Exploring The Prioritized Incremental Heuristic

Daniel Weissen, 02.11.2020

Motivation

- Finding a plan for heuristic search problems
- 80% of time spent calculating heuristic values
- Improving Heuristic calculation has a big effect on planning time
- The Prioritized Incremental Heuristic (PINCH) is an alternative way of implementing the Additive Heuristic

Content

Classical Planning

- Problem described in a planning domain language (PDDL)
- Goal: Find plan from initial state to goal state
- Method: Heuristic search algorithms

How do we extract heuristic values from encoded planning problems?

Relaxation Heuristics

- Consider relaxed version of planning task (all delete effects of actions are ignored)
- Use approximation of optimal relaxed planning cost as heuristic values
- The additive heuristic is one way of approximating the optimal relaxed planning cost

The Additive Heuristic

For Each $q \in V \cup A$:

$$x_v = \begin{cases} 0 & \text{if } v \in s \\ \min_{a \in A \mid v \in add(a)} [cost(a) + x_a] & \text{otherwise} \end{cases}$$

$$x_a = \sum_{v \in pre(a)} x_v$$

$$h^{add}(s) = \sum_{v \in G} x_v$$

Different Implementations Of The Additive Heuristic: Value Iteration (VI)

- Iterate over all variables and actions
- Update cost values according to equations
- Stop when no value changes during an iteration

$$x_v = \begin{cases} 0 & \text{if } v \in s \\ \min_{a \in A \mid v \in add(a)} [cost(a) + x_a] & \text{otherwise} \end{cases}$$

$$x_a = \sum_{v \in pre(a)} x_v$$

Different Implementations Of The Additive Heuristic: Value Iteration with Value Ordering

- Algorithms that order value updates
- Generalized Dijkstra (GD) orders value updates fully

Different Implementations Of The Additive Heuristic: Generalized Dijkstra (GD)

- Uses priority queue to track value ordering
- Updates heuristic values exactly once

Different Implementations Of The Additive Heuristic: Incremental Value Iteration (IVI)

- Similar to VI
- Does not reinitialize cost values
- Calls VI if iterations threshold is reached

Content

PINCH, The Best Of Both Worlds

- Algorithm introduce by Yaxin Liu, Sven Koenig and David Furcy in their 2002 paper ”Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning”
- Algorithm for computing the additive heuristic
- PINCH combines the benefits of Value Ordering and IVI
- Incremental GD
- **Updates cost values at most twice**

PINCH: Example

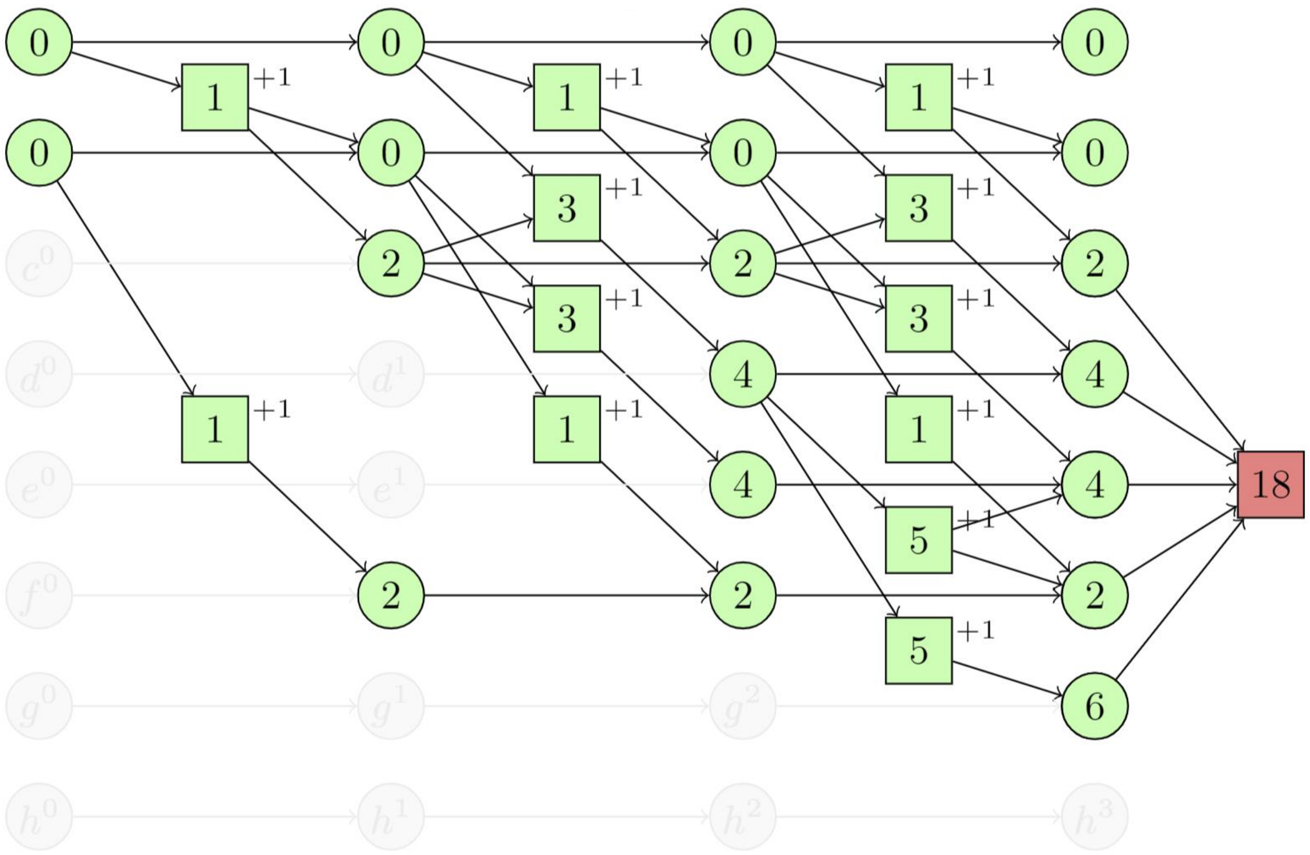
I want to explain PINCH by taking the algorithm through an example. Lets consider the following planning task $\Pi^+ = \langle V, I, G, A \rangle$ with:

- $V = \{a, b, c, d, e, f, g\}$
- $I = \{a, b\}$
- $G = \{c, d, e, f, g\}$
- $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$
- $a_1 = a \xrightarrow{1} b, c$
- $a_2 = a, c \xrightarrow{1} d$
- $a_3 = b, c \xrightarrow{1} e$
- $a_4 = b \xrightarrow{1} f$
- $a_5 = d \xrightarrow{1} e, f$
- $a_6 = d \xrightarrow{1} g$

PINCH: Example

- $s' = I = \{a, b\}$
- $s = \{a, c\}$

PINCH: Example



$h^{\text{add}}(s') = 18/2 = 9$

old state s'		
q	x_q	rhs_q
a	0	0
b	0	0
c	2	2
d	4	4
e	4	4
f	2	2
g	6	6
a_1	1	1
a_2	3	3
a_3	3	3
a_4	1	1
a_5	5	5
a_6	5	5

New Equations

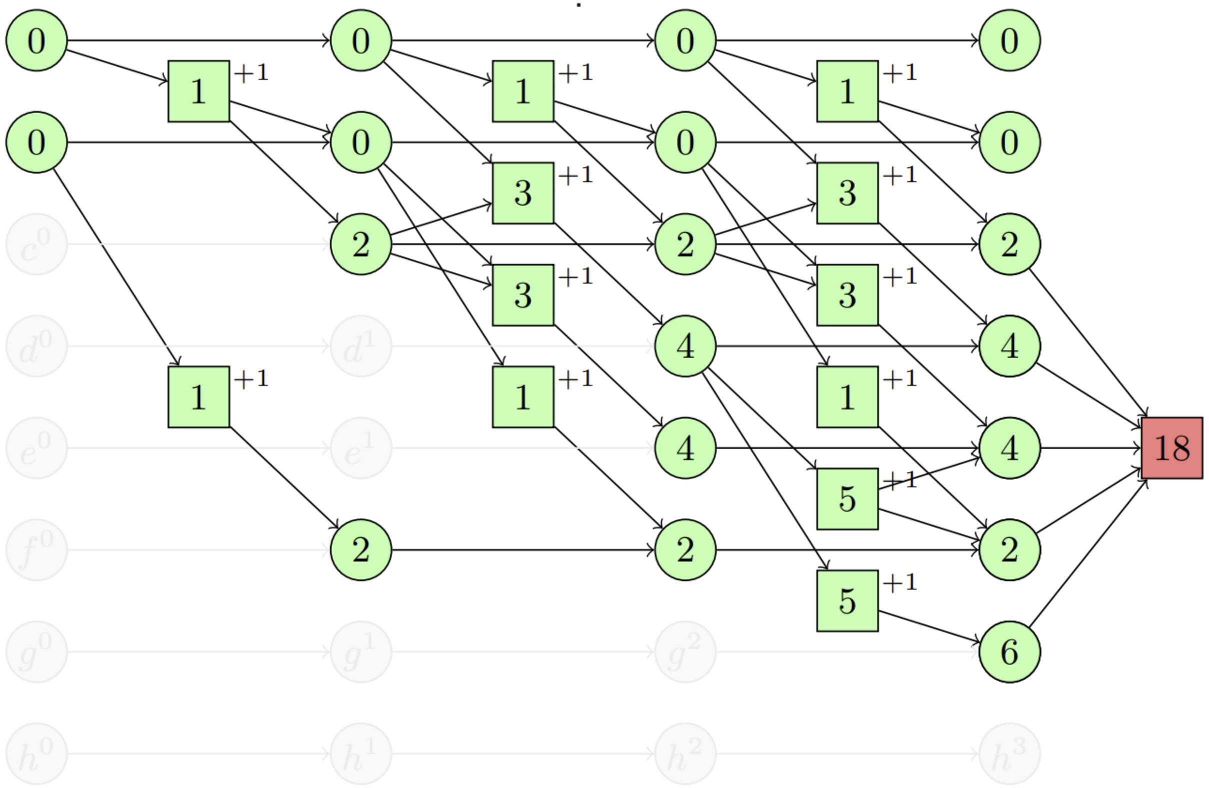
$$x'_v = \begin{cases} 0 & \text{if } v \in s \\ \min_{a \in A \mid v \in add(a)}[1 + x'_a] & \text{otherwise} \end{cases}$$

$$x'_a = 1 + \sum_{v \in pre(a)} x'_v$$

$$h^{add}(s) = \sum_{v \in G} x_v = 1/2 \sum_{v \in G} x'_v$$

PINCH: Example

- $s' = I = \{a, b\}$
- $s = \{a, c\}$

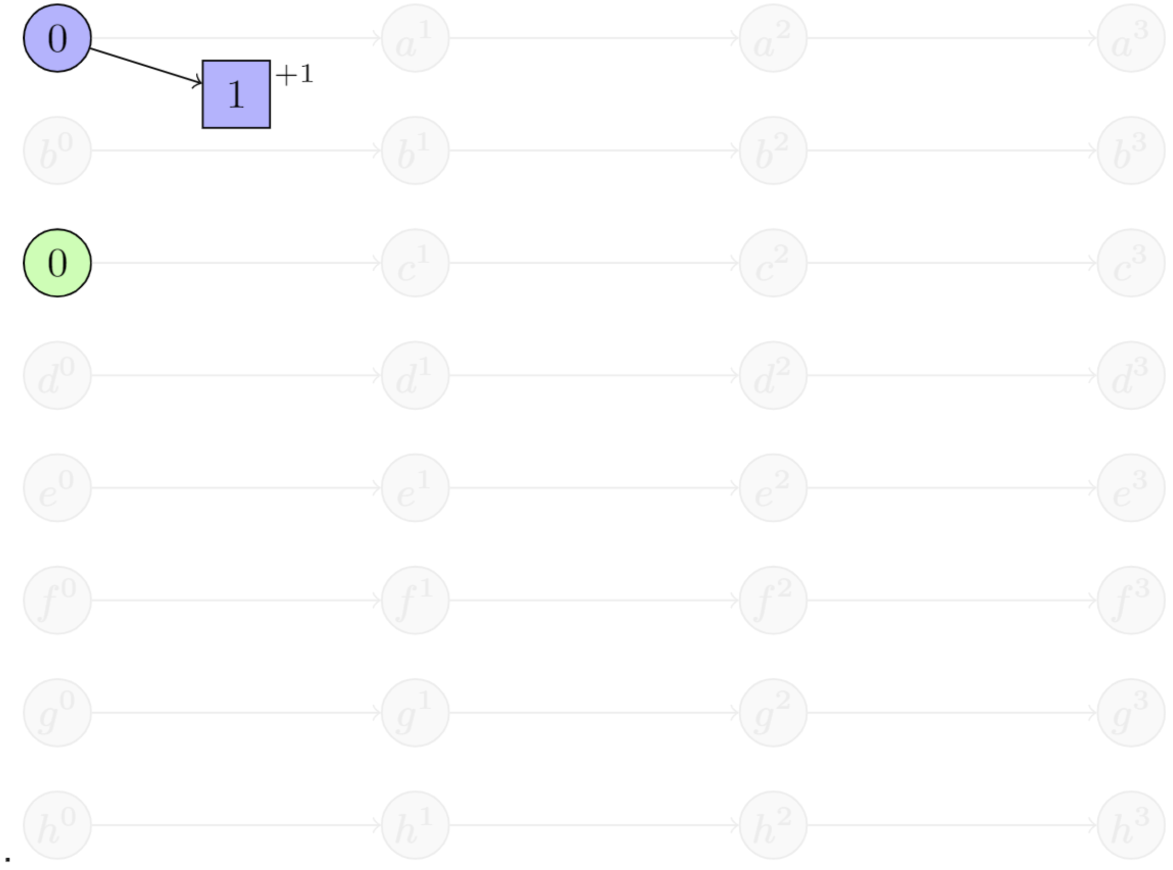


new state s		
q	x_q	rhs_q
a	0	0
b	0	2
c	2	0
d	4	4
e	4	4
f	2	2
g	6	6
a_1	1	1
a_2	3	3
a_3	3	3
a_4	1	1
a_5	5	5
a_6	5	5

PQ
b:0
c:0

PINCH: Example after we pop b and c from the priority queue

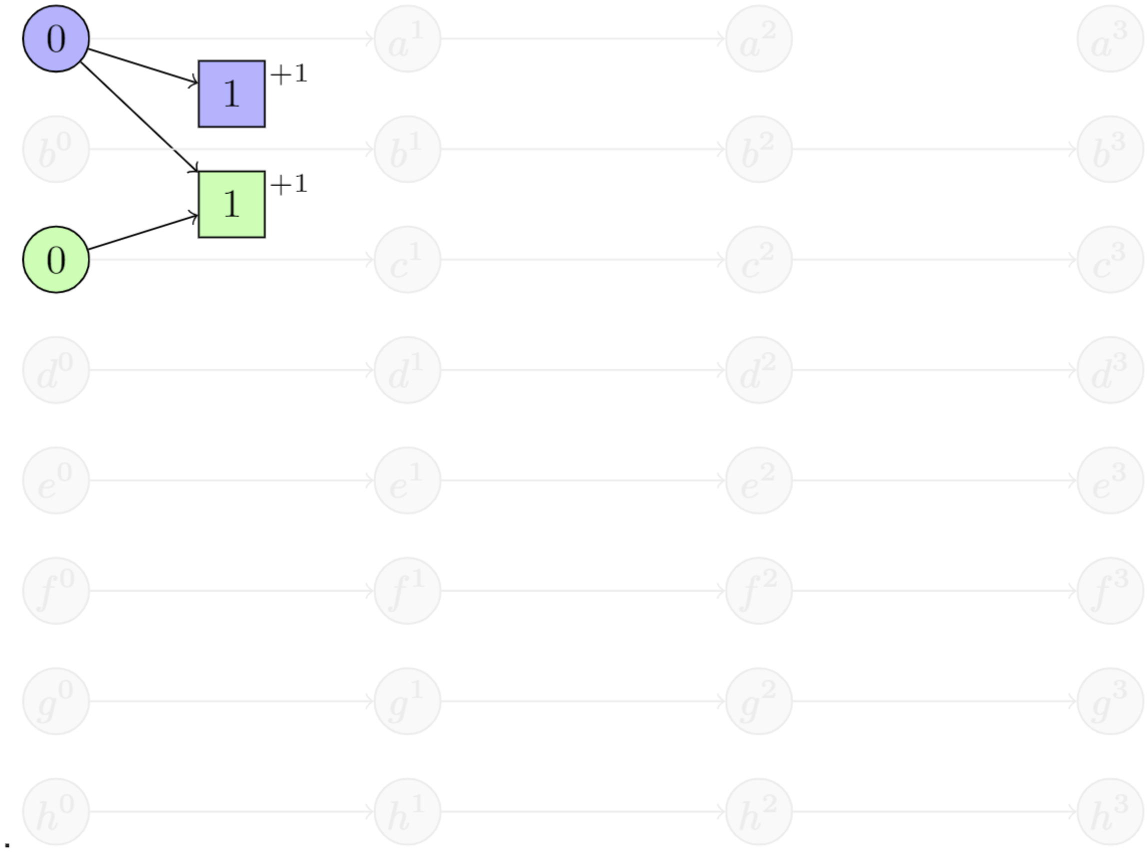
- $s' = I = \{a, b\}$
- $s = \{a, c\}$



new state s		
q	x_q	rhs_q
a	0	0
b	∞	2
c	0	0
d	4	4
e	4	4
f	2	2
g	6	6
a_1	1	1
a_2	3	1
a_3	3	∞
a_4	1	∞
a_5	5	5
a_6	5	5

PQ
a_2:1
a_4:1
b:2
a_3:3

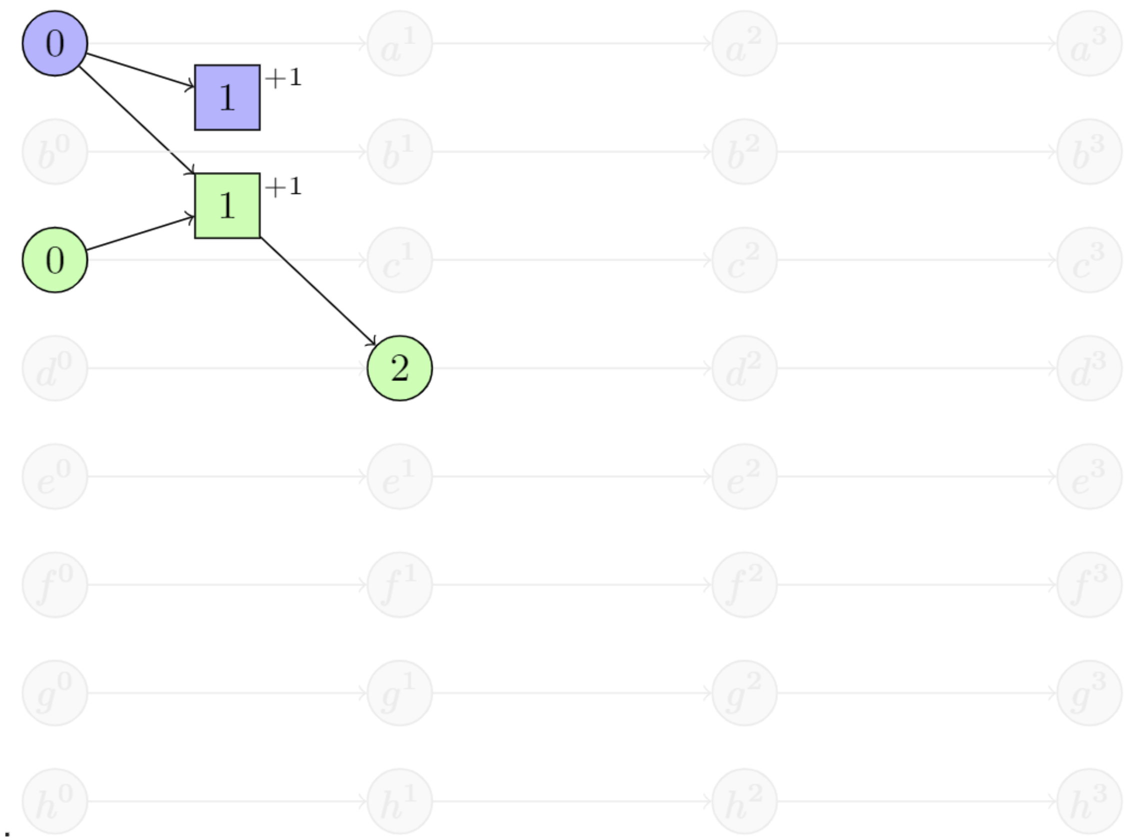
PINCH: Example after we pop a2



new state s		
q	x_q	rhs_q
a	0	0
b	∞	2
c	0	0
d	4	2
e	4	4
f	2	2
g	6	6
a_1	1	1
a_2	1	1
a_3	3	∞
a_4	1	∞
a_5	5	5
a_6	5	5

PQ
a_4:1
d:2
b:2
a_3:3

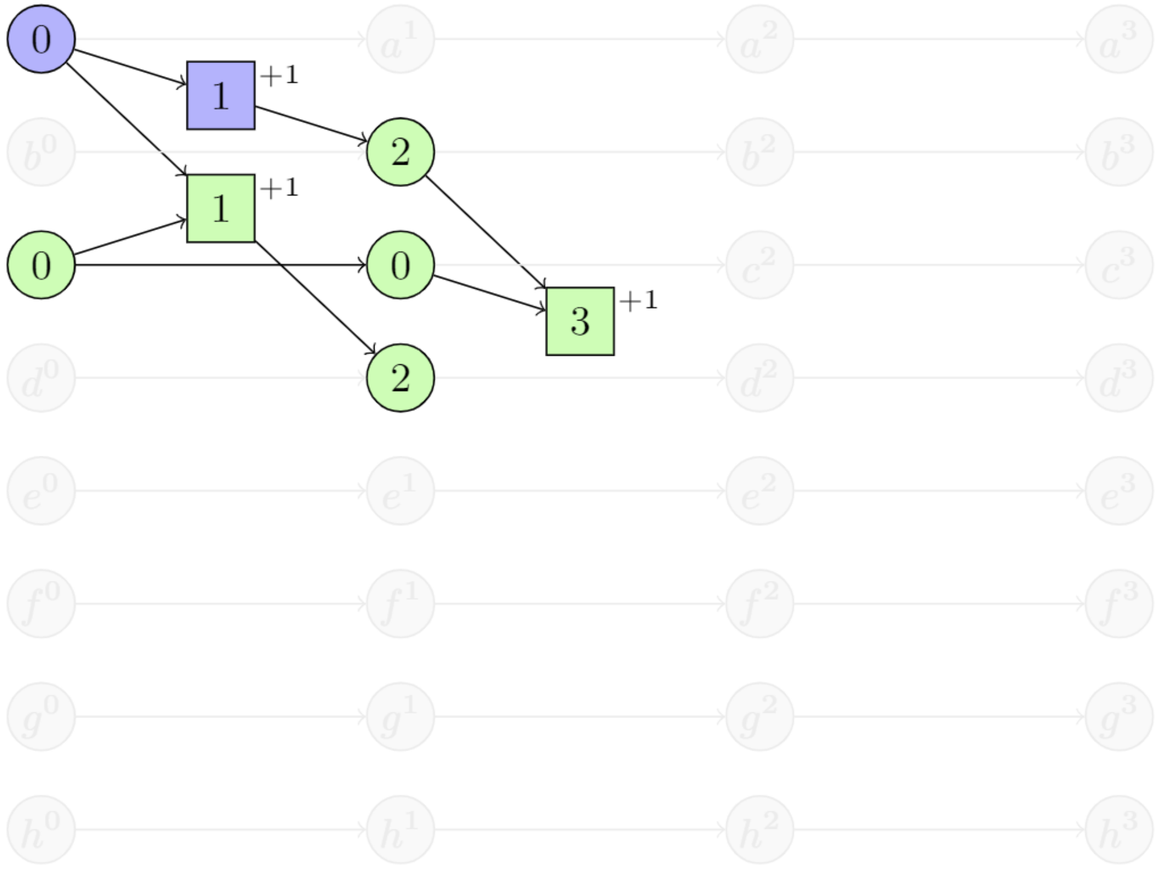
PINCH: Example after we pop a4, d



new state s		
q	x_q	rhs_q
a	0	0
b	∞	2
c	0	0
d	2	2
e	4	4
f	2	6
g	6	6
a_1	1	1
a_2	1	1
a_3	3	∞
a_4	∞	∞
a_5	5	3
a_6	5	3

PQ
f:2
b:2
a_5:3
a_6:3
a_3:7

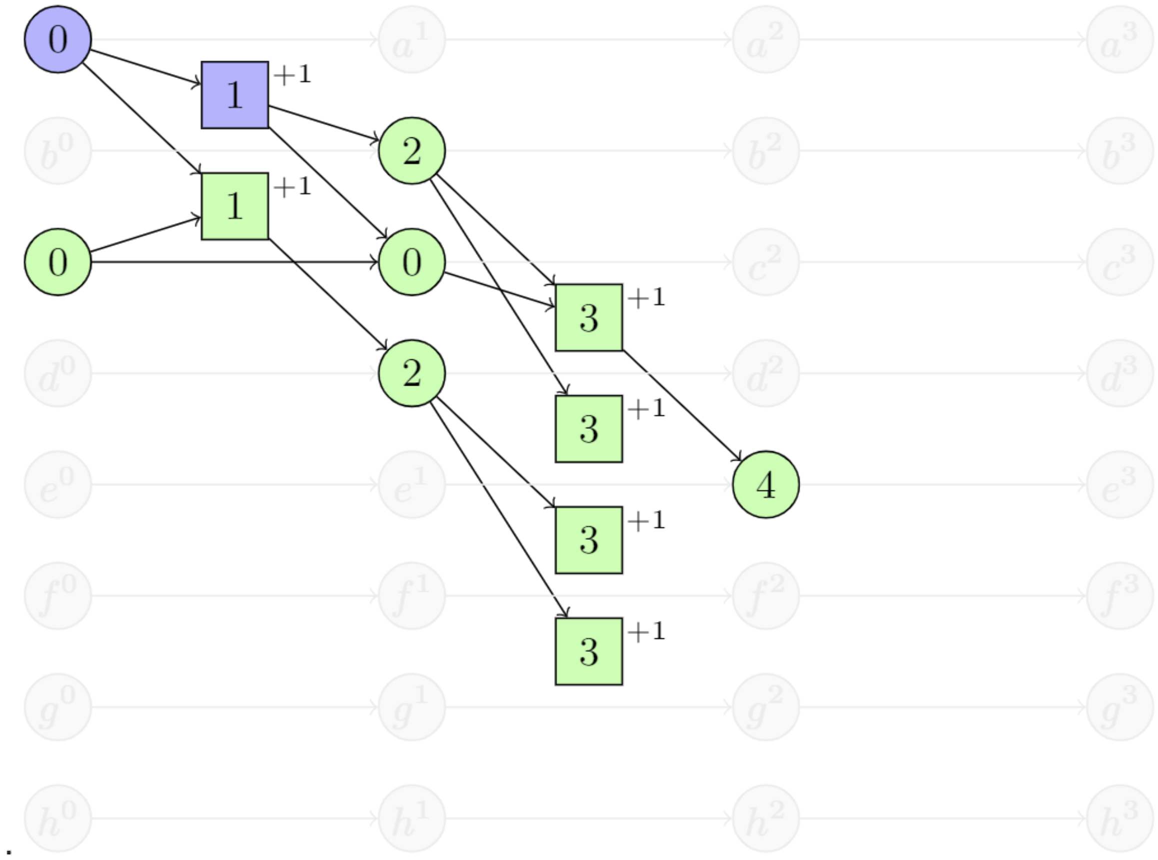
PINCH: Example after we pop f, b



new state s		
q	x_q	rhs_q
a	0	0
b	2	2
c	0	0
d	2	2
e	4	4
f	∞	6
g	6	6
a_1	1	1
a_2	1	1
a_3	3	3
a_4	∞	3
a_5	5	3
a_6	5	3

PQ
a_5:3
a_6:3
a_4:3
f:6

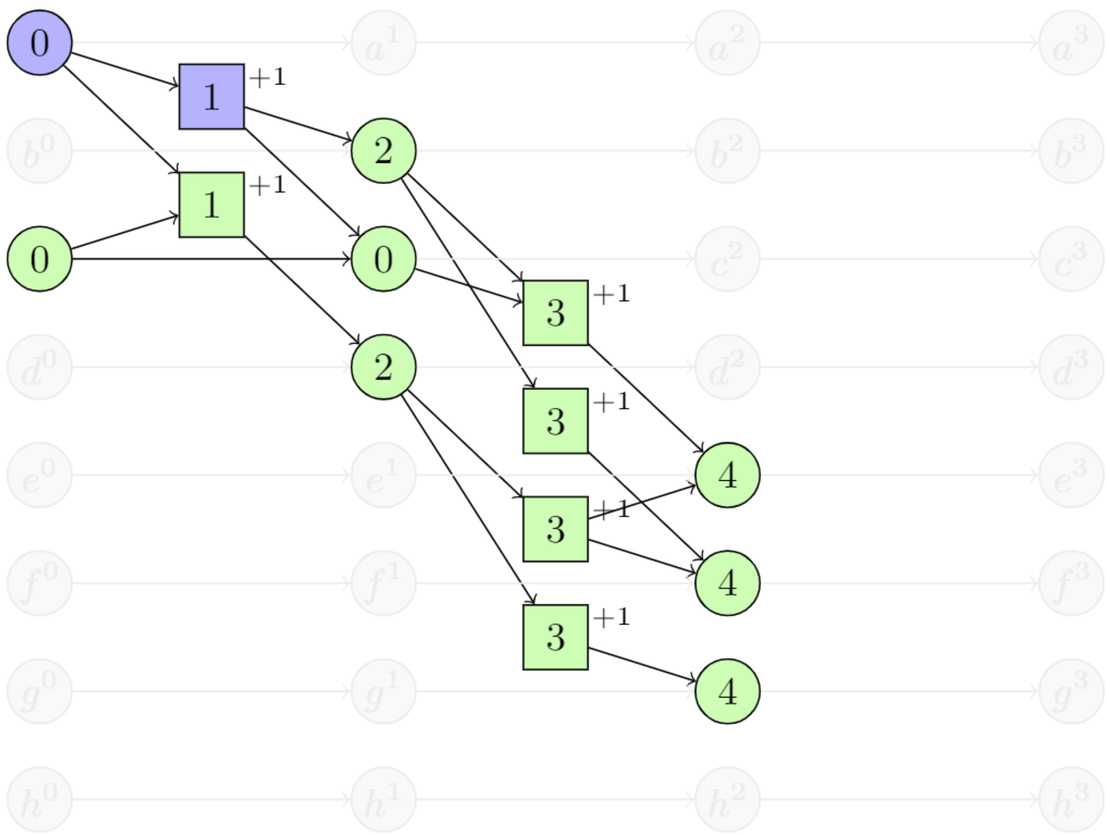
PINCH: Example after we pop a5,a6,a4



new state s		
q	x_q	rhs_q
a	0	0
b	2	2
c	0	0
d	2	2
e	4	4
f	∞	4
g	6	4
a_1	1	1
a_2	1	1
a_3	3	3
a_4	3	3
a_5	3	3
a_6	3	3

PQ
f:4
g:4

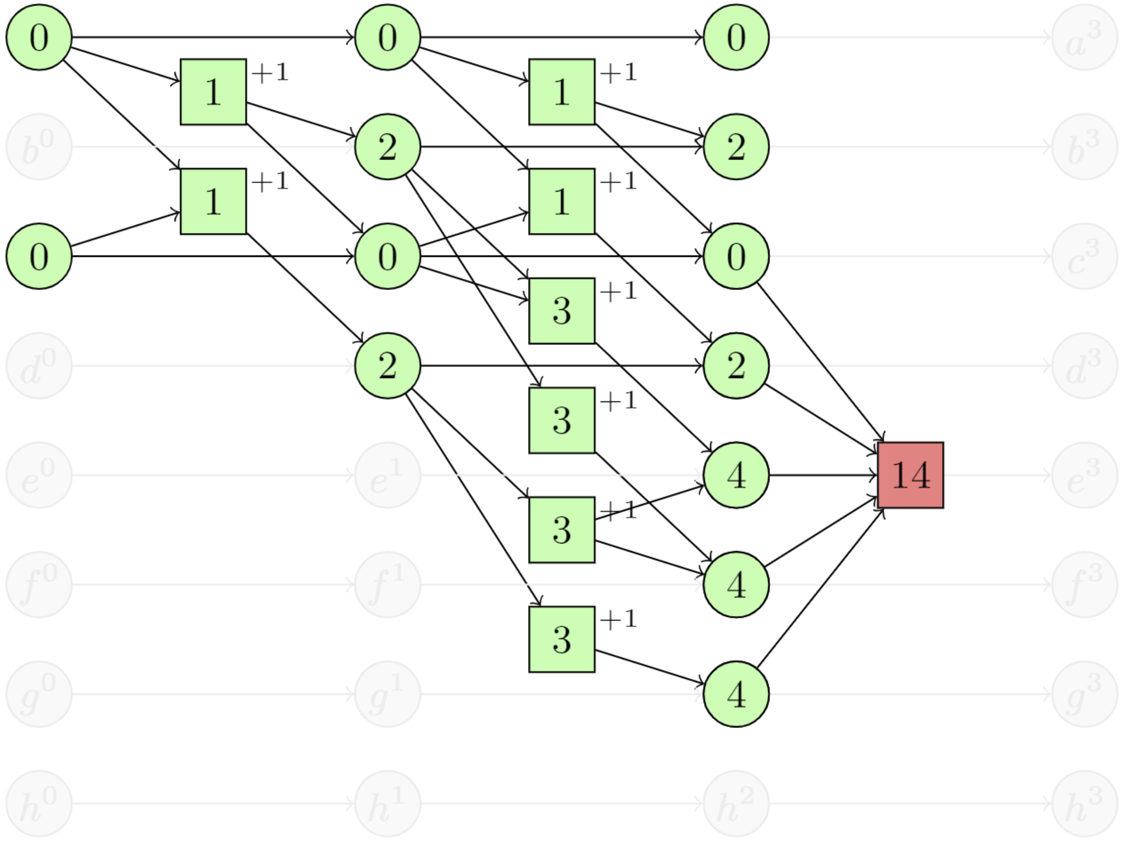
PINCH: Example after we pop f,g



new state s		
q	x_q	rhs_q
a	0	0
b	2	2
c	0	0
d	2	2
e	4	4
f	4	4
g	4	4
a_1	1	1
a_2	1	1
a_3	3	3
a_4	3	3
a_5	3	3
a_6	3	3

PQ

PINCH: Example



$h^{\text{add}}(s) = 14/2 = 7$

new state s		
q	x_q	rhs_q
a	0	0
b	2	2
c	0	0
d	2	2
e	4	4
f	4	4
g	4	4
a_1	1	1
a_2	1	1
a_3	3	3
a_4	3	3
a_5	3	3
a_6	3	3

PINCH: Main Takeaways

- PINCH is an incremental version of GD
- PINCH updates cost values at most twice
- PINCH uses a priority queue with variables and actions
- PINCH treats variables and actions based on the relationship of rhsq and xq

Content

Evaluation

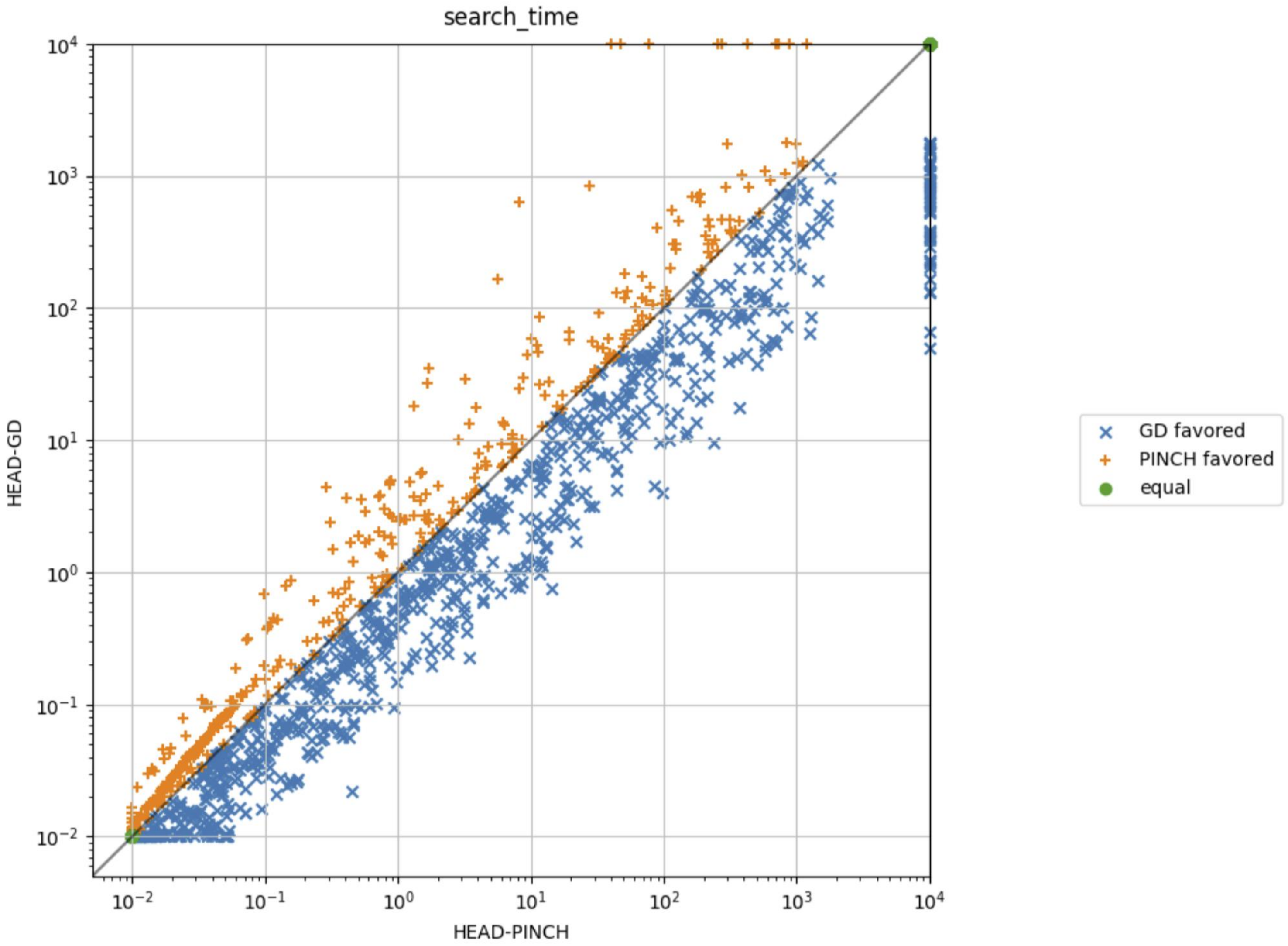
- Implementation in the Fast Downward Planning System
- Tests are done on the Satisficing Track of IPC 1998-2018
- Weighted A* with a weight of 2
- Action cost > 0
- Testing: PINCH vs. GD

Results

Results		
Property	GD	PINCH
Number of Runs	2542	2542
Coverage	1663	1630
Search out of Memory	138	229
Search out of Time	696	638
Search Time	1.08	1.84

GD is ~1.7 times faster than PINCH

Results

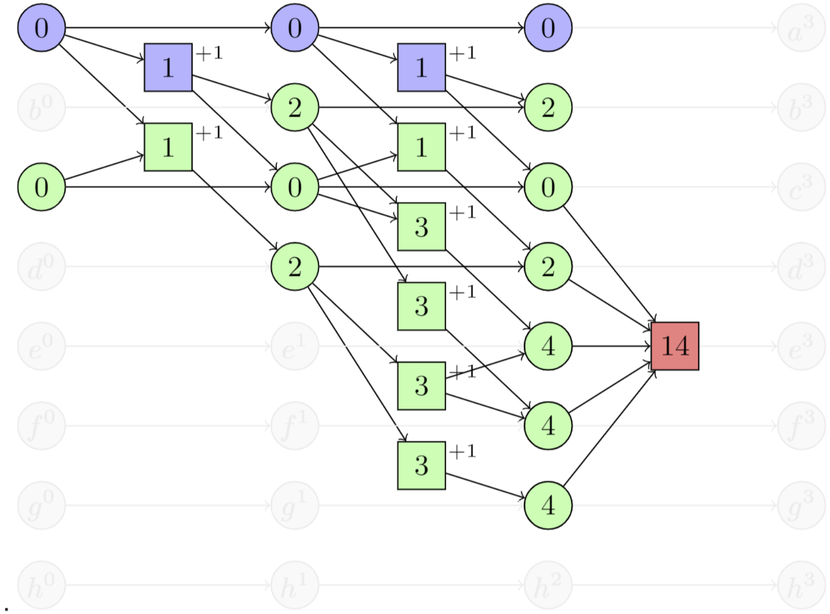


PINCH outperforms GD in ~1/4 of the covered instances

What do PINCH favored domains have in common?

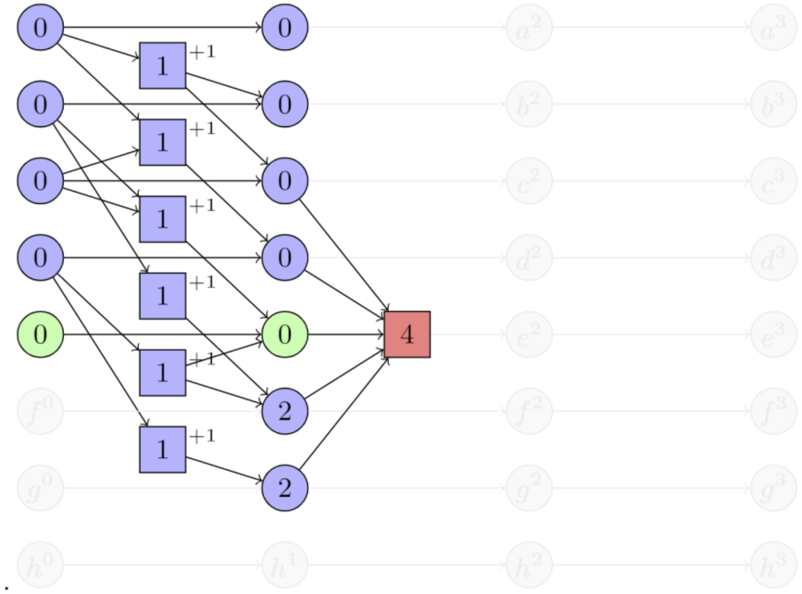
Idea: Similarity Factors

- $s' = I = \{a, b\}$
- $s = \{a, c\}$



vs.

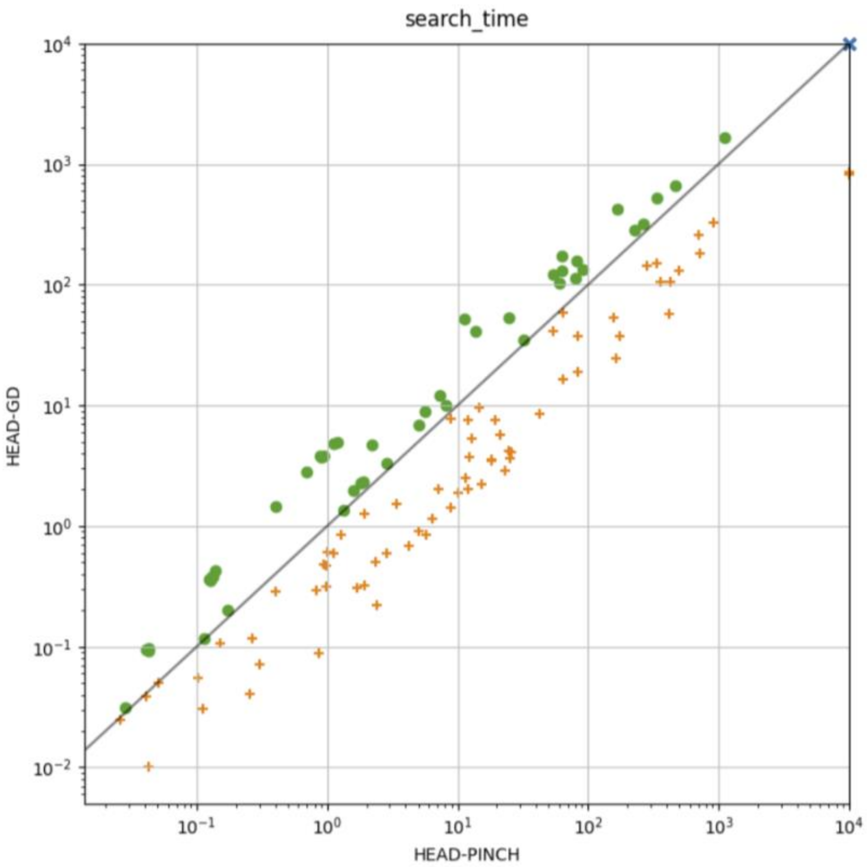
- $s' = \{a, b, c, d\}$
- $s = \{a, b, c, d, e\}$



Factor 1: PINCH benefits if s and s‘ are similar

Factor 2: PINCH benefits if s and s‘ include a large number of variables in relation to the total number of variables

Plot: Similarity Factors

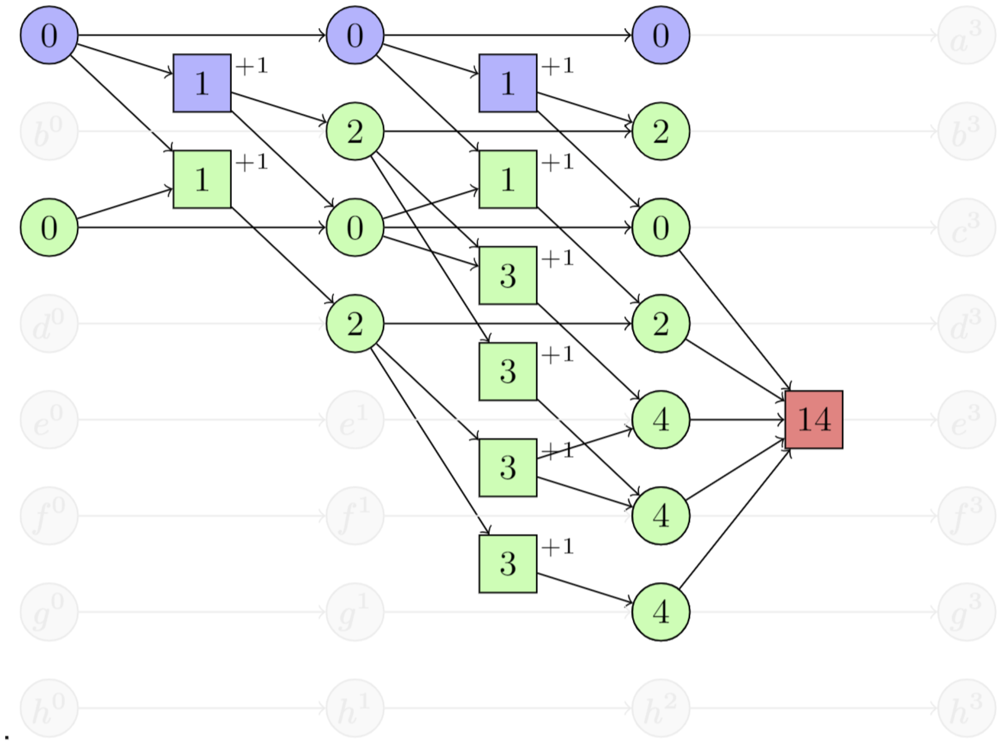


Green: PINCH favored

Orange: GD favored

PINCH outperforms GD in ~1/3 of the covered instances

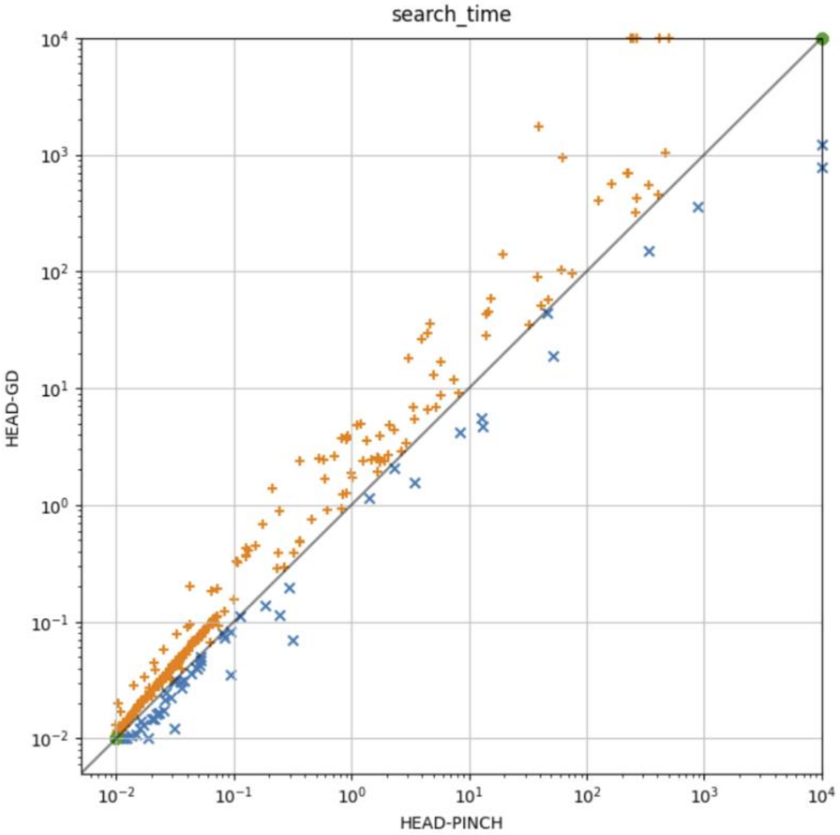
2. Idea: Action Factors



Factor 3: PINCH benefits from actions having a low number of preconditions

Factor 4: PINCH benefits from there being a high number of variables in relation to actions

Plot: low number of preconditions

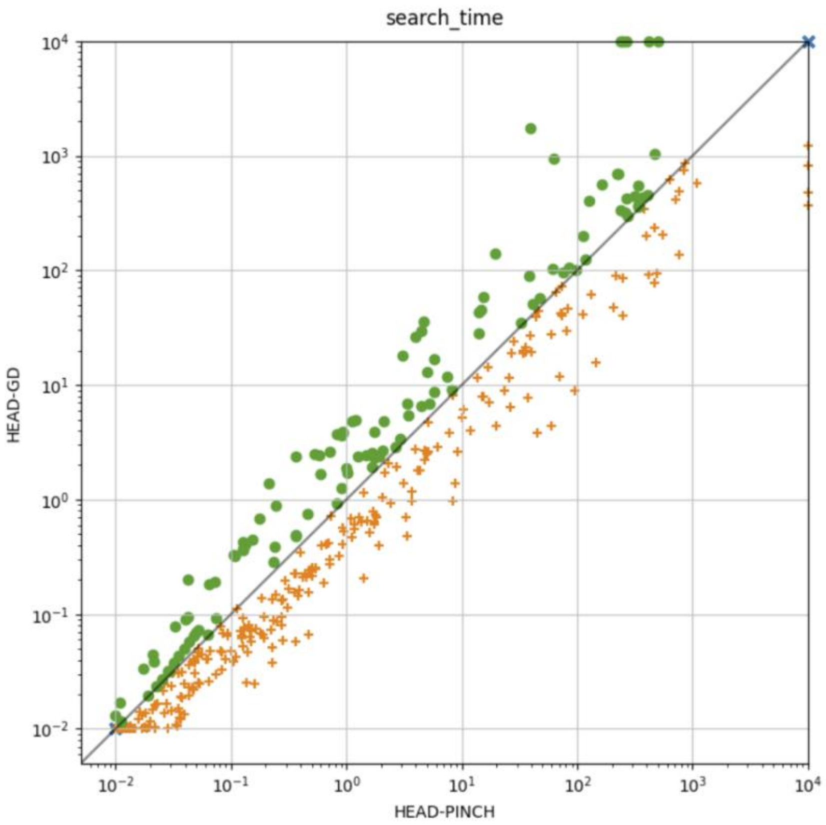


Orange: PINCH favored

Blue: GD favored

PINCH outperforms GD in ~3/4 of the covered instances

Plot: High ratio of variables



Green: PINCH favored

Orange: GD favored

PINCH outperforms GD in ~2/5 of the covered instances

Comparing PINCH and GD directly

How many times do PINCH and GD update cost values?

PINCH updates cost values 15502 times on average

GD updates cost values 15701 times on average

How many times do PINCH and GD pop something out of their priority queue?

PINCH pops 15519 times on average

GD pops 432 times on average

The Issue With The Priority Queue

PINCH inserts variables and actions into its priority queue

GD only inserts variables into its priority queue

This is the main reason for the poor performance of PINCH

Conclusion

- On average GD outperforms PINCH
- PINCH outperforms GD on domains with a low number of preconditions per operator
- Main drawback of PINCH: priority queue!
- Future Work: Consider possibility of changing priority queue such that it only uses variables

If the issue with the priority queue can be resolved,

I expect PINCH to outperform GD

Universität
Basel

Thank You
for Your Attention.