

# Cost Based Satisficing Search Considered Harmful

William Cushing and J. Benton and Subbarao Kambhampati\*

Dept. of Comp. Sci. and Eng.  
Arizona State University  
Tempe, AZ 85281

## Abstract

Recently, several researchers have found that cost-based satisficing search with A\* often runs into problems. Although some “work arounds” have been proposed to ameliorate the problem, there has not been any concerted effort to pinpoint its origin. In this paper, we argue that the origins can be traced back to the wide variance in action costs that is easily observed in planning domains. We show that such cost variance misleads A\* search, and that this is a systemic weakness of the very concept: “cost-based evaluation functions + systematic search + combinatorial graphs”. We argue that purely size-based evaluation functions are a reasonable default, as these are trivially immune to cost-induced difficulties. We further show that cost-sensitive versions of size-based evaluation function — where the heuristic estimates the size of cheap paths provides attractive quality vs. speed tradeoffs.

## 1 Introduction

Much of the scale-up, as well as the research focus, in the automated planning community in the recent years has been on satisficing planning. Unfortunately, there hasn’t been a concomitant increase in our understanding of satisficing search. Too often, the “theory” of satisficing search defaults to doing (W)A\* with inadmissible heuristics. While removing the requirement of admissible heuristics certainly relaxes the guarantee of optimality, there is no implied guarantee of efficiency. A combinatorial search can be seen to consist of two parts: a “discovery” part where the (optimal) solution is found and a “proof” part where the optimality of the solution is verified. While an optimizing search depends crucially on both these phases, a satisficing search is instead affected more directly by the discovery phase. Now, standard A\* search conflates the discovery and proof phases together and terminates only when it picks the optimal path for expansion. By default, satisficing planners use the same search regime, but relax the admissibility requirement on the heuristics.<sup>1</sup> This may not cause too much of a problem in domains with uniform action costs, but when actions can have

non-uniform costs, the the optimal and second optimal solution can be arbitrarily far apart in depth. Consequently, (W)A\* search with cost-based evaluation functions can be an arbitrarily bad strategy for satisficing search, as it waits until the solution is both discovered *and* proved to be (within some bound of) optimal.

To be more specific, consider a planning problem for which the cost-optimal and second-best solution to a problem exist on 10 and 1000 unspecified actions. *The optimal solution may be the larger one.* How long should it take just to find the 10 action plan? How long should it take to prove (or disprove) its optimality? In general (presuming PSPACE/EXSPACE  $\neq$  P):

1. Discovery should require time exponential in, at most, 10.
2. Proof should require time exponential in, at least, 1000.

That is, in principle, the only way to (domain-independently) prove that the 10 action plan is better or worse than the 1000 action one is to in fact go and discover the 1000 action plan. Thus, A\* search with cost-based evaluation function will take time proportional to  $b^{1000}$  for either discovery or proof. Simple breadth-first search discovers a solution in time proportional to  $b^{10}$  (and proof in  $O(b^{1000})$ ).

Using both abstract and benchmark problems, we will demonstrate that this is a systematic weakness of any search that uses cost-based evaluation function. In particular, we shall see that if  $\varepsilon$  is the smallest cost action (after all costs are normalized so the maximal cost action costs 1 unit), then the time taken to discover a depth  $d$  optimal solution will be  $b^{\frac{d}{\varepsilon}}$ . If all actions have same cost, then  $\varepsilon \approx 1$  where as if the actions have significant cost variance, then  $\varepsilon \ll 1$ . We shall see that for a variety of reasons, most real-world planning domains do exhibit high cost variance, thus presenting an “ $\varepsilon$ -cost trap” that forces any cost-based satisficing search to dig its own ( $\frac{1}{\varepsilon}$  deep) grave.

Consequently, we argue that satisficing search should resist the temptation to directly use cost-based evaluation functions (*i.e.*,  $f$  functions that return answers in cost units) even if they are interested in the quality (cost measure) of the resulting plan. We will consider two size-based branch-and-bound alternatives: the straightforward one which completely ignores costs and sticks to a purely size-based evaluation function, and a more subtle one that uses a cost-sensitive size-based evaluation function (specifically, the heuristic estimates the size of the cheapest cost path; see Section 2). We show that both of these outperform cost-based evaluation

\*An extended abstract of this paper appeared in the proceedings of SOCS 2010. This research is supported in part by ONR grants N00014-09-1-0017 and N00014-07-1-1049, the NSF grant IIS-0905672, and by DARPA and the U.S. Army Research Laboratory under contract W911NF-11-C-0037.

<sup>1</sup>In the extreme case, by using an infinite heuristic weight: “greedy best-first search”.

functions in the presence of  $\varepsilon$ -cost traps, with the second one providing better quality plans (for the same run time limits) than the first in our empirical studies.

While some of the problems with cost-based satisficing search have also been observed, in passing, by other researchers (e.g., (Benton et al. 2010; Richter and Westphal 2010)), and some work-arounds have been suggested, our main contribution is to bring to the fore its fundamental nature. The rest of the paper is organized as follows. In the next section, we present some preliminary notation to formally specify cost-based, size-based as well as cost-sensitive size-based search alternatives. Next, we present two abstract and fundamental search spaces, which demonstrate that *cost-based* evaluation functions are ‘always’ needlessly prone to such traps (Section 3). Section 4 strengthens the intuitions behind this analysis by viewing best-first search as flooding topological surfaces set up by evaluation functions. We will argue that of all possible topological surfaces (i.e., evaluation functions) to choose for search, cost-based is the worst. In Section 5, we put all this analysis to empirical validation by experimenting with LAMA (Richter and Westphal 2010) and SapaReplan. The experiments do show that size-based alternatives can out-perform cost-based search. Modern planners such as LAMA use a plethora of improvements beyond vanilla A\* search, and in the appendix we provide a deeper analysis on which extensions of LAMA seem to help it mask (but not fully overcome) the pernicious effects of cost-based evaluation functions.

## 2 Setup and Notation

We gear the problem set up to be in line with the prevalent view of state-space search in modern, state-of-the-art satisficing planners. First, we assume the current popular approach of reducing planning to graph search. That is, planners typically model the state-space in a causal direction, so the problem becomes one of extracting paths, meaning whole plans do not need to be stored in each search node. More important is that the structure of the graph is given *implicitly* by a procedure  $\Gamma$ , the child generator, with  $\Gamma(v)$  returning the local subgraph leaving  $v$ ; i.e.,  $\Gamma(v)$  computes the subgraph  $(N^+[v], E(\{v\}, V - v)) = (\{u \mid (v, u) \in E\} + v, \{(v, u) \mid (v, u) \in E\})$  along with all associated labels, weights, and so forth. That is, our analysis depends on the assumption that *an implicit representation of the graph is the only computationally feasible representation*, a common requirement for analyzing the A\* family of algorithms (Hart, Nilsson, and Raphael 1968; Dechter and Pearl 1985).

The search problem is to find a path from an initial state,  $i$ , to some goal state in  $\mathcal{G}$ . Let costs be represented as edge weights, say  $c(e)$  is the cost of the edge  $e$ . Let  $g_c^*(v)$  be the (optimal) cost-to-reach  $v$  (from  $i$ ), and  $h_c^*(v)$  be the (optimal) cost-to-go from  $v$  (to the goal). Then  $f_c^*(v) := g_c^*(v) + h_c^*(v)$ , the cost-through  $v$ , is the cost of the cheapest  $i$ - $\mathcal{G}$  path passing through  $v$ . For discussing smallest solutions, let  $f_s^*(v)$  denote the smallest  $i$ - $\mathcal{G}$  path through  $v$ . It is also interesting to consider the size of the cheapest  $i$ - $\mathcal{G}$  path passing through  $v$ , say  $\hat{f}_s^*(v)$ .

We define a search node  $n$  as equivalent to a path represented as a linked list (of edges). In particular, we distinguish this from the state of  $n$  (its last vertex),  $n.v$ . We say

$n.a$  (for action) is the last edge of the path and  $n.p$  (for parent) is the subpath excluding  $n.a$ . Let  $n' = na$  denote extending  $n$  by the edge  $a$  (so  $a = (n.v, n'.v)$ ). The function  $g_c(n)$  ( $g$ -cost) is just the recursive formulation of path cost:  $g_c(n) := g_c(n.p) + c(n.a)$  ( $g_c(n) := 0$  if  $n$  is the trivial path). So  $g_c^*(v) \leq g_c(n)$  for all  $i$ - $v$  paths  $n$ , with equality for at least one of them. Similarly let  $g_s(n) := g_s(n.p) + 1$  (initialized at 0), so that  $g_s(n)$  is an upper bound on the shortest path reaching the same state ( $n.v$ ).

A goal state is a target vertex where a plan may stop and be a valid solution. We fix a computed predicate  $\mathcal{G}(v)$  (a black-box), the *goal*, encoding the set of goal states. Let  $h_c(v)$ , the *heuristic*, be a procedure to estimate  $h_c^*(v)$ . (Sometimes  $h_c$  is considered a function of the search node, i.e., the whole path, rather than just the last vertex.) The heuristic  $h_c$  is called *admissible* if it is a *guaranteed* lower bound. (An inadmissible heuristic lacks the guarantee, but might anyways be coincidentally admissible.) Let  $h_s(v)$  estimate the remaining depth to the nearest goal, and let  $\hat{h}_s(v)$  estimate the remaining depth to the cheapest reachable goal. Anything goes with such heuristics — an inadmissible estimate of the size of an inadmissible estimate of the cheapest continuation is an acceptable (and practical) interpretation of  $\hat{h}_s(v)$ .

We focus on two different definitions of  $f$  (the evaluation function). Since we study cost-based planning, we consider  $f_c(n) := g_c(n) + h_c(n.v)$ ; this is the (standard, cost-based) *evaluation function* of A\*: cheapest-completion-first. We compare this to  $f_s(n) := g_s(n) + h_s(n.v)$ , the canonical size-based (or search distance) evaluation function, equivalent to  $f_c$  under uniform weights. Any combination of  $g_c$  and  $h_c$  is *cost-based*; any combination of  $g_s$  and  $h_s$  is *size-based* (e.g., breadth-first search is size-based). The evaluation function  $\hat{f}_s(n) := g_s(n) + \hat{h}_s(n.v)$  is also size-based, but nonetheless cost-sensitive and so preferable.

BEST-FIRST-SEARCH( $i, \mathcal{G}, \Gamma, h_c$ )

```

1 initialize search
2 while open not empty
3   n = open.remove()
4   if BOUND-TEST(n, h_c) then continue
5   if GOAL-TEST(n, \mathcal{G}) then continue
6   if DUPLICATE-TEST(n) then continue
7   s = n.v
8   star = \Gamma(s) // Expand s
9   for each edge a = (s, s') from s to a child s' in star
10    n' = na
11    f = EVALUATE(n')
12    open.add(n', f)
13 return best-known-plan // Optimality is proven.
```

Pseudo-code for best-first branch-and-bound search of implicit graphs is shown above. It continues searching after a solution is encountered and uses the current best solution value to prune the search space (line 4). The search is performed on a graph implicitly represented by  $\Gamma$ , with the assumption being that the explicit graph is so large that it is better to invoke expensive heuristics (inside of EVALUATE) during the search than it is to just compute the graph up front. The question considered by this paper is how to implement EVALUATE.

With respect to normalizing costs, we can let  $\varepsilon :=$

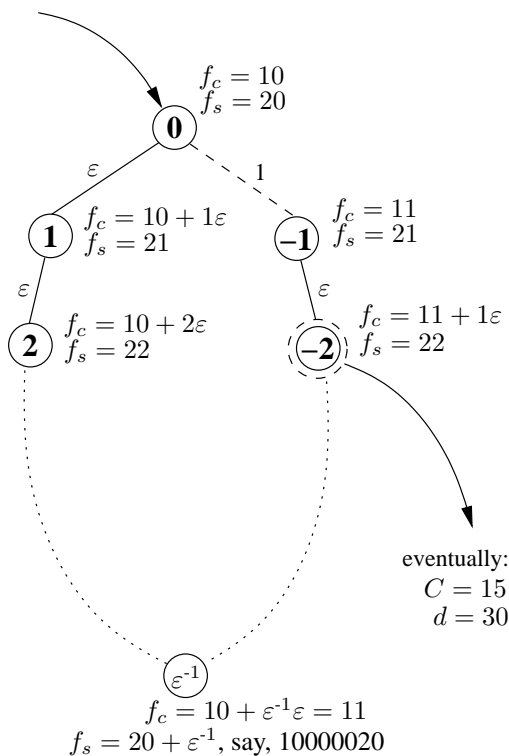


Figure 1: A trap for cost-based search. The heuristic perceives all movement on the cycle to be irrelevant to achieving high quality plans. The state with label **-2** is one interesting way to leave the cycle, there may be (many) others.  $C$  denotes the cost of one such continuation from **-2**, and  $d$  its depth. Edge weights nominally denote changes in  $f_c$ : as given, locally, these are the same as changes in  $g_c$ . But increasing  $f_s$  by 1 at **-1** (and descendants) would, for example, model instead the special edge as having cost  $\frac{1}{2}$  and being perceived as worst-possible in an undirected graph.

$\frac{\min_a c(a)}{\max_a c(a)}$ , that is,  $\varepsilon$  is the least cost edge after normalizing costs by the maximum cost (to bring costs into the range  $[0, 1]$ ). We use the symbol  $\varepsilon$  for this ratio as we anticipate actions with high cost variance in real world planning problems. For example: boarding versus flying (ZenoTravel), mode-switching versus machine operation (Job-Shop), and (unskilled) labor versus (precious) material cost.

### 3 $\varepsilon$ -cost Trap: Two Canonical Cases

In this section we argue that the mere presence of  $\varepsilon$ -cost misleads cost-based search, and that this is no trifling detail or accidental phenomenon, but a systemic weakness of the very concept of “cost-based evaluation functions + systematic search + combinatorial graphs”. We base this analysis in two abstract search spaces, in order to demonstrate the fundamental nature of such traps. The first abstract space we consider is the simplest, non-trivial, non-uniform cost, intractably large, search space: the search space of an enormous cycle with one expensive edge. The second abstract space we consider is a more natural model of search (in planning): a uniform branching tree. Traps in these spaces are just exponentially sized *and connected* sets of  $\varepsilon$ -cost edges:

not the common result of, say, a typical random model of search. We briefly consider why planning benchmarks naturally give rise to such structure.

For a thorough analysis of models of search see (Pearl 1984); for a planning specific context see (Helmert and Röger 2008).

#### 3.1 Cycle Trap

In this section we consider the simplest abstract example of the  $\varepsilon$ -cost ‘trap’. The notion is that applying increasingly powerful heuristics, domain analysis, learning techniques, ..., to one’s search problem transforms it into a simpler ‘effective graph’ — the graph for which Dijkstra’s algorithm (Dijkstra 1959) produces isomorphic behavior. For example, let  $c'$  be a new edge-cost function obtained by setting edge costs to the difference in  $f$  values of the edge’s endpoints: Dijkstra’s algorithm on  $c'$  is  $A^*$  on  $f$ .<sup>2</sup> Similarly take  $\Gamma'$  to be the result of applying one’s favorite incompleteness-inducing pruning rules to  $\Gamma$  (the child generator), say, helpful actions (Hoffmann and Nebel 2001); then Dijkstra’s algorithm on  $\Gamma'$  is  $A^*$  with helpful action pruning.

Presumably the effective search graph remains very complex despite all the clever inference (or there is nothing to discuss); but certainly complex graphs contain simple graphs as subgraphs. So if there is a problem with search behavior in an exceedingly simple graph then we can suppose that no amount of domain analysis, learning, heuristics, and so forth, will incidentally address the problem: such inference must specifically address the issue of non-uniform weights. Suppose not: none of the bells and whistles consider non-uniform costs to be a serious issue, permitting wildly varying edge ‘costs’ even in the effective search graph:  $\varepsilon \approx \varepsilon' = \frac{\min_e c'(e)}{\max_e c'(e)}$ . We demonstrate that that by itself is enough to produce very troubling search behavior:  $\varepsilon$ -cost is a fundamental challenge to be overcome in planning.

There are several candidates for simple non-trivial state-spaces (e.g., cliques), but clearly the cycle is fundamental (what kind of ‘state-space’ is acyclic?). So, the state-space we consider is the cycle, with associated exceedingly simple metric consisting of all uniform weights but for a single expensive edge. Its search space is certainly the simplest non-trivial search space: the rooted tree on two leaves. So the single unforced decision to be made is in which direction to traverse the cycle: clockwise or counter-clockwise. See Figure 1. Formally:

**$\varepsilon$ -cost Trap:** Consider the problem of making some variable, say  $x$ , encoded in  $k$  bits represent  $2^k - 2 \equiv -2 \pmod{2^k}$ , starting from 0, using only the operations of increment and decrement. There are 2 minimal solutions: incrementing  $2^k - 2$  times, or decrementing twice. Set the cost of incrementing and decrementing to 1, except for transitioning between  $x \equiv 0$  and  $x \equiv -1$  costs, say,  $2^{k-1}$  (in either direction). Then the 2 minimal solutions cost  $2^k - 2$  and  $2^{k-1} + 1$ , or, normalized,  $2(1 - \varepsilon)$  and  $1 + \varepsilon$ . Cost-based search loses: While both approaches prove optimality in exponential time ( $O(2^k)$ ), size-based search discovered that optimal plan in constant time.

<sup>2</sup>Systematic inconsistency of a heuristic translates to analyzing the behavior of Dijkstra’s algorithm with many *negative* ‘cost’ edges, a typical reason to assume consistency in analysis.

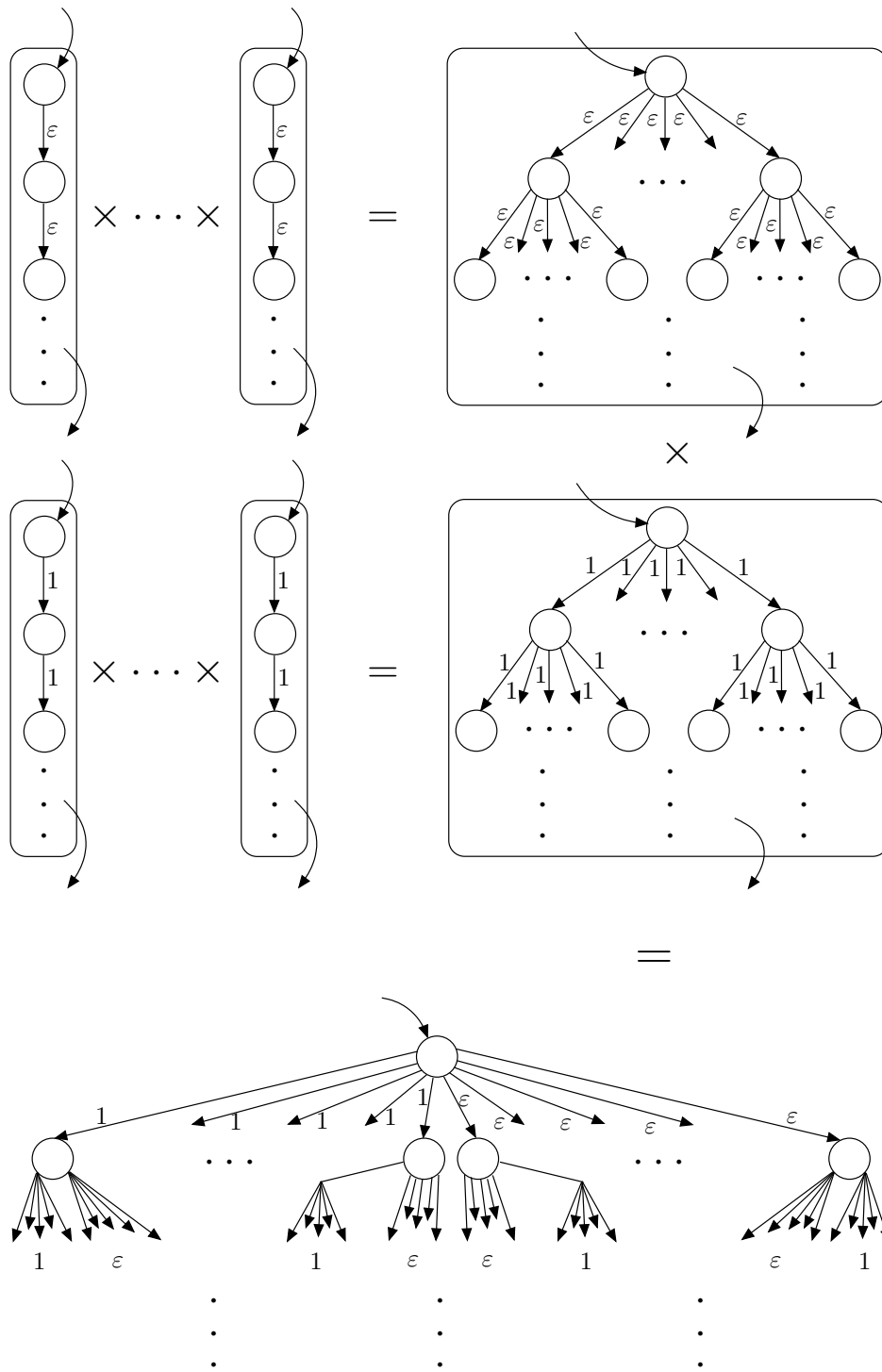


Figure 2: A trap for cost-based search. Two rather distinct kinds of physical objects exist in the domain, with primitive operators at rather distinct orders of magnitude; supposing uniformity and normalizing, then one type involves  $\epsilon$ -cost and the other involves cost 1. So there is a low-cost subspace, a high-cost subspace, and the full space, each a uniform branching tree. As trees are acyclic, it is probably best to think of these as search, rather than state, spaces. As depicted, planning for an individual object is trivial as there is no choice besides going forward. Other than that no significant amount of inference is being assumed, and in particular the effects of a heuristic are not depicted. For cost-based search to avoid death, the heuristic would need to forecast every necessary cost 1 edge, so as to reduce its weight closer to 0. (Note that the aim of a heuristic is to drive all the weights to 0 along optimal/good paths, and to infinity for not-good/terrible/dead-end choices.) If any cut of the space across such edges (separating good solutions) is not foreseen, then backtracking into *all* of the low-cost subspaces so far encountered commences, to multiples of depth  $\epsilon^{-1}$  — one such multiple for every unforeseen cost 1 cut. Observe that in the object-specific subspaces (the paths), a single edge ends up being multiplied into such a cut of the global space.

**Performance Comparison: All Goals.** Of course the goal  $x \equiv -2$  is chosen to best illustrate the trap. So consider the discovery problem for other goals. With the goal in the interval  $2^k \cdot [0, \frac{1}{2}]$  cost-based search is twice as fast. With the goal in the interval  $2^k \cdot [\frac{1}{2}, \frac{2}{3}]$  the performance gap narrows to break-even. For the last interval,  $2^k \cdot [\frac{2}{3}, 1)$ , the size-based approach takes the lead — *by an enormous margin*. There is one additional region of interest. Taking the goal in the interval  $2^k \cdot [\frac{2}{3}, \frac{3}{4}]$  there is a trade-off: size-based search finds a solution before cost-based search, but cost-based search finds the optimal solution first. Concerning time till optimality is proven, the cost-based approach is monotonically faster (of course). Specifically, the cost-based approach is faster by a factor of 2 for goals in the region  $2^k \cdot [0, \frac{1}{2}]$ , not faster for goals in the region  $2^k \cdot [\frac{3}{4}, 1)$ , and by a factor of  $(\frac{1}{2} + 2\alpha)^{-1}$  (bounded by 1 and 2) for goals of the form  $x \equiv 2^k(\frac{1}{2} + \alpha)$ , with  $0 < \alpha < \frac{1}{4}$ .

**Performance Comparison: Feasible Goals.** Considering all goals is inappropriate in the satisficing context; to illustrate, consider large  $k$ , say,  $k = 1000$ . Fractions of exponentials are still exponentials — even the most patient reader will have forcibly terminated either search *long* before receiving any useful output. Except if the goal is of the form  $x \equiv 0 \pm f(k)$  for some sub-exponential  $f(k)$ . Both approaches discover (and prove) the optimal solution in the positive case in time  $O(f(k))$  (with size-based performing twice as much work). In the negative case, only the size-based approach manages to discover a solution (the optimal one, in time  $O(f(k))$ ) before being killed. Moreover, while it will fail to produce a proof of such before death, we, based on superior understanding of the domain, can, and have, shown it to be posthumously correct.  $(2^k - f(k)) > 2^k \cdot \frac{3}{4}$  for any sub-exponential  $f(k)$  with large enough  $k$ .

**How Good is Almost Perfect Search Control?** Keep in mind that the representation of the space as a simple  $k$  bit counter is *not* available. In particular what ‘increment’ actually stands for is an inference-motivated choice of a single operator out of a large number of executable and promising operators at each state — in the language of Markov Decision Processes, we are allowing inference to be so close to perfect that the optimal policy is known at all but 1 state. Only one decision remains . . . but no methods cleverer than search remain. Still the graph is intractably large. Cost-based search only explores in one direction: left, say. In the satisficing context such behavior is entirely inappropriate. What is appropriate? Of course explore left first, for considerable time even. But certainly not for 3 years before even trying just a few expansions to the right, for that matter, even mere thousands of expansions to the left before one or two to the right are tried is perhaps too greedy.

### 3.2 Branching Trap

In the counter problem the trap is not even *combinatorial*; the search problem consists of a single decision at the root, and the trap is just an exponentially deep path. Then it is abundantly clear that appending Towers of Hanoi to a planning benchmark, setting its actions at  $\varepsilon$ -cost, will kill cost-based search — even given the perfect heuristic for the puzzle! Besides Hanoi, though, exponentially deep paths

are not typical of planning benchmarks. So in this section we demonstrate that exponentially large subtrees on  $\varepsilon$ -cost edges are also traps.

Consider  $x > 1$  high cost actions and  $y > 1$  low cost actions in a uniform branching tree model of search space. The model is appropriate up to the point where duplicate state checking becomes significant. (See Figure 2.) Suppose the solution of interest costs  $C$ , in normalized units, so the solution lies at depth  $C$  or greater. Then cost-based search faces a grave situation:  $O((x + y^{\frac{1}{\varepsilon}})^C)$  possibilities will be explored before considering all potential solutions of cost  $C$ .

A size-based search only ever considers at most  $O((x + y)^d) = O(b^d)$  possibilities before consideration of all potential solutions of size  $d$ . Of course the more interesting question is how long it takes to find solutions of fixed cost rather than fixed depth. Note that  $\frac{C}{\varepsilon} \geq d \geq C$ . Assuming the high cost actions are relevant, that is, some number of them are needed by solutions, then we have that solutions are not actually hidden as deep as  $\frac{C}{\varepsilon}$ . Suppose, for example, that solutions tend to be a mix of high and low cost actions in equal proportion. Then the depth of those solutions with cost  $C$  is  $d = 2 \frac{C}{1+\varepsilon}$  (i.e.,  $\frac{d}{2} \cdot 1 + \frac{d}{2} \cdot \varepsilon = C$ ). At such depths the size-based approach is the clear winner:  $O((x + y)^{\frac{2C}{1+\varepsilon}}) \ll O((x + y^{\frac{1}{\varepsilon}})^C)$  (normally).

Consider, say,  $x = y = \frac{b}{2}$ , then:

size effort/cost effort  $\approx$

$$\begin{aligned} b^{\frac{2C}{1+\varepsilon}} / \left(x + y^{\frac{1}{\varepsilon}}\right)^C &< b^{\frac{2C}{1+\varepsilon}} / y^{\frac{C}{\varepsilon}}, \\ &< 2^{\frac{C}{\varepsilon}} / b^{\frac{C}{\varepsilon} \frac{1-\varepsilon}{1+\varepsilon}}, \\ &< \frac{2}{b^{\frac{1-\varepsilon}{1+\varepsilon}}} \end{aligned}$$

and, provided  $\varepsilon < \frac{1 - \log_b 2}{1 + \log_b 2}$  (for  $b = 4$ ,  $\varepsilon < \frac{1}{3}$ ), the last is always less than 1 and, for that matter, goes, quickly, to 0 as  $C$  increases and/or  $b$  increases and/or  $\varepsilon$  decreases.

Generalizing, the size-based approach is faster at finding solutions of any given cost, as long as (1) high-cost actions constitute at least some constant fraction of the solutions considered (high-cost actions are relevant), (2) the ratio between high-cost and low-cost is sufficiently large, (3) the effective search graph (post inference) is reasonably well modeled by an infinite uniform branching tree (i.e., huge enough to render duplicate checking negligible, or at least not especially favorable to cost-based search), and most importantly, (4) the cost function in the effective search graph *still* demonstrates a sufficiently large ratio between high-cost and low-cost edges (no inference has attempted to compensate).

## 4 Search Effort as Flooding Topological Surfaces of Evaluation Functions

We view evaluation functions ( $f$ ) as topological surfaces over search nodes, so that generated nodes are visited in, roughly, order of  $f$ -altitude. With non-monotone evaluation functions, the set of nodes visited before a given node is all those contained within some basin of the appropriate depth — picture water flowing from the initial state: if there are dams then such a flood could temporarily visit high altitude

nodes before low altitude nodes. (With very inconsistent heuristics — large heuristic weights — the metaphor loses explanatory power, as there is nowhere to go but downhill.)

All reasonable choices of search topology will eventually lead to identifying and proving the optimal solution (*e.g.*, assume finiteness, or divergence of  $f$  along infinite paths). Some will produce a whole slew of suboptimal solutions along the way, eventually reaching a point where one begins to wonder if the most recently reported solution is optimal. Others report nothing until finishing. The former are *interruptible* (Zilberstein 1998), which is one way to more formally define satisficing.<sup>3</sup> Admissible cost-based topology is the least interruptible choice: the only reported solution is also the last path considered. Define the cost-optimal footprint as the set of plans considered. Gaining interruptibility is a matter of raising the altitude of large portions of the cost-optimal footprint in exchange for lowering the altitude of a smaller set of non-footprint search nodes — allowing suboptimal solutions to be considered. Note that interruptibility comes at the expense of total work.

So, somewhat confirming the intuition that interruptibility is a reasonable notion of satisficing: the cost-optimal approach is the worst-possible approach (short of deliberately wasting computation) to satisficing. Said another way, proving optimality is about increasing the lower bound on true value, while solution discovery is about decreasing the upper bound on true value. It seems appropriate to assume that the fastest way to decrease the upper bound is more or less the opposite of the fastest way to increase the lower bound — with the notable exception of the very last computation one will ever do for the problem: making the two bounds meet (proving optimality).

For size-based topology, with respect to any cost-based variant, the ‘large’ set is the set of *longer yet cheaper* plans, while the ‘small’ set is the *shorter yet costlier* plans. In general one expects there to be many more longer plans than shorter plans in combinatorial problems, so that the increase in total work is small, relative to the work that had to be done eventually (exhaust the many long, cheap, plans). The additional work is considering exactly plans that are costlier than necessary (potentially suboptimal solutions). So the idea of the trade-off is good, but even the best version of a purely size-based topology will not be the best trade-off possible — intuitively the search shouldn’t be completely blind to costs: just defensive about possible  $\epsilon$ -traps.

Actually, there is a common misconception here about *search nodes* and *states* due to considering uniformly branching trees as a model of state-space: putting states and search nodes in correspondence. Duplicate detection and re-expansion are, in practice, important issues. In particular not finding the cheapest path first comes with a price, re-expansion, so the satisficing intent comes hand in hand with re-expansion of states. So, for example, besides the obvious kind of re-expansion that IDA\* (Korf 1985) performs between iterations, it is also true that it considers paths which A\* never would (even subsequent to arming IDA\* with a transposition table) — it is not really true that one can re-

<sup>3</sup>Another way that Zilberstein suggests is to specify a *contract*; the 2008 planning competition has such a format (Helmert, Do, and Refanidis 2008).

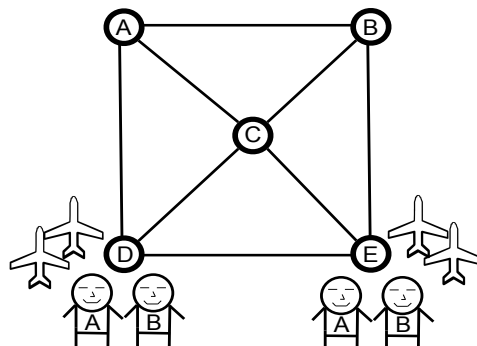


Figure 3: Rendezvous problems. Diagonal edges cost 7,000, exterior edges cost 10,000. Board/Debarck cost 1.

order consideration of paths however one pleases. In particular at least some kind of breadth-first bias is appropriate, so as to avoid finding woefully suboptimal plans to states early on, triggering giant cascades of re-expansion later on.

For thorough consideration of blending size and cost considerations in the design of evaluation functions see (Thayer and Ruml 2010). Earlier work in evaluation function design beyond just simplistic heuristic weighting is in (Pohl 1973). Dechter and Pearl give a highly technical account of the properties of generalized best-first search strategies, focusing on issues of computational optimality, but, mostly from the perspective of search constrained to proving optimality in the path metric (Dechter and Pearl 1985).

## 5 $\epsilon$ -cost Trap in Practice

In this section we demonstrate existence of the problematic planner behavior in a realistic setting: running LAMA on problems in the travel domain (simplified ZenoTravel, zoom and fuel removed), as well as two other IPC domains. Analysis of LAMA is complicated by many factors, so we also test the behavior of SapaReplan on simpler instances (but in all of ZenoTravel). The first set of problems concern a rendezvous at the center city in the location graph depicted in Figure 3; the optimal plan arranges a rendezvous at the center city. The second set of problems is to swap the positions of passengers located at the endpoints of a chain of cities.

For thorough empirical analysis of cost issues in standard benchmarks see (Richter and Westphal 2010).

### 5.1 LAMA

In this section we demonstrate the performance problem wrought by  $\epsilon$ -cost in a state-of-the-art (2008) planner — LAMA (Richter and Westphal 2010), the leader of the cost-sensitive (satisficing) track of IPC’08 (Helmert, Do, and Refanidis 2008). With a completely trivial recompilation (set a flag) one can make it ignore the given cost function, effectively searching by  $f_s$ . With slightly more work one can do better and have it use  $\hat{f}_s$  as its evaluation function, *i.e.*, have the heuristic estimate  $\hat{d}$  and the search be size-based, but still compute costs correctly for branch-and-bound. Call this latter modification LAMA-size. Ultimately, the observation is that LAMA-size outperforms LAMA — an astonishing feat for such a trivial change in implementation.

Domain	LAMA	LAMA-size
Rendezvous	70.8%	83.0%
Elevators	79.2%	93.6%
Woodworking	76.6%	64.1%

Table 1: IPC metric on LAMA variants.

LAMA<sup>4</sup> defies analysis in a number of ways: *landmarks, preferred operators, dynamic evaluation functions, multiple open lists*, and *delayed evaluation*, all of which effect potential search plateaus in complex ways. Nonetheless, it is essentially a cost-based approach.

**Results.**<sup>5</sup> With more than about 8 total passengers, LAMA is unable to complete any search stage except the first (the greedy search). For the same problems, LAMA-size finds the same first plan (the heuristic values differ, but not the structure), but is then subsequently able to complete further stages of search. In so doing it sees marked improvement in cost; on the larger problems this is due only to finding better variants on the greedy plan. Other domains are included for broader perspective, woodworking in particular was chosen as a likely counter-example, as all the actions concern just one type of physical object and the costs are not wildly different. For the same reasons we would expect LAMA to out-perform LAMA-size in some cost-enhanced version of Blocksworld.

## 5.2 SapaReplan

We also consider the behavior of SapaReplan on the simpler set of problems.<sup>6</sup> This planner is much less sophisticated in terms of its search than LAMA, in the sense of being much closer to a straight up implementation of weighted A\* search. The problem is just to swap the locations of passengers located on either side of a chain of cities. A plane starts on each side, but there is no actual advantage to using more than one (for optimizing either of size or cost): the second plane exists to confuse the planner. Observe that smallest and cheapest plans are the same. So in some sense the concepts have become only superficially different; but this is just what makes the problem interesting, as despite this similarity, still the behavior of search is strongly affected by the nature of the evaluation function. We test the performance of  $\hat{f}_s$  and  $f_c$ , as well as a hybrid evaluation function similar to  $\hat{f}_s + f_c$  (with costs normalized). We also test hybridizing via tie-breaking conditions, which ought to have little effect given the rest of the search framework.

**Results.**<sup>7</sup> The size-based evaluation functions find better cost plans faster (within the deadline) than cost-based evaluation functions. The hybrid evaluation function also does

<sup>4</sup>Options: ‘fFILI’.

<sup>5</sup>New best plans for Elevators were found (largely by LAMA-size). The baseline planner’s score is 71.8% against the better reference plans.

<sup>6</sup>Except that these problems are run on all of ZenoTravel.

<sup>7</sup>The results differ markedly between the 2 and 3 city sets of problems because the sub-optimal relaxed plan extraction in the 2-cities problems coincidentally produces an essentially perfect heuristic in many of them. One should infer that the solutions found in the 2-cities problems are sharply bimodal in quality and that the meaning of the average is then significantly different than in the 3-cities problems.

Mode	2 Cities		3 Cities	
	Score	Rank	Score	Rank
Hybrid	88.8%	1	43.1%	2
Size	83.4%	2	43.7%	1
Size, tie-break on cost	82.1%	3	43.1%	2
Cost, tie-break on size	77.8%	4	33.3%	3
Cost	77.8%	4	33.3%	3

Table 2: IPC metric on SapaReplan variants in ZenoTravel.

relatively well, but not as well as could be hoped. Tie-breaking has little effect, sometimes negative.

We note that Richter and Westphal (2010) also report that replacing cost-based evaluation function with a pure size-based one improves performance over LAMA in multiple other domains. Our version of LAMA-size uses a cost-sensitive size-based search ( $\hat{h}_s$ ), and our results, in the domains we investigated, seem to show bigger improvements over the size-based variation on LAMA obtained by *completely* ignoring costs ( $h_s$ , *i.e.*, setting the compilation flag). Also observe that one need not accept a tradeoff: calculating  $\log_{10} \epsilon^{-1} \leq 2$  (3? 1.5?) and choosing between LAMA and LAMA-size appropriately would be an easy way to improve performance simultaneously in ZenoTravel (4 orders of magnitude) and Woodworking (< 2 orders of magnitude).

Finally, while LAMA-size outperforms LAMA, our theory of  $\epsilon$ -cost traps suggests that cost-based search should fail even more spectacularly. In the appendix, we take a much closer look at the travel domain and present a detailed study of which extensions of LAMA help it temporarily mask the pernicious effects of cost-based search. Our conclusion is that both LAMA and SapaReplan manage to find solutions to problems in the travel domain despite the use of a cost-based evaluation function by using various tricks to induce a limited amount of *depth-first behavior* in an A\*-framework. This has the potential effect of delaying exploration of the  $\epsilon$ -cost plateaus slightly, past the discovery of a solution, but still each planner is ultimately trapped by such plateaus before being able to find really good solutions. In other words, such tricks are mostly serving to mask the problems of cost-based search (and  $\epsilon$ -cost), as they merely delay failure by just enough that one can imagine that the planner is now effective (because it returns a solution where before it returned none). Using a size-based evaluation function more directly addresses the existence of cost plateaus, and not surprisingly leads to improvement over the equivalent cost-based approach — even with LAMA.

## 6 Conclusion

The practice of combinatorial search in automated planning is (often) *satisficing*. There is a great call for deeper theories of satisficing search (*e.g.*, a formal definition agreeing with practice is a start), and one perhaps significant obstacle in the way of such research is the pervasive notion that perfect problem solvers are the ones giving only perfect solutions. Actually implementing cost-based, systematic, combinatorial, search reinforces this notion, and therein lies its greatest harm. (Simply defining search as if “strictly positive edge weights” is good enough in practice is also harmful.)

In support of the position we demonstrated the technical difficulties arising from such use of a cost-based evaluation

function, largely by arguing that the size-based alternative is a notably more effective default strategy. We argued that using cost as the basis for plan evaluation is a purely exploitative/greedy perspective, leading to least interruptible behavior. Being least interruptible, it follows that implementing cost-based search will typically be immediately harmful to that particular application. Exceptions will abound, for example if costs are *not* wildly varying. The lasting harm, though, in taking cost-based evaluation functions as the default approach, failing to document any sort of justification for the risk so taken, is in reinforcing the wrong definition of *satisficing* in the first place. In conclusion, as a rule: Cost-based search is harmful.

## References

- Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *AIJ* 81–138.
- Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-value plateaus: A challenge for planning. In *ICAPS*. AAAI Press.
- Bonet, B. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS*, 12–21.
- Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A\*. *ACM* 32(3):505–536.
- Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.
- Dijkstra, E. W. 1968. Go to statement considered harmful. *Commun. ACM* 11:147–148.
- Edelkamp, S., and Kissmann, P. 2009. Optimal symbolic planning with action costs and preferences. In *IJCAI*, 1690–1695.
- Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.
- Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1972. Correction to “a formal basis for the heuristic determination of minimum cost paths”. *SIGART* 28–29.
- Helmert, M., and Röger, G. 2008. How good is almost perfect. In *AAAI*, 944–949.
- Helmert, M.; Do, M.; and Refanidis, I. 2008. The deterministic track of the international planning competition. <http://ipc.informatik.uni-freiburg.de/>.
- Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.
- Kearns, M.; Capital, S.; and Singh, S. 1998. Near-optimal reinforcement learning in polynomial time. In *Machine Learning*, 260–268. Morgan Kaufmann.
- Kocsis, L., and Szepesvari, C. 2006. Bandit based monte-carlo planning. In *ECML*.
- Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *AIJ* 27:97–109.
- McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, 569–576.
- Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley.
- Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI*.
- Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.
- Sanner, S.; Goetschalckx, R.; Driessens, K.; and Shani, G. 2009. Bayesian real-time dynamic programming. In *IJCAI*.
- Streeter, M., and Smith, S. F. 2007. New techniques for algorithm portfolio design. In *UAI*.
- Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *ICAPS*. AAAI Press.
- Thayer, J., and Ruml, W. 2010. Finding acceptable solutions faster using inadmissible information. In *Proceedings of the International Symposium on Combinatorial Search*.
- Zilberstein, S. 1998. Satisficing and bounded optimality. In *AAAI Spring Symposium on Satisficing Models*.

## A Deeper Analysis of the Results in Travel Domain

In this section we analyze the reported behavior of LAMA and SapaReplan in greater depth. We begin with a general analysis of the domain itself and the behavior of (simplistic) systematic state-space search upon it, concluding that cost-based methods suffer an enormous disadvantage. The empirical results are not nearly so dramatic as the dire predictions of the theory, or at least do not appear so. We consider to what extent the various additional techniques of the planners (violating the assumptions of the theory) in fact mitigate the pitfalls of  $\epsilon$ -cost, and to what extent these only serve to mask the difficulty.

### A.1 Analysis of Travel Domain

We argue that search under  $f_c$  pays a steep price in time and memory relative to search under  $f_s$ . The crux of the matter is that the domain is reversible, so relaxation-based heuristics cannot penalize fruitless or even counter-productive passenger movements by more than the edge-weight of that movement. Then plateaus in  $g$  are plateaus in  $f$ , and the plateaus in  $g_c$  are enormous.

First note that the domain has a convenient structure: The global state space is the product of the state space of shuffling planes around between cities/airports via the fly action (expensive), and the state space of shuffling people around between (stationary) planes and cities/airports via the board/debark actions (cheap). For example, in the rendezvous problems, there are  $5^4 = 625$  possible assignments of planes to cities, and  $(5 + 4)^{2k}$  possible assignments of passengers to locations (planes + cities), so that the global state space has exactly  $5^4 \cdot 9^{2k}$  reachable states (with  $k$  the number of passengers at one of the origins).<sup>8</sup>

Boarding and debarking passengers is extremely cheap, say on the order of cents, while flying planes between cities

<sup>8</sup>Fuel and zoom are distracting aspects of ZenoTravel-STRIPS, so we remove them. Clever domain analysis could do the same.



is quite a bit more expensive, say on the order of hundreds of dollars (from the perspective of passengers). So  $\frac{1}{\varepsilon} \approx 10000$  for this domain — a constant, but much too large to ignore.

To analyze state-space approaches in greater depth let us make all of the following additional assumptions: The heuristic is relaxation-based, imperfect, and in particular heuristic error is due to the omission of actions from relaxed solutions relative to real solutions. Heuristic error is not biased in favor of less error in estimation of needed fly actions — in this problem planes are *mobiles* and *containers* whereas people are only *mobiles*. Finally, there are significantly but not overwhelmingly more passengers than planes.

Then consider a child node, in plane-space, that is in fact the correct continuation of its parent, but the heuristic fails to realize it. So its  $f$  is higher by the cost or size of one plane movement: 1 under normalized costs. Moreover assume that moving passengers is not heuristically good (in this particular subspace). (Indeed, moving passengers is usually a bad idea.) Then moving a passenger increases  $f_c$  by at most  $2\varepsilon$  (and at least  $\varepsilon$ ), once for  $g_c$  and once for  $h_c$ . As  $\frac{1}{2\varepsilon} \approx 5000$  we have that search under  $f_c$  explores the passenger-shuffling space of the parent to, at least, *depth* 5000. Should the total heuristic error in fact exceed one fly action, then each such omission will induce backtracking to a further 5000 levels: for any search node  $n$  reached by a fly action set  $e_c(n) = f_c(x) - f_c(n)$  with  $x$  some solution of interest (set  $e_s$  similarly). Then if search node  $n$  ever appears on the open list it will have its passenger-shuffling subspace explored, under  $f_c$ , to at least depth  $e_c \cdot 5000$  before  $x$  is found (and at most depth  $e_c \cdot \frac{1}{\varepsilon}$ ). Under  $\hat{f}_s$ , we have instead exploration up to at least depth  $e_s \cdot \frac{1}{2}$  and at most depth  $e_s \cdot \frac{1}{4}$ .

As 5000 objects is already far above the capabilities of any current domain-independent planners, we can say that at most plane-shuffling states considered, cost-based search *exhausts* the entire associated passenger-shuffling space during backtracking. That is, it stops exploring the space due to exhausting finite possibilities, rather than by adding up sufficiently many instances of  $2\varepsilon$  increases in  $f$  — the result is the same as if the cost of passenger movement was 0. Worse, such exhaustion commences immediately upon backtracking for the first time (with admissible heuristics). Unless *very* inadmissible (large heuristic weights), then even with inadmissible heuristics, still systematic search should easily get trapped on cost plateaus — before finding a solution.

In contrast, size-based search will be exhausting only those passenger assignments differing in at most  $e_s$  values; in the *worst* case this is equivalent to the cost-based method, but for good heuristics is a notable improvement. (In addition the size-based search will be exploring the plane-shuffling space deeper, but that space is [assumed to be] much smaller than any single passenger-shuffling space.) Then it is likely the case that cost-based search dies before reporting a solution while size-based search manages to find one or more.

## A.2 Analyzing LAMA’s Performance

While LAMA-size out-performs LAMA, it is hardly as dramatic a difference as predicted above. Here we analyze the results in greater depth, in an attempt to understand how LAMA avoids being immediately trapped by the passenger-

shuffling spaces. Our best, but not intuitive, explanation is its pessimistic delayed evaluation leads to a temporary sort of depth-first bias, allowing it to skip exhaustion of many of the passenger-shuffling spaces until *after* finding a solution. So, (quite) roughly, LAMA is able to find one solution, but not two.

**Landmarks.** The passenger-shuffling subspaces are search plateaus, so, the most immediate hypothesis is that LAMA’s use of landmarks helps it realize the futility of large portions of such plateaus (*i.e.*, by pruning them). However, LAMA uses landmarks only as a heuristic, and in particular uses them to order an additional (also cost-based) open list (taking every other expansion from that list), and the end result is actually greater breadth of exploration, not greater pruning.

**Multiple Open Lists.** Then an alternative hypothesis is that LAMA avoids immediate death by virtue of this additional exploration, *i.e.*, one open list may be stuck on an enormous search plateau, but if the other still has guidance then potentially LAMA can find solutions due to the secondary list. In fact, the lists interact in a complex way so that conceivably the multiple-list approach even allows LAMA to ‘tunnel’ out of search plateaus (in either list, so long as the search plateaus do not coincide). Indeed the secondary list improves performance, but turning it off still did not cripple LAMA in our tests (unreported), let alone outright kill it.

**Small Instances.** It is illuminating to consider the behavior of LAMA and LAMA-size with only 4 passengers total; here the problem is small enough that optimality can be proved. LAMA-size terminates in about 12 minutes. LAMA terminates in about 14.5 minutes. Of course the vast majority of time is spent in the last iteration (with heuristic weight 1 and all actions considered) — and both are unrolling the exact same portion of state space (which is partially verifiable by noting that it reports the same number of unique states in both modes). There is only one way that such a result is at all possible: the cost-based search is re-expanding many more states. That is difficult to believe; if anything it is the size-based approach that should be finding a greater number of suboptimal paths before hitting upon the cheapest. The explanation is two-fold. First of all pessimistic delayed evaluation leads to a curious sort of depth-first behavior. Secondly, cost-based search pays far more dearly for failing to find the cheapest path first.

**Delayed Evaluation.** LAMA’s delayed evaluation is not equivalent to just pushing the original search evaluation function down one level. This is because it is the *heuristic* which is delayed, not the full evaluation function. LAMA’s evaluation function is the sum of the *parent’s* heuristic on cost-to-go and the *child’s* cost-to-reach:  $f_L(n) = g(n) + h(n.p.v)$ . One can view this technique, then, as a transformation of the original heuristic. Crucially, the technique increases the inconsistency of the heuristic. Consider an optimal path and the perfect heuristic. Under delayed evaluation of the perfect heuristic, each sub-path has an  $f_L$ -value in excess of  $f^*$  by exactly the cost of the last edge. So a high cost edge followed by a low cost edge demonstrates the non-monotonicity of  $f_L$  induced by the inconsistency wrought by delayed evaluation. The problem with non-monotonic evaluation functions is not the decreases *per se*, but the increases that precede them. In this case, a low cost edge followed by a high cost edge along an optimal path *induces backtracking*

despite the perfection of the heuristic prior to being delayed. **Depth-first Bias.** Consider some parent  $n$  and two children  $x$  and  $y$  ( $x.p = n$ ,  $y.p = n$ ) with  $x$  reached by some cheap action and  $y$  reached by some expensive action. Observe that siblings are always expanded in order of their cost-to-reach (as they share the same heuristic value), so  $x$  is expanded before  $y$ . Now, delaying evaluation of the heuristic was pessimistic:  $h(x.v)$  was taken to be  $h(n.v)$ , so that it appears that  $x$  makes no progress relative to  $n$ . Suppose the pessimism was unwarranted, for argument’s sake, say entirely unwarranted:  $h(x.v) = h(n.v) - c(x.a)$ . Then consider a cheap child of  $x$ , say  $w$ . We have:

$$\begin{aligned} f_L(w) &= g(w) + h(x.v), \\ &= g(x) + c(w.a) + h(n.v) - c(x.a), \\ &= f_L(x) - c(x.a) + c(w.a), \\ &= f(n) + c(w.a), \end{aligned}$$

so in particular,  $f_L(w) < f_L(y)$  because  $f(n) + c(w.a) < f(n) + c(y.a)$ . Again suppose that  $w$  makes full progress towards the goal (the pessimism was entirely unwarranted), so  $h(w.v) = h(x.v) - c(w.a)$ . So any of its cheap children, say  $z$ , satisfies:

$$\begin{aligned} f_L(z) &= g(w) + c(z.a) + h(x.v) - c(w.a), \\ &= f_L(w) - c(w.a) + c(z.a), \\ &= f_L(x) - c(x.a) + c(w.a) - c(w.a) + c(z.a), \\ &= f_L(x) - c(x.a) + c(z.a), \\ &= f(n) + c(z.a). \end{aligned}$$

Inductively, any low-cost-reachable descendant, say  $x'$ , that makes full heuristic progress, has an  $f_L$  value of the form  $f(n) + c(x'.a)$ , and in particular,  $f_L(x') < f_L(y)$ , that is, all such descendants are expanded prior to  $y$ .

Generalizing, any low-cost-reachable and not heuristically bad descendant of any cheaply reachable child ( $x$ ) is expanded prior to any expensive sibling ( $y$ ).<sup>9</sup> Call that the good low-cost subspace.

Once such an expensive sibling is finally expanded (and the cost is found to be justified by the heuristic), then its descendants can start to compete on even footing once more. Except for the good low-cost subspaces: the good low-cost subspace of  $x$  is entirely expanded prior to the good low-cost subspace of  $y$ . In practice this means that LAMA is quite consistent about taking all promising low cost actions immediately, globally, like boarding all passengers in a problem, rather than starting some fly action halfway through a boarding sequence.

Then LAMA exhibits a curious, temporary, depth-first behavior initially, but in the large exhibits the normal breadth-first bias of systematic search. Depth-first behavior certainly results in finding an increasingly good sequence of plans to the same state. In this case, at every point in the best plan to some state where a less-expensive sibling leads to a slightly worse plan to the same state is a point at which LAMA finds a worse plan first. The travel domain is very strongly connected, so there are many such opportunities, and so we have a reasonable explanation for how LAMA could possibly be

re-expanding more states than LAMA-size in the smallest instances of the travel domain.

**Overhead.** Moreover, the impact of failing to find the right plan first is quite distinct in the two planners. Consider two paths to the same plane-shuffling state, the second one actually (but not heuristically) better. Then LAMA has already expanded the vast majority, if not the entirety, of the associated passenger-shuffling subspace before finding the second plan. That entire set is then re-expanded. The size-based approach is not compelled to exhaust the passenger-shuffling subspaces in the first place (indeed, it is compelled to backtrack to other possibilities), and so in the same situation ends up performing less re-expansion work within each passenger-shuffling subspace. Then even if the size-based approach is overall making more mistakes in its use of planes (finding worse plans first), which is to be expected, the price per such mistake is notably smaller.

**Summary.** LAMA is out-performed by LAMA-size, due to the former spending far too much time expanding and re-expanding states in the  $\varepsilon$ -cost plateaus. It fails in “depth-first” mode: finding not-cheapest almost-solutions, exhausting the associated cheap subspace, backtracking, finding a better path to the same state, re-exhausting that subspace, . . . , in particular exhausting memory extremely slowly (it spends all of its time re-exhausting the *same* subspaces).

### A.3 Analyzing the Performance of SapaReplan

The contrasting failure mode, “breadth-first”, is characterized by exhausting each such subspace as soon as it is encountered, thereby rapidly exhausting memory, without ever finding solutions. This is largely the behavior of SapaReplan (which does eager evaluation), with cost-based methods running out of memory (much sooner than the deadline, 30 minutes) and size-based methods running out of time. So for SapaReplan it is the size-based methods that are performing many more re-expansions, as in a much greater amount of time they are failing to run out of memory. From the results, these re-expansions must be in a useful area of the search space.

In particular it seems that the cost-based methods must indeed be exhausting the passenger-shuffling spaces more or less as soon as they are encountered — as otherwise it would be impossible to both consume all of memory yet fail to find better solutions. (Even with fuel there are simply too few distinct states modulo passenger-shuffling.) However, they do find solutions before getting trapped, in contradiction with theory.

The explanation is just that the cost-based methods are run with large (5) heuristic weight, thereby introducing significant depth-first bias (but not nearly so significant as with pessimistic delayed evaluation), so that it is possible for them to find a solution before attempting to exhaust such subspaces. It follows that they find solutions within seconds, and then spend minutes exhausting memory (and indeed that is what occurs). The size-based methods are run with small heuristic weight (2) as they tend to perform better in the long run that way. It would be more natural to use the same heuristic weight for both types, but, the cost-based approaches do conform to theory with small heuristic weights — producing no solutions, hardly an interesting comparison.

<sup>9</sup>The bound on heuristic badness is  $c(y.a)$ .