Combining Heuristic Estimators for Satisficing Planning

Gabriele Röger and **Malte Helmert**
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
{roeger,helmert}@informatik.uni-freiburg.de

Abstract

The problem of effectively combining multiple heuristic estimators has been studied extensively in the context of optimal planning, but not in the context of satisficing planning. To narrow this gap, we empirically examine several ways of exploiting the information of multiple heuristics in a satisficing best-first search algorithm, comparing their performance in terms of coverage, plan quality and runtime. Our empirical results indicate that using multiple heuristics for satisficing search is indeed useful and that the best results are not obtained by the most obvious combination methods.

Introduction

Heuristic forward search is one of the most popular approaches in classical planning. In the last decade, researchers have put a lot of effort into the development of new heuristics so that a wide range of heuristics are available these days. None of these heuristics consistently outperforms all others across all benchmark domains. Therefore, it appears worthwhile to use the information of several heuristics during search instead of only one.

In the case of optimal planning, which most commonly means using A^* with an admissible heuristic, arbitrary admissible estimates can simply be combined by using their maximum. The resulting heuristic dominates all individual ones, which usually translates into a reduction of the state evaluations required to solve the task. Often, even better combinations are possible: using action-cost partitioning methods (Haslum, Bonet, and Geffner 2005; Katz and Domshlak 2008), we can add heuristic estimates in an admissible way, dominating their maximum. The main drawback of these techniques is that efficiently finding good cost partitionings remains a widely open research problem despite significant recent progress (Katz and Domshlak 2008).

In the case of satisficing planning, where greedy best-first search is the most common approach, the setting for combining heuristic values is quite different: the heuristics do not have to estimate the true distance to the goal in any quantitatively meaningful way, since greedy search only cares about their *relative* values: states further from the goal should receive larger estimates than states close to the goal. Since there is no need to respect a criterion like admissibility, we can combine estimates of several heuristics into a single numeric value in essentially arbitrary ways.

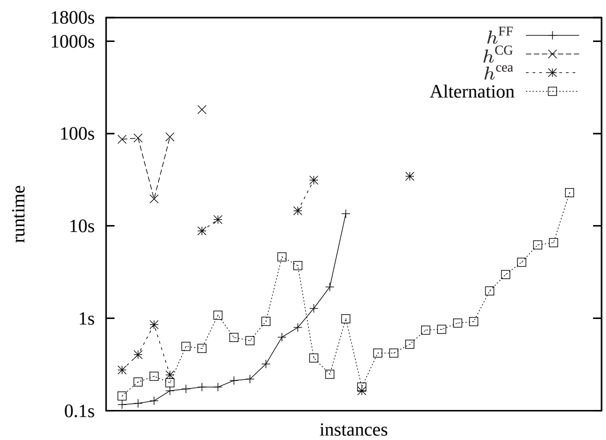

Figure 1: Runtimes in the Assembly domain. (Ordering of tasks does not correspond to the original benchmark suite.)

Combining several heuristic estimates in a satisficing planner can potentially lead to large performance and scalability improvements. Figure 1 shows a striking example of this. The graphs show the runtime, in seconds, for solving instances of the IPC-2000 Assembly domain using the FF heuristic h^{FF} (Hoffmann and Nebel 2001), the causal graph heuristic h^{CG} (Helmert 2004), and the context-enhanced additive heuristic h^{cea} (Helmert and Geffner 2008). None of the individual heuristics solves more than 15 instances. However, their combination (labeled "Alternation" in the figure) solves 29 out of 30 instances, including 13 instances not solved by any of the three heuristics it is based on.

The question, then, is *how* to combine the individual heuristic estimates to achieve the best possible performance. One obvious way to do so, by analogy to optimal planning, is to take their maximum or sum. However, for the Assembly example this does not turn out to be very useful: none of the heuristics that can be obtained by taking two or three of the candidate heuristics and computing their maximum or sum solves more than 13 of the 30 tasks within usual time and memory limits (30 minutes, 2 GB), so they are all outperformed by the FF heuristic used alone.

An alternative idea is to use *weighted* sums, but this immediately raises the question of how to determine suitable

```
open := new open-list
open.insert(s_init)
closed := ∅
while not open.empty():
    s = open.remove-best()
    if s ∉ closed:
        closed := closed ∪ {s}
        if is-goal(s):
            return extract-solution(s)
        for each s' ∈ succ(s):
            if not is-dead-end(s'):
                open.insert(s')
return unsolvable
```

Figure 2: Greedy best-first search (with duplicate detection).

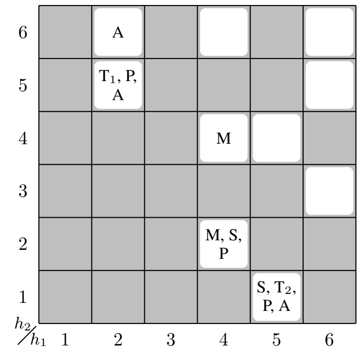

Figure 3: Buckets of an open list with heuristics h_1 and h_2. The symbols within some of the buckets are explained later.

weights. In the given domain, we experimented with all 33 combinations of the form $h(s) = p \cdot h_1(s) + (1-p)h_2(s)$ where $p \in \{0, 0.1, 0.2, \dots, 1.0\}$ and h_1 and h_2 are two heuristics from the given set. None of these combinations improves over the basic FF heuristic. It might be the case that better results could be obtained by using weighted sums of all three heuristics, but then the space of possible weights quickly explodes combinatorially.

So clearly, there are cases where maximization or summation is not the best way of combining heuristic estimates for satisficing planning. Indeed, in Fig. 1, the *Alternation* method, described later in this paper, is vastly superior. In the rest of the paper, we describe several methods for combining heuristic estimates and compare them experimentally.

Greedy Search with Multiple Heuristics

All search methods presented in this paper are variations of greedy-best first search (Pearl 1984), differing only in the choice of which state to expand next. Greedy best-first search is a well-known algorithm, so we only present it briefly to introduce some terminology (Fig. 2).

Starting from the initial state, the algorithm expands states until it has found a path to a goal state or until it has completely explored the state space. *Expanding* a state means generating its successors and adding them to the *open list*. The open list plays a very important role because its remove-best operation determines the order in which states are expanded. In single-heuristic search, it is usually simply a min-heap ordered by $s \mapsto h(s)$, where s is a search state and $h : s \to \mathbb{N}_0 \cup \{\infty\}$ estimates the length of the shortest path from s to any goal state. Hence, states with a low estimate are expanded first. If states share the same estimate, they are usually ordered according to the FIFO principle.

This paper deals with the question of how to use the estimates of multiple heuristics $h_1, \dots, h_n$ within this algorithm. In principle, the methods we present only differ in which states are selected by the *remove-best* operation.

We can see the open list as a collection of *buckets* (Fig. 3), each associated with an estimation vector $(e_1, \dots, e_n)$ and containing all open states s with $(h_1(s), \dots, h_n(s)) = (e_1, \dots, e_n)$. (We assume that is-dead-end(s) evaluates to true iff any of the heuristic estimators regards s as a dead end by mapping it to ∞, so estimates e_i of states in the open list are always finite.) All combination approaches we present can be understood as first selecting a *bucket* to expand a state from, and then picking a state from this bucket according to the FIFO principle. Hence, an approach can be largely characterized in terms of its *candidate buckets*, i. e., the buckets that are possible candidates for expansion at each step.

For example, the candidate buckets for the *maximum* method are exactly those where $\max\{e_1, \dots, e_n\}$ is minimized. In Fig. 3, this means that either the bucket with estimation vector $(4, 2)$ or the bucket with estimation vector $(4, 4)$ is chosen. Which of these buckets is actually selected again depends on FIFO tie-breaking: the bucket with the "oldest" state is given preference. Of course, an actual implementation of the method should not maintain separate buckets for each estimation vector, but rather use a one-dimensional vector of buckets indexed by $\max\{e_1, \dots, e_n\}$.

Maximum and Sum

The first combination methods we discuss are the already mentioned *maximum* and *sum* approaches. The candidate buckets for the maximum approach are those which minimize $\max\{e_1, \dots, e_n\}$, and the candidate buckets for the sum approach are those which minimize $e_1 + \dots + e_n$. In the example of Fig. 3, these buckets are marked with an **M** for the maximum approach and **S** for the sum approach. Among all states in these buckets, the oldest one is expanded first.

The maximum and sum methods are very easy to implement: since they reduce each estimation vector to a single numeric value, a standard single-heuristic open list can be used. However, we will later see that maximum and sum are among the weakest methods for combining heuristic estimates and rarely offer a compelling advantage over using one of the component heuristics individually. One explanation for this is that they are easily misled by bad information. If one of the component heuristic provides very inaccurate values, then these inaccuracies affect every single search decision of the sum method, because each heuristic directly contributes to the final estimation. For the maximum method, *large* inaccurate values from one heuristic can completely cancel the information from all other heuristics.

Of course, one can try to balance a disproportionate influence of a single heuristic by applying weights to the different estimates, but it is not clear how reasonable weight values can be determined automatically, or if weighting can help overcome the fundamental problems of these methods at all. One approach we experimented with is to calculate weighted sums with weights determined from the estimates of the initial state, trying to "balance" the contribution of each heuristic. However, this approach did not show any positive effect on planning benchmarks. One possible explanation for this is that such a normalization not just levels the influence of bad estimates, but also of good estimates.

Because initial experiments were discouraging and it is not clear how to assign reasonable weights, our empirical evaluation does not include the case of weighted sums. However, we do report experiments with the unweighted sum and maximum methods, which serve as baselines for the other approaches, to be introduced next.

Tie-breaking

From our experiments with the addition and sum methods, we got the impression that aggregating the heuristic estimates into one value tends to dilute the quality and characteristics of the individual heuristics. Therefore, in the following we concentrate on methods that preserve the individual estimates. One obvious approach is to rank the heuristics and to use the less important ones only for breaking ties. The idea behind this is that the search is mainly directed by one good heuristic and only if there are several states with the same minimum estimate do we successively consult other heuristics until we have identified a single most promising state. If two states have exactly the same estimation vector, they are again expanded according to the FIFO principle.

Tie-breaking always selects a single candidate bucket. In the example of Fig. 3, this bucket is labeled as $\mathbf{T}_1$ for the case where h_1 is the main heuristic and h_2 is used to break ties, and it is labeled as $\mathbf{T}_2$ for the opposite case.

We considered two implementations of the tie-breaking method. One natural approach is to calculate only the main heuristic and to order the open list according to these estimates. Upon each remove-best operation, we check if several states share the same minimum estimate. Only then do we successively calculate the tie-breaking heuristics, until we have identified a single state to expand. The advantage of this approach is that a heuristic estimate for a tie-breaking heuristic is never computed if it is never needed.

However, in typical planning tasks the range of encountered heuristic values is much smaller than the size of the search frontier, and there are usually many states with the same estimate of the main heuristic. Therefore, the disadvantage of this approach is that we must perform the same tie-breaking calculations again and again, which is costly even if heuristic values are cached after their first computation. While additional data structures may reduce the effort of these recomputations, this causes overhead, and it is not clear if it is worth the additional implementation complexity.

For this reason, we use a different implementation of tie-breaking: for each state inserted into the open list, we calculate the estimates of all heuristics and directly sort it to the appropriate position. With this approach, we can again implement the open list as a min-heap, ordering states lexicographically by their estimate vector. Our experimental data suggests that the cost of always computing all heuristics is not problematic at least in the cases we consider. (One important mitigating factor is that in our case, the main heuristic is more computationally intensive than the tie-breaking heuristics and hence tends to dominate overall runtime.)

Note that both implementations differ only in the time that is needed for inserting and removing states from the open list and in the space requirements for the open list data structure, but behave equivalently in all other aspects. In particular, there is no difference in the number of expanded states.

A major drawback of tie-breaking is that we have to define a ranking of the heuristics. For our experiments, we decided to order the heuristics according to their (empirical) quality in single-heuristic search. It is apparent that combining multiple heuristics via tie-breaking does not fully exploit the available information: we only use the additional estimates if the main heuristic does not distinguish two states. If it does, even if it performs very badly, we ignore the estimates of the additional heuristics. Hence, the approach is clearly not robust against bad estimates of the main heuristic.

Finally, we note that unlike the previous approaches, tie-breaking is unaffected by changing the "scale" of the component heuristics. Increasing estimates by an additive or multiplicative constant or applying any other strictly increasing transformation to a heuristic function does not affect the choices of the tie-breaking method. We see this as a strength rather than a weakness because it offers a certain robustness against systematic errors in heuristic estimates.

Selecting from the Pareto Set

We now present a method that, like tie-breaking, is robust to transformations of heuristic estimates, but does not require us to arbitrarily favour one heuristic over another. Such a method can be derived from the concept of Pareto optimality that is well-known in economics and game theory. Pareto optimality has been successfully applied in multi-objective search (Stewart and White 1991), where the goal is finding a state that is good in terms of multiple objectives whose measures cannot be meaningfully compared.

In order to introduce this method, we need to define the notion of *dominance*. We say that a state s *dominates* a state s' if all heuristics consider s at least as promising as s' and there is at least one heuristic that strictly prefers s over s'.

Definition 1. *A state* s dominates *a state* s'*, written* $s < s'$*, with respect to heuristics* $h_1, \dots, h_n$ *if* $h_i(s) \leq h_i(s')$ *for all* $i \in \{1, \dots, n\}$ *and* $h_i(s) < h_i(s')$ *for at least one heuristic.*

It is reasonable to require that if state s dominates s', then s should be expanded before s'. Hence, we are interested in the Pareto set of nondominated states, defined as

$$\text{nondom} \stackrel{\text{def}}{=} \{s \in \textit{open} \mid \nexists s' \in \textit{open} \text{ with } s' < s\}.$$

In the Pareto approach, the candidate buckets are exactly those buckets whose states belong to nondom. In the example in Fig. 3, these buckets are labeled with **P**. We see that the set includes many of the candidate buckets of the previous approaches, but not all of them. In particular, bucket $(4, 4)$ which is a candidate for the maximization approach is not Pareto-optimal because it is dominated by $(4, 2)$.

We experimented with two variants of the Pareto approach. Both variants first randomly select one of the candidate buckets and then expand the oldest state in that bucket. The two variants differ in how the random choice of buckets is performed: in the *uniform* approach, each candidate bucket is chosen with equal probability, while in the *weighted* approach each candidate bucket is chosen with probability proportional to the number of states it contains.

Note that all previous combination methods define a total preorder on the states. This is somewhat restricting because estimate vectors where neither dominates the other cannot always be reasonably compared. However, algorithmically it is very useful because it allows implementing the open list as a min-heap. This is not possible in the Pareto approach because the preorder is not total. For example, in a given situation the nondominated buckets might have associated estimate vectors of $(2, 4, 4)$ and $(4, 4, 2)$, so that the oldest states with these heuristic profiles, say s_1 and s_2, are candidates for expansion. Now assume that we insert a new state with heuristic profile $(2, 4, 3)$. This new states dominates s_1 but not s_2, so one of the previously "best" states remains a candidate for expansions, while another does not. Such effects complicate the open list implementation for the Pareto approach, and therefore this approach can carry a much larger search overhead than the others. Moreover, this overhead quickly increases with the number of heuristic estimators.

On the positive side, the Pareto method has none of the disadvantages of the previous approaches: we neither have to aggregate estimates in an unrobust way, nor do we have to fix a magic order of the heuristics. Instead, we use all available ordering information, and whenever we prefer a state over another one, we can theoretically justify this decision.

Alternation

Like the Pareto method, the final approach we present avoids aggregating the individual heuristic estimates and makes equal use of all heuristics. It was originally proposed by Helmert (2004; 2006) under the name *multi-heuristic best-first search*. In the context of this paper, this name could equally well be applied to the other combination methods, so we use the term *alternation* to refer to this method here.

The *alternation* method gets its name because it alternates between heuristics across search iterations. The first time a state is expanded, it selects the oldest state minimizing h_1. On the next iteration, it selects the oldest state minimizing h_2, and so on, until all heuristics have been used. At this point, the process repeats from h_1. The candidate buckets for the alternation method are those whose estimate vectors minimize at least one component (labeled with **A** in Fig. 3).

The alternation method is built on the assumption that different heuristics might be useful in different parts of the search space, so each heuristic gets a fair chance to expand the state it considers most promising. One heuristic might provide good guidance in one part of the search space, but be weak in another. A second heuristic might have its strong and weak areas distributed differently in the search space. By alternating between the heuristics, it is always possible to escape a plateau as long as at least one heuristic can give good guidance. There are two important differences between alternation and the Pareto approach:

- Alternation only expands states that are considered *most promising* by some heuristic. The Pareto approach can also expand states which offer a good *trade-off* between the different heuristics, such as bucket $(4, 2)$ in Fig. 3.
- For states that *are* most promising to the currently used heuristic, the alternation method completely ignores all other heuristic estimates. The Pareto approach also attempts to optimize the other heuristics in such situations. For example, it would not consider bucket $(2, 6)$ in Fig. 3 because it is dominated by bucket $(2, 5)$.

Alternation can be efficiently implemented by maintaining a set of min-heaps, one ordered by each heuristic. The approach has been used by several successful planners, including Fast Downward (Helmert 2006), using the causal graph and FF heuristics, and LAMA (Richter, Helmert, and Westphal 2008), using the FF and landmark heuristics.

Experimental Results

We now turn to the central questions of this paper: is the use of multiple heuristics for satisficing best-first search actually useful for typical benchmarks? And if so, which combination method performs best? To answer these questions, we conducted experiments with all planning tasks from the first five international planning competitions, IPC 1–5. We report results on coverage (number of solved instances), solution quality, speed, and heuristic guidance (number of state expansions). We consider three different heuristic estimators:

- h^{FF}: the FF heuristic (Hoffmann and Nebel 2001),
- h^{CG}: the causal graph heuristic (Helmert 2006), and
- h^{cea}: the context-enhanced additive heuristic (Helmert and Geffner 2008).

We evaluate each approach on all two- and three-element subsets of these heuristics. For the tie-breaking approach we fixed the ranking of the heuristics as $h^{\text{cea}} \succ h^{\text{FF}} \succ h^{\text{CG}}$ (so h^{cea} is given the highest priority) based on the coverage these heuristics achieve on the benchmark set in single-heuristic search. For the Pareto method we only report results for the *weighted* approach, because it performs slightly better than the uniform approach and the difference between these variants is low compared to the difference to other methods.

Our implementation is based on the Fast Downward planning system (Helmert 2006), whose greedy-best first search algorithm (with deferred evaluation) we extended with implementations of the different combination approaches. As we are interested in measuring the impact of heuristic combinations, not other search enhancements, we did not use the preferred operator information provided by the heuristics. All experiments were conducted on 2.66 GHz Intel Xeon CPUs with a 30 minute timeout and a 2 GB memory limit.

We first present the overall results, shown in Table 1. The table reports scores according to four metrics: coverage, (solution) quality, speed, and (heuristic) guidance. All scores

	Coverage	Quality	Speed	Guidance
h^{FF}	73.57	70.03	67.14	53.50
h^{CG}	70.92	64.21	63.82	49.46
h^{cea}	74.69	68.45	66.49	54.80
h^{FF}, h^{CG}				
Maximum	*75.11	69.00	66.45	*54.77
Sum	*74.28	65.72	66.29	*54.64
Tie-breaking	73.30	64.43	65.51	*54.35
Pareto	*75.20	66.21	*67.33	*56.35
Alternation	***78.64**	***72.35**	***69.86**	***57.87**
h^{FF}, h^{cea}				
Maximum	74.04	68.03	63.92	53.71
Sum	74.04	66.66	65.62	*55.14
Tie-breaking	73.32	65.87	64.98	54.54
Pareto	*76.01	68.87	*67.96	*58.31
Alternation	***77.72**	***73.18**	***69.67**	***58.95**
h^{CG}, h^{cea}				
Maximum	74.27	67.85	65.14	*54.81
Sum	*74.93	67.52	65.98	*55.12
Tie-breaking	73.82	66.61	65.01	54.63
Pareto	*75.02	67.44	***66.57**	***56.36**
Alternation	***75.43**	***69.26**	66.47	*55.82
h^{FF}, h^{CG}, h^{cea}				
Maximum	73.96	67.74	62.96	53.77
Sum	74.08	65.99	64.82	*55.20
Tie-breaking	73.48	65.46	63.54	54.36
Pareto	*76.32	68.58	*67.41	*58.84
Alternation	***79.13**	***74.37**	***69.74**	***59.99**

Table 1: Overall result summary. The best combination method for a given set of heuristics and metric is highlighted in bold. Entries marked with a star indicate results that are better than all respective single-heuristic approaches.

are in the range 0–100, where larger values indicate better performance. For each metric, the score is computed by assigning a value between 0 and 100 to each task, then averaging the scores for the tasks of each domain to compute a domain score, and finally averaging the domain scores to compute an overall score. Unsolved tasks are always scored as 0, while the score for solved tasks depends on the metric:

- **Coverage:** Solved tasks receive a score of 100. This metric corresponds to the probability (in percent) that the approach solves a "typical" benchmark task.
- **Quality:** Solved tasks receive a score of $100 \cdot l^*/l$, where l is the length of the generated solution and l^* is the length of the best solution generated by any of the approaches.
- **Speed:** Tasks solved within one second receive a score of 100, and tasks that require the full 1800 seconds receive a score of 0. Between these extremes, scores are interpolated logarithmically, so that doubling the runtime decreases the score by about 9.25.
- **Guidance:** Tasks solved within 100 state expansions receive a score of 100, and tasks solved with more than 1,000,000 expansions receive a score of 0. Between these extremes, scores are interpolated logarithmically, so that doubling expansions decreases the score by about 7.53.

We now turn to the interpretation of the results of Table 1.

Comparison between combination approaches. Apart from the comparison between the maximum and sum methods which perform very similarly, the results suggest a clear ranking of the different combination approaches.

Alternation generally performs best: it gives the best results in terms of coverage and quality on all four heuristic sets, and is best in terms of speed and guidance in all cases except for one where the Pareto approach is slightly better.

The Pareto approach is clearly second best, always performing better than the remaining approaches in terms of coverage, speed and guidance. In terms of quality, the maximum and sum approaches sometimes obtain better results.

Maximum and sum rank 3rd and 4th. They perform very similarly to each other in terms of coverage, with no clear winner. In terms of quality, maximum performs better than sum; in terms of speed and guidance, the opposite is true.

Tie-breaking clearly ranks last. It always performs worst according to all metrics except for two cases where it outperforms the maximum method in terms of speed and guidance.

Comparison on commonly solved tasks. It is worth noting that planners which solve many tasks also profit for metrics other than coverage, because unsolved tasks are scored as 0. We believe this to be fair: a planner that *solves* a task within 30 minutes can be rightfully considered faster than one which times out. Hence, a better speed score is justified.

Still, it is interesting to also consider the quality, speed and guidance metrics on the subset of benchmarks solved by *all* configurations. While we cannot provide details for space reasons, we point out that the ranking of approaches remains the same, although the gaps become narrower. Alternation remains best, Pareto second-best and tie-breaking worst for all metrics. This ordering is only broken for the quality metric, where the maximum approach scores higher than Pareto (a trend that already exists in Table 1).

Comparison to single-heuristic methods. Another clear outcome of the experiment is that using multiple heuristics can give significant benefits, especially with the alternation method. For any set of heuristics and any of the four metrics, the alternation method improves the performance over the best single heuristic from the set, with only one small exception (speed for the combination of h^{CG} and h^{cea}).

Indeed, adding more heuristics is almost universally a good idea for the alternation method in our experiment. There are nine ways to choose a single heuristic or two-heuristic set and a new heuristic to add, and there are four metrics to measure. In 34 of these 36 cases, the *marginal contribution* of adding the new heuristic is positive.

For the Pareto method, the advantage over single-heuristic search is less pronounced, but we believe that the method still offers improvements. While it consistently leads to better results in terms of coverage, speed and guidance, its results in terms of quality are worse than those of the best individual heuristics. The best configuration, as for the alternation approach, is again the one that uses all three heuristics.

For the maximum and sum methods, it is hard to argue that they offer any compelling advantage over single-heuristic search, and the tie-breaking method is clearly not worth using according to our data. It consistently performs

Domain	h^{FF}	h^{CG}	h^{cea}	**Max.**	**Sum**	**Tie-br.**	**Pareto**
Airport	+6/–2	+15	+3/–7	+3/–4	+4/–6	+7/–2	+4/–3
Assembly	+14	+24	+19	+19	+18	+19	+15
Depot	–1	+2	+1	+1	+1/–1	+1/–1	–1
Driverlog	+1	+1	+1	+1	+2	+1	+2
FreeCell	+3/–2	+10/–1	+2/–1	+2/–1	+2/–1	+3/–1	+2/–1
Grid	+1	+1					
Logistics-1998	+9	–1	–1	–1	–1	–1	
Miconic-FullADL	+4/–1	+2	–2	–2	–1	–2	+1
MPrime	+9		+1	+7	+1	+1	+1
Mystery	+3	+1	+2	+2	+2	+2	+1
Openstacks		+6	+4	+4	+4	+4	
OpticalTelegraphs		+3					
Pathways	+4	+4	+3/–1	+3/–1	+3/–1	+4/–1	+3/–1
Pipesw.-NoTankage	+5	+12/–3	+8/–2	+9	+10/–1	+10	+7/–1
Pipesw.-Tankage	+2/–6	+5/–3	+3/–3	+4/–3	+2/–2	+4/–3	+2/–3
PSR-Large	–2	–1	–1	–2	+1/–1	+1/–1	+1/–1
PSR-Middle					+1	+1	+1
Rovers	+5	+5	+8	+7	+7	+8	+5
Satellite	+4	+2	+1	+1	+1	+1	
Schedule		+6/–4	+6/–4	+6/–4	+6/–4	+6/–4	+4/–4
Storage	–1	+3	+4	+4	+3	+4	+3
TPP	+2	+3	+3/–1	+3/–2	+4	+4/–1	–6
Trucks		+2	–1	–1	+1		+1
Total	+72/–15	+107/–13	+69/–24	+76/–21	+73/–19	+81/–17	+53/–21

Table 2: Tasks solved by Alternation compared to single heuristics and other combination approaches. Entry $+x/-y$ means that Alternation solves x tasks not solved by the other approach and fails to solve y tasks solved by the other approach. Domains where all methods solve the same set are omitted. All combination methods use all three heuristics.

worse on all metrics than just using the main heuristic on its own, with only one exception.

Coverage details. We have established that we obtain the best results when using the alternation method applied to all three heuristics. Hence, we conclude our discussion of experimental results with some detailed data for this particular approach, in order to see whether its benefits are limited to a few planning domains or distributed more evenly.

Table 2 reports, for all IPC 1–5 benchmark domains, in what ways the set of tasks solved by the alternation method differs from other approaches. We compare to all single heuristics and to all combination methods that use the same (full) set of heuristics. The table shows improvements in many domains. Moreover, there are very few cases where the alternation method fails to solve a substantial number of tasks solved by one of the single heuristics, indicating that it is indeed very robust.

Conclusion

We have argued that the problem of combining heuristic estimates for satisficing planning calls for different approaches than the problem of combining heuristic estimates for optimal planning. We have presented five different combination methods and compared them experimentally. The *alternation* method, which performs best in our experiments, is not new: under the name *multi-heuristic best-first search*, it has been used in the Fast Downward and LAMA planners. However, prior to our experiments, the alternation method has never been systematically evaluated, and it was not clear to what extent it contributes to the performance of these planners. Moreover, it has never been compared to other approaches for combining heuristic estimates.

Our results show that aggregating different heuristic estimates into a single numeric value through arithmetic operations like taking the maximum or sum is not a good idea, even though it is the common approach for optimal planning. Our explanation for this is that such aggregation methods are easily led astray even if only one heuristic generates bad distance estimates. The Pareto and alternation approaches are much more robust to such misleading estimates.

In future work, it would be interesting to see if the benefits of using multiple heuristics can be combined with the benefits of using preferred operator information. Again, this is an approach that is commonly used, but has never been evaluated against baseline approaches or alternative methods. Finally, it would be interesting to see if even better results can be obtained by including yet more estimators such as the additive (Bonet and Geffner 2001) or landmark heuristic (Richter, Helmert, and Westphal 2008), or if performance begins to degrade when four or more estimators are used.

Acknowledgments

The computing resources for the experiments reported in this paper were graciously provided by Universitat Pompeu Fabra. We thank Héctor Palacios for his support in conducting the experiments.

This work was supported by the German Research Council (DFG) by DFG grant NE 623/10-2 and as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See http://www.avacs.org/ for more information.

References

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

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. AAAI 2005*, 1163–1168.

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

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004*, 161–170.

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.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In *Proc. ICAPS 2008*, 174–181.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

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

Stewart, B. S., and White, III, C. C. 1991. Multiobjective A*. *JACM* 38(4):775–814.