Preferred Operators and Deferred Evaluation in Satisficing Planning

Silvia Richter
Griffith University, Queensland, Australia
and
NICTA, Queensland, Australia
silvia.richter@nicta.com.au

Malte Helmert
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

Heuristic forward search is the dominant approach to satisficing planning to date. Most successful planning systems, however, go beyond plain heuristic search by employing various search-enhancement techniques. One example is the use of helpful actions or preferred operators, providing information which may complement heuristic values. A second example is deferred heuristic evaluation, a search variant which can reduce the number of costly node evaluations. Despite the wide-spread use of these search-enhancement techniques however, we note that few results have been published examining their usefulness. In particular, while various ways of using, and possibly combining, these techniques are conceivable, no work to date has studied the performance of such variations. In this paper, we address this gap by examining the use of preferred operators and deferred evaluation in a variety of settings within best-first search. In particular, our findings are consistent with and help explain the good performance of the winners of the satisficing tracks at IPC 2004 and 2008.

Introduction

In the past decade, heuristic forward search in the state space has been the dominant approach to satisficing planning. This is made evident by the fact that 4 out of 5 winners of the satisficing track in the biennial international planning competition (IPC) since 2000 have been heuristic forward-search planners. There has also clearly been a propagation of ideas in the way that certain aspects of past successful planning systems have been adopted by more recent systems. One example is the use of *helpful actions* in the venerable FF planner by Hoffmann and Nebel (2001), winner of the satisficing track at IPC-2000. Helpful actions are actions that contribute to solving a simplified version of the task. As they are likely to also contribute to solving the original task, they can be preferred over actions that are not considered helpful. This search enhancement, which has been shown to improve FF's performance notably (Hoffmann and Nebel 2001), has been adapted by the Fast Downward system (Helmert 2006), winner of the satisificing track at IPC-2004, under the name *preferred operators*. Fast Downward, in turn, inspired many newer planners like LAMA (Richter and Westphal 2008), the satisficing winner at IPC-2008; Temporal Fast Downward (Röger, Eyerich, and Mattmüller 2008), and systems used for experiments in various publications (Keyder and Geffner 2009; Richter, Helmert, and Westphal 2008; Helmert and Geffner 2008). As a result, these systems all use the *deferred evaluation* popularised by Fast Downward. Deferred evaluation is a variant of best-first where the successors of a node are not heuristically evaluated when they are generated, but later when they are expanded. This may save time if many more nodes are generated than expanded.

The starting point for this paper is to note that even though many current planning systems use the search enhancements mentioned above (preferred operators and deferred evaluation), there exists little data on their respective usefulness. For example, the use of preferred operators (resp. helpful actions) in Fast Downward is different from their use in FF. While the authors of both planners report significantly improved performance compared to *not* using preferred operators, the question remains which usage is better, or how some of the many other conceivable approaches for using preferred operators would perform. Deferred evaluation, on the other hand, has been suggested to be particularly useful in combination with preferred operators (Helmert 2006). The question is open whether deferred evaluation is also advantageous in the absence of preferred operators, or when using preferred operators in a different fashion than the one examined by Helmert.

We address these questions by examining the performance of various approaches to using preferred operators as well as their interaction with deferred evaluation. One finding of this work is that deferred evaluation does not offer any benefit on standard benchmark tasks on average, but leads to strong improvement in some contexts and to strong deterioration in others. On the other hand, we find that the best method for using preferred operators strongly depends on whether deferred evaluation is being employed or not. Using an artificial search space, we demonstrate how the benefit that can be gleaned from preferred operators varies with different aspects of the search space.

Preferred Operators

One of the core features of the FF planner (Hoffmann and Nebel 2001) is the use of *helpful actions*. Helpful actions

are operators that are deemed promising by the FF heuristic because they form part of or achieve a precondition of the *relaxed plan*, a solution for a simplified version of the task which ignores delete effects. FF uses helpful actions to prune the search space and only evaluates successor states reached via helpful actions. This restriction makes the search incomplete, so that FF restarts without this pruning technique if it fails.

In the spirit of FF's helpful actions, Helmert (2006) subsequently coined the term *preferred operators* for all operators that are deemed promising for some reason. We will in the following use that general term to subsume helpful actions. The Fast Downward planner makes use of preferred operators in a dual-queue approach, where it keeps the states that are reached via a preferred operator (the *preferred successors*) in an additional open list. It selects the next state to be expanded alternately from its regular open list containing all successors and from the open list containing just preferred successors, pruning duplicates upon removal. A preferred successor is thus expanded sooner, on average, than an arbitrary successor of a state. In particular, at least every second state selected for expansion is a preferred successor.[1] In contrast to FF which evaluates only preferred successors (as long as possible), Fast Downward may gain less leverage from its preferred operators, but at the advantage that its search stays complete.

Deferred Heuristic Evaluation

The textbook version of best-first search keeps an open list of states to be expanded, ordered according to the heuristic estimates of their goal distance. In each step, a state with smallest heuristic value is removed from the open list and expanded by applying all operators applicable in the state. All successor states generated in this way are then evaluated and inserted into the open list.

The Fast Downward planner (Helmert 2006) incorporates a variation of best-first search called *deferred heuristic evaluation*, where the successors of a state are not evaluated upon generation. Instead, they are inserted into the open list with the heuristic estimate of their parent. Only upon being removed from the open list for expansion are they evaluated, so that their heuristic value can in turn be used for their successors. According to Helmert (2006), this technique can decrease the number of state evaluations substantially, especially when combined with preferred operators. For instance, consider the way preferred operators are used within Fast Downward (see above). If s' is a preferred successor of a state s, then s' will be expanded sooner than most of its siblings (as the fraction of preferred successors is usually small). If there exists a path with non-increasing heuristic values from s' to the goal, then the remaining siblings of s' will never be evaluated. By contrast, standard best-first search would evaluate all successors of s.

While deferred evaluation may reduce the number of state evaluations compared to standard best-first search, it usually *increases* the number of state expansions. This is due to the heuristic values being less informative, since they stem from the parent of a state rather than the state itself. Note however that successor states need not be generated upon expansion of their parent. Instead it is sufficient to store pointers in the open list to the parent state and to the creating operator, thus reducing memory requirements.

[1] Note that if preferred successors were put *exclusively* into the second queue and the fraction of preferred successors was more than 50%, we would expand more non-preferred successors.

Alternative Uses of Preferred Operators

The FF planner uses preferred operators for pruning the search space while Fast Downward uses them in a dual-queue approach. In both planning systems, preferred operators have been shown to improve performance (Hoffmann and Nebel 2001; Helmert 2006). However, the question is open as to which of the two usages leads to better results, assuming that all other aspects of a planner stay fixed. Furthermore, other ways of using preferred operators are conceivable which have not been analysed in the literature to date. An obvious idea is to use preferred operators for tie-breaking and expand, among states of equal heuristic value, preferred successors first. The dual has also been proposed (Vidal 2004), namely using the heuristic values for tie-breaking among equal preferredness: choose preferred successors whenever they exist, otherwise expand non-preferred successors. Within each of the two groups, choose according to heuristic values. This latter method places more emphasis on preferred operators than the former, but is not as restrictive as expanding only preferred successors like FF does.

The dual-queue approach can also be modified to give higher precedence to preferred successors. For example, the IPC versions of both Fast Downward and its offspring LAMA used a setting that increases the use of preferred successors at certain points in time. The planners keep a priority counter for each open list, initialised to 0. Whenever a state is removed from a list, the priority of that list is decreased by 1. At each iteration, the next state is removed from the list that has higher priority. Unless the priorities are changed outside of this routine, this method will alternate between choosing a state from the list of preferred successors and from the list of all successors. The setting used in Fast Downward and LAMA to increase the use of preferred operators is as follows: Whenever progress is made, i. e., whenever a state is discovered with a better heuristic estimate than the states before, the priority of the open list with only preferred successors is "boosted" by adding 1000. Therefore, at least the next 1000 states will now be removed from the list of preferred successors, and only if none of them leads to further improvement does the turn go back to the regular open list. If another improving state is found within the 1000 states, the boosts accumulate and it takes respectively longer until states from the regular queue are expanded again.

In this paper, we address the question which of these different uses for preferred operators leads to best performance. The answer may vary for different types of search; we examine greedy best-first search. We look at both the standard implementation of best-first search (in the following called *Eager*), and the deferred-evaluation variant (in the following called *Lazy*). The reason for examining both variants is

that they can behave very differently, with Lazy being less informed than Eager, but paying a smaller price for wrong expansions. Hence, the best use of preferred operators may be different for the two. Along the way, we answer the question whether deferred evaluation is useful in general and whether synergies exist between preferred operators and deferred evaluation.

Experimental Results

We have conducted a range of experiments to measure the relative performance of several uses for preferred operators (POs). The uses we compare are

- *No POs*, normal search without preferred operators.
- *PO pruning*, where the search space is restricted to only preferred successor states and the search restarts without POs if it exhausts the restricted search space without a solution. This corresponds to the use made of preferred operators in the FF planner (though FF uses enforced hill-climbing rather than best-first search as the underlying search algorithm).
- *Heur* $>$ *PO*, where preferred operators are used only to break ties among states of equal heuristic values.
- *PO* $>$ *Heur*, where preferred operators are used whenever possible, breaking ties with heuristic values. This approach has been used in the YAHSP planner (Vidal 2004).
- *Dual Queue*, where preferred successors are kept in a second open list in addition to the regular open list, and states are expanded by alternating between the two open lists. This approach has been used in work involving Fast Downward (Helmert and Geffner 2008).
- *Boosted Dual Queue*, the dual-queue approach where the open list with preferred successors is boosted whenever progress is made during the search, leading to more preferred successors being expanded. This approach was used in the IPC versions of Fast Downward and LAMA.

We ran experiments on all planning tasks from past international planning competitions between 1998 and 2006, totalling 1612 tasks. Four different planning heuristics were used: the FF heuristic (Hoffmann and Nebel 2001), the context-enhanced additive (cea) heuristic (Helmert and Geffner 2008), the causal graph (CG) heuristic (Helmert 2006) and the additive heuristic (Bonet and Geffner 2001). We follow the Fast Downward implementation of the FF heuristic by defining preferred operators to be those applicable operators which appear in the relaxed plan. By contrast, FF's "helpful actions" additionally include all applicable operators which add a *precondition* of an operator in the relaxed plan that is not true in the current state. For the additive heuristic we generate preferred operators in the same way as for the FF heuristic. (Without referring to relaxed plans, these preferred operators can alternatively be characterised as those applicable operators which contribute to the heuristic value, i.e. we select a best supporting action for the goal, according to the heuristic, and, recursively, a best supporting action for unsatisfied preconditions of selected actions.) The preferred operators in the CG and cea heuristics are those applicable operators that change the current value of a goal variable to a value that is on a lowest-cost path (as computed by the heuristic) to the goal value in the domain transition graph of that variable. If no such operator exists for a goal variable, the process recurses for variables that are involved in conditions for changing the value of the goal variable (Helmert 2006).

The experiments were conducted on a heterogeneous cluster of Intel Xeon and AMD Opteron CPUs, ranging from 2.2 GHz to 2.83 GHz. (The magnitude of the experiments precluded experiments on homogeneous machines.) To allow fair comparisons, for each given planning task all planner configurations using the same heuristic were run on the same CPU. The timeout was 30 minutes and the memory limit 1.75 GB in all cases.

For each planner configuration, we report results regarding coverage (the number of problems solved), the number of heuristic state evaluations, the runtime and the quality of the plans found. We report evaluations rather than expansions to be able to compare the two search types Eager and Lazy in a meaningful way. While the cost for an evaluation is the same for both search variants, the cost of an expansion differs greatly for the two. To expand a state, Lazy only puts two pointers into the open list, while Eager generates and evaluates all successor states. For Lazy, evaluations and expansions are the same; for both search types the number of evaluations is the number of states actually generated. Most importantly, roughly 80% of the total runtime of the planner is spent calculating heuristic values, making evaluations a platform-independent indicator of the search effort.

All reported results are measured via scores from the range 0–100, where best possible performance in a task is counted as 100, while failure to solve a task and worst performance are counted as 0. For coverage, a solved task counts 100. Evaluations and runtime are measured on a log scale due to the exponential nature of the problem. Performance better than a lower limit (100 states for evaluations and 1 second for runtime) counts as 100, performance worse than an upper limit (1,000,000 states for evaluations and the timeout of 30 minutes for runtime) counts as 0. In between, we interpolate with a logarithmic function. Plan quality is measured using the criterion of IPC 2008 (scaled by a factor): a solved task counts as $100 \cdot l^*/l$, where l is the length of the generated plan and l^* is the shortest plan length found by any of the configurations. The scores reported for domains are averaged over the tasks in the domain; when summarising results from multiple domains we show normalised averages such that each domain is weighed equally.

Table 1 shows the results for all configurations and heuristics. We begin by discussing the coverage (number of problems solved). The first thing we note is that the use of preferred operators almost always improves performance, and that the impact of preferred operators is significantly larger even than the choice of heuristic. With suitable use of preferred operators, the weakest heuristics (CG and additive) perform better than the strongest heuristics without preferred operators (FF and cea). Which use of preferred operators gives best results is dependent on the type of search: for eager search the simple dual-queue approach works best, for deferred evaluation the boosted dual-queue is best. The

Search	PO Use	Coverage Score	Eval. Score	Quality Score	Time Score
FF Heuristic					
Lazy	No POs	74.03	54.34	68.19	67.63
	Heur > PO	79.11	62.92	72.65	72.29
	PO > Heur	83.71	70.75	75.40	79.08
	PO Pruning	84.02	70.52	76.01	79.49
	Dual Queue	83.43	65.11	77.00	76.14
	B. Dual Queue	**86.74**	**72.58**	**78.83**	**81.95**
Eager	No POs	75.07	52.42	72.08	68.08
	Heur > PO	75.80	54.28	72.24	69.31
	PO > Heur	81.42	58.20	76.86	74.66
	PO Pruning	84.64	**68.12**	79.33	**79.74**
	Dual Queue	**86.89**	60.91	**83.22**	79.17
	B. Dual Queue	83.65	58.60	79.22	75.96
Cea Heuristic					
Lazy	No POs	74.06	54.97	67.85	66.08
	Heur > PO	75.35	60.63	69.25	68.44
	PO > Heur	83.57	68.39	74.71	76.50
	PO Pruning	83.19	68.04	75.16	76.37
	Dual Queue	80.14	63.68	73.45	72.68
	B. Dual Queue	**86.24**	**70.23**	**77.48**	**78.81**
Eager	No POs	74.02	51.99	70.91	65.03
	Heur > PO	74.00	52.44	70.77	65.26
	PO > Heur	81.90	56.48	76.49	71.75
	PO Pruning	83.48	**66.17**	77.98	**76.44**
	Dual Queue	**85.23**	59.05	**81.40**	74.86
	B. Dual Queue	84.10	56.89	78.39	72.58
CG Heuristic					
Lazy	No POs	71.15	50.01	64.97	64.50
	Heur > PO	72.96	54.29	65.46	66.53
	PO > Heur	73.38	54.69	66.81	66.62
	PO Pruning	72.38	52.36	66.05	65.37
	Dual Queue	75.10	**56.53**	67.94	68.68
	B. Dual Queue	**76.38**	56.47	**69.57**	**69.42**
Eager	No POs	71.82	47.90	68.51	64.20
	Heur > PO	71.97	48.17	68.41	64.46
	PO > Heur	71.22	47.34	67.78	64.30
	PO Pruning	72.39	**50.20**	67.70	64.70
	Dual Queue	**75.74**	49.67	**72.61**	**67.52**
	B. Dual Queue	71.80	47.37	68.34	64.50
Additive Heuristic					
Lazy	No POs	70.84	50.64	63.72	63.93
	Heur > PO	73.80	57.27	66.46	67.09
	PO > Heur	83.56	66.96	75.08	77.58
	PO Pruning	84.07	66.88	75.66	78.17
	Dual Queue	78.18	59.75	70.57	70.75
	B. Dual Queue	**85.55**	**68.95**	**77.32**	**79.93**
Eager	No POs	71.59	49.02	67.83	64.24
	Heur > PO	72.26	49.94	67.94	64.97
	PO > Heur	83.35	55.88	78.01	74.05
	PO Pruning	**84.88**	**65.24**	79.47	**78.60**
	Dual Queue	84.24	57.06	**79.91**	75.47
	B. Dual Queue	84.71	56.33	79.45	75.08

Table 1: Summary of results. Scores are in the interval 0–100; higher scores are better. The best PO use for each search type and heuristic is indicated in bold. Results are reported as normalised averages that weigh all domains equally (e.g., a coverage score of 74.03 means that the configuration solves on average 74.03% of the tasks in a randomly picked domain). Details on the scoring mechanisms for the various criteria can be found in the text.

results are remarkably consistent across the four heuristics. We now discuss the various options in turn.

Using preferredness as a tie-breaker among states of equal h-value ("Heur $>$ PO") is a relatively subtle way of using preferred operators. Compared to not using preferred operators, this option usually improves performance slightly for Eager and notably for Lazy. However, for both search types the results are far from those obtainable via other options. Using preferred operators whenever possible ("PO $>$ Heur") leads with one exception to more improvement than "Heur $>$ PO" for both search types. For Lazy, this option works very well, giving consistently second- or third-best results among the 6 possible PO options. For Eager, this configuration does not work quite as well and usually ranks fourth. PO pruning is similarly good as "PO $>$ Heur" for Lazy, and better than that option for Eager, ranking second-best on average. In particular, when using PO pruning the performance of the two search types is comparable. The simple dual-queue approach is the best option for Eager, but gives mixed results for Lazy, ranging from second-best (CG heuristic) to fourth-best (cea and additive heuristic). The boosted dual-queue approach is the best option for Lazy, but for Eager the performance is notably worse than that of the simple dual queue.

Summarising the results, it seems that for Eager a certain amount of preferred operators is useful (e. g., "Heur $>$ PO" or dual-queue), but strong uses of preferred operators can be counter-productive (e. g., "PO $>$ Heur" and boosted dual-queue are worse than simple dual queue). When using deferred evaluation, however, stronger use of preferred operators always seems to improve results. The use of PO pruning works well for both search types, probably due to the significantly reduced branching factor as only preferred successors are evaluated (unless the restricted search fails, which we found to be rare). Note that for Lazy, PO pruning and "PO $>$ H" give similar results as they lead to identical behaviour in those cases where a goal can be found by only expanding preferred successors. For Eager, however, "PO $>$ H" is worse than PO pruning because with "PO $>$ H" Eager evaluates all successors, whereas with PO pruning it evaluates only preferred successors.

Comparing Eager with Lazy, we note that Eager performs better in most of the configurations; with boosted dual queue, however, deferred evaluation is better. The best performance obtainable from each search variant (using the dual-queue setting for Eager, and boosted dual-queue for Lazy) is very similar. Lazy consistently evaluates fewer states than Eager and is usually faster. Eager, on the other hand, finds plans of better quality. This is not surprising as Eager is better informed with respect to heuristic values than Lazy. For both search variants, plan quality is improved when using preferred operators.

Interactions Between Search Type and PO Use

When using deferred evaluation, all successors of a state are placed in the open list with the same heuristic value (that of the parent). Preferred operators are thus particularly useful for lazy search as they can help recognise the best successor of a state. This is why stronger uses of preferred operators

Perf. Crit.	Lazy Search						Eager Search					
(avg. scores)	No PO	H > PO	PO > H	Prun.	DQ	B. DQ	No PO	H > PO	PO > H	Prun.	DQ	B. DQ
Assembly: heur = FF												
Coverage	63.33	66.67	86.67	86.67	96.67	100.00	56.67	56.67	80.00	86.67	100.00	90.00
Eval.	40.02	42.88	82.00	82.00	81.27	87.75	29.29	29.94	55.31	74.85	70.52	55.80
Quality	61.63	64.97	84.99	84.99	93.55	97.19	55.13	55.44	78.61	85.14	97.25	87.14
Time	59.57	59.33	86.61	86.61	93.34	98.73	51.05	51.27	78.84	85.19	100.00	84.07
Openstacks: heur = Cea												
Coverage	83.33	86.67	90.00	93.33	90.00	86.67	76.67	76.67	90.00	90.00	86.67	90.00
Eval.	43.50	43.63	62.07	62.44	50.22	61.37	35.25	35.23	59.76	61.49	50.35	59.76
Quality	82.79	85.83	89.59	92.93	89.16	86.26	76.12	76.12	89.68	89.68	86.34	89.68
Time	66.72	67.20	79.98	79.83	72.34	78.86	58.70	58.63	79.17	79.79	72.97	78.89
TPP: heur = Additive												
Coverage	63.33	76.67	96.67	96.67	80.00	100.00	63.33	70.00	93.33	93.33	100.00	90.00
Eval.	33.50	40.19	67.17	65.78	44.57	68.44	33.14	35.09	58.33	64.15	62.25	58.12
Quality	58.33	70.36	92.89	92.55	73.34	96.04	56.33	62.51	89.54	87.91	91.51	86.30
Time	50.48	58.98	81.99	80.01	62.08	82.99	50.17	52.57	76.85	80.53	82.04	76.91
Pipesworld-Tankage: heur = Additive												
Coverage	36.00	38.00	76.00	74.00	44.00	72.00	34.00	34.00	80.00	80.00	54.00	78.00
Eval.	20.40	25.46	54.21	55.89	27.48	52.61	19.98	19.46	42.61	58.09	30.05	42.28
Quality	30.06	33.43	57.13	55.35	38.17	56.58	31.62	31.59	64.24	63.93	50.41	62.81
Time	25.26	29.62	57.08	60.66	31.22	54.74	25.45	23.99	51.89	62.92	36.20	50.79
Storage: heur = CG												
Coverage	53.33	60.00	70.00	56.67	63.33	70.00	53.33	60.00	60.00	53.33	60.00	60.00
Eval.	44.28	46.66	58.53	43.97	52.54	58.09	43.26	44.73	47.60	43.11	48.94	47.60
Quality	50.93	53.30	66.61	53.70	61.75	66.28	50.91	56.02	56.31	50.68	56.96	56.31
Time	52.67	54.94	65.84	53.03	58.91	65.40	52.61	54.84	59.03	52.39	59.31	59.05
PSR-Large: heur = Additive												
Coverage	58.00	66.00	46.00	50.00	64.00	64.00	58.00	60.00	46.00	50.00	54.00	48.00
Eval.	36.95	53.03	29.09	31.88	50.67	42.64	37.46	37.89	24.81	31.93	31.45	25.41
Quality	50.88	61.15	39.75	44.61	57.25	56.22	51.41	52.44	41.72	45.51	49.46	42.64
Time	54.18	61.63	42.32	48.72	60.56	58.75	54.69	55.50	41.81	48.77	50.08	43.09
Philosophers: heur = FF												
Coverage	100.00	100.00	43.75	43.75	100.00	81.25	100.00	52.08	35.42	47.92	100.00	37.50
Eval.	37.87	38.31	19.80	19.80	38.37	26.62	37.61	23.73	15.73	19.92	37.67	15.98
Quality	100.00	100.00	29.24	29.24	97.87	66.04	100.00	52.08	23.49	30.91	100.00	25.81
Time	74.46	73.96	36.07	36.05	73.79	58.16	75.58	44.52	31.06	38.56	73.61	32.32
Schedule: heur = FF												
Coverage	18.00	16.00	86.67	86.67	75.33	99.33	22.00	16.67	75.33	88.00	100.00	76.00
Eval.	12.65	15.19	72.85	72.36	46.39	79.18	14.27	12.00	33.42	67.35	46.25	33.42
Quality	17.09	15.53	79.12	79.31	67.12	90.05	21.55	16.32	70.77	81.46	94.65	71.43
Time	17.51	15.79	82.81	82.38	62.19	93.95	21.25	16.26	61.96	82.69	84.50	62.17
Satellite: heur = FF												
Coverage	69.44	100.00	100.00	100.00	88.89	100.00	69.44	72.22	77.78	91.67	77.78	77.78
Eval.	38.59	79.21	81.34	81.34	66.98	81.24	36.74	40.53	43.11	70.78	42.05	43.11
Quality	63.70	94.47	94.33	94.33	83.76	94.33	68.48	71.49	76.18	89.85	76.77	76.18
Time	57.24	81.02	82.29	82.53	74.12	82.99	57.97	60.77	65.59	78.30	64.82	65.63
Logistics-98: heur = CG												
Coverage	100.00	94.29	91.43	94.29	100.00	97.14	97.14	97.14	85.71	94.29	97.14	88.57
Eval.	68.66	66.96	63.12	65.08	71.51	64.37	49.51	49.30	40.02	48.69	47.87	40.23
Quality	98.20	87.53	84.33	92.61	92.97	89.66	96.37	95.54	80.21	93.91	95.38	82.94
Time	89.39	85.14	81.66	84.60	90.17	83.06	75.34	75.53	65.52	74.17	74.03	65.50
Pathways: heur = Cea												
Coverage	26.67	30.00	63.33	63.33	40.00	63.33	33.33	30.00	93.33	90.00	93.33	93.33
Eval.	18.48	21.51	49.68	49.68	26.56	49.31	17.90	19.27	47.43	62.65	45.92	47.43
Quality	25.67	29.28	60.93	60.93	38.26	60.89	33.14	29.49	92.22	88.70	92.24	92.22
Time	25.08	28.02	62.91	62.92	36.42	62.50	27.28	27.16	84.05	87.61	83.04	84.06

Table 2: Detailed results for select domains.

consistently improve results for Lazy. Eager search, on the other hand, evaluates all successors of a state at once. Eager search thus has an estimate (through the h-values of the successors) of which successors are best, and preferred operators may not provide as much additional information to Eager than to Lazy. Preferred operators can be helpful to decide among states of equal h-value, for example, as they act as a second type of information about which states likely lie on a path to the goal. However, preferred operators may even be detrimental to the performance of Eager if they are ill-informed (i. e. if successors are deemed preferred even though they are in fact worse than some of their siblings).

Summary of Findings

We summarise the main findings that hold on average, as shown by Table 1.

- **Preferred operators are very helpful.** For both search types, an average coverage improvement of 10-15% can be obtained with the respective best PO use; this is far more than the difference between the various heuristics.
- **Best PO use depends on the search type.** While both eager and lazy search improve by a similar amount through using POs, Eager performs best with PO pruning and the standard dual-queue approach, while Lazy excels when using POs more strongly via the boosted dual queue.
- **Deferred evaluation trades time for quality.** Both Eager and Lazy obtain similar coverage. But Lazy is faster while Eager finds better plans. Our work thus confirms a claim by Helmert (2006) which had been lacking empirical support to date.

Details and Special Cases

In Table 2, we report individual results for a number of domain-heuristic combinations. The top part of the table contains "typical" cases which reflect the general findings discussed above. Preferred operators increase performance notably, and lazy and eager search perform similarly well. While there is some variation (e. g. in Pipesworld-Tankage), in general it holds that "H $>$ PO" is slightly better than not using POs, while the other PO uses are noticeably better; and PO pruning or dual-queue usage is best for Eager while the boosted dual-queue approach is best for Lazy.

The rest of the table contains exceptions from the general trend, which we discuss in turn. Firstly, we examine cases which are outliers with regard to PO use. In PSR-Large with the additive or FF heuristic, we find that both search types gain some improvement from subtle use of preferred operators in the "H $>$ PO" setting. However, stronger use of POs is detrimental, which becomes particularly evident in the settings "PO $>$ H" and PO pruning. The reason seems to be that few preferred operators are found in this domain; this holds for both the additive and the FF heuristic. The setting "H $>$ PO" uses the information about preferred operators only when it exists and adds to the knowledge about h-values, and thus improves results. By contrast, restricting the search to POs means that we may often be precluded from expanding the current best state because it may not happen to be preferred.

The Philosophers domain with the additive or FF heuristic is interesting because even the subtle use of POs via "H $>$ PO" is detrimental for eager search, and with the exception of the dual-queue approach, all strong uses of POs lead to worse performance for both search types. When examining the domain, we found that it contains large plateaus or near-plateaus, and many ill-informed preferred operators. In large areas of the search space 80–100% of the operators are preferred, of which only 5–10% lead to states with better h-values. Breadth-first search may be the best way to escape from plateaus, and when not using POs, this is what our search does (due to the open list acting FIFO on states of equal h-value). When using the dual queue, the search performs breadth-first search half of the time, which still works very well. By contrast, strong use of POs means that the search spends most of the time following ill-informed POs, and by *not* doing breadth-first search it may not manage to leave the plateaus as quickly.

In Schedule with the FF heuristic, the use of POs in a dual-queue approach is a substantial improvement over not using POs, however, the subtle use of POs as tie-breakers in "H $>$ PO" is notably worse than not using POs. While we are at present not sure what causes this behaviour, one possible explanation is that POs are helpful in some parts of the search space but useless or even detrimental in others.

It is noteworthy that the dual-queue approach may be the most "robust" way of using preferred operators. For all other PO uses, bad cases exist where they perform significantly worse than not using POs. By contrast, the dual-queue approach always performs fairly well and seems to be able to make use of helpful POs while not getting overly distracted by ill-informed POs.

Lastly, the third part of Table 2 contains examples where the two search variants Eager and Lazy exhibit substantially different behaviour. In Satellite and Logistics, a large branching factor in combination with an informative heuristic leads to lazy search being extremely useful, so that it dominates eager search notably with respect to coverage, evaluations and runtime. We will discuss this effect in Satellite in more detail below. On the other hand, Pathways is a domain where deferred evaluation seems to be particularly harmful, so that Eager performs significantly better than Lazy.

Table 3 provides a closer look at the effect of deferred evaluation in the Satellite domain when using the context-enhanced additive heuristic (results for the other heuristics are similar). We show the number of evaluations for the last 12 instances of the domain. With the configurations that are not shown (H $>$ PO, PO $>$ H, and boosted dual queue), Lazy performs exactly the same as with PO pruning, while Eager performs no better than without POs. Eager obtains best performance with PO pruning in this domain as the pruning greatly reduces the large branching factor of the search space. However, even in this setting Lazy outperforms Eager notably with regards to evaluations and, consequently, runtime. Table 3 shows that with exception of the first two instances, Lazy evaluates substantially fewer states than Eager, with differences as high as two orders of magnitude in some cases. On instance 34, eager search with the dual

inst.	Lazy			Eager		
	No PO	Prun.	DQ	No PO	Prun.	DQ
25	150K	1K	2K	71K	2K	71K
26	211K	1K	3K	116K	3K	80K
27	24K	1K	2K	96K	3K	96K
28	46K	2K	3K	195K	6K	195K
29	—	1K	3K	—	7K	285K
30	—	3K	6K	—	10K	420K
31	—	4K	5K	—	14K	—
32	—	10K	—	—	28K	—
33	—	8K	—	—	41K	—
34	—	4K	7K	265K	15K	265K
35	—	10K	—	—	26K	—
36	—	7K	—	—	27K	—

Table 3: Evaluations in Satellite with the cea heuristic.

queue finds a plan of length 287 expanding only 287 states, i. e., it is guided straight to the goal. However, it still needs to evaluate more than 265,000 states, and runs for 29 minutes, while Lazy only evaluates 4,000 states and terminates after 43 seconds.

A Controlled Experiment

To more closely analyse how the benefit from preferred operators varies with different characteristics of the search space, we conducted a set of experiments on a manually-designed search space. We chose the parameters of this artificial search space without much experimentation, as it was straightforward to find settings that show informative behaviour, i. e. where the task is not too easy and not too hard. Other settings are possible and may often lead to similar results. Our goal was to create a scenario where preferred operators can be useful but also harmful, depending on how well the preferredness of an operator predicts that it actually leads to a better state, and how well informed the heuristic is in comparison. The parameters we vary are:

- The quality of the heuristic. We use an admissible heuristic that randomly deviates by up to a factor *dev. fac.* $\leq$ 1 from the approximate goal distance *agd* (see below) of a state. We experimented with values from 0.1 to 0.9. in 0.1-step increases.
- The recognition rate *rec. rate* for preferred operators, i. e., in what percentage of cases a useful operator (one that leads to a better state) is labelled as preferred. For this parameter, we tested values between 0% and 100% using steps of 10%.

We fixed a branching factor of 25 and a typical solution depth of 50. States are characterised by their *approximate goal distance (agd)*. The initial state has an *agd*-value of 50 and states with a value of 0 are labelled as goals. For each state with *agd*-value d, we generate the values of the 25 successors randomly as follows: with probability 0.04 they are $d-1$, with probability 0.16 they are $d+1$, and with probability 0.8 they are d. This means that there exist on average one successor with a lower *agd*-value, 20 successors with equal *agd*-values, and 4 successors with higher *agd*-values than the parent. Likewise, the decision whether or not an operator is labelled as preferred is made randomly: if an operator is useful, i. e. if it leads to a successor state with better *agd*-value, it is preferred in the percentage of cases given by *rec. rate*; else it is preferred in 10% of the cases. If the recognition rate decreases, this means that the relative chance of a preferred operator *not* being useful increases.

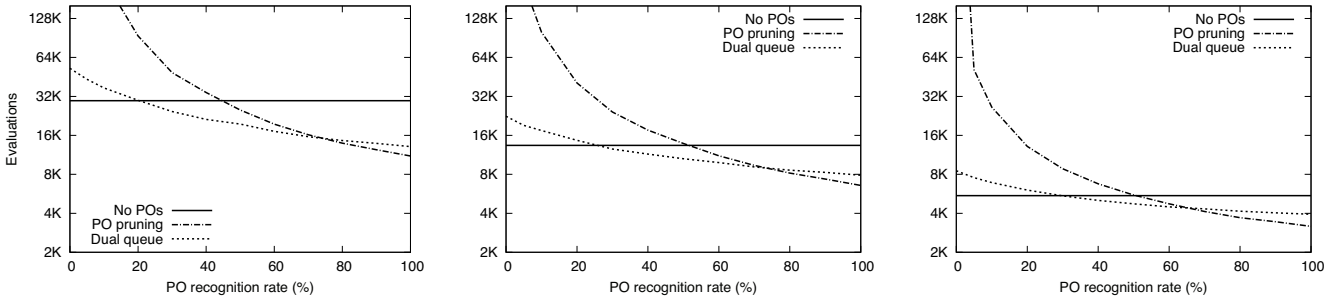
Figures 1 and 2 show the results of our experiment. In Fig. 1, we fixed the heuristic quality at three different values (low in the left panel, medium in the centre and high on the right), and varied the recognition rate of preferred operators. The numbers shown are averaged over 100 runs with different random seeds. As can be seen, a low recognition rate leads to preferred operators being detrimental to the search, while with a high recognition rate, using preferred operators improves performance compared to not using them. It is also evident that PO pruning suffers more strongly from ill-informed preferred operators than the dual-queue approach. The relative performance of the approaches shown in Fig. 1 is notably independent of the quality of the heuristic. The dual-queue approach typically becomes better than not using POs if the recognition rate is more than 20–30%; PO pruning has a higher threshold of 40–50%.

Fig. 2 provides a different look on the same data, where we fix the recognition rate of preferred operators and vary the heuristic quality. As can be seen in the centre and right panels, the advantage of using preferred operators decreases for high values of heuristic quality, i. e., preferred operators are most useful if the heuristic is not well informed. But even given very high heuristic quality, POs can still notably improve performance if the recognition rate is high.

Conclusion

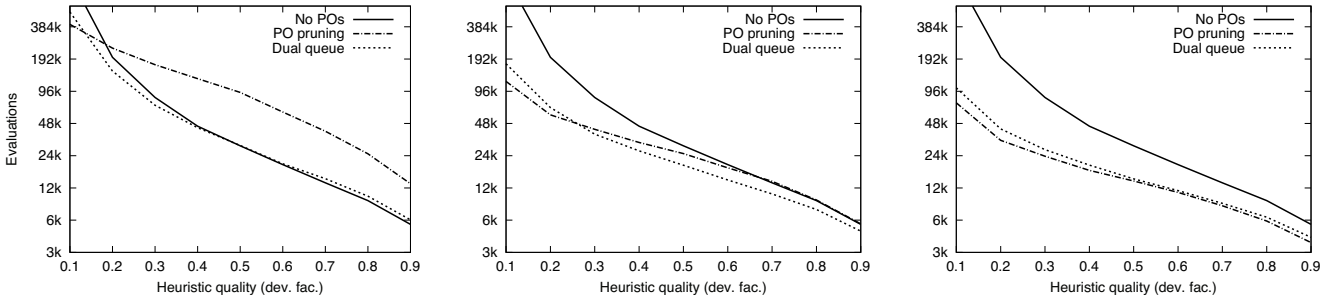
We have examined two search enhancement techniques for planning that are widely used, preferred operators and deferred evaluation. Our findings include that the obvious method of using preferred operators as tie-breakers has little use, compared to other approaches; and that the dual-queue approach, which keeps preferred operators in a second open list, performs best amongst all tested approaches in a standard best-first search. In particular, the dual-queue approach dominates pruning, the method that was proposed when preferred operators were first introduced. With controlled experiments in an artificial search space, we showed that pruning performs badly if the preferred operators are ill-informed. Our results thus suggest that FF might be improved by using the dual-queue approach, though experiments with FF's enforced hill-climbing search algorithm would be needed to confirm this hypothesis.

Our work confirms previous claims that deferred evaluation can be very useful in the presence of large branching factors and that this method trades time for plan quality. Our findings also help to explain the good performance of Fast Downward at IPC 2004 and LAMA at IPC 2008. Both planners used deferred evaluation in combination with the boosted dual-queue option, a combination which led to best performance in our experiments. However, the conceptually simpler approach of using standard best-first search with the unboosted dual queue performs equally well on average, albeit having different strengths and weaknesses. Our results



(a) bad heuristic: dev. fac. 0.5 (b) mediocre heuristic: dev. fac. 0.7 (c) good heuristic: dev. fac. 0.9

Figure 1: Experiments in an artificial search space. On the x-axis we vary the rate at which good operators are recognised (labelled as preferred). Results are shown for three different levels of heuristic quality: rather uninformative, moderately informative and very informative.



(a) bad PO recognition: 20% (b) mediocre PO recognition: 50% (c) good PO recognition: 80%

Figure 2: Experiments in an artificial search space. On the x-axis we vary the quality of the heuristic. Results are shown for three different rates of PO recognition: recognising 20%, 50% and 80% of the useful operators in a state as preferred.

show that boosting the dual queue in Fast Downward and LAMA should always be done for best results, but it would be detrimental in a standard best-first search.

Acknowledgements

We thank Patrik Haslum for helpful discussions.

NICTA is funded by the Australian Government, as represented by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council, through the ICT Centre of Excellence program.

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). For more information, see `http://www.avacs.org/`.

The computing resources for the experiments reported in this paper were provided by the Black Forest Grid Initiative.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1):5–33.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, 140–147.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Keyder, E., and Geffner, H. 2009. Trees of shortest paths vs. Steiner trees: Understanding and improving delete relaxation heuristics. In *Proc. IJCAI 2009*.

Richter, S., and Westphal, M. 2008. The LAMA planner — Using landmark counting in heuristic search. IPC 2008 short papers, `http://ipc.informatik.uni-freiburg.de/Planners`.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI 2008*, 975–982.

Röger, G.; Eyerich, P.; and Mattmüller, R. 2008. TFD: A numeric temporal extension to Fast Downward. IPC 2008 short papers, `http://ipc.informatik.uni-freiburg.de/Planners`.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proc. ICAPS 2004*, 150–159.