# The FF Heuristic for Lifted Classical Planning

Augusto B. Corrêa[1], Florian Pommerening[1], Malte Helmert[1], Guillem Francès[2]

[1]University of Basel, Switzerland
[2]Universitat Pompeu Fabra, Spain
{augusto.blaascorrea,florian.pommerening,malte.helmert}@unibas.ch
guillem.frances@upf.edu

# Lifted Planning

what we consider lifted classical planning:

- planning only with the PDDL description
  - predicate symbols, objects, action schemas, initial state, goal

heuristic search:

- action schemas are lifted
- states are ground

# What We Already Know in Lifted Planning

what we already know:

- efficient successor generation
- several heuristics
  - $h^{\text{add}}$, unary relaxation, goalcount, ...
- extract preferred operators

problem: no lifted version of a state-of-the-art heuristic

- our contribution: lifted $h^{\text{FF}}$

# Running Example

## Running Example

```
(:init (P 0 1) (S 0))
(:action A
 :parameters (?X ?Y)
 :precondition (and (P ?X ?Y)
                    (S ?X))
 :effect (and (Q ?X)
              (R ?Y)))
(:goal (Q 0))
```

delete-free planning task → Datalog program

# Initial State into Datalog Facts

**Running Example**

```
(:init (P 0 1) (S 0))
```

$$\mathcal{F} \coloneqq \{P(0,1), S(0)\}$$

# Action Schemas into Datalog Rules

## Running Example

```
(:action A
 :parameters (?X ?Y)
 :precondition (and (P ?X ?Y)
                    (S ?X))
 :effect (and (Q ?X)
              (R ?Y)))
```

$$\overbrace{A\text{-}applicable(X,Y)}^{\text{head}} :- \overbrace{P(X,Y), S(X)}^{\text{body}}$$

$$Q(X) :- A\text{-}applicable(X,Y)$$

$$R(Y) :- A\text{-}applicable(X,Y)$$

# Goal into Datalog Rule

**Running Example**

```
(:goal (Q 0))
```

$$goal() :- Q(0)$$

# Datalog Program

$$
\begin{aligned}
\mathcal{F} :=& \{P(0,1), S(0)\}\\
\mathcal{R} :=& \{\textit{A-applicable}(X, Y) \coloneq P(X, Y), S(X),\\
& \quad Q(X) \coloneq \textit{A-applicable}(X, Y),\\
& \quad R(Y) \coloneq \textit{A-applicable}(X, Y),\\
& \quad \textit{goal}() \coloneq Q(0)\}
\end{aligned}
$$

# One Step Further...

annotated Datalog:

- annotate each rule with instructions
- Python-like imperative instructions

in our case: annotations add ground actions to a relaxed plan $\pi_{\mathtt{FF}}$

# Datalog Program

$A\text{-}applicable(X, Y) :- P(X, Y), S(X)$ [Add $A(X, Y)$ to $\pi_{\texttt{FF}}$]

$Q(X) :- A\text{-}applicable(X, Y)$ [ ]

$R(Y) :- A\text{-}applicable(X, Y)$ [ ]

$goal() :- Q(0)$ [ ]

# How It Works

step-by-step:

1. ground program until we reach *goal*()
2. construct derivation tree
3. execute instructions in order

# Step 1: Ground

$$\mathcal{M} = \{P(0,1), S(0)\}$$
$$GroundRules = \{\}$$

# Step 1: Ground

$$\mathcal{M} = \{P(0,1), S(0), \textit{A-applicable}(0,1)\}$$
$$\textit{GroundRules} = \{r_1\}$$

$$r_1 \coloneqq \textit{A-applicable}(0,1) \coloneq P(0,1), S(0) \qquad \texttt{[Add } A(0,1) \texttt{ to } \pi_{\texttt{FF}}\texttt{]}$$

# Step 1: Ground

$$\begin{aligned} \mathcal{M} &= \{P(0,1), S(0), \mathit{A\text{-}applicable}(0,1)\} \\ \mathit{GroundRules} &= \{r_1\} \end{aligned}$$

# Step 1: Ground

$$\mathcal{M} = \{P(0,1), S(0), \textit{A-applicable}(0,1), Q(0), R(1)\}$$

$$\textit{GroundRules} = \{r_1, r_2, r_3\}$$

$r_2 := Q(0) \mathrel{:-} \textit{A-applicable}(0,1)$ [ ]

$r_3 := R(1) \mathrel{:-} \textit{A-applicable}(0,1)$ [ ]

# Step 1: Ground

$$\mathcal{M} = \{P(0,1), S(0), \textit{A-applicable}(0,1), Q(0), R(1)\}$$

$$\textit{GroundRules} = \{r_1, r_2, r_3\}$$

# Step 1: Ground

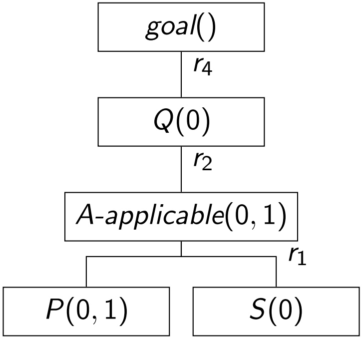
$$\mathcal{M} = \{P(0,1), S(0), \mathit{A\text{-}applicable}(0,1), Q(0), R(1), \mathit{goal}()\}$$
$$\mathit{GroundRules} = \{r_1, r_2, r_3, r_4\}$$

$r_4 := \mathit{goal}() \text{ :− } Q(0)$ [ ]

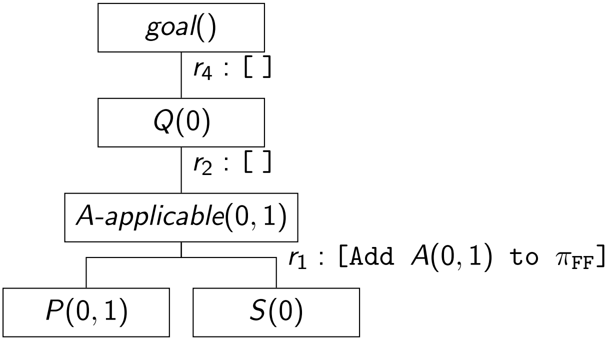
note: in practice, we ground atoms ordered by $h^{\text{add}}$ values

# Step 2: Derivation Tree

atom $A$ derives $B$ if $A$ is in the body of the rule reaching $B$



# Step 3: Execute Instructions




execution:

- order rule annotations bottom-up and execute
- our example: $r_1, r_2, r_4$

after execution: $\pi_{\mathtt{FF}} = \langle A(0,1) \rangle$

- $h^{\mathrm{FF}}$ = cost of $\pi_{\mathtt{FF}}$

# Framework

With this type of annotations, we can compute $h^{\mathsf{FF}}$.
But we can do more than that.

annotated Datalog as a framework:

- useful atoms
- other heuristics
- more info in the paper

# Problem

problem: straightforward encoding used does not scale

- atoms like *A-applicable* might have high arity
- duplicated sub-expressions
- inefficient joins

solution: program rewriting transformations

# Example of Transformation: Rule Merging

$A\text{-}applicable(X, Y) :- P(X, Y), S(X)$ `[Add` $A(X, Y)$ `to` $\pi_{\texttt{FF}}$`]`

$Q(X) :- A\text{-}applicable(X, Y)$ `[ ]`

$R(Y) :- A\text{-}applicable(X, Y)$ `[ ]`

# Example of Transformation: Rule Merging

$Q(X) :- P(X, Y), S(X)$ $\quad$ [Add $A(X, Y)$ to $\pi_{\texttt{FF}}$]

$R(Y) :- P(X, Y), S(X)$ $\quad$ [Add $A(X, Y)$ to $\pi_{\texttt{FF}}$]

# Example of Transformation: Rule Splitting

$P_1(X) \coloneq Q(X, Z), T(X, Y), S(Y)$ [Add $A_1(X, Y, Z)$ to $\pi_{\mathtt{FF}}$]

$P_2(X) \coloneq R(X, Z), T(X, Y), S(Y)$ [Add $A_2(X, Y, Z)$ to $\pi_{\mathtt{FF}}$]

# Example of Transformation: Rule Splitting

$$\alpha(X) :\!-\ T(X,Y), S(Y) \qquad [\texttt{Instantiation}[\alpha(X)] = (X,Y)]$$

$$P_1(X) :\!-\ Q(X,Z), \alpha(X) \qquad [\texttt{X,Y = Instantiation}[\alpha(X)]; \ \texttt{Add } A_1(\texttt{X},\texttt{Y},Z) \texttt{ to } \pi_{\texttt{FF}}]$$

$$P_2(X) :\!-\ R(X,Z), \alpha(X) \qquad [\texttt{X,Y = Instantiation}[\alpha(X)]; \ \texttt{Add } A_2(\texttt{X},\texttt{Y},Z) \texttt{ to } \pi_{\texttt{FF}}]$$

more: predicate collapsing, variable renaming

# Transformations Preserve Relaxed Plans

in the paper: transformations preserve relaxed plans

- step-by-step process for the transformations
- how to handle annotations

transformations preserve semantics of annotations in general

- under certain circumstances
- by transforming annotations together with the rules

# Experiments

two benchmarks:

- 1001 IPC tasks
- 862 hard-to-ground (HTG) tasks

setup:

- 30 minutes per run
- 16 GiB

# Comparison to Ground Version

using $h^{\text{FF}}$ with lifted and ground implementations of

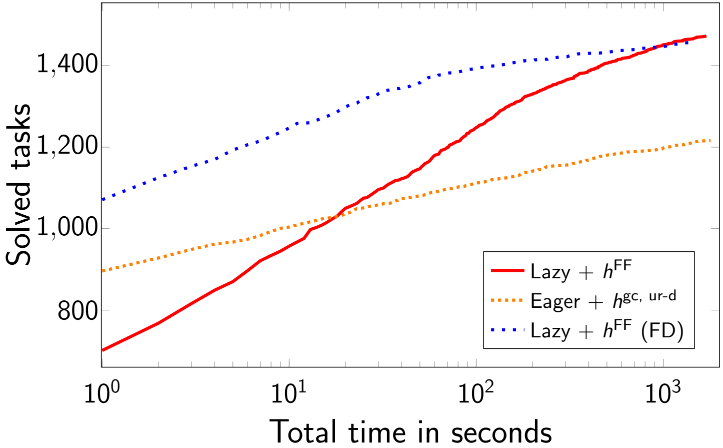
- eager GBFS
- lazy GBFS with preferred operators

| | Ground | | Lifted | |
|---|---|---|---|---|
| Coverage | Eager | Lazy + PO | Eager | Lazy + PO |
| **IPC Sum (1001)** | 775 | **862** | 653 | 782 |
| blocksworld-large (40) | 4 | **12** | 3 | 9 |
| childsnacks-large (144) | 51 | **115** | 27 | 77 |
| genome-edit-distance (312) | **312** | **312** | 286 | 310 |
| logistics-large (40) | 30 | 32 | 6 | **40** |
| organic-synthesis (56) | 20 | 20 | 46 | **47** |
| pipesworld-tankage-nosplit (50) | 15 | 19 | 17 | **28** |
| rovers-large (40) | 11 | 13 | 26 | **40** |
| visitall-multidimensional (180) | 72 | 72 | 108 | **140** |
| **HTG Sum (862)** | 515 | 595 | 519 | **691** |
| **Total Sum (1863)** | 1290 | 1457 | 1172 | **1473** |

# Comparison to Other Lifted Methods

| Coverage | $h^{\text{gc, ur-d}}$ | $h^{\text{add}}$ Eager | $h^{\text{add}}$ Lazy + PO | $h^{\text{FF}}$ Eager | $h^{\text{FF}}$ Lazy + PO |
|---|---|---|---|---|---|
| **IPC Sum (1001)** | 575 | 608 | 762 | 653 | **782** |
| blocksworld-large (40) | 7 | 1 | 5 | 3 | **9** |
| childsnacks-large (144) | **98** | 34 | 81 | 27 | 77 |
| genome-edit-distance (312) | **312** | 181 | 285 | 286 | 310 |
| logistics-large (40) | 0 | 6 | **40** | 6 | **40** |
| organic-synthesis (56) | **47** | 46 | **47** | 46 | **47** |
| pipesworld-tankage-nosplit (50) | 10 | 22 | **32** | 17 | 28 |
| rovers-large (40) | 16 | 11 | 31 | 26 | **40** |
| visitall-multidimensional (180) | **151** | 117 | 142 | 108 | 140 |
| **HTG Sum (862)** | 641 | 418 | 663 | 519 | **691** |
| **Total Sum (1863)** | 1216 | 1026 | 1425 | 1172 | **1473** |

# Solved Tasks over Time



# Conclusion

key ideas:

- lifted $h^{\mathrm{FF}}$
- state-of-the-art lifted planner
- framework to compute delete-relaxed heuristics