Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling

Heuristics for Bounded-Cost Search

Patrik Haslum
Australian National University & NICTA Optimisation Research Group
firstname.lastname@anu.edu.au

Abstract

The problem of searching for a plan with cost at most equal to a given absolute bound has attracted interest recently, and several search algorithms tailored specifically to this problem have been proposed. We investigate instead how to adapt planning heuristics to this setting. A few of the resulting heuristics, used in a greedy bounded-cost search, perform better than previous and baseline methods, but only by a small margin. Making effective use of the cost bound in bounded-cost planning, it appears, remains a challenge.

Introduction

Bounded-cost (heuristic) search is a graph search problem with an absolute bound on path cost. That is, given a problem and a cost bound C, we ask for a path of cost at most C. It does not have to be the cheapest path: any solution will do as long as its cost is within the bound, and our sole objective is to find such a path as quickly as possible. This is different from bounded suboptimal search, which asks for a solution with a cost that is at most a given multiple of the optimal cost, in that the bound is absolute. It is reminiscent of the resource-constrained shortest path (RCSP) problem, but different in that the path does not have to be shortest. Even so, RCSP algorithms that require access to the graph only through a successor function (e.g. Aneja, Aggarwal, and Nair 1983) can also be applied to bounded-cost search.

Recently, there has been a resurgence in interest in the bounded-cost search problem in planning (Stern, Puzis, and Felner 2011; Thayer et al. 2012). It is useful if we have a plan and wish to find a cheaper plan. Indeed, a bounded-cost search procedure can be turned into an anytime planner, by running successive searches using a bound strictly less than the cost of the last plan found for the next search. Bounded-cost search is also a natural model for planning under certain types of resource constraints, where any solution cannot consume more of the resource than what is available (Nakhost, Hoffman, and Müller 2012).

Perhaps the simplest bounded-cost search procedure we can imagine is a greedy search, using one heuristic to guide the search and another heuristic, which estimates cost to goal and is admissible, to calculate a cost f-value that is used to prune search nodes that provably cannot lead to a solution within the bound. But what kind of heuristic should we use to guide this search, to meet our objective minimising the time to find a solution? Planning heuristics come in two flavours: cost-sensitive heuristics, which try to estimate the *cost* of the *cheapest* plan (from the current state) and cost-ignorant heuristics, which estimate the *length* of the *shortest* plan. Neither of these is what we want: Cost-sensitive planning heuristics are notoriously poor at guiding the search towards any solution (Cushing, Benton, and Kambhampati 2010; Richter and Westphal 2010), and searching for the cheapest plan is unnecessary if the cost bound is loose enough to admit many other plans. Cost-ignorant (plan length) heuristics are good at estimating distance-to-goal in directional search spaces (forward or backward search), but the shortest plan that exists within the bound may be completely different from the shortest plan in the unconstrained search space. In other words, even if the heuristic is perfectly accurate, it may still misdirect the search because the heuristic is not estimating the right thing.

This paper examines the problem of designing a heuristic that is appropriate for the bounded-cost search problem; that is, a heuristic $h(s, C)$ that given a state and a cost bound tries to estimate the distance to the nearest goal state that is reachable within the cost bound. Algorithms proposed for bounded-cost heuristic search have attempted to use more than a single heuristic estimate, but have left the heuristics unchanged. Stern et al. (2011) assume access to a model of the error of a cost-sensitive heuristic, and use this to convert the heuristic value into a "potential" which better guides the search. Thayer et al. (2012) combine estimates of cost and unconstrained distance from different heuristics.

It turns out that greedy bounded-cost search with the standard (approximately) shortest delete-relaxed plan heuristic is quite effective; on planning benchmark problems it outperforms the PTS, BEES and BEEPS bounded-cost search algorithms, using unmodified planning heuristics. By blending shortest and cheapest relaxed plans in the heuristic computation, and using an explicit heuristic penalty for over-cost relaxed plans, we are able to improve over it, but not by much. Creating heuristics or search algorithms for bounded-cost planning that substantially improve over algorithms and heuristics not designed specifically for the bounded-cost setting appears to be a challenge.

Relaxed Plan Heuristics

Relaxed plan heuristics, in general, find a solution (plan) to a relaxation of the planning problem and use some value computed from this plan, such as its length or cost, as the heuristic estimate. However, modern heuristic search-based planners use more information extracted from the relaxed plan than a single number. In particular, actions in the relaxed plan that are applicable in the current state, known as "helpful" (Hoffmann and Nebel 2001) or "preferred" (Helmert 2006) actions, can be used to focus a greedy search. Accurate recognition of preferred actions (i.e., a strong correlation between an action's appearances in the relaxed and real plans) can largely compensate for an inaccurate heuristic, as well as reduce search in combination with an accurate heuristic (Richter and Helmert 2009). Thus, the *structure* of the relaxed plan is as important as the heuristic value that is obtained from it. This is particularly relevant in bounded-cost search, since relaxed plans with different costs (below and above the bound) often have similar length but very different sets of constituent actions.

Here, we will concentrate on plans for the delete relaxation of planning problems. The delete relaxation of a planning problem ignores negative action effects (deletes). Delete-relaxed plan heuristics were introduced in the FF planner (Hoffmann and Nebel 2001), but here we follow roughly the formulation of Keyder & Geffner (2008). A relaxed plan is found in two steps: First, an estimated cost of achieving each proposition subset, from the current state s, is computed. From this estimate, a *best supporting action* for each proposition p is chosen as the action a with $p \in \mathrm{add}(a)$ that minimises $h(\mathrm{pre}(a); s) + \mathrm{cost}(a)$, where $h(\mathrm{pre}(a); s)$ is the heuristic estimate of the cost of achieving a's preconditions from s. (If we seek a short relaxed plan, $\mathrm{cost}(a)$ is 1 for all actions.) Second, the relaxed plan is extracted by backchaining from the goals of the planning problem, selecting for each subgoal the best supporter to achieve it. That is,

$$\pi(p; s) = \begin{cases} \emptyset & \text{if } p \in s \\ \{\mathrm{bs}(p; s)\} \cup \bigcup_{q \in \mathrm{pre}(\mathrm{bs}(p;s))} \pi(q; s) & \text{else,} \end{cases}$$

where $\mathrm{bs}(p; s)$ is the best supporter of p, and,

$$\pi(G; s) = \bigcup_{p \in G} \pi(p; s)$$

The relaxed plan found this way is not guaranteed to be optimal. Finally, the heuristic estimate is computed as a function of the extracted plan. To minimise the number of steps to reach a goal state, we use $h(s) = |\pi(G; s)|$.

This formulation leaves a lot of leeway to influence the relaxed plan we find by modifying the criterion for selecting the best supporter. If we want an (approximately) shortest relaxed plan, we use for $h(p; s)$ a heuristic that estimates the number of actions needed to achieve p, and count all actions' costs as 1. If we want an (approximately) cheapest relaxed plan, we use a $h(p; s)$ that estimates cost. Common choices for this heuristic are the $h^{\max}$ and h^{add} heuristics, both of which can be computed either with actual action costs with unit costs. Here, we use the additive heuristic, h^{add}, since it has been shown to give more accurate relaxed plans (Keyder and Geffner 2008). We denote the additive heuristic computed with action costs by $h^{\mathrm{add}}[\mathrm{cost}]$ and the heuristic computed with unit costs by $h^{\mathrm{add}}[1]$; we use the same annotations to indicate the best supporter selected according to each heuristic.

A simple idea to balance the length and cost of the relaxed plan is to focus on length, but use cost for tie-breaking. That is, define the best supporter $\mathrm{bs}(p; s)$ as the action a with $p \in \mathrm{add}(a)$ that minimises $h^{\mathrm{add}}[\mathrm{cost}](\mathrm{pre}(a); s) + \mathrm{cost}(a)$ among those that minimise $h^{\mathrm{add}}[1](\mathrm{pre}(a); s)$.

In cost-bounded search, we have at every search node a "budget" of $B(n) = C - g(n)$, where $g(n)$ is the cost of the path to the node. This is as much as the rest of the plan is allowed to cost, if we are to respect the cost bound. We can (and should) take advantage of this information in computing the relaxed plan, while still keeping a focus on finding a short acceptable plan. In fact, the ideal solution is the shortest relaxed plan whose cost is less than $B(n)$, but to find this plan is equivalent to solving the delete relaxed problem (cost-)optimally, which is known to be NP-hard. Instead, we suggest an iterative improvement scheme: First compute an (approximately) shortest relaxed plan, π_0, based on $\mathrm{bs}[1]$. If the cost of this plan ($\mathrm{cost}(\pi_0) = \sum_{a \in \pi_0} \mathrm{cost}(a)$) exceeds the budget, $B(n)$, compute a new relaxed plan, π_1, using the cheapest best supporter $\mathrm{bs}[\mathrm{cost}](p; s)$ for all subgoals in some select set P of propositions. This process repeats as long as the plan is too costly, using in each iteration a larger set P. The process ends with a relaxed plan π_n whose cost is either within the budget, or that is extracted entirely according to $\mathrm{bs}[\mathrm{cost}]$, i.e., an approximately cheapest plan.

The remaining question is how to select, in each iteration, the set of propositions P whose best supporters are chosen by cost. There are many possible strategies: we explore three, which differ in how aggressively they expand the set P. The first, called "improve once", simply takes as P the entire set of propositions immediately. Thus, this strategy computes at most two relaxed plans: first, a short plan, and if this plan is too expensive, an (approximately) cheapest. This limits the overhead that all improvement strategies incur for computing multiple relaxed plans. The second, called "improve top-down", adds in each iteration one of the top-level goals, and all subgoals recursively relevant to this goal (following the cheapest supporter). Goals are added in order of decreasing cost, i.e., the subplan for the most expensive goal is replaced first. The number of iterations under this strategy is at most the number of top-level goals. The third strategy, called "improve bottom-up", adds in each iteration one subgoal that appears in the relaxed plan, in order of distance from the current state, i.e., by increasing $h^{\mathrm{add}}[1]$. Replacing subplans from the bottom ensures that these subgoals are still relevant, since they appear in the (unchanged) recursive plan extraction from subgoals above. A new subplan is kept only if it decreases the total cost of the relaxed plan. This strategy is potentially the most expensive, since it can perfom a large number of iterations.

Penalising Too Expensive Relaxed Plans

If the heuristic was admissible, we would simply cut off any node whose estimated cost-to-go exceeds the budget, but as the cost of the relaxed plan is not optimal we cannot be

so drastic and retain completeness. Instead, we can apply a "soft" pruning, in the form of a penalty to the heuristic value if the cost of the relaxed plan exceeds the budget. This moves such nodes further back in the open queue, but leaves them to be explored if no seemingly better node leads to a solution. The penalty is multiplicative, that is,

$$h(n) = \begin{cases} f_{\text{penalty}}|\pi(G;n.s)| & \text{if } \text{cost}(\pi(G;n.s)) > B(n) \\ |\pi(G;n.s)| & \text{otherwise} \end{cases}$$

Budget overruns tend to occur relatively deep in the search, where heuristic values are small. This suggests that the penalty factor should be fairly high. Experiments, however, indicate a smaller penalty may be better.

Experiments

We compare the different heuristics in a bounded-cost greedy search on problems from the satisficing track of the 2008 and 2011 International Planning Competition[1]. With the exception of the Tidybot domain, these are all problems with non-unit action costs. Cost bounds are set to the cost of the second-best and the best plan found by any planner in the respective competition, minus 1. In other words, the first set of problems seek a plan better than the 2^{nd} best, and the second set a plan better than the best known. In the first case, a plan within the cost bound is known to exist. Because plan costs vary greatly between domains, the cost of the 2^{nd} best plan is thought to be a better measure of a "less constrained" bounded-cost problem than, for example, the cost of the best plan plus some constant. For problems in the second set, a solution may not exist. Note that when no plan exists within the bound, all search algorithms must explore the entire space up the point where the admissible cost estimate cuts off, and thus will behave more or less the same. The first test set excludes instances for which no 2^{nd} best plan exists, and the second instances for which the best plan is known to be optimal. For domains that appeared in both competitions we use only the subset of instances from 2011 that were new in that year.

The heuristics compared are the shortest, cheapest, shortest tie-breaking on cheapest, and improve once, top-down and bottom-up relaxed plans, as described above. We use preferred actions (based on the relaxed plan) in a dual queue (Richter and Helmert 2009), but not deferred evaluation. All configurations use the admissible LM-Cut heuristic (Helmert and Domshlak 2009) for cut-offs. Unlike a normal greedy search, bounded-cost search reopens closed states when reached by a new path of lower cost. This is necessary to retain completeness.

We also compare the greedy searches with the $\hat{\text{PTS}}$, BEES and BEEPS bounded-cost search algorithms. $\hat{\text{PTS}}$ uses the cost of the cheapest relaxed plan to calculate the potential. BEES and BEEPS use the shortest relaxed plan heuristic as the distance estimator and the cost of the cheapest relaxed plan as the inadmissible cost estimator. All three use LM-Cut as the admissible cost estimate for pruning. All search algorithms and heuristics are implemented in the Fast Downward planner.[2] All planners were run with 30-minute CPU time and 3Gb memory limits.

Discussion of Results

The number of problems solved are summarised in Table 1. Clearly no search algorithm or heuristic dominates: each performs relatively poorly in some domain. Greedy search using the shortest relaxed plan is much better than greedy search using the (length of the) cheapest relaxed plan, confirming previous observations that cost-sensitive heuristics are less effective at guiding search to a plan (Cushing, Benton, and Kambhampati 2010; Richter and Westphal 2010).

Of the hybrid strategies, improve once and top-down are better than following the cheapest relaxed plan, while improve bottom-up is not. Somewhat surprisingly, there is no evidence that this is due to the overhead of computing many relaxed plans during the improvement process: averaged across instances solved by all greedy search strategies, total run time divided by total node expansions shows no significant differences.

Comparing the relaxed plan heuristics pairwise, on instances solved by both in each pair, reveals a little more: As expected, using the cheapest relaxed plan expands more nodes than using the shortest, about 30% more. But all three relaxed plan improvement strategies expand even more (about 50–60% more). Even the two heuristics that solve more instances than the shortest relaxed plan (improve once improve top-down, with penalty) expand around 30% more nodes (on those instances that are solved also using the shortest relaxed plan). Comparing the three improvement strategies, once and top-down expand very similar numbers of nodes, while the bottom-up strategy expands slightly fewer. We may conjecture that good performance of the shortest plan heuristic is due to "luck", i.e., that the plan it leads to is within the bound most of the time only because the cost bounds are loose. However, the relative performance of all relaxed plan heuristics, with and without penalty, remains roughly the same also using the tighter cost bound (last row in Table 1).

The impact of the penalty for over-budget relaxed plans is far from even. It has the most beneficial effect when applied to a relaxed plan that was extracted taking cost into account: $f_{\text{penalty}} = 10$ leads to an increase by 10 instances solved when applied to the cheapest relaxed plan, and 19 instances solved when applied to the relaxed plan found by the improve once strategy. Experiments with a subset of strategies suggest that a smaller penalty factor is better (though 9 of the additional problems solved with the cheapest relaxed plan and $f_{\text{penalty}} = 2$ are in the ParcPrinter domain).

The $\hat{\text{PTS}}$, BEES and BEEPS search algorithms are much less effective than greedy search, regardless of which relaxed plan heuristic is used (though note the exception in the PegSol domain). This raises the question, why? First, note that $\hat{\text{PTS}}$, BEES and BEEPS do not benefit from preferred actions. Implementing preferred actions in these algorithms is non-trivial: in $\hat{\text{PTS}}$, the applicable actions in the

[1] http://ipc.informatik.uni-freiburg.de/; http://www.plg.inf.uc3m.es/ipc2011-deterministic

[2] http://fast-downward.org

First problem set: C = cost of 2^{nd} best plan $- 1$

Domain	#	Greedy search												P	BE	BE
		Sh.	Ch.	StC	I.1	I.T.	I.B.	Sh.	Ch.	StC	I.1.	I.T.	I.B.	T	ES	EP
		No penalty						Penalty factor 10						S		S
Elevators	40	21	8	16	21	22	11	26	11	19	26	25	14	9	21	21
Openstacks	40	23	25	23	22	21	23	18	25	16	22	21	23	12	23	23
ParcPrinter	39	24	34	21	26	26	23	23	24	21	25	23	22	8	9	9
PegSol	34	25	20	25	23	22	26	25	26	25	26	25	28	34	34	34
Scanalyzer	30	26	21	21	21	21	18	22	22	22	22	22	18	22	23	23
Sokoban	23	10	8	10	9	9	11	11	10	11	10	10	11	10	10	10
Transport	40	12	11	13	10	11	10	13	13	11	12	11	11	6	8	8
Woodworking	39	23	22	21	23	23	19	20	22	20	25	24	19	4	3	3
Barman	20	9	9	9	9	9	5	9	9	9	9	9	5	0	0	0
Floortile	8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
NoMystery	19	11	11	11	11	11	11	16	16	16	16	15	11	15	11	11
Parking	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
VisitAll	20	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Tidybot	20	4	4	4	4	4	4	5	5	5	5	5	5	6	7	6
Total:	392	190	175	176	181	181	163	190	185	178	200	192	169	128	151	150
								Penalty factor 2								
Total:								195	195		200					
								Penalty factor 1000								
Total:								189	185		199					

Second problem set: C = cost of best plan $- 1$

Total:	286	54	46	44	51	48	34	54	50	45	57	60	42	19	31	30

Table 1: Number of bounded-cost problems solved, IPC 2008 & 2011 satisficing problem set. Column "#" shows the number of instances. Relaxed plan types are abbreviated as follows: Shortest: Sh. Cheapest: Ch. Shortest, tie-breaking on cheapest: StC. Improve once: I.1. Improve top-down: I.T. Improve bottom-up: I.B.

relaxed plan do not necessarily correlate with the change in potential, and BEES and BEEPS already have multiple open queues with rules for selecting which queue to expand from. However, disabling preferred actions in the greedy search only decreases problems solved by 15% with the shortest plan heuristic and 13% with the improve once heuristic. In other words, this is not enough to explain the difference in performance. Another possible explanation is that these algorithms rely on assumptions about the (inadmissible) cost-estimating heuristic that are not satisfied by relaxed plan heuristics. $\hat{\text{PTS}}$ assumes access to a model of heuristic error.[3] The explicit estimation search algorithm, on which BEES and BEEPS build, assumes "an unbiased estimate of the cost-to-go" (Thayer and Ruml 2011), i.e., that the inadmissible cost estimating heuristic neither over-estimates nor under-estimates systematically. It is questionable to what extent this is true of relaxed plan heuristics, or of planning heuristics generally. Looking only at the initial state estimate of length, and comparing it to the length of the final plan, over instances solved with the shortest relaxed plan heuristic, the cases where the heuristic errs to one side outnumber the cases where it errs to the other at least $10 : 1$ in every domain except Scanalyzer.

[3]The $\hat{\text{PTS}}$ implementation uses a linear heuristic error model. This is the same model that was used for experimental evaluation in in previous work (Stern, Puzis, and Felner 2011; Thayer et al. 2012).

Conclusions

This investigation into heuristics for bounded-cost planning perhaps opens more problems and questions for the future than it provides answers. Designing heuristics, or any mechanism, that effectively uses the cost bound information to *guide* – not just prune – the search for a plan appears to be a challenge. Given the potential usefulness of efficient bounded-cost planning, this seems an important challenge for future research to address.

The ideas applied here to delete-relaxed plans can, in principle, be used for any kind of heuristic estimate based on finding a solution for some kind of problem relaxation. For example, abstraction of a planning problem yields a problem with a smaller state space, typically one small enough that it can be represented explicitly. Finding the shortest plan within the cost bound (budget) for the abstract problem amounts to solving a resource-constrained shortest path problem on the state space graph. Algorithms for RCSP are only pseudo-polynomial (i.e., polynomial in the magnitude of the cost bound, rather than its representation), but finding the shortest or cheapest path is easy, so hybrid strategies similar to those we have proposed can be devised. Also, unlike delete relaxed planning, RCSP admits polynomial time approximation schemes (Hassin 1992).

Acknowledgements

I'm grateful to Jordan Thayer for providing the PTS, BEES and BEEPS implementations. NICTA is funded by the Australian Government represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Aneja, Y.; Aggarwal, V.; and Nair, K. 1983. Shortest chain subject to side constraints. *Networks* 13:295–302.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost-based search considered harmful. In *Proc. Symposium on Combinatorial Search (SoCS'10)*.

Hassin, R. 1992. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research* 17(1):36–42.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Helmert, M. 2006. The Fast Downward planning system. *Journal of AI Research* 26:191–246.

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

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. European Conference on AI*.

Nakhost, H.; Hoffman, J.; and Müller, M. 2012. Resource-constrained planning: A monte carlo random walk approach. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 181–189.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 273–280.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of AI Research* 39:127–177.

Stern, R.; Puzis, R.; and Felner, A. 2011. Potential-search: A bounded-cost search algorithm. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 234–241.

Thayer, J., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proc. 22nd International Conference on Artificial Intelligence (IJCAI'11)*, 674–679.

Thayer, J.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible heuristics. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 270–278.