

Preferred Operators and Deferred Evaluation in Satisficing Planning

Silvia Richter¹ Malte Helmert²

¹Griffith University & NICTA, Australia

²Albert-Ludwigs-Universität Freiburg, Germany

ICAPS 2009

Outline

- 1 Introduction
 - Deferred Evaluation
 - Preferred Operators
- 2 Experiments on IPC benchmarks
- 3 Experiment with Artificial Search Space
- 4 Summary

Outline

- 1 Introduction
 - Deferred Evaluation
 - Preferred Operators
- 2 Experiments on IPC benchmarks
- 3 Experiment with Artificial Search Space
- 4 Summary

Motivation

Preferred operators and **deferred evaluation** used by many planners (Fast Downward, LAMA, Temporal Fast Downward, many research systems)

However, **little empirical evidence to date:**

- Best use of preferred operators?
(cf. FF's helpful actions vs. Fast Downward's pref. ops.)
- How useful is deferred evaluation?
- Is there interaction?

We fill this gap in common knowledge

Experiments conducted with greedy best-first search

Deferred Evaluation

Standard greedy best-first search (“Eager”):

Given a state,

- Generate successors and compute their heuristic values
- Enqueue successors with their h -values
- Remove state with minimal h -value from queue, iterate

Observation: Most time (80%!) spent on heuristic computation, often many more states **generated** than **expanded**

↪ Idea: compute h -values only for expanded states

Deferred Evaluation (cont.)

Deferred evaluation (“Lazy”):

- Do **not** compute h -values of successors immediately
- Enqueue successors with **parent's h -value**
- When removing state from queue, compute its h -value

Lazy **reduces expansion time**, but results in **loss of heuristic accuracy**

Preferred Operators

Idea of preferred operators:

Prefer actions that are **likely to lead to better state**

- Those which appear in simplified solution (and are applicable)
- Identified during computation of heuristic

Generation:

- FF heuristic: actions from relaxed plan, e. g. **best supports** for necessary facts, calculated recursively from goals
- Additive heuristic: same method as in FF heuristic
- Causal Graph (CG) & Context-enhanced additive (cea) heuristics: actions that achieve necessary changes in DTGs

Preferred Operators (cont.)

Methods of exploitation:

- 1 Pruning (FF)
- 2 Dual queue (Fast Downward)
- 3 Boosted dual queue (IPC Fast Downward, LAMA)
- 4 *Pref.* > *heur.*, i. e. “preferredness before heuristic” (YAHSP)
- 5 *Heur.* > *pref.*, i. e. “heuristic before preferredness”

Preferred Operators (cont.)

- 1 Pruning (FF)
 - Prune all states that are not preferred successors
 - If that fails (no solution), restart without pref. ops.
 - **Radical**, but potentially very helpful method due to **reduced branching factor**
- 2 Dual queue (Fast Downward)
- 3 Boosted dual queue (IPC Fast Downward, LAMA)
- 4 *Pref.* > *heur.*, i. e. “preferredness before heuristic” (YAHSP)
- 5 *Heur.* > *pref.*, i. e. “heuristic before preferredness”

Preferred Operators (cont.)

- 1 Pruning (FF)
- 2 Dual queue (Fast Downward)
 - Add separate queue for preferred successors
 - When expanding states, **alternate between queues**
 \rightsquigarrow preferred successors expanded sooner, on average
 - Less radical than pruning, search remains **complete**
- 3 Boosted dual queue (IPC Fast Downward, LAMA)
- 4 *Pref.* > *heur.*, i. e. “preferredness before heuristic” (YAHSP)
- 5 *Heur.* > *pref.*, i. e. “heuristic before preferredness”

Preferred Operators (cont.)

- 1 Pruning (FF)
- 2 Dual queue (Fast Downward)
- 3 **Boosted dual queue (IPC Fast Downward, LAMA)**
 - Expand states from **preferred queue more often**
 - Whenever progress was made (new best h-value), expand 1000 more preferred successors
 - Boosts accumulate if progress made again within 1000 states
 - **Complete** as dual queue, but **more aggressive** if search progresses quickly
- 4 *Pref.* > *heur.*, i. e. “preferredness before heuristic” (YAHSP)
- 5 *Heur.* > *pref.*, i. e. “heuristic before preferredness”

Preferred Operators (cont.)

- 1 Pruning (FF)
- 2 Dual queue (Fast Downward)
- 3 Boosted dual queue (IPC Fast Downward, LAMA)
- 4 *Pref. > heur.*, i. e. “preferredness before heuristic” (YAHSP)
 - Expand preferred successors **whenever possible**
 - Only if pref. succs. run out, use non-preferred successors
 - Within each group, choose as usual (according to h -values)
 - **Similar to pruning** (expands same states as long as pref. succs. present)
 - Unlike pruning **complete**, but **does not reduce branching factor**
- 5 *Heur. > pref.*, i. e. “heuristic before preferredness”

Preferred Operators (cont.)

- 1 Pruning (FF)
- 2 Dual queue (Fast Downward)
- 3 Boosted dual queue (IPC Fast Downward, LAMA)
- 4 *Pref.* > *heur.*, i. e. “preferredness before heuristic” (YAHSP)
- 5 *Heur.* > *pref.*, i. e. “heuristic before preferredness”
 - Use preferred operators for **tie-breaking**
 - Always expand the state with lowest *h*-value
 - If several states equally good, expand any preferred ones first
 - Comparatively **subtle** way of using preferred operators
 - Expectation: likely to be useful (and unlikely to be harmful)

Outline

- 1 Introduction
 - Deferred Evaluation
 - Preferred Operators
- 2 Experiments on IPC benchmarks
- 3 Experiment with Artificial Search Space
- 4 Summary

Experimental Setup

- All IPC tasks from 1998–2006
- 4 heuristics: FF, cea, CG, additive
- Results measured via scores with range 0–100 (higher score better)
- **Coverage**: normalised percentage of tasks solved
- **Quality** (plan length): IPC-2008 criterion $q^*/q \cdot 100$
- **Evaluations**: $\leq 100 \Rightarrow 100$ pts., $\geq 1,000,000 \Rightarrow 0$ pts., log interpolation
- **Runtime**: $\leq 1\text{sec} \Rightarrow 100$ pts., $\geq 30\text{min} \Rightarrow 0$ pts., log interpolation

Results FF Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
FF Heuristic					
Lazy	None	74.03	68.19	54.34	67.63
	Heur. > Pref.	79.11	72.65	62.92	72.29
	Pref. > Heur.	83.71	75.40	70.75	79.08
	Pruning	84.02	76.01	70.52	79.49
	Dual Queue	83.43	77.00	65.11	76.14
	B. Dual Queue	86.74	78.83	72.58	81.95
Eager	None	75.07	72.08	52.42	68.08
	Heur. > Pref.	75.80	72.24	54.28	69.31
	Pref. > Heur.	81.42	76.86	58.20	74.66
	Pruning	84.64	79.33	68.12	79.74
	Dual Queue	86.89	83.22	60.91	79.17
	B. Dual Queue	83.65	79.22	58.60	75.96

Results FF Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
FF Heuristic					
Lazy	None	74.03	68.19	54.34	67.63
	Heur. > Pref.	79.11	72.65	62.92	72.29
	Pref. > Heur.	83.71	75.40	70.75	79.08
	Pruning	84.02	76.01	70.52	79.49
	Dual Queue	83.43	77.00	65.11	76.14
	B. Dual Queue	86.74	78.83	72.58	81.95
Eager	None	75.07	72.08	52.42	68.08
	Heur. > Pref.	75.80	72.24	54.28	69.31
	Pref. > Heur.	81.42	76.86	58.20	74.66
	Pruning	84.64	79.33	68.12	79.74
	Dual Queue	86.89	83.22	60.91	79.17
	B. Dual Queue	83.65	79.22	58.60	75.96

Results FF Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
FF Heuristic					
Lazy	None	74.03	68.19	54.34	67.63
	Heur. > Pref.	79.11	72.65	62.92	72.29
	Pref. > Heur.	83.71	75.40	70.75	79.08
	Pruning	84.02	76.01	70.52	79.49
	Dual Queue	83.43	77.00	65.11	76.14
	B. Dual Queue	86.74	78.83	72.58	81.95
Eager	None	75.07	72.08	52.42	68.08
	Heur. > Pref.	75.80	72.24	54.28	69.31
	Pref. > Heur.	81.42	76.86	58.20	74.66
	Pruning	84.64	79.33	68.12	79.74
	Dual Queue	86.89	83.22	60.91	79.17
	B. Dual Queue	83.65	79.22	58.60	75.96

Results Cea Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
Cea Heuristic					
Lazy	None	74.06	67.85	54.97	66.08
	Heur. > Pref.	75.35	69.25	60.63	68.44
	Pref. > Heur.	83.57	74.71	68.39	76.50
	Pruning	83.19	75.16	68.04	76.37
	Dual Queue	80.14	73.45	63.68	72.68
	B. Dual Queue	86.24	77.48	70.23	78.81
Eager	None	74.02	70.91	51.99	65.03
	Heur. > Pref.	74.00	70.77	52.44	65.26
	Pref. > Heur.	81.90	76.49	56.48	71.75
	Pruning	83.48	77.98	66.17	76.44
	Dual Queue	85.23	81.40	59.05	74.86
	B. Dual Queue	84.10	78.39	56.89	72.58

Results CG Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
CG Heuristic					
Lazy	None	71.15	64.97	50.01	64.50
	Heur. > Pref.	72.96	65.46	54.29	66.53
	Pref. > Heur.	73.38	66.81	54.69	66.62
	Pruning	72.38	66.05	52.36	65.37
	Dual Queue	75.10	67.94	56.53	68.68
	B. Dual Queue	76.38	69.57	56.47	69.42
Eager	None	71.82	68.51	47.90	64.20
	Heur. > Pref.	71.97	68.41	48.17	64.46
	Pref. > Heur.	71.22	67.78	47.34	64.30
	Pruning	72.39	67.70	50.20	64.70
	Dual Queue	75.74	72.61	49.67	67.52
	B. Dual Queue	71.80	68.34	47.37	64.50

Results Additive Heuristic

Search	Pref. Op. Use	Coverage Score	Quality Score	Eval. Score	Time Score
Additive Heuristic					
Lazy	None	70.84	63.72	50.64	63.93
	Heur. > Pref.	73.80	66.46	57.27	67.09
	Pref. > Heur.	83.56	75.08	66.96	77.58
	Pruning	84.07	75.66	66.88	78.17
	Dual Queue	78.18	70.57	59.75	70.75
	B. Dual Queue	85.55	77.32	68.95	79.93
Eager	None	71.59	67.83	49.02	64.24
	Heur. > Pref.	72.26	67.94	49.94	64.97
	Pref. > Heur.	83.35	78.01	55.88	74.05
	Pruning	84.88	79.47	65.24	78.60
	Dual Queue	84.24	79.91	57.06	75.47
	B. Dual Queue	84.71	79.45	56.33	75.08

Results Overview (cont.)

Preferred operators:

- Pref. ops. drastically improve performance (by 10–15 pts.)
⇒ no. of unsolved tasks **halved**
- Pref. ops. much more important than heuristic (≤ 5 pts.)
- Best exploitation method depends on search type
- Results consistent across heuristics
- Best: **Eager + dual queue** or **Lazy + boosted dual queue**
- Stronger use of pref. ops. better for Lazy

Results Overview (cont.)

Search type:

- Eager and Lazy **comparable on average** re. coverage
- However: Lazy huge advantages in **some cases**
- Lazy **faster**, but **worse-quality** plans

Outline

- 1 Introduction
 - Deferred Evaluation
 - Preferred Operators
- 2 Experiments on IPC benchmarks
- 3 Experiment with Artificial Search Space
- 4 Summary

A Controlled Experiment

Q.: How does performance vary with **heuristic accuracy** and **quality of pref. ops.**?

Experiment with artificial search space

- Start state has **approx. goal distance** (*agd*)
- Probability distribution over *agds* of successors
- States with *agd* 0 are goals

We test Eager in 3 configurations: without pref. ops., pruning and dual queue

For each data point, results averaged over 100 runs

A Controlled Experiment (cont.)

We vary 2 parameters *dev. fac.* and *rec. rate*

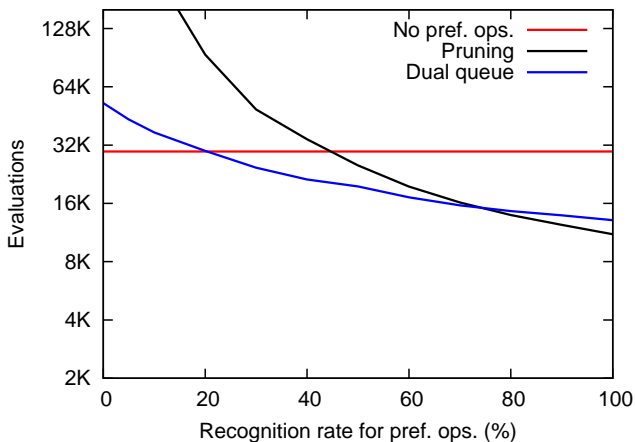
Heuristic:

- Quality parameterized by number *dev. fac.* in range 0.1–0.9
- Defines **how much *h*-value may deviate from *agd*** of state
- Then $dev(s) := \text{random}(0, dev. fac.)$, and
 $h(s) := agd(s) - (dev(s) \times agd(s))$

Preferred Operators:

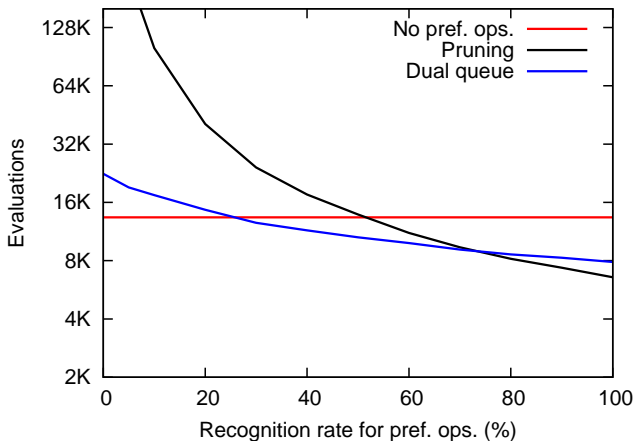
- Quality parameterized by number *rec. rate* in range 0%–100%
- If an action leads to a better state, **prefer it randomly with probability *rec. rate***
- Else, prefer it with 10% probability

Varying Quality of Preferred Operators (*Rec. Rate*)



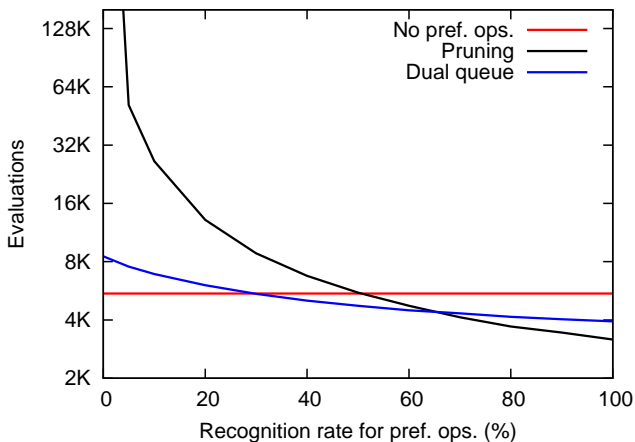
Heuristic quality low (*dev. fac.* 0.5)

Varying Quality of Preferred Operators (*Rec. Rate*)



Heuristic quality medium (*dev. fac.* 0.7)

Varying Quality of Preferred Operators (*Rec. Rate*)



Heuristic quality high (*dev. fac.* 0.9)

Results

- Even with **high-quality** heuristic, performance can be notably improved through good preferred operators
- However, **bad pref. ops.** may be **detrimental** to search
- **Pruning reacts worse to bad pref. ops.** than dual queue
- Dual queue good if *rec. rate* $> 20\text{--}30\%$
- Pruning good if *rec. rate* $> 40\text{--}50\%$

Outline

- 1 Introduction
 - Deferred Evaluation
 - Preferred Operators
- 2 Experiments on IPC benchmarks
- 3 Experiment with Artificial Search Space
- 4 Summary

Summary

- Preferred operators are **extremely useful**
- Deferred evaluation can be useful (but isn't on average)
- **Dual queue** is best method for pref. ops. in a **standard BFS**
- If using deferred evaluation, then use boosted dual queue
- **Eager + dual queue \approx Lazy + boosted dual queue**

Thank you!

Questions?