To Max or Not to Max: Online Learning for Speeding Up Optimal Planning *

Carmel Domshlak and **Erez Karpas**
Faculty of Industrial Engineering & Management
Technion, Israel

Shaul Markovitch
Faculty of Computer Science
Technion, Israel

Abstract

It is well known that there cannot be a single "best" heuristic for optimal planning in general. One way of overcoming this is by combining admissible heuristics (e.g. by using their maximum), which requires computing numerous heuristic estimates at each state. However, there is a tradeoff between the time spent on computing these heuristic estimates for each state, and the time saved by reducing the number of expanded states. We present a novel method that reduces the cost of combining admissible heuristics for optimal search, while maintaining its benefits. Based on an idealized search space model, we formulate a decision rule for choosing the best heuristic to compute at each state. We then present an active online learning approach for that decision rule, and employ the learned model to decide which heuristic to compute at each state. We evaluate this technique empirically, and show that it substantially outperforms each of the individual heuristics that were used, as well as their regular maximum.

Introduction

One of the most prominent approaches to cost-optimal planning is using the A^* search algorithm with an admissible heuristic. Many admissible heuristics have been proposed, varying from cheap to compute yet typically not very informative to expensive to compute but often very informative. Since the accuracy of heuristic functions varies for different problems, and even for different states of the same problem, we can produce a more robust optimal planner by combining several admissible heuristics. The simplest way of doing this is by using their point-wise maximum at each state. Presumably, each heuristic is more accurate, that is, provides a higher estimate, in different regions of the search space, and thus their maximum is at least as accurate as each of the individual heuristics. In some cases it is also possible to use additive (Felner, Korf, and Hanan 2004; Haslum, Bonet, and Geffner 2005; Katz and Domshlak 2008) or mixed additive/maximizing (Coles et al. 2008; Haslum et al. 2007) combinations of admissible heuristics.

An important issue with both max-based and sum-based approaches is that the benefit of adopting them over sticking to just a single heuristic is assured only if the planner is not constrained by time. Otherwise, the time spent on computing numerous heuristic estimates at each state may outweigh the time saved by reducing the number of expanded states. This is precisely the contribution of this paper: We propose a novel method for combining admissible heuristics that aims at providing the accuracy of their max-based combination while still computing just a single heuristic for each search state. This method, called *selective max*, is presented and evaluated in what follows in the context of heuristic-search planning, yet is applicable to any search problem.

At a high level, selective max can be seen as a hyper-heuristic (Burke et al. 2003) — a heuristic for choosing between other heuristics. Specifically, selective max is based on a seemingly useless observation that, if we had an oracle indicating the most accurate heuristic for each state, then computing only the indicated heuristic would provide us with the heuristic estimate of the max-based combination. In practice, of course, such an oracle is not available. However, in the time-limited settings of our interest, this is not our only concern: It is possible that the extra time spent on computing the more accurate heuristic (indicated by the oracle) may not be worth the time saved by the reduction in expanded states.

Addressing the latter concern, we first analyze an idealized model of a search space and deduce a decision rule for choosing a heuristic to compute at each state when the objective is to minimize the overall search time. Taking that decision rule as our target concept, we then describe an online active learning procedure for that concept that constitutes the essence of selective max. Our experimental evaluation with two state-of-the-art admissible heuristics for domain-independent planning, h_{LA} (Karpas and Domshlak 2009) and $h_{\text{LM-CUT}}$ (Helmert and Domshlak 2009), shows that, under various time limits, using selective max consistently results in solving more problems than using each of these two heuristics individually, as well as using their max-based combination. Furthermore, the results show that using selective max results in solving problems faster on average.

Notation

We consider planning in the SAS$^+$ formalism (Bäckström and Nebel 1995); a SAS$^+$ description of a planning task can be automatically generated from its PDDL description (Helmert 2009). A SAS$^+$ task is given by a 4-tuple

*Partly supported by ISF grant 670/07

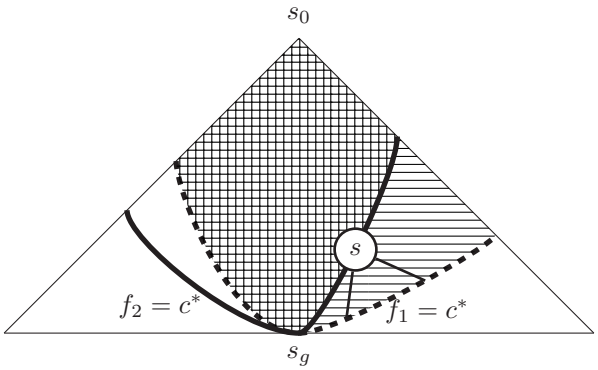


Figure 1: An illustration of the idealized search space model and the f-contours of two admissible heuristics.

$\Pi = \langle V, A, s_0, G \rangle$. $V = \{v_1, \ldots, v_n\}$ is a set of *state variables*, each associated with a finite domain $dom(v_i)$. Each complete assignment s to V is called a *state*; s_0 is an *initial state*, and the *goal* G is a partial assignment to V. A is a finite set of *actions*, where each action a is a pair $\langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$ of partial assignments to V called *preconditions* and *effects*, respectively.

An action a is applicable in a state s iff $\mathsf{pre}(a) \subseteq s$. Applying a changes the value of each state variable v to $\mathsf{eff}(a)[v]$ if $\mathsf{eff}(a)[v]$ is specified. The resulting state is denoted by $s[\![a]\!]$; by $s[\![\langle a_1, \ldots, a_k \rangle]\!]$ we denote the state obtained from sequential application of the (respectively applicable) actions $a_1, \ldots, a_k$ starting at state s. Such an action sequence is a plan if $G \subseteq s_0[\![\langle a_1, \ldots, a_k \rangle]\!]$.

A Model for Heuristic Selection

Given a set of admissible heuristics and the objective of minimizing the overall search time, we are interested in a decision rule for choosing the right heuristic to compute at each search state. In what follows, we derive such a decision rule for a pair of admissible heuristics with respect to an idealized search space model corresponding to a tree-structured search space with a single goal state, constant branching factor b, and uniform cost actions (Pearl 1984). Two additional assumptions we make are that the heuristics are consistent, and that the time t_i required for computing heuristic h_i is independent of the state being evaluated; w.l.o.g. we assume $t_2 \geq t_1$. Obviously, most of the above assumptions do not hold in typical search problems, and later we carefully examine their individual influences on our framework.

Adopting the standard notation, let $g(s)$ be the cost of the cheapest path from s_0 to s. Defining $\max_h(s) = \max(h_1(s), h_2(s))$, we then use the notation $f_1(s) = g(s) + h_1(s)$, $f_2(s) = g(s) + h_2(s)$, and $\max_f(s) = g(s) + \max_h(s)$. The A^* algorithm with a heuristic h expands states in increasing order of $f = g + h$. Assuming the goal state is at depth c^*, let us consider the states satisfying $f_1(s) = c^*$ (the dotted line in Fig. 1) and those satisfying $f_2(s) = c^*$ (the solid line in Fig. 1). The states above the $f_1 = c^*$ and $f_2 = c^*$ contours are those that are surely expanded by A^* with h_1 and h_2, respectively. The states above both these contours (the grid-marked region in Fig. 1), that is, the states $SE = \{s \mid \max_f(s) < c^*\}$, are those that are surely expanded by A^* using $\max_h$ (see Theorem 4, p. 79, Pearl 1984).

Under the objective of minimizing the search time, observe that the optimal decision for any state $s \in SE$ is not to compute any heuristic at all, since all these states are surely expanded anyway. The optimal decision for all other states is a bit more complicated. $f_2 = c^*$ contour that separates between the grid-marked and lines-marked areas. Since $f_1(s)$ and $f_2(s)$ account for the same $g(s)$, we have $h_2(s) > h_1(s)$, that is, h_2 is more accurate in state s than h_1. If we were interested solely in reducing state expansions, then h_2 would obviously be the right heuristic to compute at s. However, for our objective of reducing the actual search time, h_2 may actually be the wrong choice because it might be much more expensive to compute than h_1.

Let us consider the effects of each of our two alternatives. If we compute $h_2(s)$, then s is no longer surely expanded since $f_2(s) = c^*$, and thus whether A^* expands s or not depends on tie-breaking. In contrast, if we compute $h_1(s)$, then s is surely expanded because $f_1(s) < c^*$. Note that not computing h_2 for s and then computing h_2 for one of the descendants s' of s is surely a sub-optimal strategy as we do pay the cost of computing h_2, yet the pruning of A^* is limited only to the search sub-tree rooted in s'. Therefore, our choices are really either computing h_2 for s, or computing h_1 for all the states in the sub-tree rooted in s that lie on the $f_1 = c^*$ contour. Suppose we need to expand l complete levels of the state space from s to reach the $f_1 = c^*$ contour. This means we need to generate order of b^l states, and then invest $b^l t_1$ time in calculating h_1 for all these states that lie on the $f_1 = c^*$ contour. In contrast, suppose we choose to compute $h_2(s)$. Assuming favorable tie-breaking, the time required to "explore" the sub-tree rooted in s will be t_2.

Putting things together, the optimal decision in state s is thus to compute h_2 iff $t_2 < b^l t_1$, or if we rewrite this, if

$$l > \log_b(t_2/t_1).$$

As a special case, if both heuristics take the same time to compute, this decision rule boils down to $l > 0$, that is, the optimal choice is simply the more accurate (for state s) heuristic.

The next step is to somehow estimate the "depth to go" l. For that, we make another assumption about the rate at which f_1 grows in the sub-tree rooted at s. Although there are many possibilities here, we will look at two estimates that appear to be quite reasonable. The first estimate assumes that the h_1 value remains constant in the subtree rooted at s, that is, the additive error of h_1 increases by 1 for each level below s. In this case, f_1 increases by 1 for each expanded level of the sub-tree (because h_1 remains the same, and g increases by 1), and it will take expanding $\Delta_h(s) = h_2(s) - h_1(s)$ levels to reach the $f_1 = c^*$ contour. The second estimate we examine assumes that the absolute error of h_1 remains constant, that is, h_1 increases by 1 for each level expanded, and so f_1 increases by 2. In this case, we will need to expand $\Delta_h(s)/2$ levels. This can be generalized to the case where the estimate h_1 increases by any constant additive factor c, which results in $\Delta_h(s)/(c+1)$

levels being expanded. In either case, the dependence of l on $\Delta_h(s)$ is linear, and thus our decision rule can be reformulated to compute h_2 if

$$\Delta_h(s) > \alpha \log_b(t_2/t_1),$$

where α is a hyper-parameter for our algorithm. Note that, given b, t_1, and t_2, the quantity $\alpha \log_b(t_2/t_1)$ becomes fixed and in what follows we denote simply by *threshold* τ.

Dealing with Model Assumptions

The idealized model above makes several assumptions, some of which appear to be very problematic to meet in practice. Here we examine these assumptions more closely, and when needed, suggest pragmatic compromises.

First, the model assumes that the search space forms a tree with a single goal state and uniform cost actions, and that the heuristics in question are consistent. Although the first assumption does not hold in most planning problems, and the second assumption is not satisfied by some state-of-the-art heuristics, they do not prevent us from using the decision rule suggested by the model. Furthermore, there is some empirical evidence to support our conclusion about exponential growth of the search effort as a function of heuristic error, even when the assumptions made by the model do not hold. In particular, the experiments of Helmert and Röger (2008) with heuristics with small constant additive errors clearly show that the number of expanded nodes typically grows exponentially as the (still very small and additive) error increases.

The model also assumes that both the branching factor and the heuristic computation times are constant across the search states. In our application of the decision rule to planning in practice, we deal with this assumption by adopting the average branching factor and heuristic computation times, estimated from a random sample of search states. Finally, the model assumes perfect knowledge about the surely expanded search states. In practice, this information is obviously not available. We approach this issue conservatively by treating all the examined search states as if they were on the decision border, and thus apply the decision rule at all the search states. Note that this does not hurt the correctness of our algorithm, but only costs us some heuristic computation time on the surely expanded states. Identifying the surely expanded region during search is the subject of ongoing work, and can hopefully be used to improve search efficiency even further.

Online Learning of the Selection Rule

Our decision rule for choosing a heuristic to compute at a given search state s suggests to compute the more expensive heuristic h_2 when $h_2(s) - h_1(s) > \tau$. However, computing $h_2(s) - h_1(s)$ requires computing in s both heuristics, defeating the whole purpose of reducing search time by selectively evaluating only one heuristic at each state. To overcome this pitfall, we take our decision rule as a target concept, and suggest an *active online learning* procedure for that concept. Intuitively, our concept is the set of states where the more expensive heuristic h_2 is "significantly" more accurate than the cheaper heuristic h_1. According to our model, this corresponds to the states where the reduction in expanded states by computing h_2 outweighs the extra time needed to compute it. In what follows, we present our learning-based methodology in detail, describing the way we select and label training examples, the features we use to represent the examples, the way we construct our classifier, and the way we employ it within A^* search.

To build a classifier, we first need to collect training examples, which should be representative of the entire search space. One option for collecting the training examples is to use the first k states of the search where k is the desired number of training examples. However, this method has a bias towards states that are closer to the initial state, and therefore is not likely to well represent the search space. Hence, we instead collect training examples by sending "probes" from the initial state. Each such "probe" simulates a stochastic hill-climbing search with a depth limit cutoff. All the states generated by such a probe are used as training examples, and we stop probing when k training examples have been collected. In our evaluation, the probing depth limit was set to twice the heuristic estimate of the initial state, that is $2 \max_h(s_0)$, and the next state s for an ongoing probe was chosen with a probability proportional to $1/\max_h(s)$. This "inverse heuristic" selection biases the sample towards states with lower heuristic estimates, that is, to states that are more likely to be expanded during the search. It is worth noting here that more sophisticated procedures for search space sampling have been proposed in the literature (e.g., see Haslum et al. 2007), but as we show later, our much simpler sampling method is already quite effective for our purpose.

After the training examples T are collected, they are first used to estimate b, t_1 and t_2 by averaging the respective quantities over T. Once b, t_1 and t_2 are estimated, we can compute the threshold $\tau = \alpha \log_b(t_2/t_1)$ for our decision rule. We generate a label for each training example by calculating $\Delta_h(s) = h_2(s) - h_1(s)$, and comparing it to the decision threshold. If $\Delta_h(s) > \tau$, we label s with h_2, otherwise with h_1. If $t_1 > t_2$ we simply switch between the heuristics—our decision is always *whether to compute the more expensive heuristic or not*; the default is to compute the cheaper heuristic, unless the classifier says otherwise.

Besides deciding on a training set of examples, we need to choose a set of features to represent each of these examples. The aim of these features is to characterize search states with respect to our decision rule. While numerous features for characterizing states of planning problems have been proposed in previous literature (see, e.g., Yoon, Fern, and Givan (2008); de la Rosa, Jiménez, and Borrajo (2008)), they were all designed for inter-problem learning, and most of them are not suitable for intra-problem learning like ours. In our work we decided to use only elementary features corresponding simply to the actual state variables of the planning problem.

Once we have our training set and features to represent the examples, we can build a binary classifier for our concept. This classifier can then play the role of our hypothetical or-

evaluate(s)
$\langle h, confidence \rangle$:= CLASSIFY(s, $model$)
if ($confidence > \rho$) **then return** $h(s)$
else
    $label := h_1$
    if $h_2(s) - h_1(s) > \alpha \log_b(t_2/t_1)$ **then** $label := h_2$
    update $model$ with $\langle s, label \rangle$
    return $\max(h_1(s), h_2(s))$

Figure 2: The selective max state evaluation procedure.

acle indicating which heuristic to compute where. However, as our classifier is not likely to be a perfect such oracle, we further consult the confidence the classifier associates with its classification. The resulting state evaluation procedure of selective max is depicted in Figure 2. If state s is to be evaluated by A^*, we use our classifier to decide which heuristic to compute. If the classification confidence exceeds a parameter threshold ρ, then only the indicated heuristic is computed for s. Otherwise, we conclude that there is not enough information to make a selective decision for s, and compute the regular maximum over $h_1(s)$ and $h_2(s)$. However, we use this opportunity to improve the quality of our prediction for states similar to s, and update our classifier. This is done by generating a label based on $h_2(s) - h_1(s)$ and learning from this new example.[1] This can be viewed as the active part of our learning procedure.

The last decision to be made is the choice of classifier. Although many classifiers can be used here, there are several requirements that need to be met due to our particular setup. First, both training and classification must be very fast, as both are performed during time-constrained problem solving. Second, the classifier must be incremental to allow online update of the learned model. Finally, the classifier should provide us with a meaningful confidence for its predictions. While several classifiers meet these requirements, we found the classical Naive Bayes classifier to provide a good balance between speed and accuracy (Mitchell 1997). One note on the Naive Bayes classifier is that it assumes a very strong conditional independence between the features. Although this is not a fully realistic assumption for planning problems, using a SAS$^+$ formulation of the problem instead of the classical STRIPS helps a lot: instead of many binary variables which are highly dependent upon each other, we have a much smaller set of variables which are less dependent upon each other.[2]

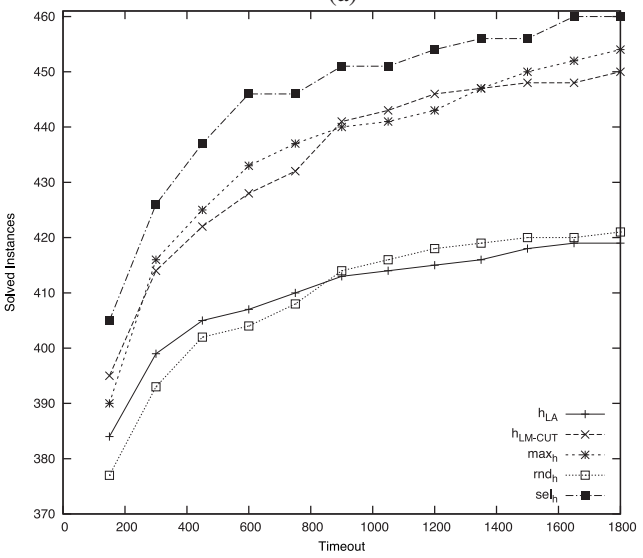
As a final note, extending selective max to use more than two heuristics is rather straightforward—simply compare the heuristics in a pair-wise manner, and choose the best heuristic by a vote, which can either be a regular vote (i.e., 1 for the winner, 0 for the loser), or weighted according to the classifier's confidence. Although this requires a quadratic number of classifiers, training and classification time (at least with Naive Bayes) appear to be much lower than the overall time spent on heuristic computations, and thus the overhead induced by learning and classification is likely to remain relatively low.

[1] We do not change the estimates for b, t_1 and t_2, so the threshold τ remains fixed.

[2] The PDDL to SAS$^+$ translator (Helmert 2009) detects mutual exclusions between propositions in the PDDL representation of the problem, and creates a single SAS$^+$ state variable that represents the whole mutually exclusive set.

Domain	h_{LA}	$h_{\text{LM-CUT}}$	$\max_h$	rnd_h	sel_h
Average Time/Problem	39.65	38.59	41.39	42.6	**24.53**
Average Time/Domain	67.08	53.02	58.97	64.44	**33.83**
Total Solved	419	450	454	421	**460**

(a)



(b)

Figure 3: Summary of the evaluation. Table (a) summarizes the average search times and number of problem instances solved with each of the five methods under comparison. Plot (b) depicts the number of solved instances under different timeouts; the x- and y-axes capture the timeout in seconds and the number of problems solved, respectively.

Experimental Evaluation

To empirically evaluate the performance of selective max we have implemented it on top of the A^* implementation of the Fast Downward planner (Helmert 2006), and conducted an empirical study on a wide range of planning domains from the International Planning Competitions 1998-2006; the domains are listed in Table 2. The search for each problem instance was limited to 30 minutes[3] and to 1.5 GB of memory. The search times do not include the PDDL to SAS$^+$ translation as it is common to all planners, and is tangential to the issues considered in our study. The search times do include learning and classification time for selective max. In the experiments we set the size of the initial training set to 100, the confidence threshold ρ to 0.6, and α to 1.

Our evaluation of selective max was based on two state-of-the-art admissible heuristics h_{LA} (Karpas and Domshlak 2009) and $h_{\text{LM-CUT}}$ (Helmert and Domshlak 2009). In contrast to abstraction heuristics that are typically based on expensive offline preprocessing, and then very fast online per-

[3] Each search was given a single core of a 3GHz Intel E8400 CPU machine.

state computation (Helmert, Haslum, and Hoffmann 2007; Katz and Domshlak 2009), both h_{LA} and $h_{\text{LM-CUT}}$ perform most of their computation online. Neither of our two base heuristics is better than the other across all planning domains, although A^* with $h_{\text{LM-CUT}}$ solves more problems overall. On the other hand, the empirical time complexity of computing h_{LA} is typically much lower than that of computing $h_{\text{LM-CUT}}$.

We compare our selective max approach (sel_h) to each of the two base heuristics individually, as well as to their standard, max-based combination ($\max_h$). In addition, to avoid erroneous conclusions about the impact of our specific decision rule on the effectiveness of selective max, we also compare sel_h to a trivial version of selective max that chooses between the two base heuristics uniformly at random (rnd_h).

As the primary purpose of selective max is speeding up optimal planning, we first examine the average runtime complexity of A^* with the above five alternatives for search node evaluation. While the results of this evaluation in detail are given in Table 2, the table in Figure 3a provides the bottom-line summary of these results. The first two rows in that table provide the average search times for all five methods. The times in the first row are plain averages over all (403) problem instances that were solved by all five methods. Based on the same problem instances, the times in the second row are averages over average search times within each planning domain. Supporting our original motivation, these results clearly show that sel_h is on average *substantially faster* than any of the other four alternatives, including not only the max-based combination of the base heuristics, but also *both* our individual base heuristics. Focusing the comparison to only $\max_h$ (averaging on 454 problem instances that were solved with both $\max_h$ and sel_h), the average search time with sel_h was 65.2 seconds, in contrast to 90.75 seconds with $\max_h$. Although it is hard to measure the exact overhead of the learning component of selective max, the average overhead for training and classification over all problems that took more than 1 second to solve was about 2%

The third row in the table provides the total number of problems solved by each of the methods. Here as well, selective max is the winner, yet the picture gets even sharper when the results are considered in more detail. For all five methods, Figure 3b plots the total number of solved instances as a function of timeout. The plot is self-explanatory, and it clearly indicates that selective max has a consistently better anytime behavior than any of the alternatives. We also point out that A^* with sel_h solved all the problems that were solved by A^* with $\max_h$, and more. Finally, note that the results with rnd_h in terms of both average runtime and number of problems solved clearly indicate that the impact of the concrete decision rule suggested by our model on the performance of selective max is spectacular.

One more issue that is probably worth discussing is the "sophistication" of the classifiers that were learned for selective max. Having read so far, the reader may wonder whether the classifiers we learn are not just trivial classifiers in a sense that, for any given problem, they either always suggest computing $h_{\text{LM-CUT}}$ or always suggest computing h_{LA}. However, Table 1 shows that typically this is not the case. For each problem solved by sel_h, we recorded the distribution of the high-confidence decisions made by our classifier, and Table 1 shows the averages of these numbers for each domain. We say that a domain has a significant preference for heuristic h if the classifier chose h for over 75% of the search states encountered while searching the problems from that domain. Only 8 domains out of 22 had a significant preference, and even those are divided almost evenly—3 domains had significant preference for $h_{\text{LM-CUT}}$, while 5 had significant preference for h_{LA}.

Domain	$h_{\text{LM-CUT}}$	h_{LA}
airport	0.68	0.32
blocks	• 0.81	0.19
depots	0.67	0.33
driverlog	0.5	0.5
freecell	0.14	• 0.86
grid	0.05	• 0.95
gripper	0.06	• 0.94
logistics-2000	0.02	• 0.98
logistics-98	0.31	0.69
miconic	0.34	0.66
mprime	0.42	0.58
mystery	0.47	0.53
openstacks	0.06	• 0.94
pathways	• 1	0
psr-small	0.38	0.62
pw-notankage	0.62	0.38
pw-tankange	0.62	0.38
rovers	0.49	0.51
satellite	0.53	0.47
tpp	0.26	0.74
trucks	• 0.98	0.02
zenotravel	0.53	0.47

Table 1: Average distribution of the high-confidence decisions made by classifiers within sel_h. Cells marked with (•) correspond to significant (over 75%) preference for the respective heuristic.

Finally, we have also compared sel_h to two of its minor variants: in one, the successor state in each "probe" used to generate the initial training set is chosen from the successors uniformly, and in the other, the decision rule's threshold τ was set simply to 0. The results of this comparison are omitted here for the sake of brevity, but they both performed worse than sel_h.

Discussion

Learning for planning has been a very active field starting in the early days of planning (Fikes, Hart, and Nilsson 1972), and is recently receiving growing attention in the community. So far, however, relatively little work has dealt with learning for heuristic search planning, one of the most prominent approaches to planning these days. Most works in this direction have been devoted to learning macro-actions (see, e.g., Finkelstein and Markovitch 1998, Botea et al. 2005, and Coles and Smith 2007). Among the other works, the one most closely related to ours is probably the work by Yoon, Fern and Givan (2008) that suggest learning an (inadmissible) heuristic function based upon features extracted from relaxed plans. In contrast, our focus is on optimal planning. Overall, we are not aware of any previous work that deals with learning for optimal heuristic search.

The experimental evaluation demonstrates that selective max is a more effective method for combining arbitrary admissible heuristics than their regular point-wise maximization. Another advantage of the selective max approach is

Domain	h_{LA}		$h_{\text{LM-CUT}}$		$\max_h$		rnd_h		sel_h	
	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time
airport (25)	25	125.96	38	35.36	36	73.8	29	54.78	36	68.44
blocks (20)	20	66.01	28	3.71	28	6.39	28	6.44	28	5.59
depots (7)	7	196.91	7	65.99	7	103.26	7	155.14	7	94.36
driverlog (14)	14	66.67	14	110.87	14	86.04	14	120.84	14	81.31
freecell (15)	28	6.04	15	249.28	22	23.93	15	44.22	28	9.25
grid (2)	2	12.05	2	33.78	2	44.27	2	38.3	2	40.26
gripper (6)	6	71.6	6	106.48	6	264.79	6	161.98	6	77.07
logistics-2000 (19)	19	73.32	20	152.27	20	255.36	20	153.89	20	79.17
logistics-98 (5)	5	18.84	6	24.11	6	29.55	5	28.69	6	24.43
miconic (140)	140	2.03	140	8.04	140	10.08	140	5.67	140	7.65
mprime (19)	21	17.52	25	17.9	25	15.68	19	111.48	25	8
mystery (12)	13	7.55	17	1.61	17	2.03	14	57.93	17	2.49
openstacks (7)	7	15.93	7	72.3	7	75.83	7	52.69	7	17.11
pathways (4)	4	5.38	5	0.08	5	0.14	4	1.15	5	0.18
psr-small (48)	48	3.55	49	4.05	48	7.92	48	5.73	48	4.87
pw-notankage (16)	16	48.8	17	71.34	17	71.49	17	73.92	17	59
pw-tankange (9)	9	211.43	11	173.61	11	189.89	10	172.99	11	130.98
rovers (6)	6	122.7	7	5.23	7	8.79	6	45.72	7	7.97
satellite (7)	7	46.22	8	3.47	9	4.51	7	21.95	9	3.58
tpp (6)	6	108.54	6	14.36	6	5.9	6	56.32	6	5.69
trucks (7)	7	238.85	10	11.69	9	16.48	7	39.64	9	15.56
zenotravel (9)	9	9.84	12	0.91	12	1.33	10	8.27	12	1.28

Table 2: Results of the evaluation in details. For each pair of domain D and state evaluation method E, we give the number of problems in D that were solved by A^* using E (left column), and the average search time (right column). The search time is averaged only on problems that were solved using all five methods; the number of these problems for each domain is listed in parentheses next to the domain name.

that it can successfully exploit pairs of heuristics where one dominates the other. For example, the h_{LA} heuristic can be used with two action cost partitioning schemes: uniform and optimal (Karpas and Domshlak 2009). The heuristic induced by the optimal action cost partitioning dominates the one induced by the uniform action cost partitioning, but takes much longer to compute. Selective max could be used to learn when it is worth spending the extra time to compute the optimal cost partitioning, and when it is not. In contrast, the max-based combination of these two heuristics would simply waste the time spent on computing the uniform action cost partitioning.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Comp. Intell.* 11(4):625–655.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *JAIR* 24:581–621.

Burke, E.; Kendall, G.; Newall, J.; Hart, E.; Ross, P.; and Schulenburg, S. 2003. Hyper-heuristics: an emerging direction in modern search technology. In *Handbook of meta-heuristics*. chapter 16, 457–474.

Coles, A. I., and Smith, A. J. 2007. Marvin: A heuristic search planner with online macro-action learning. *JAIR* 28:119–156.

Coles, A. I.; Fox, M.; Long, D.; and Smith, A. J. 2008. Additive-disjunctive heuristics for optimal planning. In *ICAPS*, 44–51.

de la Rosa, T.; Jiménez, S.; and Borrajo, D. 2008. Learning relational decision trees for guiding heuristic planning. In *ICAPS*, 60–67.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *JAIR* 22:279–318.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *AIJ* 3:251–288.

Finkelstein, L., and Markovitch, S. 1998. A selective macro-learning algorithm and its application to the NxN sliding-tile puzzle. *JAIR* 8:223–263.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, 1007–1012.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 162–169.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *AAAI*, 944–949.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AIJ* 173:503–535.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*, 1728–1733.

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

Katz, M., and Domshlak, C. 2009. Structural-pattern databases. In *ICAPS*, 186–193.

Mitchell, T. M. 1997. *Machine Learning*. New York: McGraw-Hill.

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

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *JMLR* 9:683–718.