The More, the Merrier: 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

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, speed, and search guidance. Our results indicate that using multiple heuristics for satisficing search is indeed useful. Among the combination methods we consider, the best results are obtained by the *alternation* method of the "Fast Diagonally Downward" planner.

Introduction

Heuristic forward search is a very popular approach in classical planning, and a wide range of heuristics is available today. None of these heuristics consistently outperforms all others, and hence it appears worthwhile to use the information of several heuristics during search instead of only one.

For optimal planning with A*-style algorithms, arbitrary admissible heuristics can be combined by using their maximum. The resulting heuristic dominates all individual ones and usually requires fewer state evaluations to solve a task. Often, even better combinations are possible, for example by using action cost partitioning methods that allow *adding* heuristic estimates admissibly (Haslum, Bonet, and Geffner 2005; Katz and Domshlak 2008).

For satisficing planning, where greedy best-first search is a common approach, the setting for combining heuristic values is quite different. Heuristics do not have to estimate the true goal distance in any quantitatively meaningful way, since greedy search only cares about *relative* values: states further from the goal should receive larger estimates than states closer to the goal. There is no need to respect a criterion like admissibility, and we can combine multiple estimates in essentially arbitrary ways.

Combining several heuristics in a satisficing planner can improve performance and scalability dramatically. 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 tasks within usual resource limits (30 minutes, 2 GB). However, their combination (labeled "Alternation" in the figure) solves 29 out of 30 tasks, including 13 tasks not solved by any of the three heuristics it is based on.

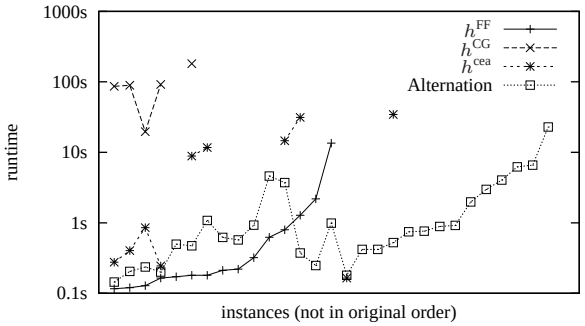

Figure 1: Runtimes in the Assembly domain.

The question, then, is *how* to combine heuristic estimates to achieve the best possible performance. One obvious way, by analogy to optimal planning, is to take their maximum or sum. However, for the Assembly example this does not turn out to be useful: none of the heuristics obtained by taking two or three of the candidate heuristics and computing their maximum or sum solves more than 13 of the 30 tasks, so they are all outperformed by the FF heuristic used alone. An alternative is to use *weighted* sums, but this immediately raises the question of how to determine suitable weights. In the given domain, we tested all 33 combinations of the form $h(s) = p \cdot h_1(s) + (1-p)h_2(s)$ where $p \in \{0, 0.1, 0.2, \ldots, 1.0\}$ and h_1 and h_2 are two heuristics from the given set. None of these combinations improves over the FF heuristic.

So clearly, there are cases where maximization or summation is not the best way to combine heuristics for satisficing planning. Indeed, in Fig. 1, the *alternation* method is vastly superior. This method is not new: it was introduced by Helmert (2006) under the name "multi-heuristic best-first search" (a term we avoid in this paper because it applies to all methods we discuss), and it is one of the ingredients underlying the Fast Downward (Helmert 2006) and LAMA (Richter, Helmert, and Westphal 2008) planners. However, neither alternation nor any other method for com-

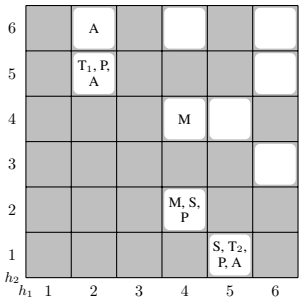

Figure 2: Buckets of an open list with heuristics h_1 and h_2 (shown as white rounded rectangles). The symbols within some of the buckets are explained later.

bining heuristic estimates in satisficing planners has ever been evaluated in a principled way, and from the literature it is completely unclear *if*, *to what extent*, and *why* alternation or any other method for combining heuristic values leads to better planner performance than just using a single heuristic.

In this paper, we attempt to rectify this situation by describing several methods for combining heuristic estimates and providing a thorough experimental study to illustrate the benefits of using multiple heuristics for satisficing planning.

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. Starting from the initial state, the algorithm expands states until it has found a goal path or explored all reachable states. *Expanding* a state means generating its successors and adding them to the *open list*. The open list plays a very important role because it determines the order in which states are expanded. In single-heuristic search, it is often 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 the goal. Hence, states with a low estimate are expanded first. States with the same estimate are usually expanded in FIFO order.

This paper deals with the question of how to use estimates of multiple heuristics $h_1, \ldots, h_n$ within this algorithm. In principle, the methods we present only differ in the expansion priorities imposed by the open list. We can see the open list as a collection of *buckets* (Fig. 2), each associated with an estimation vector $(e_1, \ldots, e_n)$ and containing all open states s with $(h_1(s), \ldots, h_n(s)) = (e_1, \ldots, e_n)$. All 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 is largely characterized by its *candidate buckets*, i. e., the buckets that are possible candidates for expansion at each step. For example, the candidate buckets for the *sum* method are those where $e_1 + \cdots + e_n$ is minimal. In Fig. 2, this means that either the bucket with estimation vector $(4, 2)$ or the bucket with estimation vector $(5, 1)$ is chosen.

Maximum and Sum

We first discuss the already mentioned *maximum* and *sum* approaches. The candidate buckets for the maximum approach are those minimizing $\max\{e_1, \ldots, e_n\}$, and the candidate buckets for the sum approach are those minimizing $e_1 + \cdots + e_n$. In the example of Fig. 2, these buckets are marked with an **M** for maximum and **S** for sum. 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 number, a standard single-heuristic open list can be used. However, we will see that maximum and sum are among the weakest methods for combining heuristic estimates. One explanation for this is that they are easily misled by bad information. If one of the component heuristic provides very inaccurate values, 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 estimates of one heuristic can completely cancel the information of all other heuristics.

Tie-breaking

Our experience with the addition and sum methods suggests that aggregating 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 idea is to rank the heuristics and use the less important ones only for breaking ties. With this approach, search is mainly directed by one good heuristic and only if there are several states with the same minimum estimate, the other heuristics are successively consulted to identify the 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. 2, this bucket is labeled with $\mathbf{T}_1$ for the case where h_1 is the main heuristic and h_2 is used to break ties and with $\mathbf{T}_2$ for the opposite case.

Unlike the maximum and sum approaches, tie-breaking is not affected by the "scale" of the component heuristics. Increasing estimates by an additive or multiplicative constant or applying any other strictly increasing transformation does not affect the choices of the tie-breaking method. We see this as a strength of the approach because it offers some resilience against systematic errors in heuristic estimates.

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 and that the approach is clearly not robust against bad estimates of the main heuristic.

Selecting from the Pareto Set

We now present a method that, like tie-breaking, is robust to transformations of heuristic estimates, but does not favour one heuristic over another. The method uses the concept of

Pareto-optimality, well-known in economics and game theory, which is based on the notion of *dominance*. We say that a state s *dominates* a state s' ($s < s'$) if all heuristics consider s at least as promising as s' (i. e., $\forall i\, h_i(s) \leq h_i(s')$) and at least one heuristic strictly prefers s over s' (i. e., $\exists j\, h_j(s) < h_j(s')$). It appears reasonable to require that if state s dominates s', then s should be expanded before s'. Hence, we consider the Pareto set of nondominated states:

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

In the Pareto approach, the candidate buckets are exactly those buckets whose states belong to *nondom*. In Fig. 2, these buckets are labeled with **P**. We see that the set includes many candidate buckets of the previous approaches, but not all of them. In particular, bucket $(4, 4)$ which is a candidate for the maximum approach is not Pareto-optimal because it is dominated by $(4, 2)$. From all Pareto-optimal candidate buckets, the one used for expansion is chosen randomly with probability proportional to the number of states it contains.

Unlike the previous methods, the Pareto criterion does not impose a total preorder on states, which makes maintaining the open list for this approach much more expensive. We discuss this issue in more detail in a technical report (Röger and Helmert 2010). On the positive side, the Pareto method has none of the disadvantages mentioned for the previous approaches: no single heuristic has an overly large influence on the overall state ranking, and we use all available ordering information. Moreover, whenever we prefer a state over another, we can theoretically justify this decision.

Alternation

The last approach we consider is the *alternation* method, originally proposed by Helmert (2004; 2006). Like the Pareto method, it avoids aggregating the individual heuristic estimates and makes equal use of all heuristics. The method gets its name because it alternates between heuristics across search iterations. The first time a state is expanded, the alternation method 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. 2).

Alternation 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. There are two important differences to the Pareto approach. Firstly, alternation only expands states that are considered *most promising* by some heuristic. The Pareto approach also expands states which offer a good *trade-off* between the different heuristics, such as bucket $(4, 2)$ in Fig. 2. Secondly, for states that *are* most promising to the currently used heuristic, alternation 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. 2 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.

Experiments

We now turn to the central questions of this paper: is the use of multiple heuristics for satisficing best-first search actually beneficial? And if so, which combination method performs best? To answer these questions, we integrated the different combination methods into a state-of-the-art planner and evaluated them on all planning tasks from the first five international planning competitions (IPC 1–5). All experiments were conducted on computers with 2.3 GHz AMD Opteron CPUs under a 30 minute timeout and 2 GB memory limit.

Our implementation is based on the Fast Downward planner (Helmert 2006), which we extended with implementations of the different combination approaches. To focus on the impact of heuristic combination methods, not other search enhancements, we did not use the preferred operator information provided by the heuristics.

We conducted experiments both with Fast Downward's lazy variant of greedy best-first search and with the textbook ("eager") algorithm (Richter and Helmert 2009), with virtually identical results. Here, we report on the more standard eager algorithm. Results for lazy search are reported in an earlier workshop paper (Röger and Helmert 2009).

We consider three heuristic estimators: h^{FF}, h^{CG}, and h^{cea}. Each approach is evaluated on all two- and three-element subsets of these heuristics. For the tie-breaking method we fixed the ranking of the heuristics as $h^{\text{cea}} \succ h^{\text{FF}} \succ h^{\text{CG}}$ based on the coverage these heuristics achieve on the benchmarks in single-heuristic search.

All planners thus obtained are scored according to four metrics: *coverage* (solved tasks), *quality* (solution length compared to best solution found by any approach; essentially the IPC-2008 scoring method), *speed* (CPU time to solve a task on a logarithmic scale), and *guidance* (state expansions to solve a task on a logarithmic scale). All scores are in the range 0–100, where larger values indicate better performance. See the paper by Richter and Helmert (2009), which uses the same scoring methods, for details.

The results of the experiment are summarized in Table 1.

Comparison between combination approaches. Comparing the five combination methods to each other, we see that 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 next best method is the Pareto approach, which always outperforms the remaining three methods on speed and guidance. In terms of coverage and quality, the maximum and sum approaches sometimes obtain comparable results.

The remaining three techniques, maximum, sum and tie-breaking, perform quite similarly to each other and are clearly worst overall, with tie-breaking slightly weaker than the others. In particular, tie-breaking preforms worse than the sum method in all cases.

Comparison to single-heuristic methods. Another clear outcome of the experiment is that using multiple heuristics can give considerable benefits, especially with the alternation method. For any set of heuristics and any of the metrics, alternation outperforms the best single heuristic from

Heuristics	Combination	Cover.	Quality	Speed	Guid.
h^{cea}		74.62	68.67	65.27	65.65
h^{FF}		73.85	70.55	66.81	64.07
h^{CG}		72.66	65.36	64.16	60.43
h^{cea}, h^{FF}	Maximum	72.69	67.26	62.15	64.02
	Sum	73.75	68.42	63.75	*65.67
	Tie-breaking	72.44	67.14	62.90	64.67
	Pareto	*76.20	*70.71	66.32	*68.90
	Alternation	***77.95**	***73.70**	***67.84**	***70.14**
h^{FF}, h^{CG}	Maximum	*74.76	68.76	65.29	*65.08
	Sum	*75.01	67.99	65.41	*65.35
	Tie-breaking	72.59	66.13	64.66	*64.41
	Pareto	*74.93	67.84	65.87	*66.19
	Alternation	***78.73**	***73.28**	***69.22**	***69.28**
h^{cea}, h^{CG}	Maximum	74.06	67.95	63.63	65.51
	Sum	*74.76	67.70	64.12	*65.67
	Tie-breaking	73.78	67.41	63.36	64.99
	Pareto	74.52	67.70	**64.48**	***66.52**
	Alternation	***75.20**	***69.18**	64.42	*66.39
h^{cea}, h^{FF}, h^{CG}	Maximum	72.21	66.54	61.13	63.71
	Sum	73.47	67.52	62.98	65.24
	Tie-breaking	72.49	66.95	61.90	64.34
	Pareto	*76.29	70.16	66.01	*69.18
	Alternation	***79.80**	***74.62**	***68.56**	***71.91**

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

the set, with only one exception (speed for the combination of h^{cea} and h^{CG}). 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 scoring metrics. In 34 of these 36 cases, the marginal contribution of adding the new heuristic is positive.

For the Pareto method, the comparison to single-heuristic search gives somewhat mixed results. While it improves coverage (except for the combination of h^{cea} and h^{CG}) and guidance, the quality and speed results are mostly worse than those of the best individual 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 consistently performs worse on all metrics than just using the main heuristic on its own, with only one exception.

Details for alternation. We have observed that the best results are obtained by the alternation method using all three heuristics. A detailed look at the experimental data, reported in the technical report accompanying this paper (Röger and Helmert 2010), shows that this improvement is not limited to a few benchmark domains but distributed quite evenly across domains. Moreover, the improvement of coverage over the other combination methods and individual heuristics is statistically significant at a level of $p \leq 0.001$, using the same nonparametric test that Hoffmann and Nebel (2001) employ in their comparison of FF and HSP.

Conclusion

Combining heuristic estimates for satisficing planning calls for different approaches than combining heuristic estimates for optimal planning. In our experiments, aggregating different heuristic estimates into a single numeric value through arithmetic operations like taking the maximum or sum turned out not to be 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 estimates. The Pareto approach, and especially the alternation approach which clearly performed best in our experiments, are much more robust to such misleading estimates.

In future work, it would be interesting to see if results can be improved further 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. Another interesting question is whether *adaptive* techniques that acquire information about the heuristics during search can lead to further performance improvements.

Acknowledgments

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" (AVACS).

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., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, 273–280.

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

Röger, G., and Helmert, M. 2009. Combining heuristic estimators for satisficing planning. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*, 43–48.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning (extended version). Technical Report 258, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.