University of Basel

Preferred Operators for Abstraction Heuristics

Carina Fehr

Department of Mathematics and Computer Science, University of Basel

June 15, 2026

Background

Planning task

- Variables: variable assignments are states
- Operators: allow transitions between states
- Initial state
- Goal states

Background

Planning task

- Variables: variable assignments are states
- Operators: allow transitions between states
- Initial state
- Goal states

Heuristic

Estimates distance from a given state to the closest goal state

Background

Greedy Best-First search (GBFS)

- Only consider the heuristic
- Suboptimal planning algorithm

Eager GBFS:
Computes heuristic values for all successors

Lazy GBFS:
uses the parent's heuristic value and defers heuristic evaluation

Background

Greedy Best-First search (GBFS)

- Only consider the heuristic
- Suboptimal planning algorithm

Eager GBFS:
Computes heuristic values for all successors

Lazy GBFS:
uses the parent's heuristic value and defers heuristic evaluation

Preferred operators (POs)

Operators that appear promising in a given state

Motivation

- So far, preferred operators have only been tested for inadmissible heuristics
- We test them for abstraction heuristics
- Is the improved guidance worth the increased time and memory costs?

Motivation

- So far, preferred operators have only been tested for inadmissible heuristics
- We test them for abstraction heuristics
- Is the improved guidance worth the increased time and memory costs?

Main question

Can preferred operators in abstraction heuristics improve the performance of GBFS?

Abstraction heuristics

- Abstractions remove distinctions between certain states
- Solve a simplified version of the planning task
- Use optimal distance in abstract state space as heuristic value

Abstraction heuristics

- Pattern Databases (PDBs)
  - Projection to a subset of state variables
  - Look-up table stores heuristic values

Abstraction heuristics

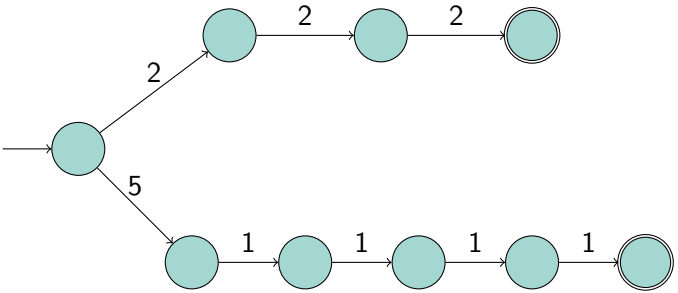
- Pattern Databases (PDBs)
  - Projection to a subset of state variables
  - Look-up table stores heuristic values
- iPDB
  - Collection of multiple patterns selected with hill-climbing search
  - Combined using the canonical heuristic

Abstraction heuristics

- Pattern Databases (PDBs)
  - Projection to a subset of state variables
  - Look-up table stores heuristic values
- iPDB
  - Collection of multiple patterns selected with hill-climbing search
  - Combined using the canonical heuristic
- Cartesian Abstractions by CEGAR
  - Start with one coarse abstraction
  - Iteratively refine the abstraction by refining flaws
  - Abstract state represented as Cartesian set of concrete states

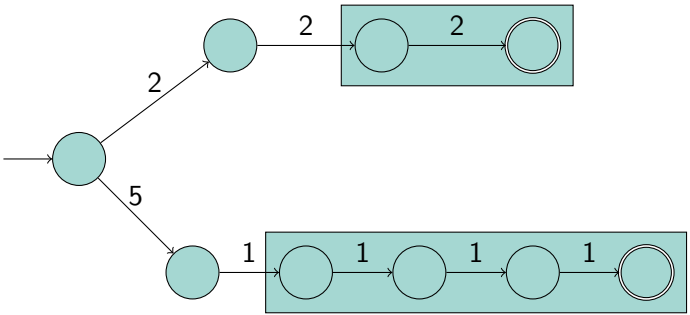
Abstraction heuristics: example

Concrete state space



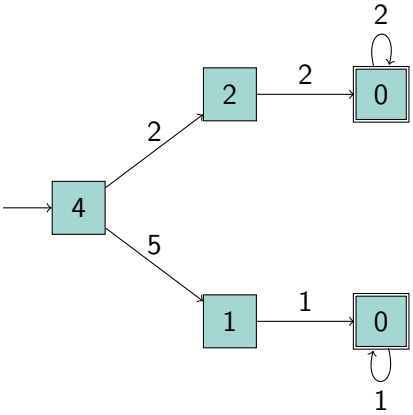
Abstraction heuristics: example

Concrete state space

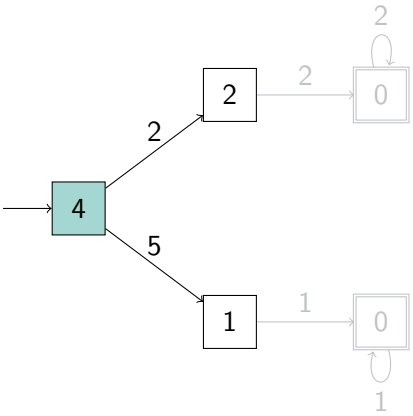


Abstraction heuristics: example

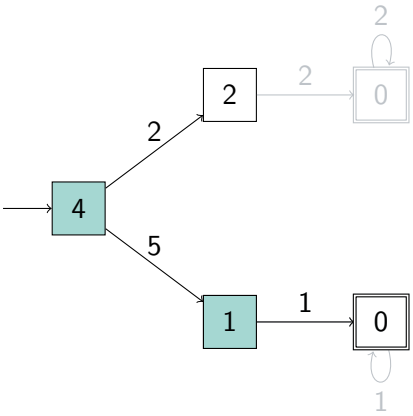
Abstract state space



GBFS: example

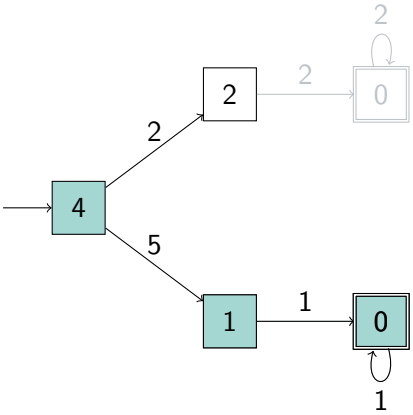


GBFS: example



GBFS: example

Expansions in concrete state space: 6



Preferred operators

- Operators lying on an optimal abstract plan
- Used as additional search guidance
- Successors generated by POs expanded earlier on average
- Two versions implemented

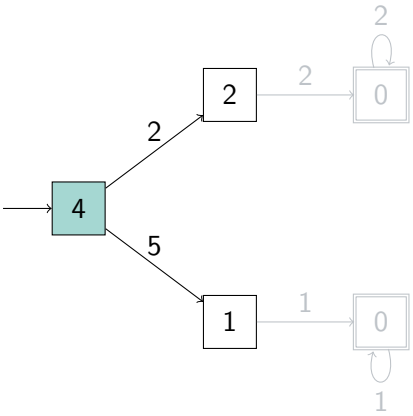
Precomputed preferred operators

- Computed before actual search starts
- Stored for every abstract state
- Identified during backward Dijkstra loop in pattern database construction

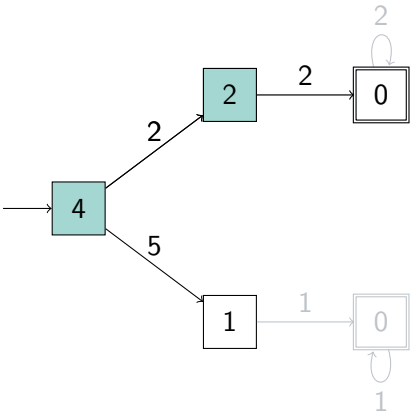
Live preferred operators

- Computed during search
- Operator lies on optimal plan if $h(s) = cost(operator) + h(s')$
- Computed only for generated states
- No additional storage required

Preferred operators: example

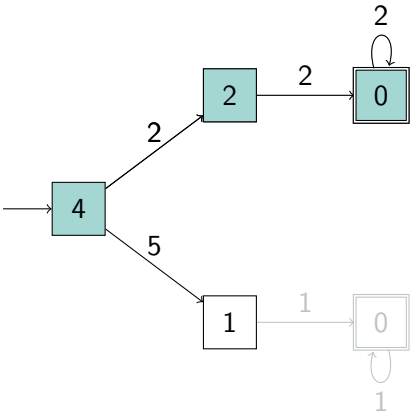


Preferred operators: example



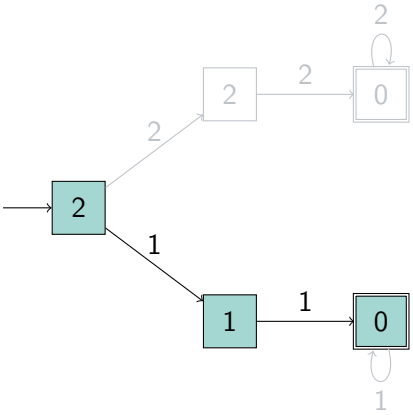
Preferred operators: example

Expansions in concrete state space: 4



Preferred operators: another example

POs and GBFS same guidance

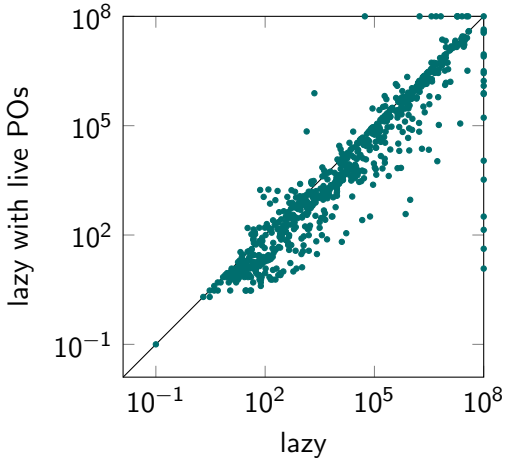


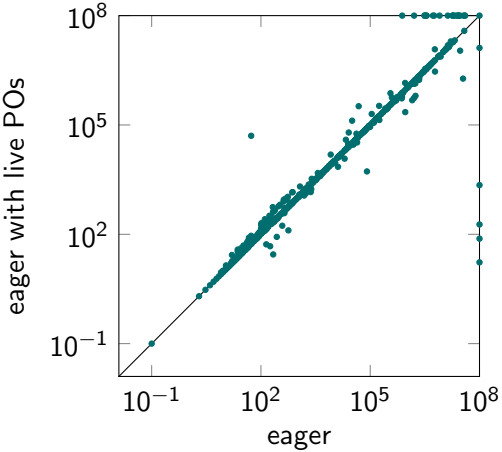
Experiments

Experimental setup

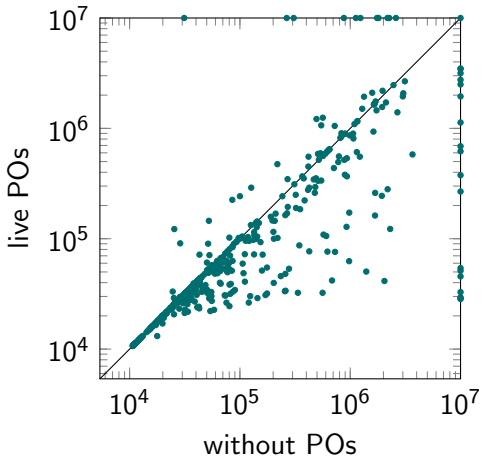
- IPC satisficing benchmarks
- Exclude domains that use axioms and conditional effects
- Time limit of 5 minutes per task
- Memory limit of 3584 MB per task

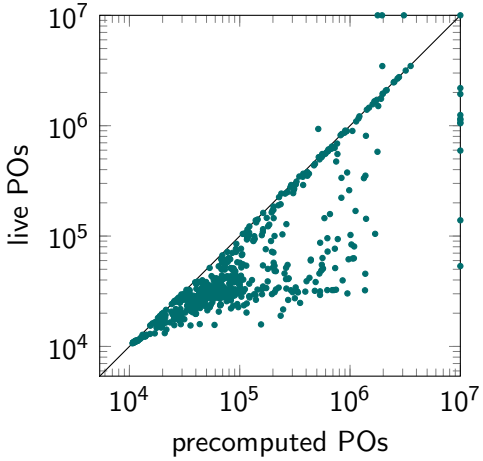
Expansions in eager and lazy GBFS



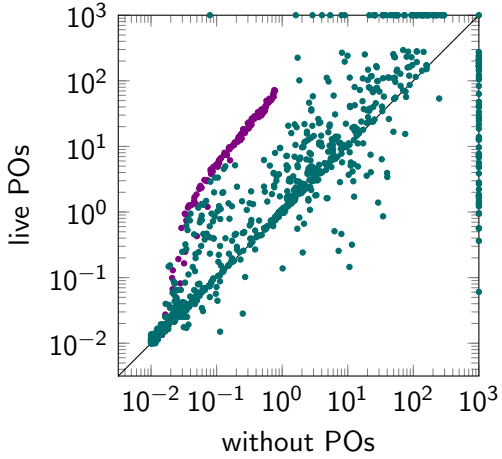


Memory with live and precomputed POs





Total time with Cartesian abstractions



- Coverage increases
- Miconic domain is a negative outlier
- Usefulness of POs is domain-dependent

Eager GBFS vs. lazy GBFS with POs

Metric	eager GBFS	lazy GBFS with POs
Coverage	**852**	836
expansions	**519.80**	509.36
memory	498.82	**499.00**
total time	**736.91**	720.18
time per expansion (ms)	0.429	**0.397**

Conclusion

Precomputed POs	live POs
high memory consumption	lower memory consumption
high preprocessing cost	low preprocessing cost
Cannot improve total time	can lower total time
worse overall performance	better overall performance

Conclusion

- Eager GBFS: no positive impact with POs on any metric
- Lazy GBFS: POs have positive impact on guidance
- Generally, weak guidance can benefit more from POs
- Improved guidance is partially offset by computational overhead

Conclusion

- Eager GBFS: no positive impact with POs on any metric
- Lazy GBFS: POs have positive impact on guidance
- Generally, weak guidance can benefit more from POs
- Improved guidance is partially offset by computational overhead

Can preferred operators in abstraction heuristics improve the performance of GBFS?

Yes, but only in specific settings and with limited overall effect.

Thank you for the attention!
Questions?