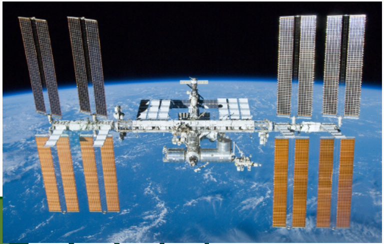
Counterexample-guided Cartesian Abstraction Refinement and Saturated Cost Partitioning for Optimal Classical Planning

Jendrik Seipp

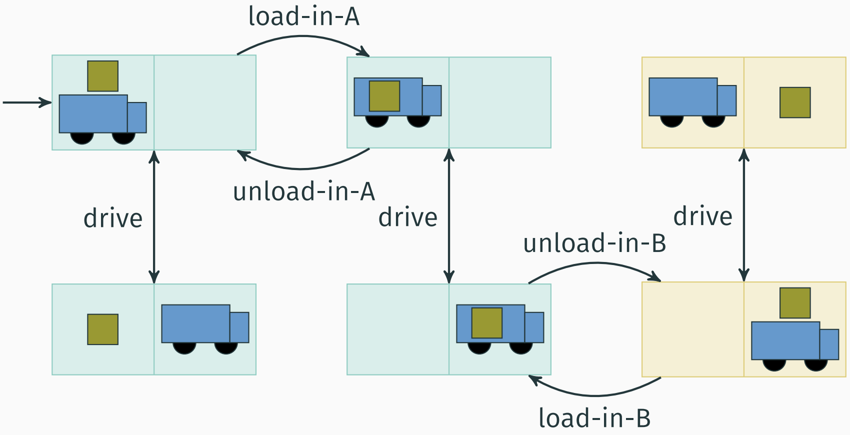
February 28, 2018

University of Basel

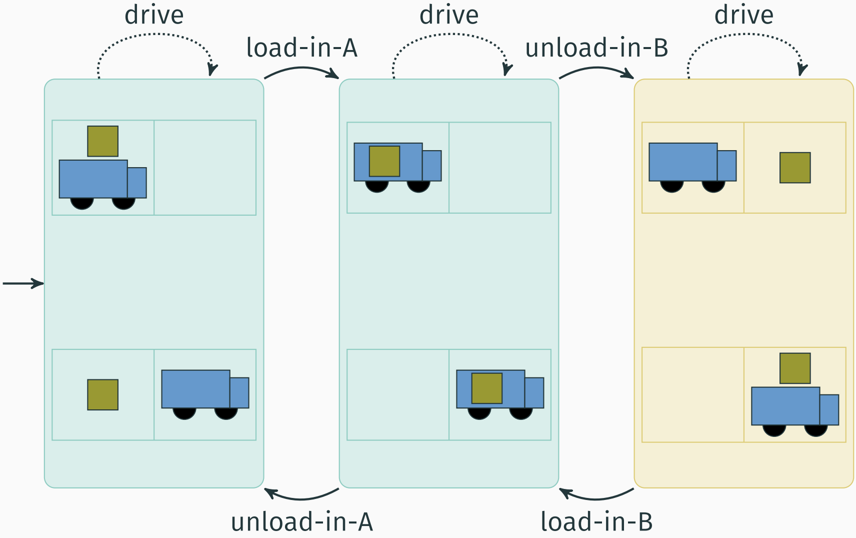
Planning



Find a sequence of actions that achieves a goal.


load-in-A
unload-in-A
drive
drive
drive
unload-in-B
load-in-B

Optimal Classical Planning: Example Abstraction



Abstraction Heuristics

- abstraction heuristics never overestimate $\rightarrow$ admissible
- A^* + admissible heuristic $\rightarrow$ optimal plan
- higher accuracy $\rightarrow$ better guidance

Abstraction Heuristics

- abstraction heuristics never overestimate $\rightarrow$ admissible
- A^* + admissible heuristic $\rightarrow$ optimal plan
- higher accuracy $\rightarrow$ better guidance
- how to create abstractions?

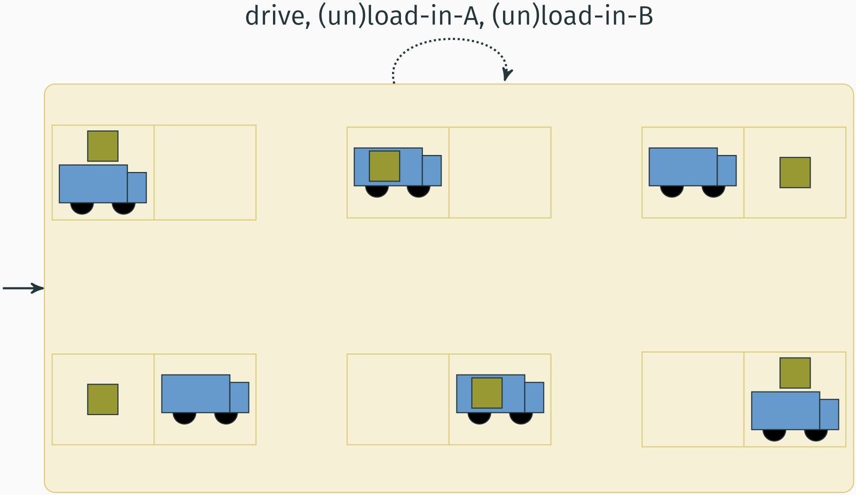
Counterexample-guided
Cartesian Abstraction Refinement

Counterexample-guided Abstraction Refinement (CEGAR)

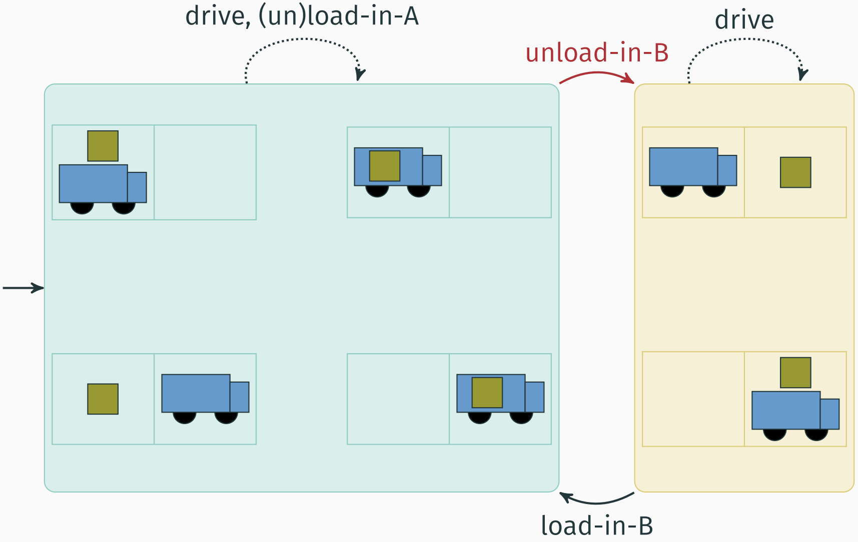
CEGAR Algorithm

- start with coarse abstraction
- until finding concrete solution or running out of time:
    - find abstract solution
    - check if and why it fails in the real world
    - refine abstraction

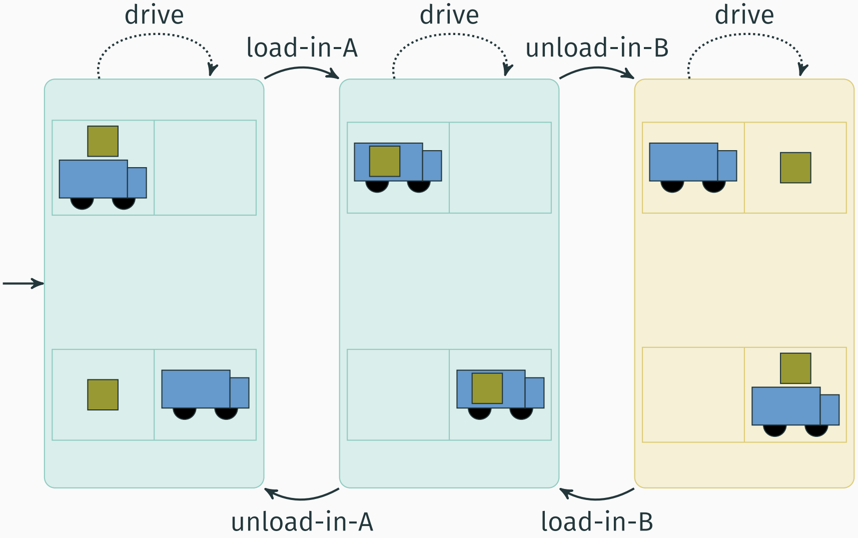
Example Refinement



Example Refinement



Example Refinement

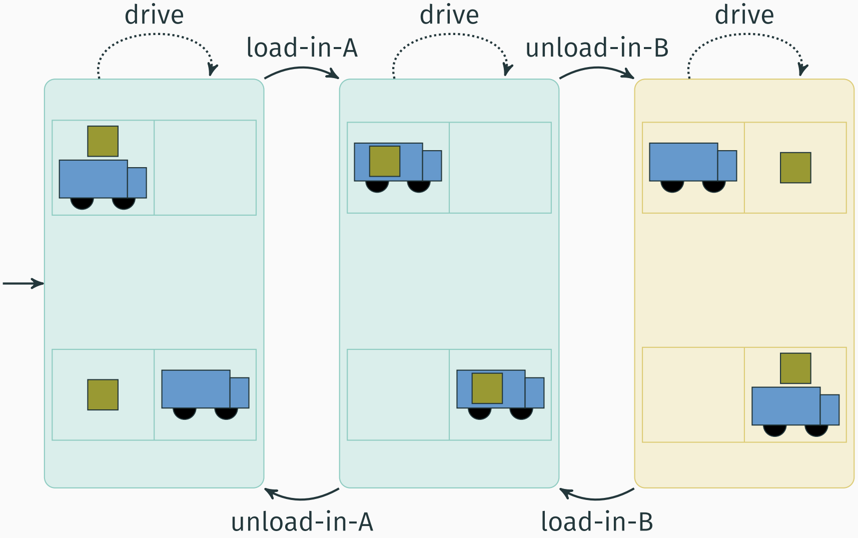


Classes of Abstractions

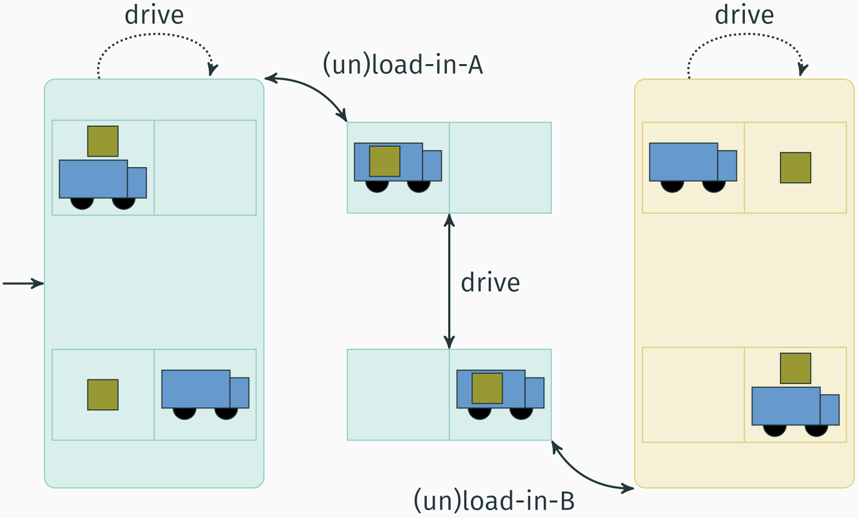
Cartesian Abstractions

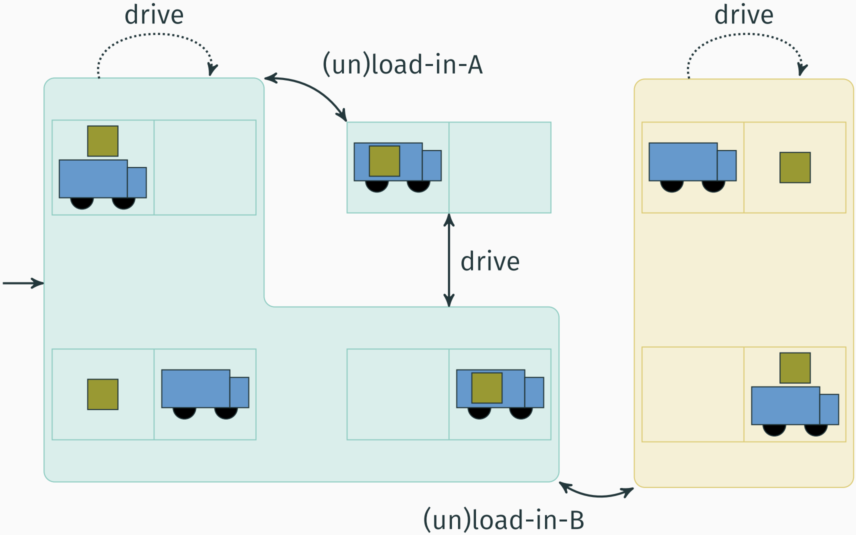
- relation to other classes of abstractions?

Projection (PDB)



Cartesian Abstraction

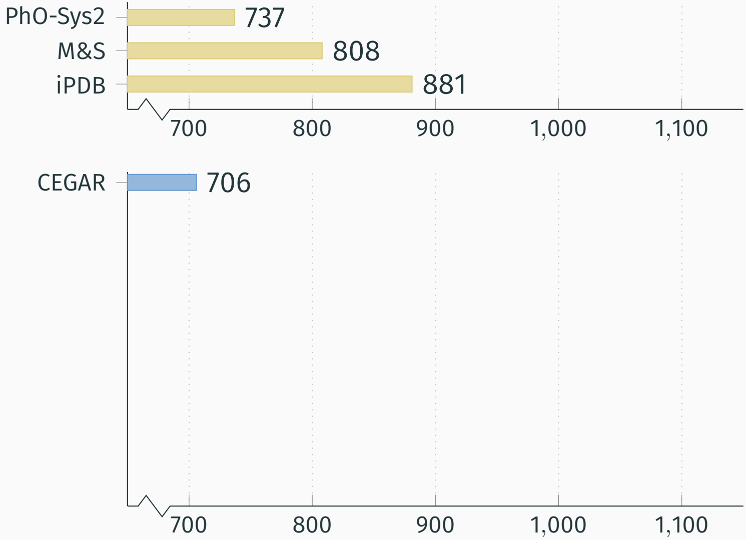



drive
(un)load-in-A
drive
drive
(un)load-in-B

Classes of Abstractions

- Projections (PDBs)
  refinement at least doubles number of states
- Cartesian Abstractions
  allow efficient and fine-grained refinement
- Merge-and-shrink Abstractions
  refinement complicated and expensive

Solved Tasks



CEGAR Drawbacks

Diminishing Returns

- finding solutions takes longer
- heuristic values only increase logarithmically

CEGAR Drawbacks

Diminishing Returns

- finding solutions takes longer
- heuristic values only increase logarithmically

$\rightarrow$ multiple smaller abstractions

Task Decomposition by Goals

- build abstraction for each goal fact

Task Decomposition by Goals

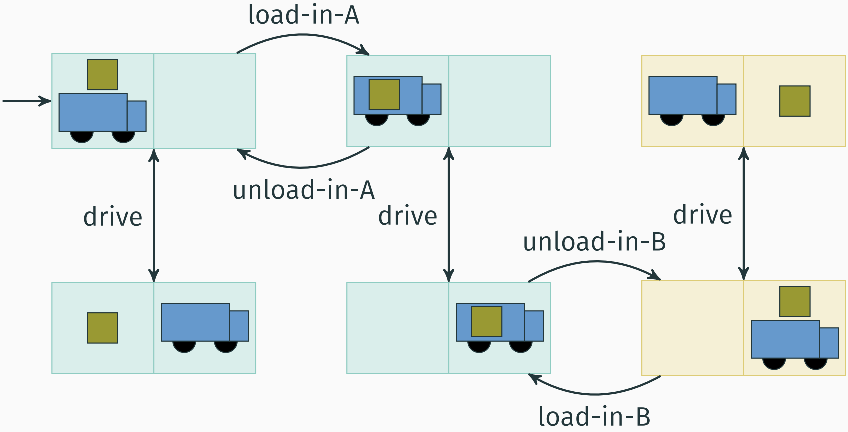
- build abstraction for each goal fact
- problem: tasks with single goal fact

Task Decomposition by Landmarks

- build abstraction for each fact landmark

Task Decomposition by Landmarks

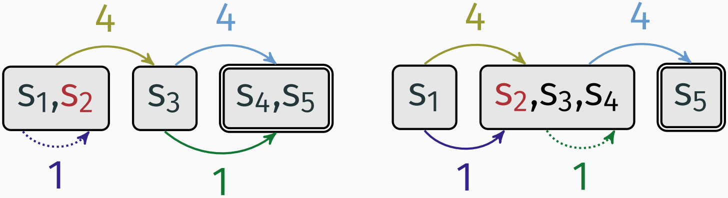
- build abstraction for each fact landmark



how to combine multiple heuristics?

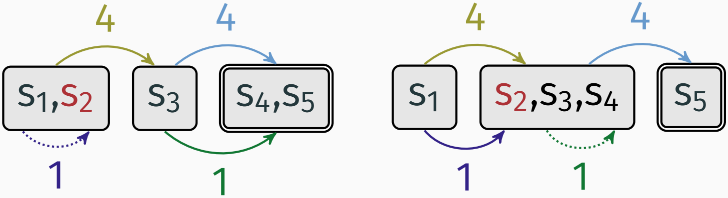
Multiple Heuristics

how to combine multiple heuristics?



Multiple Heuristics

how to combine multiple heuristics?

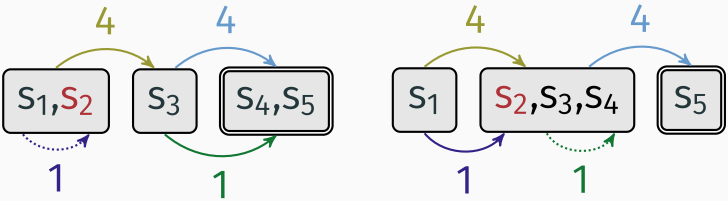


$h_1(s_2) = 5$

$h_2(s_2) = 4$

Multiple Heuristics

how to combine multiple heuristics?

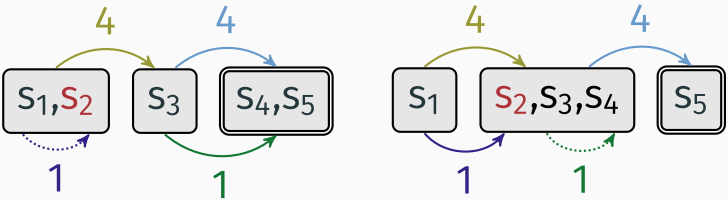


$h_1(s_2) = 5$ $h_2(s_2) = 4$

maximize over estimates:

- $h(s_2) = 5$

how to combine multiple heuristics?



$h_1(s_2) = 5$

$h_2(s_2) = 4$

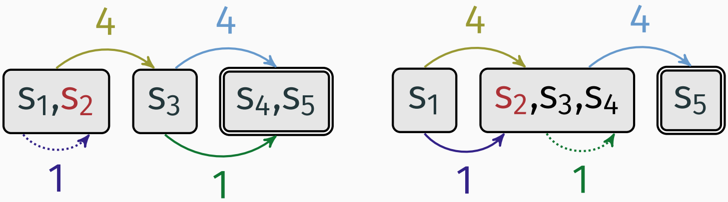
maximize over estimates:

- $h(s_2) = 5$
- only selects best heuristic
- does not combine heuristics

Multiple Heuristics: Cost Partitioning

Cost Partitioning

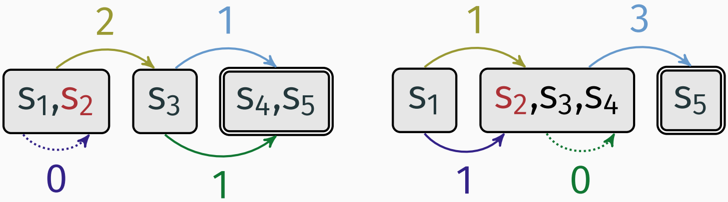
- split operator costs among heuristics
- sum of costs must not exceed original cost



Multiple Heuristics: Cost Partitioning

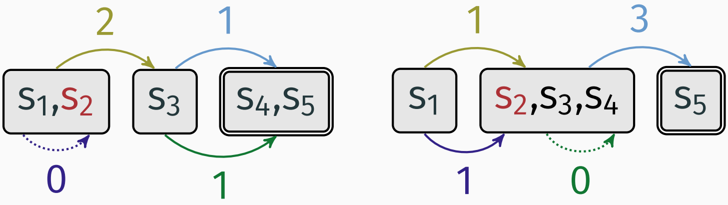
Cost Partitioning

- split operator costs among heuristics
- sum of costs must not exceed original cost



Cost Partitioning

- split operator costs among heuristics
- sum of costs must not exceed original cost

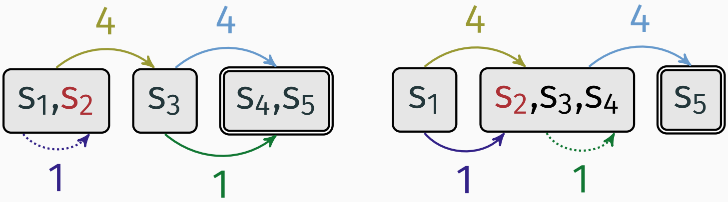


$$h(s_2) = 3 + 3 = 6$$

Saturated Cost Partitioning

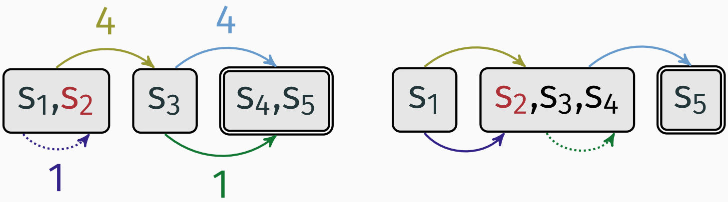
Saturated Cost Partitioning Algorithm

- order heuristics, then for each heuristic h:
    - use minimum costs preserving all estimates of h
    - use remaining costs for subsequent heuristics



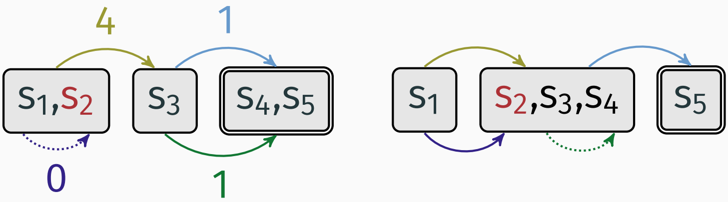
Saturated Cost Partitioning Algorithm

- order heuristics, then for each heuristic h:
    - use minimum costs preserving all estimates of h
    - use remaining costs for subsequent heuristics



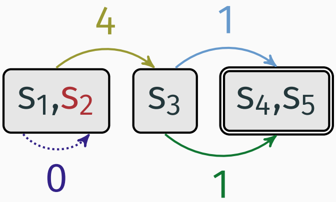
Saturated Cost Partitioning Algorithm

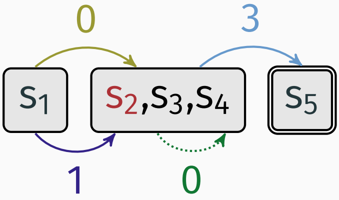
- order heuristics, then for each heuristic h:
  - use minimum costs preserving all estimates of h
  - use remaining costs for subsequent heuristics



Saturated Cost Partitioning Algorithm

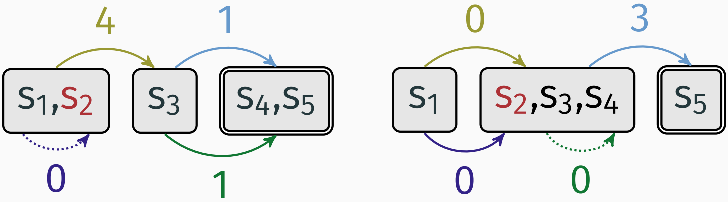
- order heuristics, then for each heuristic h:
  - use minimum costs preserving all estimates of h
  - use remaining costs for subsequent heuristics





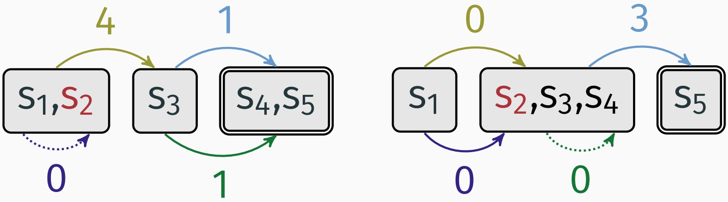
Saturated Cost Partitioning Algorithm

- order heuristics, then for each heuristic h:
  - use minimum costs preserving all estimates of h
  - use remaining costs for subsequent heuristics



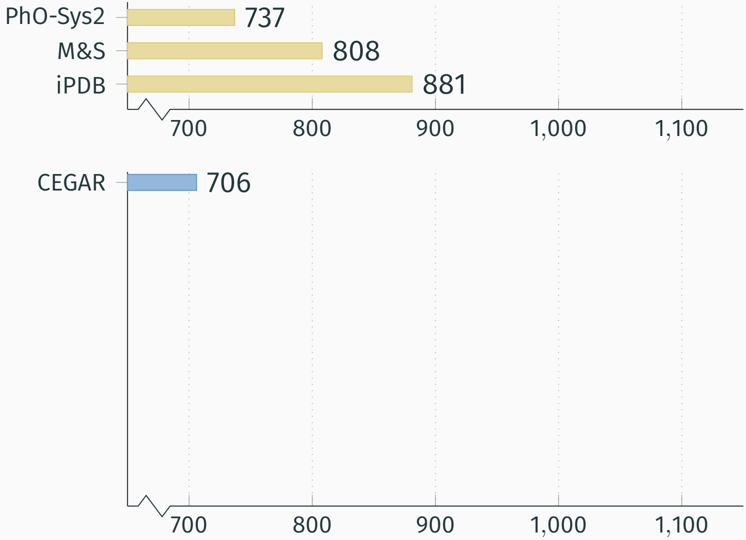
Saturated Cost Partitioning Algorithm

- order heuristics, then for each heuristic h:
  - use minimum costs preserving all estimates of h
  - use remaining costs for subsequent heuristics

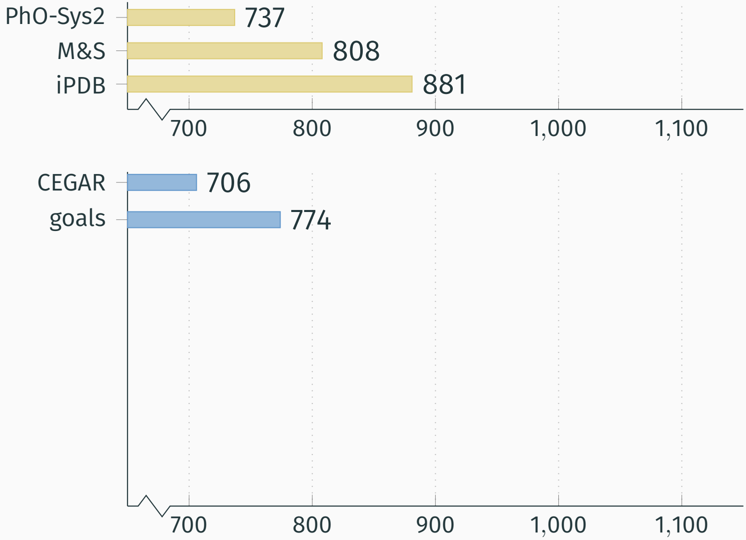


$$h(s_2) = 5 + 3 = 8$$

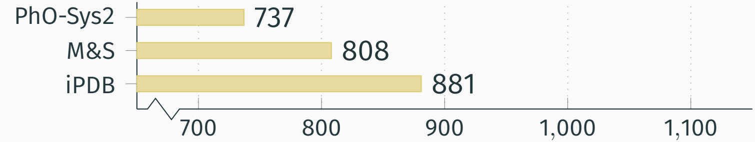
Solved Tasks

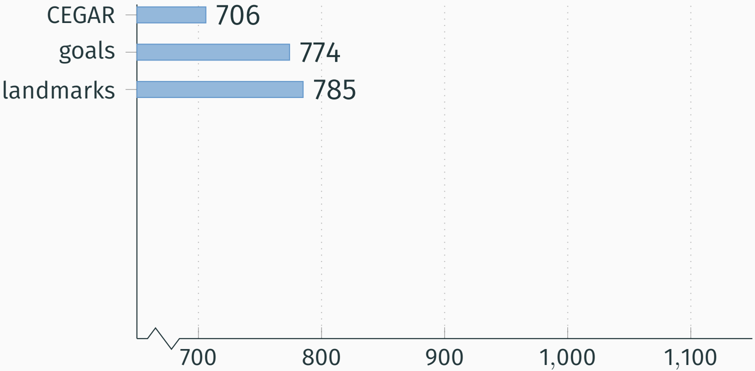


Solved Tasks

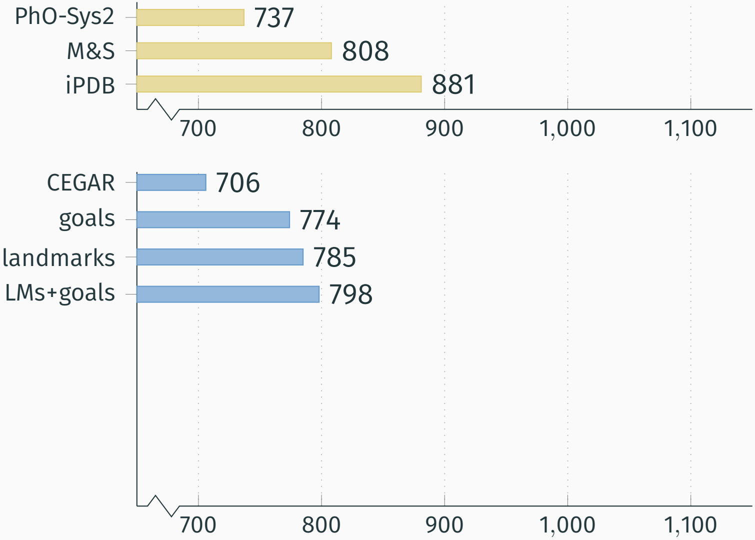


Solved Tasks

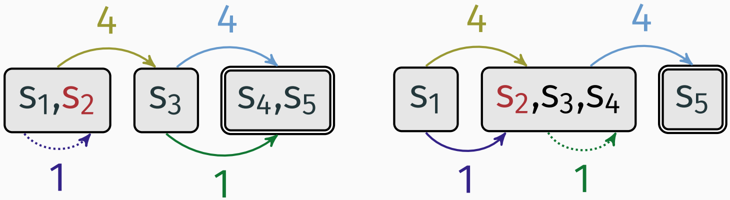




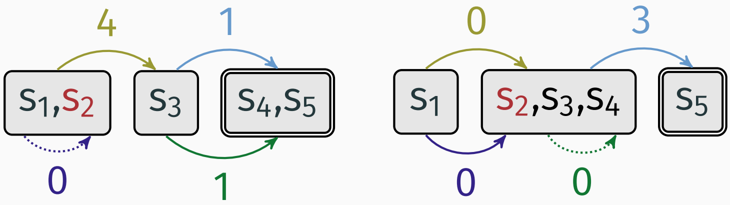
Solved Tasks



Order of Heuristics Is Important

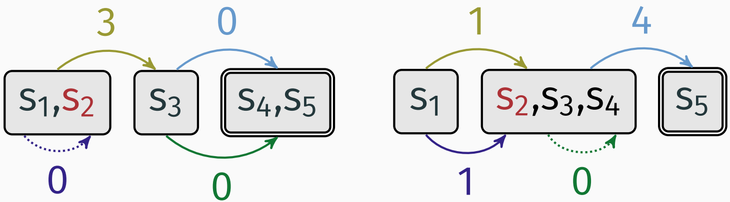


Order of Heuristics Is Important



$$h^{\mathrm{SCP}}_{\rightarrow}(s_2) = 5 + 3 = 8$$

Order of Heuristics Is Important



$$h^{\mathrm{SCP}}_{\rightarrow}(s_2) = 5 + 3 = 8$$

$$h^{\mathrm{SCP}}_{\leftarrow}(s_2) = 3 + 4 = 7$$

Finding a Good Order

- n heuristics $\rightarrow$ n! orders

Finding a Good Order

- n heuristics $\rightarrow$ n! orders

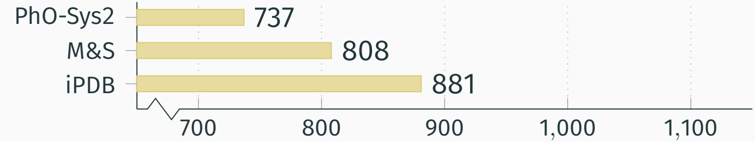
$\rightarrow$ search for good order: greedy initial order + optimization

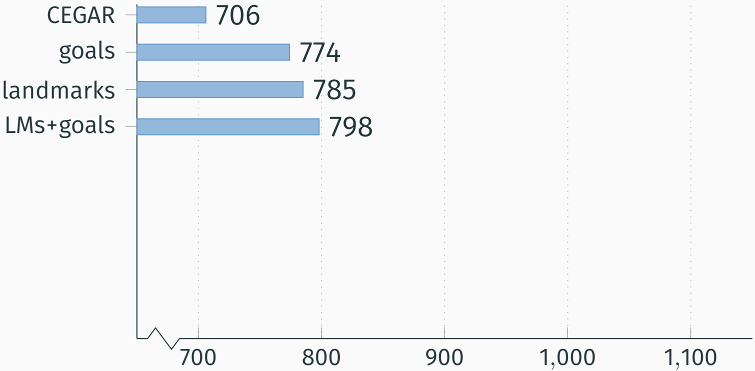
Goal: high estimates and low costs

Greedy Orders

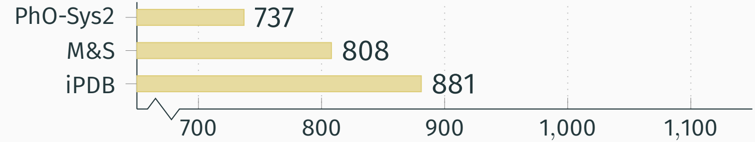
Goal: high estimates and low costs
$\rightarrow$ order by heuristic/costs ratio

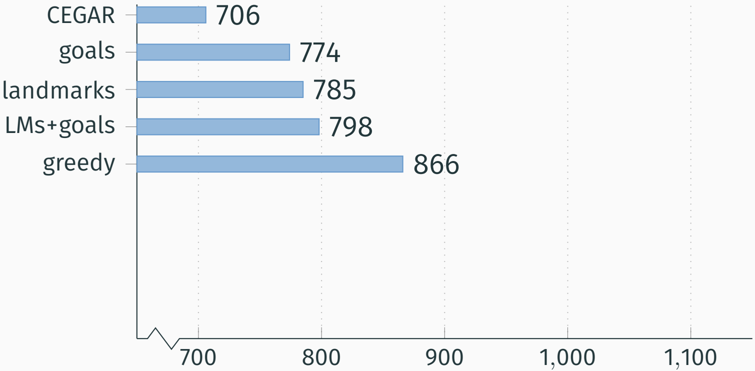
Solved Tasks





Solved Tasks





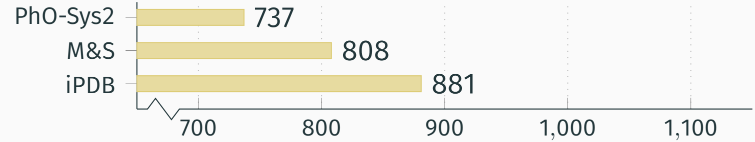
Optimized Orders

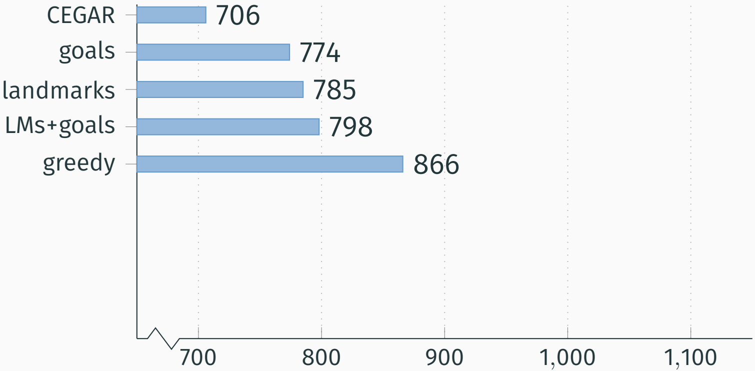
Optimization: finding initial order usually only first step

Hill-climbing Search

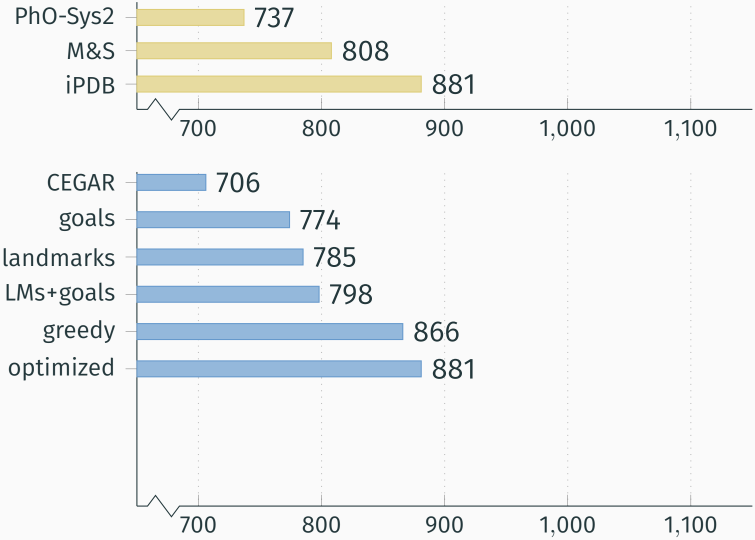
- start with initial order
- until no better successor found:
  - switch positions of two heuristics
  - commit to first improving successor

Solved Tasks

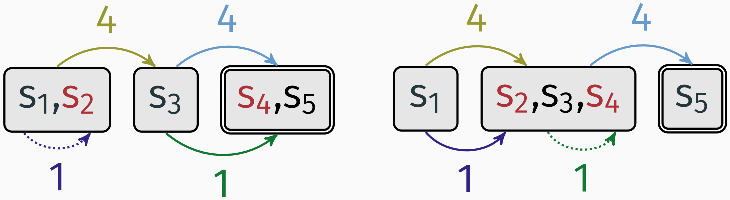




Solved Tasks



One Order Is Not Enough



$$h^{SCP}_{\rightarrow}(s_2) = 8$$
$$h^{SCP}_{\leftarrow}(s_2) = 7$$

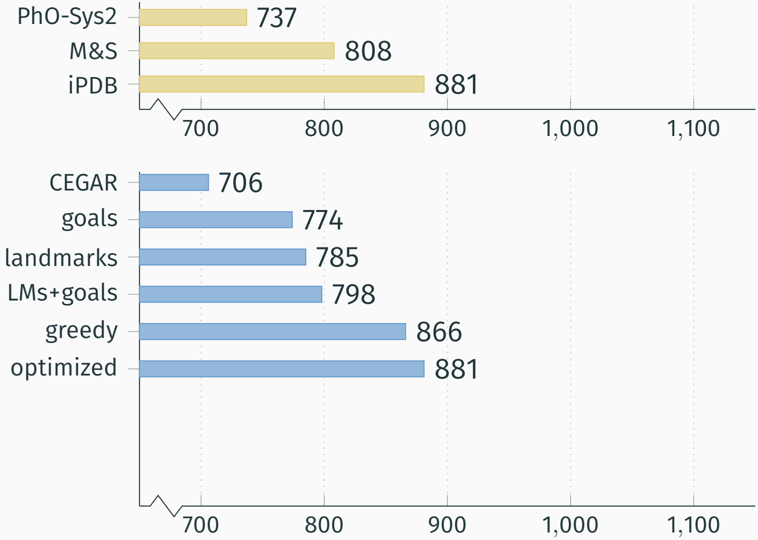
$$h^{SCP}_{\rightarrow}(s_4) = 3$$
$$h^{SCP}_{\leftarrow}(s_4) = 4$$

Multiple Orders

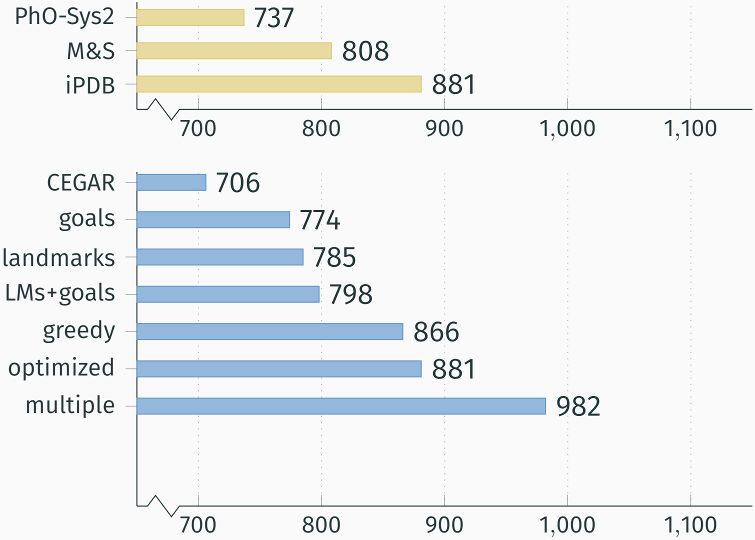
Approach:

- compute saturated cost partitioning for multiple orders
- maximize over heuristic estimates

Solved Tasks



Solved Tasks



Multiple Orders

Problems:

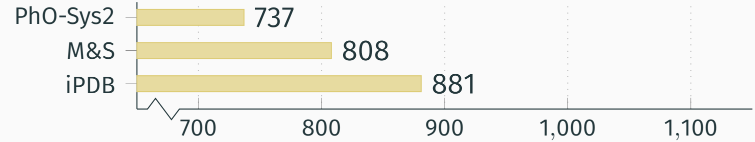
- many useless orders
- slow evaluation

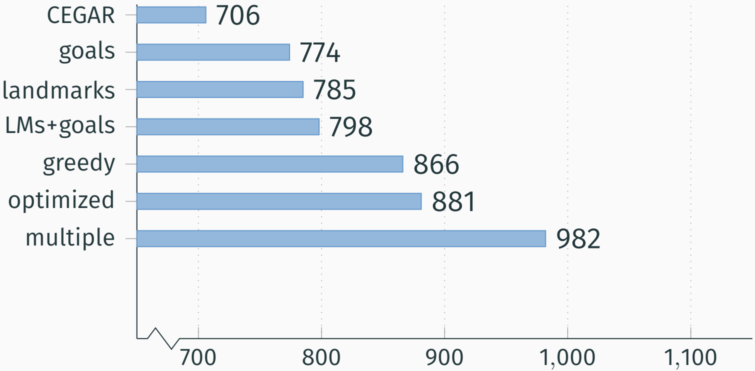
Diverse Orders

Diversification Algorithm

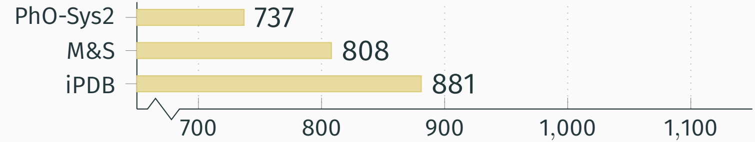
- sample 1000 states
- start with empty set of orders
- until time limit is reached:
    - generate an optimized order
    - if a sample profits from it, keep it
    - otherwise, discard it

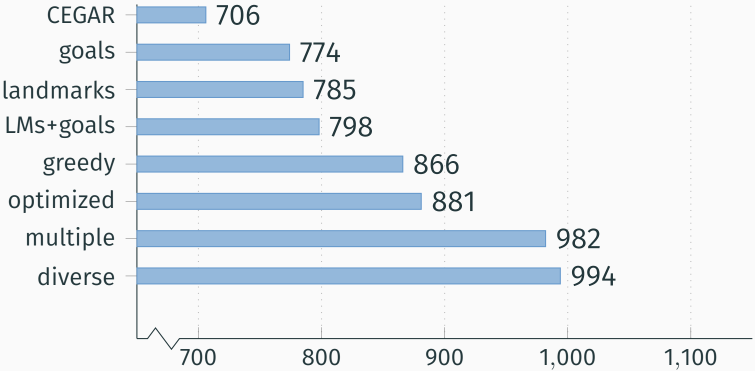
Solved Tasks





Solved Tasks





Comparison of Cost Partitioning Algorithms

Theoretical Comparison

UCP

Uniform Cost Partitioning

distribute costs evenly among relevant heuristics

GZOCP

UCP

Greedy Zero-one Cost Partitioning

order heuristics and give full cost to first relevant heuristic

Theoretical Comparison

GZOCP

PhO

UCP

Post-hoc Optimization

compute weight for each heuristic and return weighted sum

Theoretical Comparison

GZOCP

PhO

CAN

UCP

Canonical Heuristic

maximum over sums of independent heuristic subsets

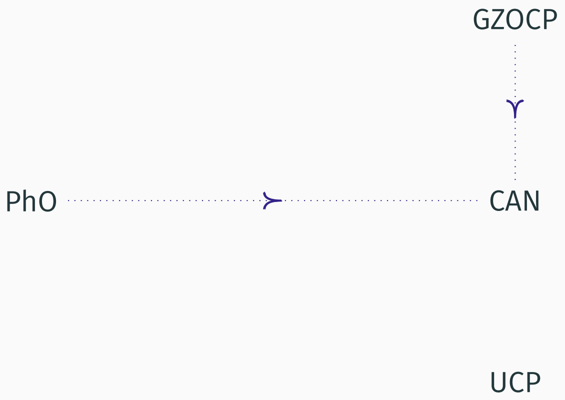
Theoretical Comparison

GZOCP

PhO $\succ$ CAN

UCP

Pommerening et al. 2013


GZOCP
PhO
CAN
UCP

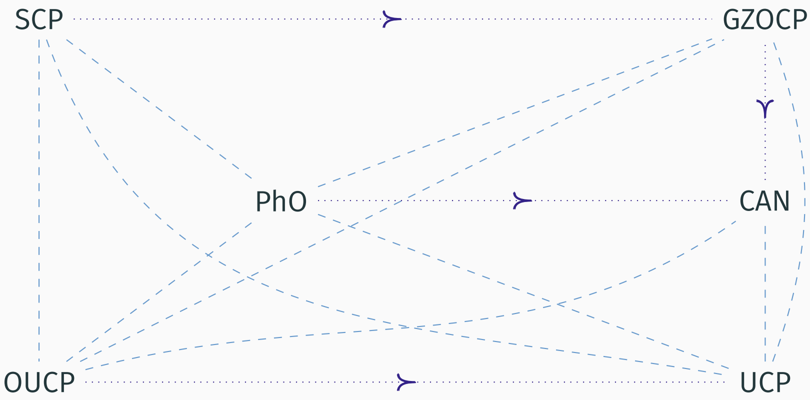
$$
\begin{tikzcd}
\mathrm{SCP} & & \mathrm{GZOCP} \arrow[d, dash, dotted, "\curlyvee" description] \\
& \mathrm{PhO} \arrow[r, dash, dotted, "\succ" description] & \mathrm{CAN} \\
& & \mathrm{UCP}
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathrm{SCP} \arrow[rr, dotted, dash, "\succ" marking] & & \mathrm{GZOCP} \arrow[d, dotted, dash, "\curlyvee" marking] \\
& \mathrm{PhO} \arrow[r, dotted, dash, "\succ" marking] & \mathrm{CAN} \\
& & \mathrm{UCP}
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathrm{SCP} \arrow[rr, dash, dotted, "\succ" description] & & \mathrm{GZOCP} \arrow[d, dash, dotted, "\curlyvee" description] \\
& \mathrm{PhO} \arrow[r, dash, dotted, "\succ" description] & \mathrm{CAN} \\
\mathrm{OUCP} & & \mathrm{UCP}
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathrm{SCP} \arrow[rr, dotted, "\succ" description] & & \mathrm{GZOCP} \arrow[d, dotted, "\curlyvee" description] \\
& \mathrm{PhO} \arrow[r, dotted, "\succ" description] & \mathrm{CAN} \\
\mathrm{OUCP} \arrow[rr, dotted, "\succ" description] & & \mathrm{UCP}
\end{tikzcd}
$$

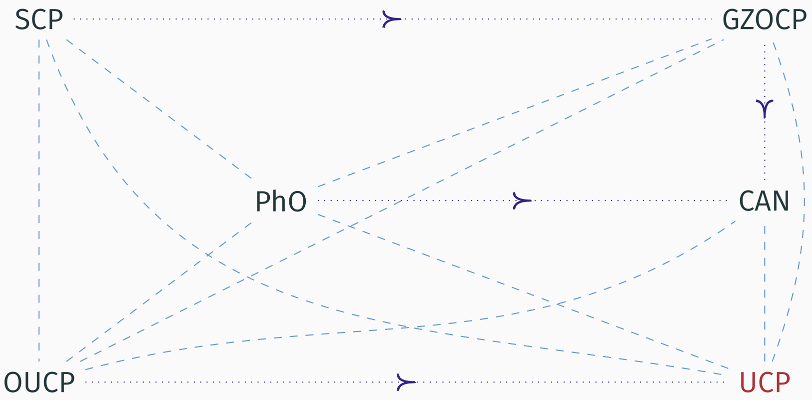
Theoretical Comparison



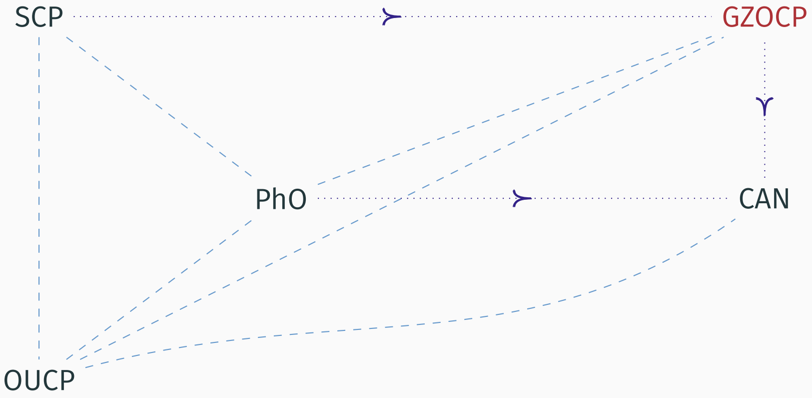
Experimental Comparison

- Heuristics: Cartesian abstraction heuristics + PDBs

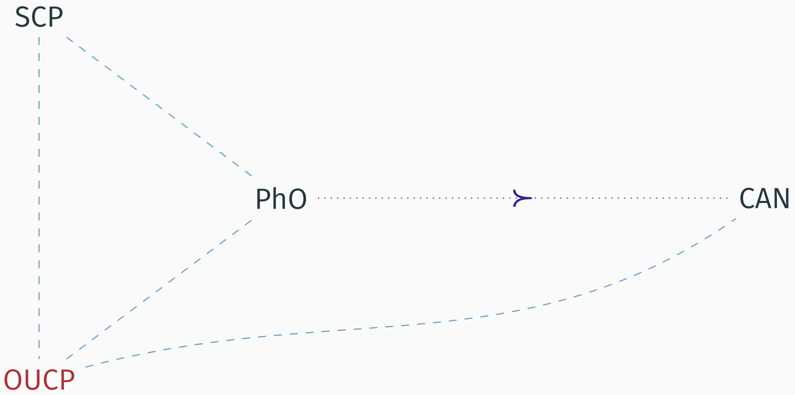
Experimental Comparison



Experimental Comparison



Experimental Comparison



Experimental Comparison

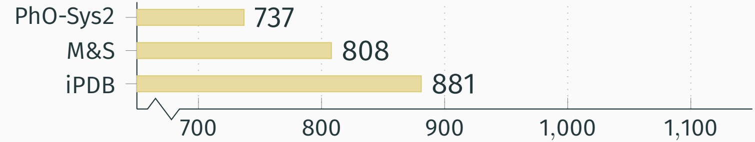
SCP

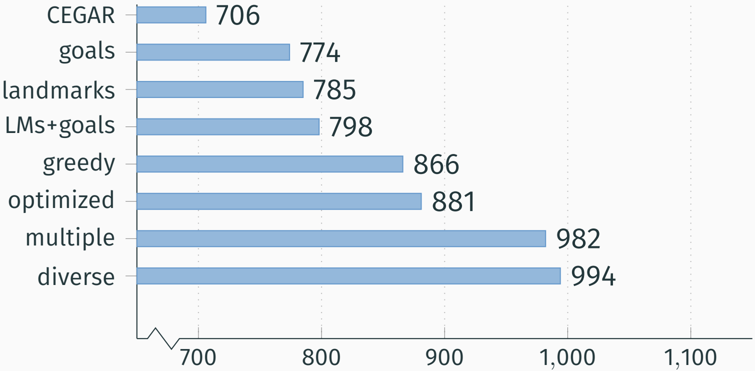
PhO $\succ$ CAN

Experimental Comparison

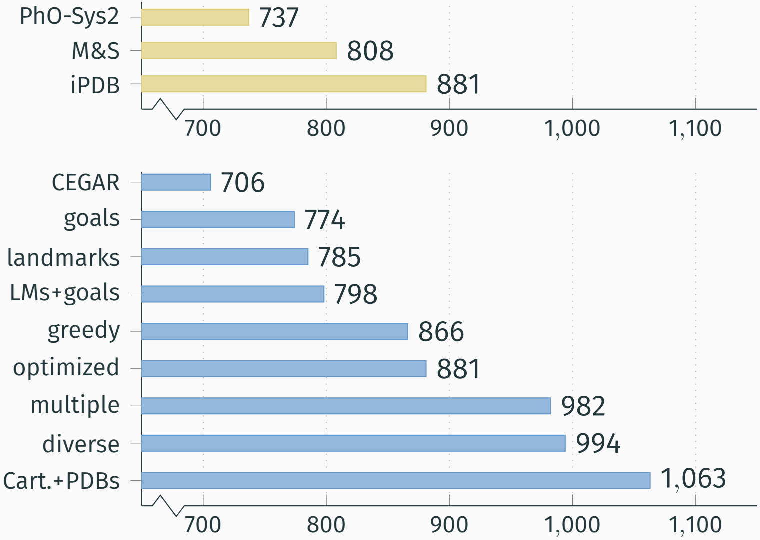
SCP

Solved Tasks





Solved Tasks



Conclusion

Counterexample-guided Cartesian Abstraction Refinement

- refines abstraction only where needed
- decompositions yield complementary heuristics

Conclusion

Counterexample-guided Cartesian Abstraction Refinement

- refines abstraction only where needed
- decompositions yield complementary heuristics

Saturated Cost Partitioning

- assigns each heuristic only the costs it needs
- best results for diverse optimized orders

Conclusion

Counterexample-guided Cartesian Abstraction Refinement

- refines abstraction only where needed
- decompositions yield complementary heuristics

Saturated Cost Partitioning

- assigns each heuristic only the costs it needs
- best results for diverse optimized orders

Comparison of Cost Partitioning Algorithms

- dominances and non-dominances
- saturated cost partitioning preferable in all settings