Engineering a Heuristic Search Planner

Gabriele Röger  Malte Helmert

University of Basel, Switzerland

ICAPS Tutorial
11 June 2013
Introduction
We will talk about Fast Downward. (That’s what we know.)

- **Part 1**: high-level picture, architecture
- **Part 2**: in-depth look: efficient FF heuristic implementation

work in progress ↞ what do you want from such a tutorial?
Why did we propose this tutorial?

- efficiency often a stumbling block
- experience with large (and painful!) code integration efforts
- not/barely covered in papers

→ communicate lessons learned
Fast Downward Overview & History
# Fast Downward

Three components of Fast Downward:

1. **Translator**
   - implemented in Python
   - takes PDDL input, produces finite-domain representation
   - **main jobs**: invariants, grounding

2. **Preprocessor (“Knowledge Compilation”)**
   - implemented in C++
   - augments translator output
   - **main jobs**: relevance analysis, successor generator

3. **Search**
   - implemented in C++
   - various search algorithms and heuristics
$ hg clone ssh://hg.fast-downward.org downward
$ cd downward/src
$ ./build_all -j4
$ ./plan PROBLEM_FILE --search 'astar(lmcut())'
Where to Find More Details

If you want to read up on this:

- **translator:**
  M. Helmert. *Concise finite-domain representations for PDDL planning tasks.*

- **preprocessor:**

- **search:**
  all over the place
Code History

<table>
<thead>
<tr>
<th>Fast Downward 2004</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Translator</td>
<td>3082 lines</td>
</tr>
<tr>
<td>Preprocessor</td>
<td>1871 lines</td>
</tr>
<tr>
<td>Search</td>
<td>2966 lines</td>
</tr>
</tbody>
</table>
## Fast Downward 2004

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translator</td>
<td>3082 lines</td>
</tr>
<tr>
<td>Preprocessor</td>
<td>1871 lines</td>
</tr>
<tr>
<td>Search</td>
<td>2966 lines</td>
</tr>
</tbody>
</table>

## Fast Downward 2013

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translator</td>
<td>4941 lines</td>
</tr>
<tr>
<td>Preprocessor</td>
<td>2161 lines</td>
</tr>
<tr>
<td>Search</td>
<td>23990 lines</td>
</tr>
</tbody>
</table>
2004:
- 1 search algorithm (lazy greedy BFS)
- 2 heuristics (CG and FF)
- 4 switches (cCfF)
With Great Variety Comes Great Mess

- **2004:**
  - 1 search algorithm (lazy greedy BFS)
  - 2 heuristics (CG and FF)
  - 4 switches (cCfF)

- **2005–2009:**
  - two more search algorithms (eager greedy BFS, A*)
  - more heuristics (M&S, $h^{cea}$, landmarks, . . .)
  - many more switches (cCyYfFakziwmhdspLDbuARS)
Pop quiz: What does Dyu do in Fast Downward 2009?
Pop quiz: What does Dyu do in Fast Downward 2009?

Run eager search with alternation between context-enhanced additive heuristic and LM-cut heuristic, using preferred operators from the additive heuristic.
**Pop quiz:** What does Dyu do in Fast Downward 2009?

Run eager search with alternation between context-enhanced additive heuristic and LM-cut heuristic, using preferred operators from the additive heuristic.

**For extra credit:**
Pop quiz: What does Dyu do in Fast Downward 2009?

Run eager search with alternation between context-enhanced additive heuristic and LM-cut heuristic, using preferred operators from the additive heuristic.

For extra credit: Why??
Command Lines: Now

```bash
./plan PROBLEM_FILE
  --heuristic 'h1=cea()'
  --heuristic 'h2=lmcut()'
  --heuristic 'h3=add()'
  --search 'eager_greedy([h1, h2], preferred=[h3])’
```

which is actually syntactic sugar for...
Command Lines: Now

Today (director’s cut)

./plan PROBLEM_FILE
   --heuristic 'h1=cea(cost_type=NORMAL)'
   --heuristic 'h2=lcse(cost_type=NORMAL)'
   --heuristic 'h3=add(cost_type=NORMAL)'
   --search 'eager(open=alt(sublists=[
         single(h1), single(h1, pref_only=true),
         single(h2), single(h2, pref_only=true)],
         boost=0),
         reopen_closed=false,
         pathmax=false,
         preferred=[h3],
         cost_type=NORMAL,
         bound=2147483647)
Another Command Line Example

Pop quiz: What is this?

If all actions are unit cost, run:

```bash
./plan PROBLEM_FILE
   --heuristic 'hlm,hff=lm_ff_syn(lm_rhw(
       reasonable_orders=true,lm_cost_type=2,cost_type=2))'
   --search 'iterated(
     lazy_greedy([hff,hlm],preferred=[hff,hlm]),
     lazy_wastar([hff,hlm],preferred=[hff,hlm],w=5),
     lazy_wastar([hff,hlm],preferred=[hff,hlm],w=3),
     lazy_wastar([hff,hlm],preferred=[hff,hlm],w=2),
     lazy_wastar([hff,hlm],preferred=[hff,hlm],w=1]),
     repeat_last=true,continue_on_fail=true)'
```
Pop quiz: What is this?

Otherwise, run:

```
./plan PROBLEM_FILE
   --heuristic 'hlm1,hff1=lm_ff_syn(lm_rhw(
       reasonable_orders=true,lm_cost_type=1,cost_type=1))'
   --heuristic 'hlm2,hff2=lm_ff_syn(lm_rhw(
       reasonable_orders=true,lm_cost_type=2,cost_type=2))'
   --search 'iterated([
      lazy_greedy([hff1,hlm1],preferred=[hff1,hlm1],
         cost_type=1,reopen_closed=false),
      lazy_greedy([hff2,hlm2],preferred=[hff2,hlm2],
         reopen_closed=false),
      lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=5),
      lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=3),
      lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=2),
      lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=1]),
      repeat_last=true,continue_on_fail=true)'
```
Another Command Line Example

Pop quiz: What is this?

Otherwise, run:

```
./plan PROBLEM_FILE
   --heuristic 'hlm1,hff1=lm_ff_syn(lm_rhw(
             reasonable_orders=true,lm_cost_type=1,cost_type=1))'
   --heuristic 'hlm2,hff2=lm_ff_syn(lm_rhw(
             reasonable_orders=true,lm_cost_type=2,cost_type=2))'
   --search 'iterated([
             lazy_greedy([hff1,hlm1],preferred=[hff1,hlm1],
                           cost_type=1,reopen_closed=false),
             lazy_greedy([hff2,hlm2],preferred=[hff2,hlm2],
                           reopen_closed=false),
             lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=5),
             lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=3),
             lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=2),
             lazy_wastar([hff2,hlm2],preferred=[hff2,hlm2],w=1],
             repeat_last=true,continue_on_fail=true)'
```

⇝ LAMA 2011
Plug-In Architecture
Why All These Long Options?

problems with the old way of handling things:

- running out of letters
- options for heuristics/search algorithms painful
- not very flexible
- merge conflicts

Idea:

- small core
- plug-ins for heuristics etc.
- no code changes to add/remove plug-ins
Search Component: Codebase

planner.cc axioms.cc causal_graph.cc combining_evaluator.cc domain_transition_graph.cc
eager_search.cc enforced_hill_climbing_search.cc exact_timer.cc g_evaluator.cc globals.cc
heuristic.cc ipc_max_heuristic.cc iterated_search.cc lazy_search.cc legacy_causal_graph.cc
max_evaluator.cc operator.cc operator_cost.cc option_parser.cc pref_evaluator.cc
relaxation_heuristic.cc rng.cc search_engine.cc search_node_info.cc search_progress.cc
search_space.cc state.cc successor_generator.cc sum_evaluator.cc timer.cc utilities.cc
weighted_evaluator.cc additive_heuristic.cc blind_search_heuristic.cc cea_heuristic.cc
cg_heuristic.cc cg_cache.cc ff_heuristic.cc goal_count_heuristic.cc hm_heuristic.cc
lm_cut_heuristic.cc max_heuristic.cc
open_lists/alternation_open_list.cc .../open_list_buckets.cc .../pareto_open_list.cc
.../standard_scalar_open_list.cc .../tiebreaking_open_list.cc
merge_and_shrink/abstraction.cc .../label_reducer.cc .../merge_and_shrink_heuristic.cc
.../shrink_bisimulation.cc .../shrink_bucket_based.cc .../shrink_fh.cc .../shrink_random.cc
.../shrink_strategy.cc .../variable_order_finder.cc
landmarks/exploration.cc .../h_m_landmarks.cc .../lama_ff_synergy.cc
.../landmark_cost_assignment.cc .../landmark_count_heuristic.cc .../landmark_status_manager.cc
.../landmark_graph_merged.cc .../landmark_graph.cc .../landmark_factory.cc
.../landmark_factory_rpg_exhaust.cc .../landmark_factory_rpg_sasp.cc
.../landmark_factory_zhu_givan.cc .../util.cc
learning/AODE.cc .../classifier.cc .../composite_feature_extractor.cc .../feature_extractor.cc
.../maximum_heuristic.cc .../naive_bayes_classifier.cc .../PDB_state_space_sample.cc
.../probe_state_space_sample.cc .../selective_max_heuristic.cc .../state_space_sample.cc
.../state_vars_feature_extractor.cc
pdbs/canonical_pdbs_heuristic.cc .../dominance_pruner.cc .../match_tree.cc .../max_cliques.cc
.../pattern_generation_edelkamp.cc .../pattern_generation_haslum.cc .../pdbs_heuristic.cc
.../util.cc .../zero_one_pdbs_heuristic.cc
planner.cc axioms.cc causal_graph.cc combining_evaluator.cc domain_transition_graph.cc
eager_search.cc enforced_hill_climbing_search.cc exact_timer.cc g_evaluator.cc globals.cc
heuristic.cc ipe_max_heuristic.cc iterated_search.cc lazy_search.cc legacy_causal_graph.cc
max_evaluator.cc operator.cc operator_cost.cc option_parser.cc pref_evaluator.cc
relaxation_heuristic.cc rng.cc search_engine.cc search_node_info.cc search_progress.cc
search_space.cc state.cc successor_generator.cc sum_evaluator.cc timer.cc utilities.cc
weighted_evaluator.cc additive_heuristic.cc blind_search_heuristic.cc cea_heuristic.cc
cg_heuristic.cc eg_cache.cc ff_heuristic.cc goal_count_heuristic.cc hm_heuristic.cc
lm_cut_heuristic.cc max_heuristic.cc
open_lists/alternation.open_list.cc .../open_list_buckets.cc .../pareto_open_list.cc
.../standard_scalar_open_list.cc .../tiebreaking_open_list.cc
merge_and_shrink/abstraction.cc .../label_reducer.cc .../merge_and_shrink_heuristic.cc
.../shrink_bisimulation.cc .../shrink_bucket_based.cc .../shrink_fh.cc .../shrink_random.cc
.../shrink_strategy.cc .../variable_order_finder.cc
landmarks/exploration.cc .../h_m_landmarks.cc .../lama_ff_synergy.cc
.../landmark_cost_assignment.cc .../landmark_count_heuristic.cc .../landmark_status_manager.cc
.../landmark_graph_merged.cc .../landmark_graph.cc .../landmark_factory.cc
.../landmark_factory_rpg_exhaust.cc .../landmark_factory_rpg_casp.cc
.../landmark_factory_zhu_givan.cc .../util.cc
learning/AODE.cc .../classifier.cc .../composite_feature_extractor.cc .../feature_extractor.cc
.../maximum_heuristic.cc .../naive_bayes_classifier.cc .../PDB_state_space_sample.cc
.../probe_state_space_sample.cc .../selective_max_heuristic.cc .../state_space_sample.cc
.../state_vars.feature_extractor.cc
pdbs/canonical_pdbs.heuristic.cc .../dominance_pruner.cc .../match_tree.cc .../max_cliques.cc
.../pattern_generation.edelkamp.cc .../pattern_generation_haslum.cc .../pdb_heuristic.cc
.../util.cc .../zero_one_pdbs_heuristic.cc

small core ↦ no need to understand the other code
~ demo: minimal codebase
~ demo: add a search algorithm and a heuristic
How do the plug-ins work?

Plug-in mechanism for heuristics (and other things):

- **factory function** that creates heuristic object from (parsed) parameter string
- **global object** that registers the factory function with the option parser for a certain keyword

⇝ demo: a simple plug-in
⇝ demo: a plug-in with options
Tools for Organizing Development
Tools We Use

Tools making our life easier, in roughly historical order:
Tools We Use

Tools making our life easier, in roughly historical order:

- version control (!!!)
Tools We Use

Tools making our life easier, in roughly historical order:

- version control (!!!)

- issue tracker (!!!)
  ~ http://issues.fast-downward.org
Tools We Use

Tools making our life easier, in roughly historical order:

- version control (!!!)

- issue tracker (!!!)
  ⇾ http://issues.fast-downward.org

- experiment infrastructure (!!!)
  ⇾ https://bitbucket.org/jendrikseipp/lab
Tools We Use

Tools making our life easier, in roughly historical order:

- distributed version control (!!!!!)
  ~> http://hg.fast-downward.org

- issue tracker (!!!)
  ~> http://issues.fast-downward.org

- experiment infrastructure (!!!)
  ~> https://bitbucket.org/jendrikseipp/lab
Tools We Use

Tools making our life easier, in roughly historical order:

- **distributed version control (!!!!!)**
  \[\Rightarrow \text{http://hg.fast-downward.org}\]

- **issue tracker (!!!)**
  \[\Rightarrow \text{http://issues.fast-downward.org}\]

- **experiment infrastructure (!!!)**
  \[\Rightarrow \text{https://bitbucket.org/jendrikseipp/lab}\]

- **wiki**
  \[\Rightarrow \text{http://www.fast-downward.org}\]
Tools We Use

Tools making our life easier, in roughly historical order:

- **distributed version control (!!!!)**
  ~> [http://hg.fast-downward.org](http://hg.fast-downward.org)

- **issue tracker (!!!)**
  ~> [http://issues.fast-downward.org](http://issues.fast-downward.org)

- **experiment infrastructure (!!!)**
  ~> [https://bitbucket.org/jendrikseipp/lab](https://bitbucket.org/jendrikseipp/lab)

- **wiki**
  ~> [http://www.fast-downward.org](http://www.fast-downward.org)

- **buildbot**
  ~> [http://buildbot.fast-downward.org](http://buildbot.fast-downward.org)
Tools We Use

Tools making our life easier, in roughly historical order:

- distributed version control (!!!!)
  ⇾ http://hg.fast-downward.org

- issue tracker (!!!)
  ⇾ http://issues.fast-downward.org

- experiment infrastructure (!!!)
  ⇾ https://bitbucket.org/jendrikseipp/lab

- wiki
  ⇾ http://www.fast-downward.org

- buildbot
  ⇾ http://buildbot.fast-downward.org

- public mailing list
  ⇾ http://groups.google.com/group/fast-downward/
Tools We Use

Tools making our life easier, in roughly historical order:

- distributed version control (!!!!)
  ➝ http://hg.fast-downward.org

- issue tracker (!!!)
  ➝ http://issues.fast-downward.org

- experiment infrastructure (!!!)
  ➝ https://bitbucket.org/jendrikseipp/lab

- wiki
  ➝ http://www.fast-downward.org

- buildbot
  ➝ http://buildbot.fast-downward.org

- public mailing list
  ➝ http://groups.google.com/group/fast-downward/

- internal mailing list
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Overview &amp; History</th>
<th>Plug-In Architecture</th>
<th>Tools</th>
<th>Relaxation Heuristics</th>
</tr>
</thead>
</table>

Relaxation Heuristics
Relaxation Heuristics

- heuristics derived from delete relaxation of task
- delete relaxation ignores negative effects
- often taught by means of relaxed planning graph

Rest of this section:
- brief reminder of relaxed planning graph, additive heuristic, and FF heuristic
- efficient implementation of the heuristics by means of exploration queues
Illustrative Example

Variables $V$, initial state $I$, goal propositions $G$, actions $A$

\[ V = \{a, b, c, d, e, f, g, h\} \]
\[ I = \{a\} \]
\[ G = \{c, d, e, f, g\} \]
\[ A = \{a_1, a_2, a_3, a_4, a_5, a_6\} \]
\[ a_1 = \langle a \rightarrow b, c \rangle_3 \]
\[ a_2 = \langle a, c \rightarrow d \rangle_1 \]
\[ a_3 = \langle b, c \rightarrow e \rangle_1 \]
\[ a_4 = \langle b \rightarrow f \rangle_1 \]
\[ a_5 = \langle d \rightarrow e, f \rangle_1 \]
\[ a_6 = \langle d \rightarrow g \rangle_1 \]
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Relaxed Planning Graph
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{add}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{add}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$
Illustrative Example: Additive Heuristic $h^{\text{add}}$

$$h^{\text{add}}(\{a\}) = 21$$
FF Heuristic $h^{FF}$

like $h^{add}$ but with additional post-processing steps

- mark goal node in last graph layer.
- apply following rules until fixpoint reached:
  - marked action or goal node?
    $\Rightarrow$ mark all predecessors
  - variable node $v^i$ in layer $i \geq 1$ marked?
    $\Rightarrow$ mark one predecessor with minimal $h^{add}$ value
      (Tie-breaking: prefer variable nodes; otherwise arbitrarily)

heuristic estimate:

- marked action nodes form a relaxed plan
- heuristic estimate is cost of this plan
Illustrative Example: $h^{FF}$

$$h^{FF} \left( \{a_0\} \right) = 3 + 1 + 1 + 1 + 1 = 7$$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$

$h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7$
Illustrative Example: $h^{FF}$

$h^{FF}(\{a_0\}) = 3 + 1 + 1 + 1 + 1 = 7$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$

$$h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7$$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$

$$h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7$$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$

$$h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7$$
Illustrative Example: $h^{FF}$

\[ h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7 \]
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{\text{FF}}$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{FF}$

$h^{FF}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7$
Illustrative Example: $h^{FF}$
Illustrative Example: $h^{\text{FF}}$

\[ h^{\text{FF}}(\{a\}) = 3 + 1 + 1 + 1 + 1 = 7 \]
Implementation of $h^{\text{add}}$ and $h^{\text{FF}}$

Naive implementation:
build up relaxed planning graph and follow the algorithms just seen

Questions for more efficient implementation:

- **What** is actually computed?
- **Which aspects** of the task can **contribute** to the heuristic estimate?
- **When** do these aspects become **relevant**?
- **Which operations** should our **data structures** cheaply support?
Again: Additive Heuristic $h^\text{add}$
Again: Additive Heuristic $h^{\text{add}}$

heuristic estimate is sum of cheapest values of goal propositions
Again: Additive Heuristic $h^{add}$

cheapest value of initially true proposition is 0
cheapest value of other proposition is cheapest value of an action achieving it
Again: Additive Heuristic $h^{\text{add}}$

The cheapest value of action nodes is the sum of the cheapest values of precondition values plus costs of action.
Again: Additive Heuristic $h^{\text{add}}$

"concurrency" of action effects irrelevant
“concurrency” of operator effects irrelevant
⇝ change task representation

Use one unary operator for each single effect of an action, e.g., replace

\[ a_1 = \langle a \rightarrow b, c \rangle_3 \]

with

\[ a_{1,b} = \langle a \rightarrow b \rangle_{a_1}^3 \]
\[ a_{1,c} = \langle a \rightarrow c \rangle_{a_1}^3 \]

Advantage: easy support of axioms and conditional effects
“concurrency” of operator effects irrelevant
\[ \leadsto \text{change task representation} \]

Use one unary operator for each single effect of an action, e.g., replace

\[ a_1 = \langle a \rightarrow b, c \rangle_3 \]

with

\begin{align*}
  a_{1,b} &= \langle a \rightarrow b \rangle^a_3 \\
  a_{1,c} &= \langle a \rightarrow c \rangle^a_3
\end{align*}

Advantage: easy support of axioms and conditional effects
Exploration Queue Algorithm for $h^{\text{add}}$

queue = priority_queue over (value, proposition) pairs ordered by value
enqueue all initially true propositions with value 0
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if we already encountered prop with a cheaper value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of best seen values of goal propositions
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < best seen value of its effect:
                update best seen value of effect proposition
                queue.push((h_add of op, effect))
return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

queue = priority_queue over (value, proposition) pairs ordered by value
enqueue all initially true propositions with value 0
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if we already encountered prop with a cheaper value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of best seen values of goal propositions
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < best seen value of its effect:
                update best seen value of effect proposition
                queue.push((h_add of op, effect))
return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

```plaintext
queue = initialize_priority_queue(I)
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if we already encountered prop with a cheaper value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of best seen values of goal propositions
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < best seen value of its effect:
                update best seen value of effect proposition
        queue.push((h_add of op, effect))
return deadend
```
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if we already encountered prop with a cheaper value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of best seen values of goal propositions
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < best seen value of its effect:
                update best seen value of effect proposition
            queue.push((h_add of op, effect))
    return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)
set value of all operators to operator cost

while queue is not empty:
    (p_value, prop) = queue.pop()
    if we already encountered prop with a cheaper value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of best seen values of goal propositions
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < best seen value of its effect:
                update best seen value of effect proposition
        queue.push((h_add of op, effect))
return deadend
queue = initialize_priority_queue(I)
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < op.effect.hadd_value:
                op.effect.hadd_value = h_add of op
                queue.push((h_add of op, effect))
return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

```python
queue = initialize_priority_queue(I)
set value of all operators to operator cost
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition:
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        increase value of op by p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if value of op < op.effect.hadd_value:
                op.effect.hadd_value = $h_{\text{add}}$ of op
            queue.push((h_add of op, effect))
return deadend
```
Exploration Queue Algorithm for $h^{add}$

queue = initialize_priority_queue(I)
op.hadd_value = op.op_cost for all operators op
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition:
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        op.hadd_value += p_value
        mark precondition prop of op as satisfied
        if all preconditions of op satisfied:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
            queue.push((op.hadd_value, effect))
return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)

op.hadd_value = op.op_cost for all operators op

while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition:
        return sum of g.hadd_value for all g
    for operator op with precondition prop:
        op.hadd_value += p_value
    mark precondition prop of op as satisfied
    if all preconditions of op satisfied:
        if op.hadd_value < op.effect.hadd_value:
            op.effect.hadd_value = op.hadd_value
        queue.push((op.hadd_value, effect))

return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)

op.unsat_preconditions = op.precond.size() for all operators op

op.hadd_value = op.op_cost for all operators op

while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition:
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
            queue.push((op.hadd_value, effect))
return deadend
Exploration Queue Algorithm for $h^{add}$

queue = initialize_priority_queue(I)

op.unsat_preconditions = op.precond.size() for all operators op

op.hadd_value = op.op_cost for all operators op

while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop is last unsatisfied goal proposition
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
                queue.push((op.hadd_value, effect))
    return deadend
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)
op.unsat_preconditions = op.precond.size() for all operators op

goal_counter = |G|
op.hadd_value = op.op_cost for all operators op

while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop.is_goal and --goal_counter == 0:
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
                queue.push((op.hadd_value, effect))

return deadend
Exploration Queue Algorithm for $h^{add}$

```python
queue = initialize_priority_queue(I)
op.unsat_preconditions = op.precond.size() for all operators op
goal_counter = |G|
op.hadd_value = op.op_cost for all operators op
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop.is_goal and --goal_counter == 0:
        return sum of g.hadd_value for all goal propositions g
    for operator op with precondition prop:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
        queue.push((op.hadd_value, effect))
return deadend
```


**Exploration Queue Algorithm for** $h^{\text{add}}$

```python
queue = initialize_priority_queue(I)
op.unsat_preconditions = op.precond.size() for all operators op
goal_counter = |G|
op.hadd_value = op.op_cost for all operators op
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop.is_goal and --goal_counter == 0:
        return sum of g.hadd_value for all goal propositions g
    for op in prop.precond_of:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
        queue.push((op.hadd_value, effect))
return deadend
```

**Prop**

- **hadd.value** : int = inf
- **is_goal** : bool
- **precond_of** :
  - vector<UnaryOperator>

**UnaryOperator**

- **precond** : vector<Prop>
- **effect** : Prop
- **op_cost** : int
- **hadd.value** : int
- **unsat.preconditions** : int
Exploration Queue Algorithm for $h^{\text{add}}$

queue = initialize_priority_queue(I)
op.unsat_preconditions = op.precond.size() for all operators op

goal_counter = |G|
op.hadd_value = op.op_cost for all operators op

while queue is not empty:
  (p_value, prop) = queue.pop()
  if prop.hadd_value < p_value:
    continue
  if prop.is_goal and --goal_counter == 0:
    return sum of g.hadd_value for all goal propositions g
  for op in prop.precond_of:
    op.hadd_value += p_value
    if --op.unsat_preconditions == 0:
      if op.hadd_value < op.effect.hadd_value:
        op.effect.hadd_value = op.hadd_value
    queue.push(((op.hadd_value, effect))

return deadend
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

\[ V = \{a, b, c, d, e, f, g, h\} \]
\[ I = \{a\} \]
\[ G = \{c, d, e, f, g\} \]
\[ A = \{a_1, a_2, a_3, a_4, a_5, a_6\} \]
\[ a_1 = \langle a \rightarrow b, c\rangle_3 \]
\[ a_2 = \langle a, c \rightarrow d\rangle_1 \]
\[ a_3 = \langle b, c \rightarrow e\rangle_1 \]
\[ a_4 = \langle b \rightarrow f\rangle_1 \]
\[ a_5 = \langle d \rightarrow e, f\rangle_1 \]
\[ a_6 = \langle d \rightarrow g\rangle_1 \]
**Illustrative Example:** $h^{\text{add}}$ with Exploration Queue

\[
V = \{a, b, c, d, e, f, g, h\}
\]

\[
I = \{a\}
\]

\[
G = \{c, d, e, f, g\}
\]

\[
A = \{a_1, a_2, a_3, a_4, a_5, a_6\}
\]

\[
a_1 = \langle a \rightarrow b, c \rangle_3
\]

\[
a_2 = \langle a, c \rightarrow d \rangle_1
\]

\[
a_3 = \langle b, c \rightarrow e \rangle_1
\]

\[
a_4 = \langle b \rightarrow f \rangle_1
\]

\[
a_5 = \langle d \rightarrow e, f \rangle_1
\]

\[
a_6 = \langle d \rightarrow g \rangle_1
\]
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

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

$a_1 = \langle a \rightarrow b, c \rangle_3$
$a_2 = \langle a, c \rightarrow d \rangle_1$
$a_3 = \langle b, c \rightarrow e \rangle_1$
$a_4 = \langle b \rightarrow f \rangle_1$
$a_5 = \langle d \rightarrow e, f \rangle_1$
$a_6 = \langle d \rightarrow g \rangle_1$
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

$I = \{a\}$

$G = \{c, d, e, f, g\}$

$A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$

$a_2 = \langle a, c \rightarrow d \rangle_1$

$a_3 = \langle b, c \rightarrow e \rangle_1$

$a_4 = \langle b \rightarrow f \rangle_1$

$a_5 = \langle d \rightarrow e, f \rangle_1$

$a_6 = \langle d \rightarrow g \rangle_1$
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

$I = \{a\}$

$G = \{c, d, e, f, g\}$
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 5

\[(0,a)\]

$I = \{a\}$
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 5

(0,a)

```
var:hadd_val
  a:0
  b:∞
  c:∞
  d:∞
  e:∞
  f:∞
  g:∞
  h:∞

op:hadd_val:unsat\_precond
  a_1,b:3:1
  a_1,c:3:1
  a_2,d:1:2
  a_3,e:1:2
  a_4,f:1:1
  a_5,e:1:1
  a_5,f:1:1
  a_6,g:1:1
```
unsatisfied_goalprops = 5

(0,a)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5
Illustrative Example: \( h^{\text{add}} \) with Exploration Queue

unsatisfied\_goalprops = 5
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied goalprops = 5

var:hadd_val

- a:0
- b:∞
- c:∞
- d:∞
- e:∞
- f:∞
- g:∞
- h:∞

op:hadd_val:unsat_precond

- $a_1,b:3:0$
- $a_1,c:3:1$
- $a_2,d:1:2$
- $a_3,e:1:2$
- $a_4,f:1:1$
- $a_5,e:1:1$
- $a_5,f:1:1$
- $a_6,g:1:1$
**Illustrative Example:** $h^{add}$ with Exploration Queue

unsatisfied_goalprops = 5

(3, b)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

(3,b)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 5

(3,b) (3,c)
Illustrative Example: $h^\text{add}$ with Exploration Queue

unsatisfied goalprops = 5

(3,b) (3,c)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

(3,b) (3,c)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 5

(3,b) (3,c)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

\[(3,b) \quad (3,c)\]
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied Goalprops = 5

(3,c)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

(3,c)
Illustrative Example: $h^{add}$ with Exploration Queue

unsatisfied_goalprops = 5

(3,c)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 5

(3,c) (4,f)
unsatisfied_goalprops = 5

(3,c) (4,f)
unsatisfied_goalprops = 4

(4,f)

Illustrative Example: $h^{\text{add}}$ with Exploration Queue
Illustrative Example: $h^\text{add}$ with Exploration Queue

unsatisfied\_goalprops = 4

(4,f) (4,d) (7,e)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 3

(4,d) (7,e)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied_goalprops = 2

(5,e) (5,g) (7,e)
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

$$\text{unsatisfied}_\text{goalprops} = 1$$

$$(5,g) \ (7,e)$$

```
var:hadd\_val

a:0
b:3
c:3
d:4
e:5
f:4
g:5
h:∞

op:hadd\_val:unsat\_precond

a₁,b:3:0
a₁,c:3:0
a₂,d:4:0
a₃,e:7:0
a₄,f:4:0
a₅,e:5:0
a₅,f:5:0
a₆,g:5:0
```
Illustrative Example: $h^{\text{add}}$ with Exploration Queue

unsatisfied\_goalprops = 0

(7,e)
Illustrative Example: $h^{\text{FF}}$
### Prop

- `hadd_value : int = infty`
- `is_goal : bool`
- `precond_of : vector<UnaryOperator>`
- `reached_by : UnaryOperator = 0`
- `marked : bool = false`

### UnaryOperator

- `precond : vector<Prop*>`
- `effect : Prop*`
- `op_cost: int`
- `hadd_value : int`
- `unsat_preconditions : int`
- `orig_op: Operator`
Exploration Queue Algorithm for $h^{FF}$

queue = initialize_priority_queue(I)
op.unsat_preconditions = op.precond.size() for all operators op
goal_counter = |G|
op.hadd_value = op.op_cost for all operators op
while queue is not empty:
    (p_value, prop) = queue.pop()
    if prop.hadd_value < p_value:
        continue
    if prop.is_goal and --goal_counter == 0:
        return sum of g.hadd_value for all goal propositions g
    for op in prop.precond_of:
        op.hadd_value += p_value
        if --op.unsat_preconditions == 0:
            if op.hadd_value < op.effect.hadd_value:
                op.effect.hadd_value = op.hadd_value
                op.effect.reached_by = op
        queue.push(op.hadd_value, effect)
return deadend
for goal_prop in G:
    mark_relaxed_plan(goal_prop)

void mark_relaxed_plan(goal_prop):
    // Only consider each subgoal once.
    if !goal_prop->marked
        goal_prop->marked = true;
    unary_op = goal.reached_by;
    // If we have not yet chained back to a start node.
    if unary_op:
        for prop in op.precond:
            mark_relaxed_plan(prop);
        mark unary_op.orig_op
## Experimental Results

<table>
<thead>
<tr>
<th>coverage</th>
<th>expqueue</th>
<th>textbook</th>
</tr>
</thead>
<tbody>
<tr>
<td>airport (50)</td>
<td>34</td>
<td>31</td>
</tr>
<tr>
<td>barman-opt11-strips (20)</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>elevators-sat08-strips (30)</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>floortile (40)</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>freecell (80)</td>
<td>79</td>
<td>76</td>
</tr>
<tr>
<td>logistics98 (35)</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>mprime (35)</td>
<td>31</td>
<td>19</td>
</tr>
<tr>
<td>mystery (30)</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>nomystery (40)</td>
<td>26</td>
<td>21</td>
</tr>
<tr>
<td>parcprinter (70)</td>
<td>43</td>
<td>70</td>
</tr>
<tr>
<td>parking (40)</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>pipesworld-notankage (50)</td>
<td>33</td>
<td>29</td>
</tr>
<tr>
<td>pipesworld-tankage (50)</td>
<td>21</td>
<td>25</td>
</tr>
<tr>
<td>satellite (36)</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>scanalyzer-opt11-strips (20)</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>tidybot (40)</td>
<td>31</td>
<td>23</td>
</tr>
<tr>
<td>tpp (30)</td>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td>trucks-strips (30)</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>woodworking-sat08-strips (30)</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>554</strong></td>
<td><strong>492</strong></td>
</tr>
</tbody>
</table>