Linköping University

Automated Planning

Tweaking Search Strategies

Jonas Kvarnström
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2019

Baseline

jonkv@ida

- Baseline: General **Search-Based Planning Algorithm**:

Initialization

Selection

Solution check

Node expansion

```
search(problem) {
    initial-node ← make-initial-node(problem) // [2]
    open ← { initial-node }
    while (open ≠ ∅) {
        node ← search-strategy-remove-from(open) // [6]
        if is-solution(node) then  // [4]
            return extract-plan-from(node) // [5]

        foreach newnode ∈ successors(node) { // [3]
            add newnode to open
        }
    }
    // Expanded the entire search space without finding a solution
    return failure;
}
```

Typically computes $h(initial - node)$

Typically computes $h(newnode)$ to place *newnode* correctly in *open*

"Helpful Actions" in FF

Running Example

jonkv@ida

- Recall the **example** from **Relaxed Planning Graph** Heuristics
  - Prepare and serve a surprise dinner,
    take out the garbage,
    and make sure the present is wrapped before waking your sweetheart!

s_0 = {clean, garbage, asleep}

g = {clean, ¬garbage, served, wrapped}

Action	Preconds	Effects
cook()	clean	dinner
serve()	dinner	served
wrap()	asleep	wrapped
carry()	garbage	¬garbage, ¬clean
roll()	garbage	¬garbage, ¬asleep
clean()	¬clean	clean
buy()	–	dinner // New action to illustrate a point

Search Tree

jonkv@ida

- Let's do **heuristic forward state space search** with h_{FF} …
  - First step: Compute $h_{FF}(s_0)$

s_0 = {clean, garbage, asleep}

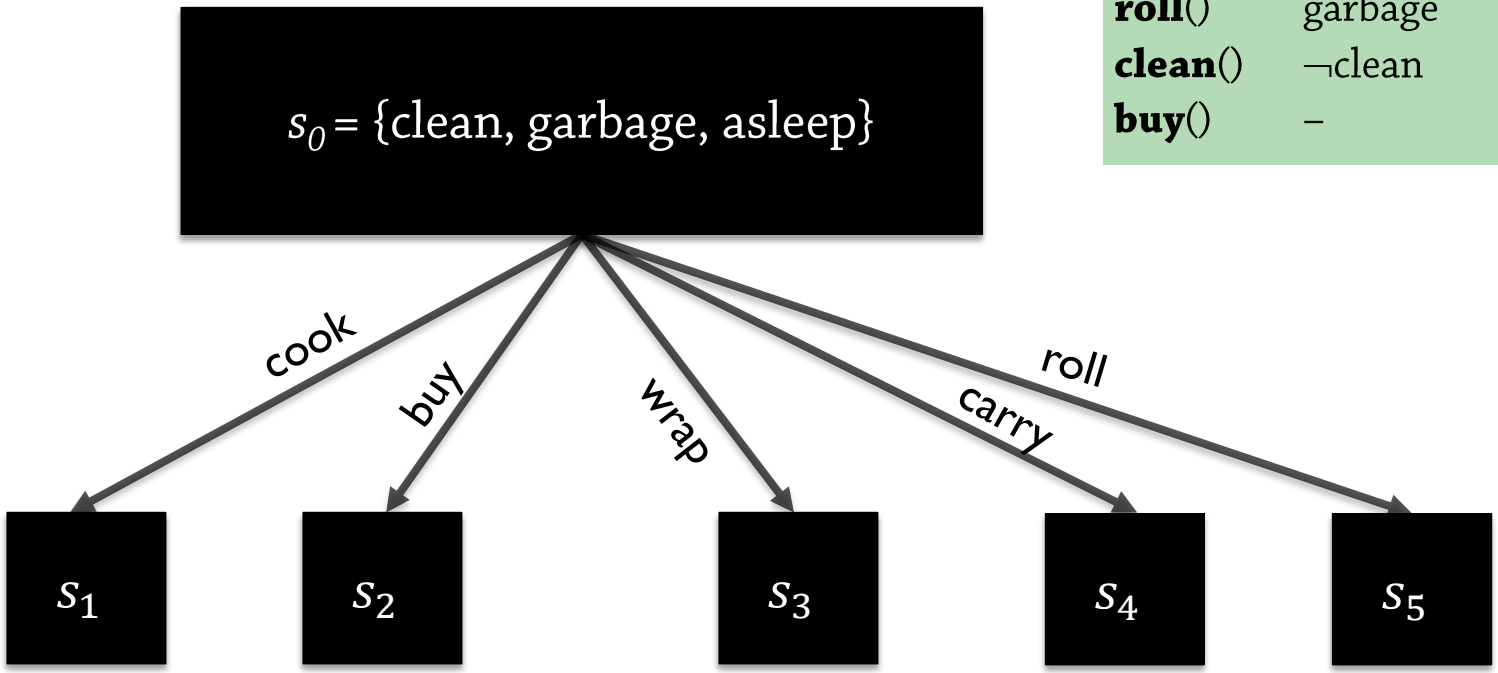
  - Does not satisfy the goal
    - Let's create successors

Search Tree

jonkv@ida

- Beginning of search tree:
  - 5 applicable actions
  - 5 successors

Action	Preconds	Effects
cook()	clean	dinner
serve()	dinner	served
wrap()	asleep	wrapped
carry()	garbage	¬garbage, ¬clean
roll()	garbage	¬garbage, ¬asleep
clean()	¬clean	clean
buy()	–	dinner

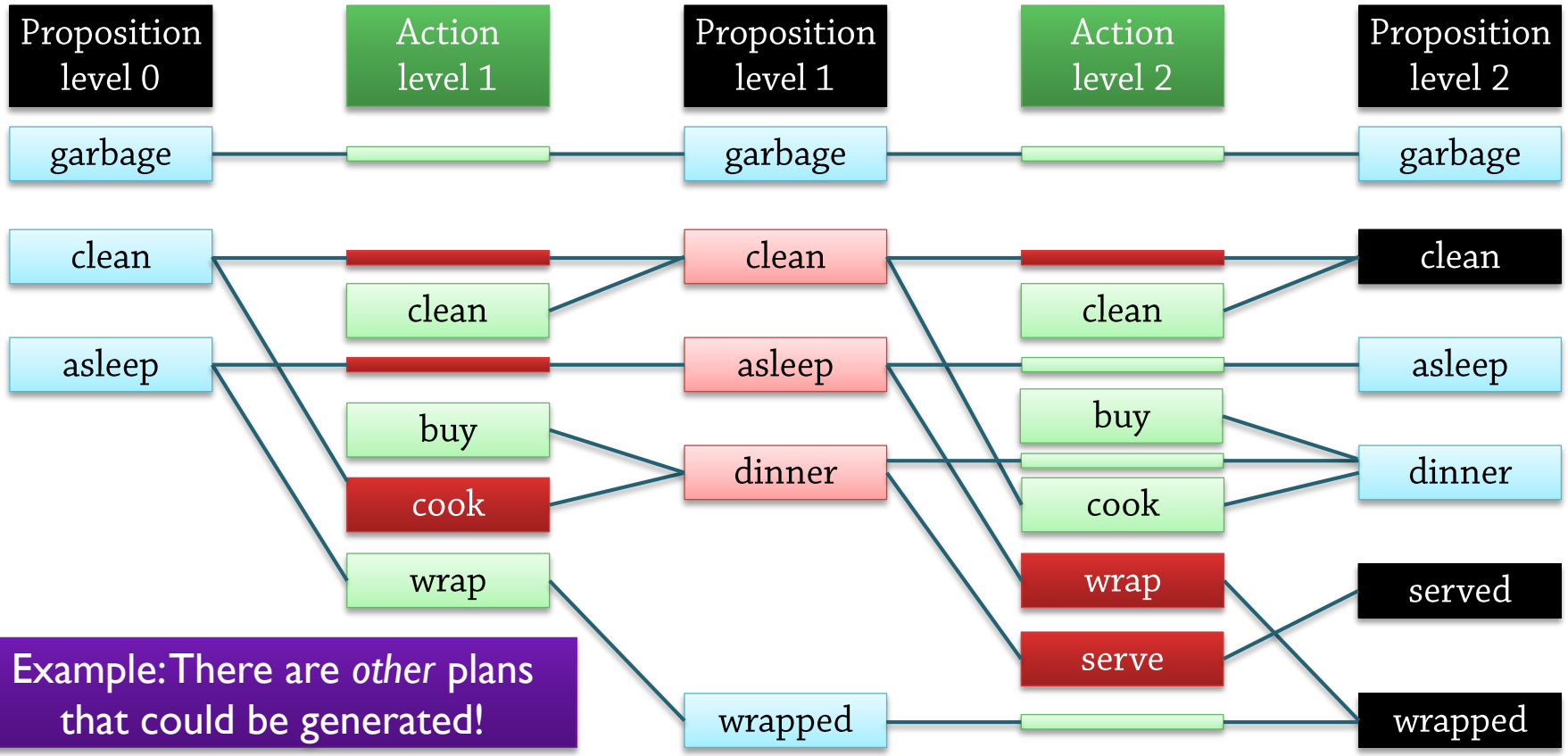


Are all successors equally likely to be ”useful”?

Relaxed Planning Graph Heuristics

jonkv@ida

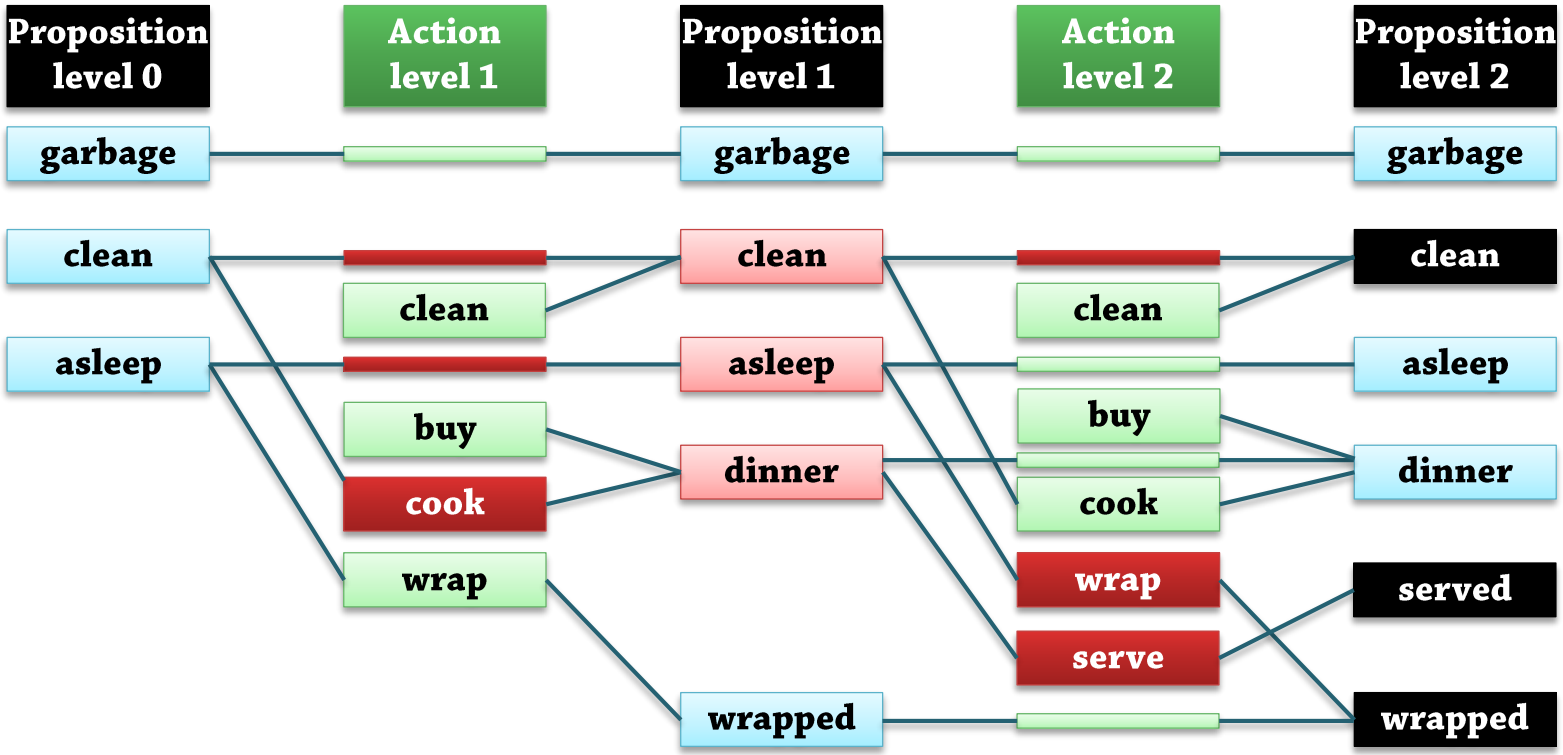
- What did we do when we **computed** $h_{FF}(s_0)$?
  - Construct a *relaxed planning graph* starting in *s*
  - Extract a *relaxed plan* (sufficient for achieving the goal in the relaxed problem)
  - $h_{FF}(s)$ = the *cost* of the relaxed plan



Action Level 1

jonkv@ida

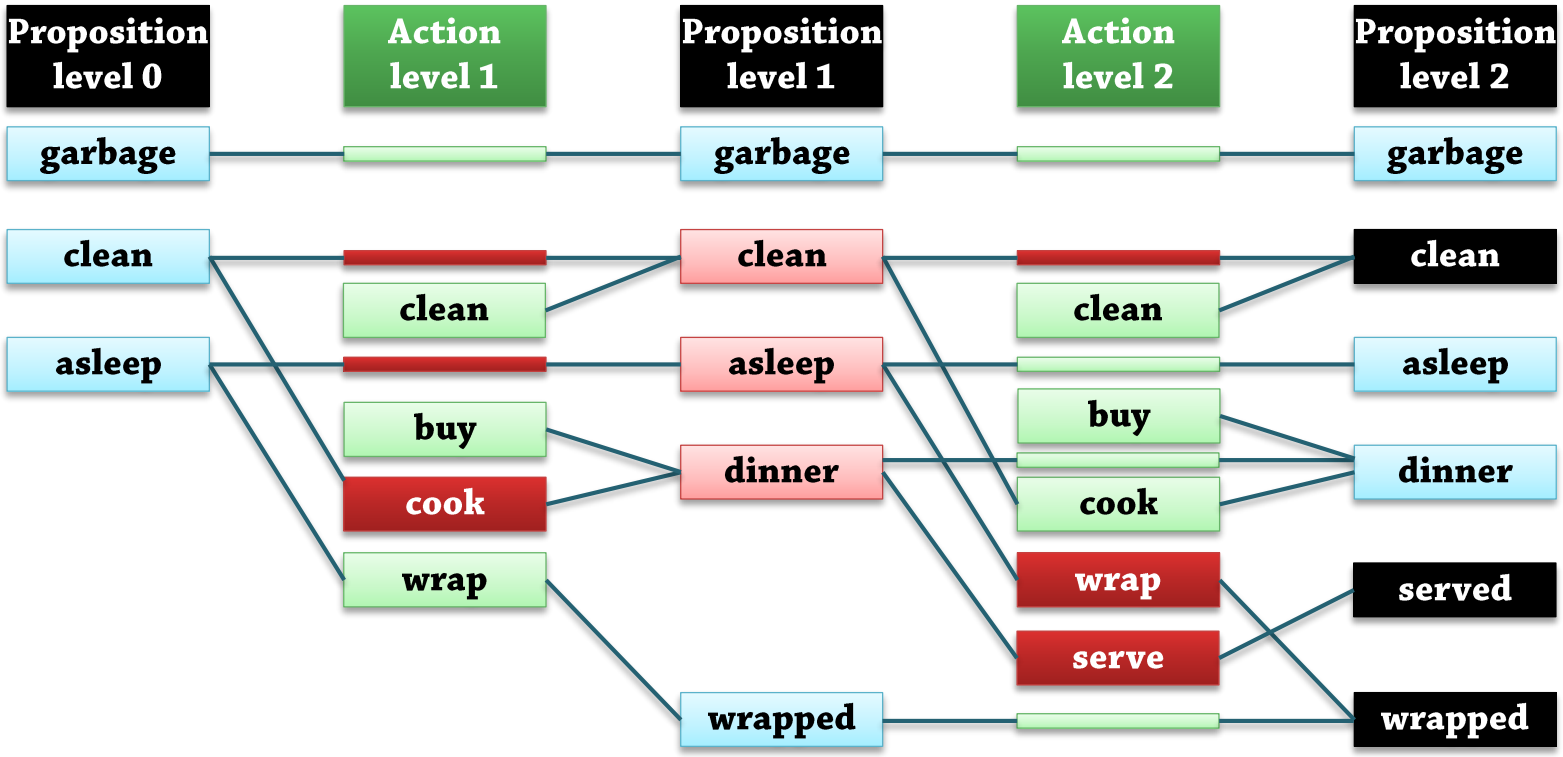
- Consider the actions **selected** at **action level 1**
  - Might be more likely to be useful as **first** actions…
    - Were useful in the **relaxed** problem, where you found a **complete** relaxed plan!
  - ➔ First action *cook*?
  - **No, too restrictive**



Action Level 1

jonkv@ida

- Consider the actions **selected** at **action level 1**
  - Then consider all **alternative actions** that could **achieve the same facts**
  - $H(s) = \{o \mid pre(o) \subseteq s \wedge effects^{+}(o) \cap PropLevel1 \neq \emptyset\}$
    = {cook, buy}
  - Called **helpful actions** or (later) **preferred operators**

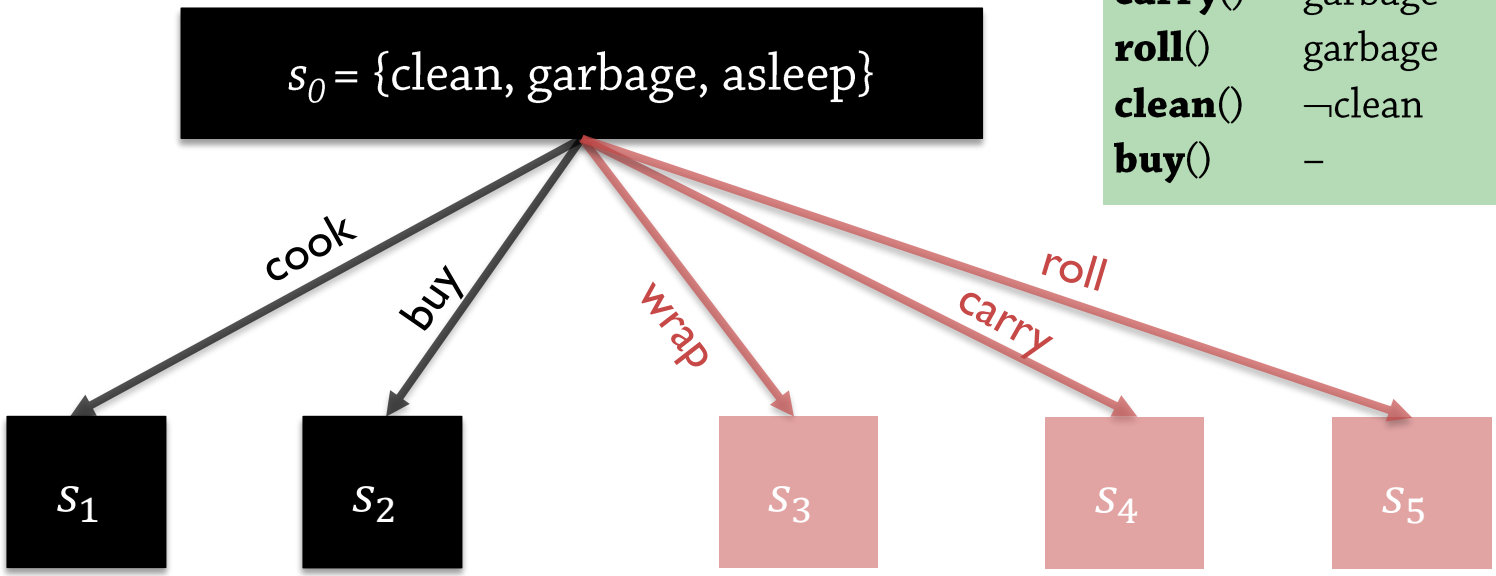


Search Tree

jonkv@ida

- New Beginning of search tree:
  - 2 applicable actions
  - 2 successors

Action	Preconds	Effects
cook()	clean	dinner
serve()	dinner	served
wrap()	asleep	wrapped
carry()	garbage	¬garbage, ¬clean
roll()	garbage	¬garbage, ¬asleep
clean()	¬clean	clean
buy()	–	dinner



Generated…

But not guaranteed to be useful in practice

Not generated at all!

Though not guaranteed to be useless…

FF: EHC with Helpful Action Pruning

jonkv@ida

- **EHC** with **helpful actions**:
  - EHC(initial state I , goal G)

```
plan    ← EMPTY
s       ← I
while h_FF(s) != 0 do
        execute breadth first search from s,
        using only helpful actions,
        to find the first s' such that h_FF(s') < h_FF(s)

        if no such state is found then fail

        plan ← plan + actions on the path to s'
        s ← s'
end while
return plan
```

The state space definition of **successors**(*node*)
is tweaked to *only* generate successors using actions in **H**(*node*)

Incomplete
if there are dead ends!

Actions not in H(s)
may be required;
not detected due to
relaxation…

If EHC fails,
FF falls back on
best-first search
using $f(s)=h_{FF}(s)$

Dual Queue Techniques

Helpful Actions and Completeness

jonkv@ida

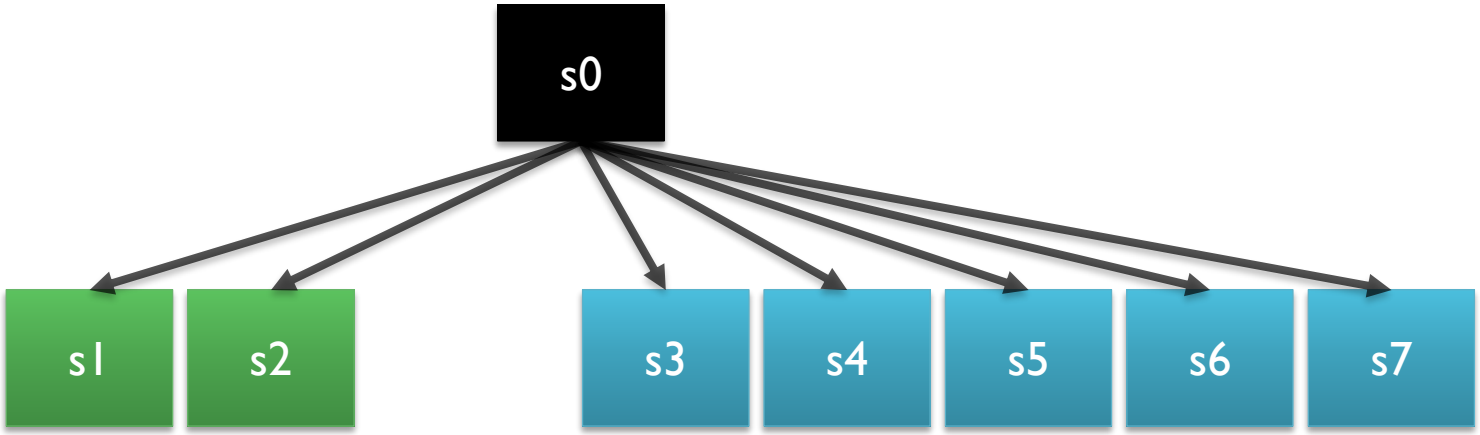
- Using **helpful actions** for **pruning** leads to **incompleteness**
  - May search for a long time,
    exhaust the search space,
    then start over using complete search

- ”Helpful actions” are **more likely** to be helpful
  - But skipping the other actions **completely** is too strict!

Pruning vs Prioritization

jonkv@ida

- Fast Downward: **Prioritize** helpful actions
  - Successors created by **helpful actions in H(s)** (called *preferred operators*) are **preferred** successors
  - Successors created by **other actions** are **ordinary** successors



Generally much fewer!

Dual Queues (1)

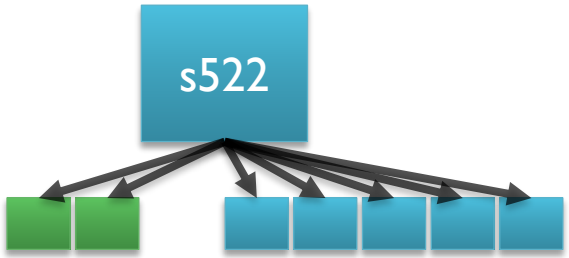
jonkv@ida

- Fast Downward introduced **dual queues** (two ”open lists”)
  - One for states generated as **preferred** successors
  - One for the **ordinary** states

Dual Queues (2)

jonkv@ida

- To expand a state:
  - Pick the **best** state from the **preferred** queue, and expand it
  - Pick the **best** state from the **ordinary** queue, and expand it



Preferred

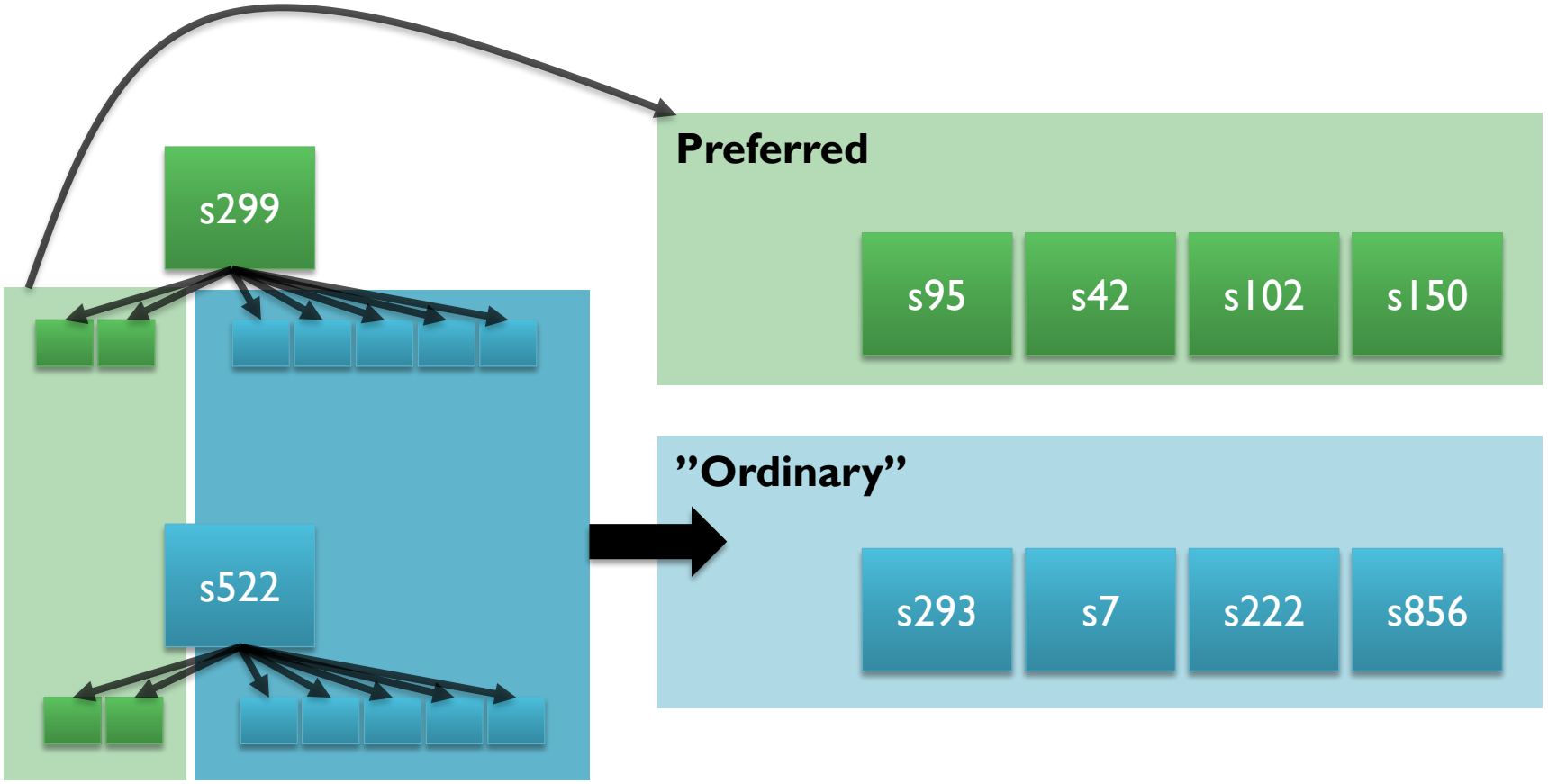
s95 s42 s102 s150

"Ordinary"

s293 s7 s222 s856

Dual Queues (3)

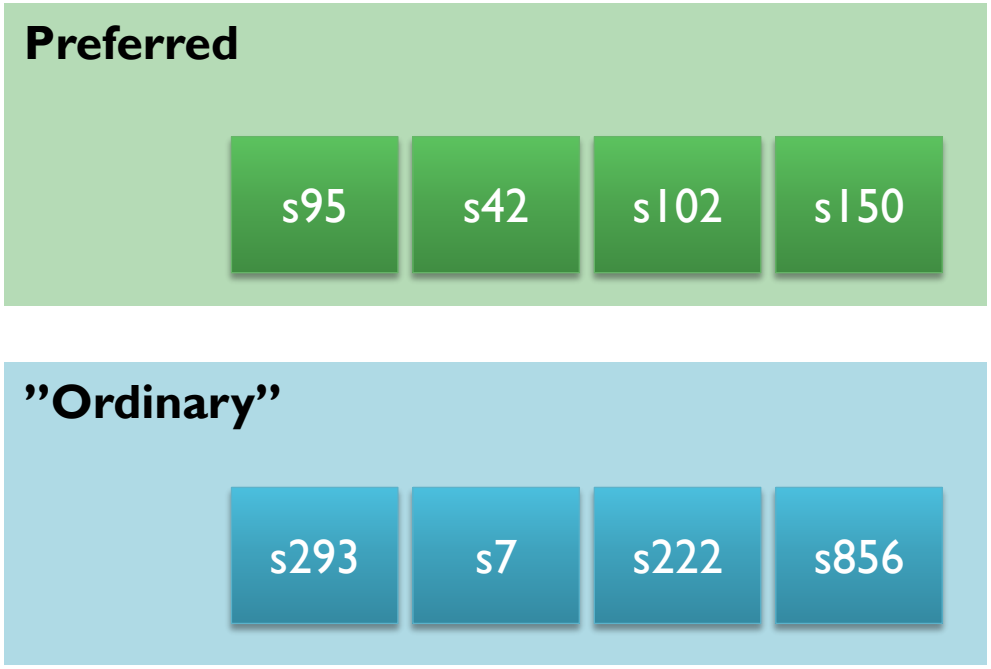
- After expansion:
  - Place all new states where they belong



Dual Queues (4)

jonkv@ida

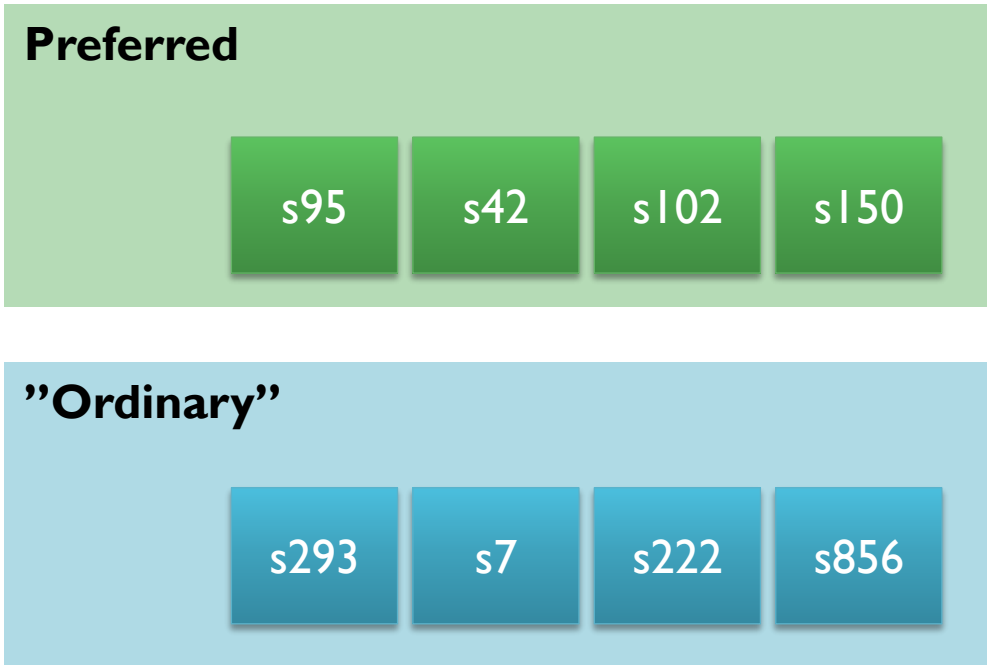
- **Fewer** states are preferred
  - Reached more quickly in the queue
- If we "**misclassified**" an action as non-helpful:
  - Don't have to exhaust the "preferred part" of the search space before we can "recover"
  - Search is *complete*



Boosted Dual Queues

jonkv@ida

- **Boosted** Dual Queues:
  - Used in later versions of Fast Downward and LAMA
  - Whenever progress is made (better *h*-value reached):
    - Choose **1000** times from the preferred queue
    - (Each chosen state is expanded as usual, *modifying* both queues… Then you pick again)



  - If progress is made again within these 1000 successors:
    - Add another 1000, accumulating
    - (Progress made after 300 ➔ keep expanding 1700 more)

Boosted Dual Queues

jonkv@ida

- **Boosted** Dual Queues:
  - After reaching the preferred successor limit:
    - Expand a **single** node from the non-preferred queue
  - Still complete
    - More aggressive than ordinary dual queues
    - Less aggressive than pure pruning

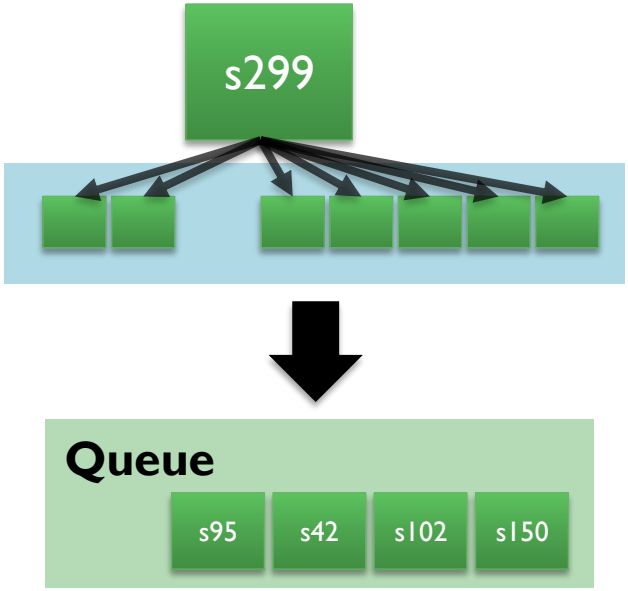
Deferred Evaluation / Lazy Search

Deferred Evaluation

jonkv@ida

- Standard **best-first** search:
  - Remove the "best" (most promising) state from the open list / priority queue
  - Check whether it satisfies the goal
  - Generate all successors
  - Calculate their heuristic values
  - Place in priority queue(s)

Typically takes most of the time



Deferred Evaluation (2)

jonkv@ida

- Potentially faster: **Deferred Evaluation** (Fast Downward, …)
  - Remove the "best" state from the priority queue
  - Check whether it satisfies the goal
  - Calculate **its** heuristic value (**only one**!)
  - Generate all successors
  - Place in priority queue using the **parent's** heuristic value

Takes less time, but less accurate heuristic – "one step behind"
Often **faster** but **lower-quality** plans