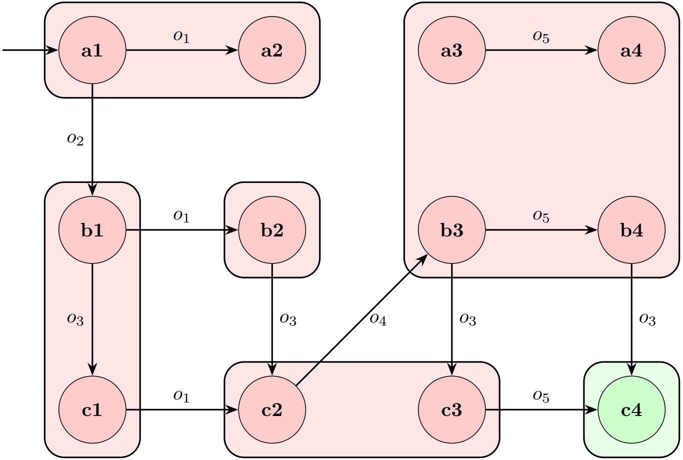
$dom(v_1) = \{a, b, c\}$
$dom(v_2) = \{1, 2, 3, 4\}$



Example – Cartesian abstraction

Each abstract state is a cross-product of variable domain subsets

$s_1 = \{a\} \times \{1, 2\}$
$s_2 = \{b, c\} \times \{1\}$
$s_3 = \{b\} \times \{2\}$
$s_4 = \{c\} \times \{2, 3\}$
$s_5 = \{a, b\} \times \{3, 4\}$
$s_6 = \{c\} \times \{4\}$



Automatic abstraction refinement

Counter-Example Guided Abstraction Refinement (CEGAR)

(Clarke et al., 2003; Seipp and Helmert, 2018)

CEGAR algorithm

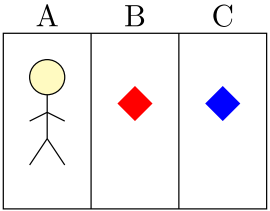
Start with single-state abstraction

Until a concrete solution is found or time runs out

1. Find abstract solution
2. Check if and why it fails for the concrete problem
3. Refine abstraction

Cartesian abstractions $\leadsto$ quick refinement operations

CEGAR example



Actions

moveAB, moveBA, moveBC, moveCB

$$\texttt{moveAB} : P = A \Longrightarrow P = B$$

pickRed, pickBlue

$$\texttt{pickBlue} : P = C, B = 0 \Longrightarrow B = 1$$

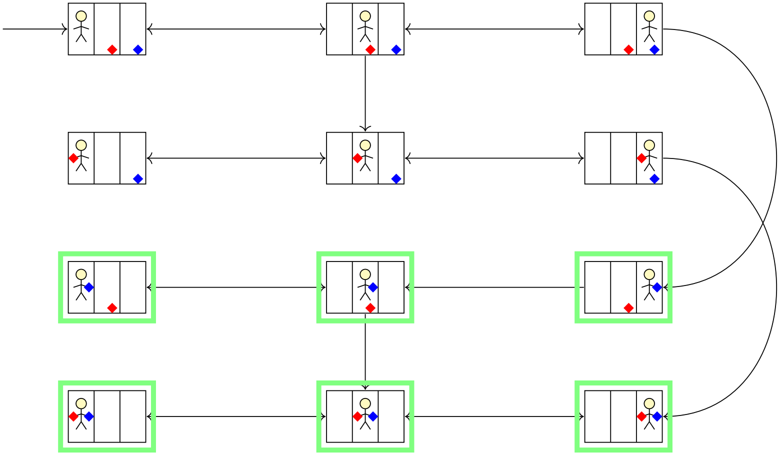
Variables

- Position (P), $dom(P) = \{A, B, C\}$
- HasRed (R), $dom(R) = \{0, 1\}$
- HasBlue (B), $dom(B) = \{0, 1\}$

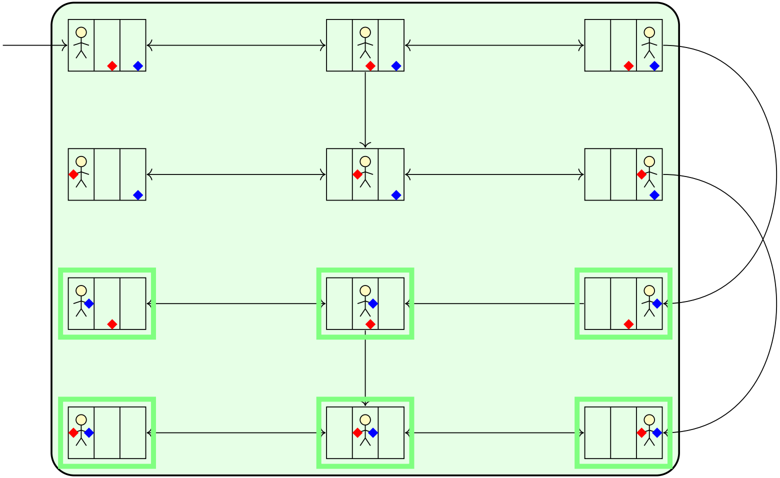
Initial state: $\langle A, 0, 0 \rangle$

Goal: $\langle ?, ?, 1 \rangle$

Example CEGAR – concrete state space

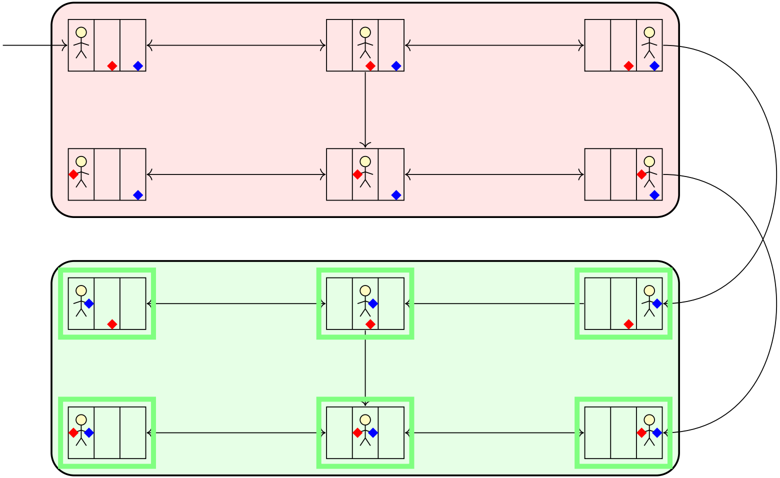


Example CEGAR – initialization – single-state abstraction



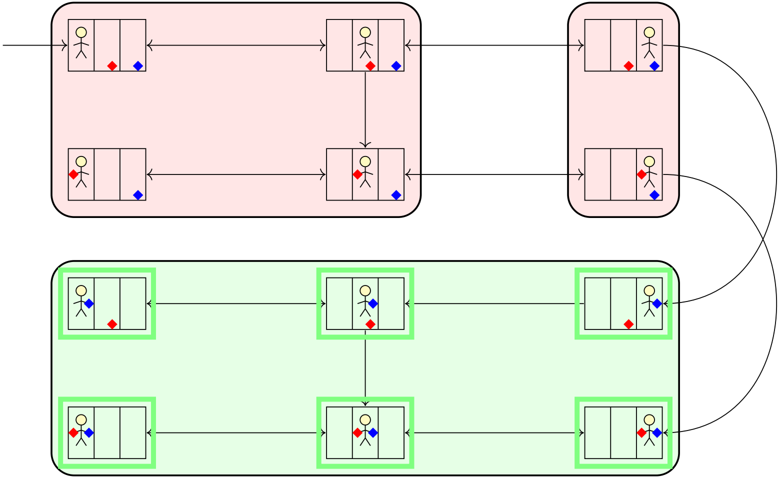
Abstract plan: $\emptyset$

Example CEGAR – iteration 1



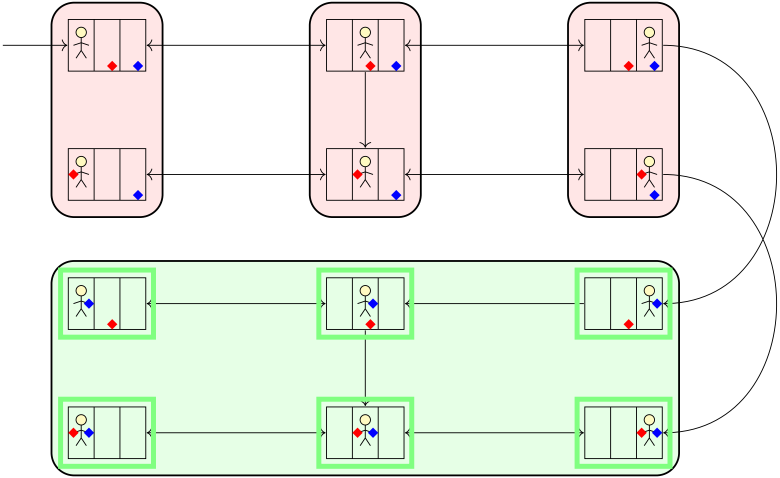
Abstract plan: `pickBlue`

Example CEGAR – iteration 2



Abstract plan: `moveBC`, `pickBlue`

Example CEGAR – iteration 3



Abstract plan: `moveAB`, `moveBC`, `pickBlue` concrete plan! $\rightsquigarrow$ h=3

Single Cartesian abstraction vs. multiple Cartesian abstractions

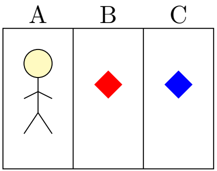
Single abstraction

- Problem: disminishing returns

Solution $\rightsquigarrow$ Multiple abstractions

- Diverse abstractions $\rightsquigarrow$ focus on different subproblems (e.g. one per goal)
- Combine heuristics admissibly $\rightsquigarrow$ cost partitioning

Combining heuristics – cost partitioning



Abstraction 1 (goal: blue diamond)

`moveAB`, `moveBC`, `pickBlue` ($h_1 = 3$)

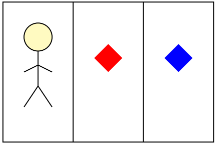
Abstraction 2 (goal: red diamond)

`moveAB`, `pickRed` ($h_2 = 2$)

Combine heuristics

- Addition $h = 5 \rightsquigarrow$ non-admissible heuristic ($h^* = 4$)
- Maximum $h = 3 \rightsquigarrow$ admissible but usually poor
- Solution $\rightsquigarrow$ addition with cost partitioning —for each action distribute its cost among the abstractions

Combining heuristics – cost partitioning



Abstraction 1 (goal: blue diamond)

moveAB, moveBC, pickBlue ($h_1 = 3$)

Abstraction 2 (goal: red diamond)

moveAB, pickRed ($h_2 = 2 \rightsquigarrow h_2 = 1$)

Cost partitioning example

	moveAB	moveBA	moveBC	moveCB	pickRed	pickBlue
Abstraction 1	1	1	1	1	0	1
Abstraction 2	0	0	0	0	1	0

Cost partitioning techniques

- uniform cost partitioning
- 0-1 cost partitioning
- optimal cost partitioning (Katz and Domshlak, 2010)
- posthoc optimization (Pommerening et al., 2013)
- saturated cost partitioning (Seipp et al. 2020)
- saturated posthoc optimization (Seipp et al. 2021)

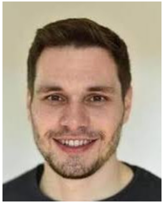
Summary

- Classical Planning as state space search
- General algorithms $\rightsquigarrow$ planning formalism, problem independent heuristics
- Computationally challenging
- Deriving heuristics
  - Cartesian abstractions
  - CEGAR: automatic generation of abstractions
  - Cost partitioning

Our current work

Merging Cartesian Abstractions for Classical Planning






Mauricio Salerno Raquel Fuentetaja David Speck Jendrik Seipp

Universidad Carlos III de Madrid Basel University Linköping University

Our current work – main idea

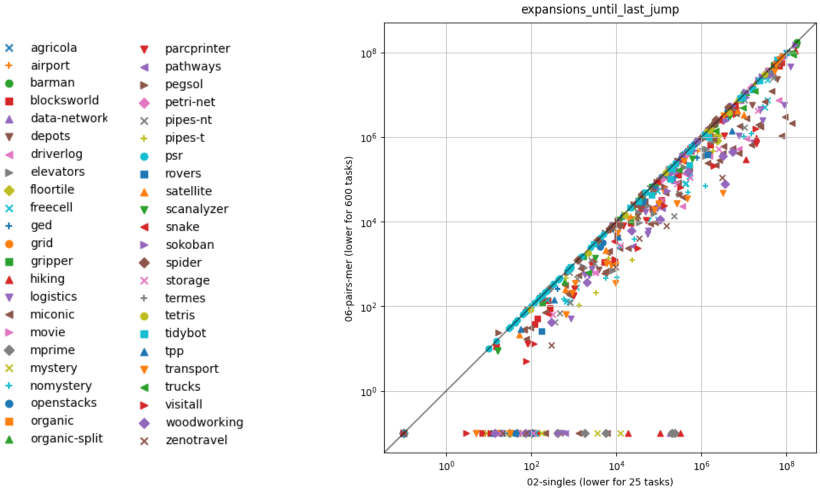
Before, for Cartesian abstractions

- One “big” abstraction
- Multiple “small” abstractions: one per goal

Now

Generate efficiently “medium-size” abstractions by merging small abstractions

Our current work – some results



Solved problems

844 —single abs

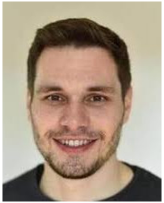
853 —1-goal abs

888 —2-goal abs

Our current work

Merging Cartesian Abstractions for Classical Planning






Mauricio Salerno Raquel Fuentetaja David Speck Jendrik Seipp

Universidad Carlos III de Madrid Basel University Linköping University