Cost-Based Heuristic Search Is Sensitive to the Ratio of Operator Costs

Christopher Wilt and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
{*wilt*, *ruml*} at *cs.unh.edu*

Abstract

In many domains, different actions have different costs. In this paper, we show that various kinds of best-first search algorithms are sensitive to the ratio between the lowest and highest operator costs. First, we take common benchmark domains and show that when we increase the ratio of operator costs, the number of node expansions required to find a solution increases. Second, we provide a theoretical analysis showing one reason this phenomenon occurs. We also discuss additional domain features that can cause this increased difficulty. Third, we show that searching using distance-to-go estimates can significantly ameliorate this problem. Our analysis takes an important step toward understanding algorithm performance in the presence of differing costs. This research direction will likely only grow in importance as heuristic search is deployed to solve real-world problems.

Introduction

In real world domains, different actions rarely all have the same cost. For example, in the logistics domain, the cost of moving a package from one vehicle to another is very small compared to the cost of moving the vehicle, and the cost of moving different kinds of vehicles can vary by more than an order of magnitude. Other examples include traversing a map where there are different kinds of terrain, or a manufacturing domain where the cost of materials, machines, and labor can vary by orders of magnitude. In robotic motion planning, sometimes large macro-actions are considered in addition to the base actions (Likhachev and Ferguson 2009), but these macro actions have a very high cost compared to the base ones. Despite this reality, many of the common benchmark domains used to validate search algorithms (sliding tile puzzle, pancake puzzle, topspin puzzle, Rubik's cube, grid path planning, STRIPS planning, etc.) are all unit cost. To understand how heuristic search can be used on real-world problems, it is crucial to examine algorithm behavior on problem domains that exhibit a variety of action costs.

In this paper, we first present an empirical analysis that shows that, when costs are not uniform, problems become much more difficult for standard heuristic search algorithms. We then provide a theoretical analysis that demonstrates that a single error in the heuristic can cause A* (Hart, Nilsson, and Raphael 1968), weighted A* (Pohl 1970), and greedy search (Doran and Michie 1966) to all expand a number of nodes exponential in the ratio of the heuristic error to the smallest operator cost. We argue that this effect plays an important role in explaining the observation that non-unit cost domains are much more difficult to solve than a comparable unit cost domain, and we argue that this phenomenon will occur whenever the domain has low cost operators that can be applied from any node, and few duplicates. Last, we show that exploiting a heuristic that identifies short paths, as opposed to cheap paths, fares much better than basic best-first searches in domains with non-unit operators.

This work shows that the common strategy in search research of considering only unit-cost domains has an important weakness: algorithm behavior can be very different when actions have different costs. Our analysis takes a step toward understanding algorithm performance in the presence of differing costs, and demonstrates the effectiveness of recently-proposed distance-based methods in providing suboptimal solutions. This research direction will likely only grow in importance as heuristic search is increasingly deployed to solve real-world problems, which often feature a variety of costs.

Background

We define a search problem as the task of finding a least cost path through a directed weighted graph G where all weights are finite and strictly positive. If all edges in the graph have the same weight, we say the graph has *unit cost*. Otherwise, we say the graph has *non-unit cost*. In a graph G we define the *operator cost ratio* to be the ratio of the largest edge weight in G to the smallest edge weight in G.

For any node n in the graph, $succ(n)$ returns the successors of n in G. In the graph, some nodes are goal nodes, identified by a predicate $goal(n)$. For all nodes, $h^*(n)$ denotes the cost of a cheapest path in G from n to a node that satisfies the $goal$ predicate. Since it is generally computationally demanding to calculate h^* for any given node, we have a function $h(n)$ that accepts a node and returns an estimate of $h^*(n)$. We assume that this heuristic is admissible. An admissible heuristic is one that satisfies:

$$\forall n : h^*(n) \geq h(n) \tag{1}$$

$c(n, n')$ denotes the cost of a cheapest path from n to n'. A consistent heuristic is one that satisfies:

$$\forall n, n' : h(n) \leq c(n, n') + h(n') \quad (2)$$

A search problem is formally defined by the tuple containing G, a goal predicate, a $h()$ function and a node $start \in G$. Search algorithms like A*, weighted A*, and greedy search construct paths through the graph starting at the $start$ node, so any node n that has been encountered by the search algorithm has a known path from $start$ to n through G. The cost of this path is defined as $g(n)$.

Related Work

Other researchers have demonstrated that domains with high cost ratios can cause best-first search to perform poorly. The problem, in its most insidious form, was first identified by Benton et al. (2010). They discuss g value plateaus, a collection of nodes that are connected to one another, all with the same g value. Domains with g value plateaus have zero cost operators, so these domains have an infinite operator cost ratio. Such plateaus arise in temporal planning with concurrent actions and the goal of minimizing makespan, where extraneous concurrent actions can be inserted into a partially constructed plan without increasing its makespan, creating a group of connected nodes with the same g value. Benton et al. (2010) prove that when using either an optimal or a suboptimal search algorithm, large portions of g value plateaus have to be expanded, and they then demonstrate empirically that this phenomenon can be observed in modern planners. They propose a solution that uses a node evaluation function considering estimated makespan to go, makespan thus far, and estimated total time consumed by all actions, independent of any parallelism. The estimate of the total time without parallelism is weighted and added to the estimated remaining makespan and incurred makespan to form the final node evaluation function, which is then tested empirically in a planner, which performs well on the planning domains they discuss. Benton et al. discuss the effects of zero cost operators, but do not discuss problems associated with small cost operators.

Cushing, Benton, and Kambhampati (2010) argue that cost based search performs poorly when the ratio of the largest operator cost to the smallest operator cost is large, and that cost-based search can take prohibitively long to find solutions when this occurs. They first argue that it is straightforward to construct domains in which cost-based search performs extremely poorly simply by having a very low cost operator that is always applicable, but does not immediately lead to a goal. They empirically demonstrate this phenomenon in a travel domain where passengers have to be moved around by airplanes, but boarding is much less expensive than flying the airplane. They provide empirical evidence using planners to demonstrate that the presence of low cost operators causes traditional best-first search to perform poorly, and show that best-first searches that use plan size or hybridized size-cost metrics to rank nodes performs much better as compared to when using cost alone. Their analysis of how the heuristic factors into the analysis is limited to an empirical analysis of two planning systems. This work invited questions about the theoretical underpinnings of precisely why the range of operator costs was causing such a huge problem, and also questions about why the presence of an informed heuristic was not ameliorating the problem.

Problem	Expansions
Unit Pancake	69
Sum Pancake	3,711

Table 1: A* expansions on different problems

Concurrently with our work, Cushing, Benton, and Kambhampati (2011) extended their previous analysis, discussing examples of domains that exhibit the problematic operator cost ratios. Once again, their theoretical analysis is almost exclusively confined to uniform cost search, and does not consider the mitigating effects an informed heuristic can have, although they speculate that heuristics can help improve the bound, but not asymptotically. Their consideration of heuristics is purely empirical, limited to extensive analysis of the performance of two planning systems, discussing how the various components of the planners deal with a wide variety of operator costs, and either exasperate or mitigate the underlying problem.

In this paper, we extend this previous work in two ways. First, we illustrate the problem empirically in a number of standard benchmark domains from the search literature. Second we deepen the theoretical underpinnings of the problem associated with a wide range of operator costs. We do this by showing that, given a heuristic with a bound on its change between any parent and child node, the number of extra nodes introduced into the search by a heuristic error will be exponential, and that this can only be mitigated by change in the branching factor (e.g. branching factor is not uniform) or the detection of duplicates; the heuristic cannot rise sufficiently quickly to change the underlying asymptotic complexity. We also show that the property of bounded change of heuristic error arises in a number of common robotics heuristics, as well as any domain with a consistent heuristic and invertible operators.

Difficulty of High Cost Ratio Domains

The first step in our analysis is to show that, empirically, non-unit cost domains are more difficult than their unit-cost counterparts. In order to demonstrate the difficulty associated with non-unit cost edges, we consider modifications of two common benchmark domains: the sliding tile puzzle and the pancake puzzle. We do this to maintain the same connectivity and heuristic.

We define failure of a search to be exhausting main memory, which is 8 GB on our machines, without finding a solution.

The pancake puzzle consists of an array of numbers, and the array must be sorted by flipping a prefix of the array. In the standard variety of the pancake puzzle, each flip costs 1. In our sum pancake puzzle, the cost of each flip is determined by adding up the values contained in the prefix to be flipped. It is unclear how to effectively adapt the gap

Cost Function	Failure Rate	95% Conf
Unit	21%	7.9%
Cost = $\sqrt{face}$	48%	9.6%
Cost = $\frac{1}{face}$	66%	9.5%

Table 2: Behavior of A* on tile puzzles with different cost functions

heuristic (Helmert 2010) to the non-unit puzzle, so we use a pattern database (Culberson and Schaeffer 1998) for both varieties of pancake puzzle to make sure the heuristics used are comparable to one another. We consider a 10 pancake problem with a pattern database that tracks 7 pancakes, in order to make sure that A* is able to solve both the unit and the non-unit cost versions of the problem. We considered 100 randomly generated problems, and solved the same set of problems with both cost functions. With the sum cost function, the cost of a move ranges from 3 to 55, giving a ratio of 18.3. As can be seen in Table 1, the non-unit sum cost function requires more than 50 times the number of expansions than its unit cost cousin.

The sliding tile puzzle we consider is a 4x4 grid with a number, 1-15, or blank in each slot. A move consists of swapping the position of the blank with a tile next to it, either up, down, left, or right, with the restriction that all tiles must remain on the 4x4 grid. In the standard sliding tile puzzle, each move costs 1, but we also consider variants where the cost of doing a move is related to the face of the tile. We use the Manhattan distance heuristic for all puzzles. For the non-unit puzzles, the standard Manhattan distance heuristic is weighted to account for the different operator costs while retaining both admissibility and consistency. We use the 100 instances published by Korf (1985). As can be seen in Table 2, as we increase the ratio of operator costs from 1 in Unit to 3.9 in square root tiles to 15 in inverse tiles, the problems get more difficult. These differences are statistically highly significant.

We have seen that a simple modification to the cost function of two standard benchmark domains makes the problems require significantly more computational effort to solve. These examples establish the fact that non-unit cost domains can be much more difficult as compared to a unit cost domain with the same state space and connectivity.

Localized Heuristic Error

In this section, we consider one possible explanation for the difficulty of high cost ratio problems. The problem occurs when the amount the heuristic can change across a transition is bounded. We show that this condition holds when we have a consistent heuristic in a domain with invertible operators, but we argue that some heuristics have this property even if the operators in the domain are not invertible. We use this bound on the rate at which h can change to prove bounds on the rate at which error in the heuristic can change. We then apply these results to draw conclusions about the size of heuristic depressions in non-unit cost domains.

Felner et al. (2011) show that in a graph with invertible operators, any consistent heuristic will satisfy:

$$\forall n, s : \Delta_h(n, s) = |h(n) - h(s)| \leq c(n, s) \tag{3}$$

This is one method of establishing a bound on how $\Delta_h(n, c)$ can change along a path. Note that having reversible operators is not the only way in which a bound on Δ_h can be established. Some heuristics inherently have this property. For example, in the dynamic robot motion planning domain, operators are often not reversible. In this domain, the usual heuristic is calculated by solving the problem without any dynamics. Without dynamics, operators can be reversed, so the rate at which the heuristic can change is bounded by the cost of the transition.

Establishing a bound on Δ_h allows us to establish properties about not only the heuristic error, but also about the behavior of search algorithms that rely upon the heuristic for guidance.

Bounds on the rate at which h can change have an important consequence for how the error in $h(n)$, defined as $\epsilon(n) = h^*(n) - h(n)$, can change across any transition.

Theorem 1. *In any domain where $\Delta_h(n, s) \leq c(n, s)$ holds for any pair of nodes n, s such that s is a successor of n:*

$$\Delta_\epsilon(n, s) = |\epsilon(n) - \epsilon(s)| \leq 2 \cdot c(n, s) \tag{4}$$

Proof. This equation can be rewritten as

$$|(h^*(n) - h(n)) - (h^*(s) - h(s))| \leq 2 \cdot c(n, s) \tag{5}$$

which itself can be rearranged to be

$$|(h^*(n) - h^*(s)) - (h(n) - h(s))| \leq 2 \cdot c(n, s) \tag{6}$$

The fact that h^* is monotone implies that the most h^* can change between two nodes n and s where s is a successor of n is $c(n, s)$. The statement of the theorem limits $h(n)$ to within $c(n, s)$ of $h(s)$. Since the change in both h and h^* is bounded by the $c(n, s)$, the difference between the $\epsilon(n)$ and $\epsilon(s)$ is at most twice $c(n, s)$. The 2 is necessary because $(h^*(n) - h^*(s)) = \Delta_{h*}$ and $(h(n) - h(s)) = -\Delta_h$ may have opposite signs. □

This bound Δ_ϵ places important restrictions on the creation of a local minimum in the heuristic. When the change in $h(n)$ is bounded and all nodes have similar cost, in order to accumulate a large heuristic error, the heuristic has to provide incorrect assessments for several transitions. For example, in order for a heuristic in a unit cost domain to have an error of 4, there must be at least two transitions where the heuristic does not change correctly; it is not possible for any single transition to add more than 2 to the total amount of error present in h.

In a non-unit cost domain, the error associated with a high cost operator can contribute much more to the overall heuristic error as compared to the error potentially contributed by a low cost operator. The result of this is that, in a domain with non-unit cost, heuristic error does not accumulate uniformly because the larger operators can contribute more to the heuristic error. This does not occur in domains with unit costs, because each operator can contribute exactly the same

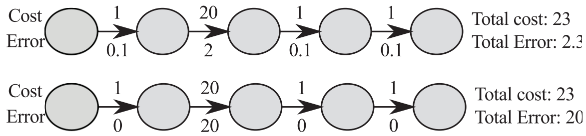

Figure 1: Examples of heuristic error accumulation along a path

amount to the heuristic error. A simple example of this phenomenon can be seen in the top part of Figure 1; all operators contribute the same percent to the total error, but due to size differences, a single large operator contributes almost all of the error in h. If we consider a different model of error where any operator can either add its size to the total error, or not contribute to the error, we have the example in the bottom part of Figure 1, where a single large operator can contribute so much error to a path that the effects of the other operators is insignificant. This can occur when, for example, an important aspect of the domain has been abstracted away in the heuristic. In either case, the large operators have the ability to contribute much more error than the smaller operators.

Consequences for Best-First Search

Any deviation from $h^*(n)$ in an expanded node can cause A* to expand extra nodes. In this section, we show that the number of extra nodes is exponential in the ratio of the size of the error to the smallest cost operator.

Corollary 1. *In a domain where $\Delta_h(n,s) \leq c(n,s)$ with an operator of cost δ, if a node n is expanded by A* and all descendants of n have b applicable operators of cost δ, at least $b^{\epsilon(n)/2\cdot\delta}$ extra nodes will be introduced to the open list, unless there are duplicate nodes or dead ends to prematurely terminate the search within the subtree of descendants of n.*

Proof. Consider the situation where an A* search encounters a node n that has a heuristic error of size $\epsilon(n)$. This means that $g(n) + h(n) + \epsilon(n) = g_{opt}(goal)$. In order to expand the goal, all descendants s of n which have $f(s) \leq g_{opt}(goal)$ must be expanded.

We have assumed that the change in h going from the parent to the child is bounded by the cost of the operator used to get from the parent to child. We have also assumed that we can apply b operators with cost δ. Since b operators with cost δ can be applied, then we have the result that f can increase by at most 2δ per transition, at least while in the subtree consisting of operators of cost δ. Note that there can be additional nodes reachable by different operators, but these nodes are not counted by this theorem.

This bound on the change in f stems from the fact that $g(s)$ always increases from $g(n)$ by δ, and in the best case $h(s)$ will also increase by δ over $h(n)$. This means that in order to raise $f(s)$ to something higher than $g_{opt}(goal)$, we must apply a cost δ operator at least $\frac{\epsilon(n)}{2\cdot\delta}$ times to make it so that $f(s) > g_{opt}(goal)$.

The best case scenario is that the heuristic always rises by δ across each transition, along with g. If this happens, we can apply the δ cost operator $\frac{\epsilon(n)}{2\cdot\delta}$ times before $f(s) > g_{opt}(goal)$. If we assume there are no duplicate states, this produces a tree of size $b^{\epsilon(n)/2\cdot\delta}$ □

	2	1	3
4	5	6	7
8	9	10	11
12	13	15	14

?	?	?	?
?	?	?	?
?	?	?	
?	?	15	14

Figure 2: Left: 15 puzzle instance with a large heuristic minimum. Right: State in which the 14 tile can be moved.

The exact number of nodes that will be within the local minimum is more difficult to calculate. First, it might not always be possible to apply an operator with cost δ. If this is the case, then the local minimum might not have $b^{\epsilon(n)/\delta}$ nodes in it, because δ will be replaced by the cost of the smallest operator that is applicable. If the domain has small operators that are not always applicable, then this equation will overestimate the number of nodes that will be introduced to the open list.

In addition to that, the closed list can provide protection against expanding $b^{\epsilon(n)/\delta}$ nodes. If the local minimum does not have $b^{\epsilon(n)/2\cdot\delta}$ unique nodes in it, then the closed list will detect the duplicate states, allowing search to continue with a more promising area of the graph. This observation plays a critical role in determining whether cost-based search will perform well in domains with cycles.

An Example

We consider a sliding tile puzzle in which the cost of moving a tile is the value of the tile, or the face value of the tile squared. On the instance shown in the left part of Figure 2, the Manhattan distance heuristic will underestimate the cost of the root node by a very large margin. In order to get to the solution, the 14 and the 15 tiles have to switch places. In order for this to occur, one of the tiles has to move out of the way. Let n denote a configuration as in the right part of Figure 2, where we can move the 14 tile up 1 slot, to make room for the 15 tile, and the other tiles are arranged in any fashion. If we assume the optimal path involves first moving the 14 out of the way (as opposed to first moving the 15 out of the way), this node, or one like it, must eventually make its way to the front of the open list. Let s_{14} denote the node representing the state where we have just moved the 14 tile. If we have unit cost, $f(s_{14}) - 2 = f(n)$, since $g(s_{14}) = g(n)+1$ and $h(s_{14}) = h(n)+1$. If cost is proportional to the tile face, then we have $f(s_{14}) - 28 = f(n)$, since $g(s_{14}) = g(n) + 14$ and $h(s_{14}) = h(n) + 14$. If cost is proportional to the square of the tile face, then we have $f(s_{14}) - 392 = f(n)$, since $g(s_{14}) = g(n) + 14^2$ and $h(s_{14}) = h(n) + 14^2$, and $2 \cdot 14^2 = 392$. Since s_{14} is along the optimal path, eventually it must be expanded by A*. Unfortunately, in order to get this node to the front of the open list A* must first expand enough nodes such that the minimum f on the open list is $f(n) + \{2, 28, 392\}$, where the appropriate value depends upon the cost function under

Cost (ratio)	Algorithm	Exp	Cost	Length
Unit (1:1)	A*	76,599	28	28
	WA*(3)	19,800	34	34
	Greedy	881	168	168
Face (1:15)	A*	482,948	210	28
	WA*(3)	94,301	314	48
	Greedy	36,932	956	180
Face^2 (1:225)	A*	3,575,939	DNF	DNF
	WA*(3)	1,699,265	2,884	56
	Greedy	1,156,044	6,272	94

Table 3: Difficulty of solving the puzzle from Figure 2

consideration.

The core problem is that raising the f value of the head of the open list in non-unit domains can require expanding a very large number of nodes due to the low cost operators. We can observe this phenomenon empirically by considering Table 3, where we can see that the more we vary the operator costs, more nodes must be expanded to get out of the local minimum associated with the root node.

Suboptimal Cost-Based Search

One way to scale A* to difficult problems is to relax the admissibility criterion associated with A* by using either weighed A* (Pohl 1970) or greedy search ($f(n) = h(n)$) (Doran and Michie 1966). In Figure 2, any solution has to move either the 14 or the 15 tile from its initial location, which will make the heuristic larger. Let the collection of nodes such that either the 14 or the 15 tile has been moved be denoted by $\{n_{lm}\}$, since one of these nodes must be included in any path to a solution. In greedy search, in order to bring one of the nodes in $\{n_{lm}\}$ to the front of the open list, all nodes that have a lower h value have to be removed from the open list. In particular, the siblings of the node from $\{n_{lm}\}$ under consideration have to be expanded, as do all of their successors that have a lower h value. Since the sliding tiles domain has reversible operators and a consistent heuristic, by Theorem 1, Δ_h is bounded by the cost of the operator used to transition. The end result is that it could potentially take a very long time to get one of the nodes in $\{n_{lm}\}$ to the front of the open list if it is always possible to apply a low cost operator.

As can be seen in Table 3, when faced with a problem where there is a landmark node where the heuristic increases, both weighted A* and greedy search must expand significantly more nodes when the operator costs in the domain vary, and this number increases as the range of operator costs increases.

When does this Problem Arise?

We have shown that in certain domains, the A* search algorithm has to expand a very high number of nodes as compared to a identical search space with a different cost function. In addition to that, we have shown that weighted A* and greedy search are not immune to this problem. A natural question to ask is how to predict when such problems will

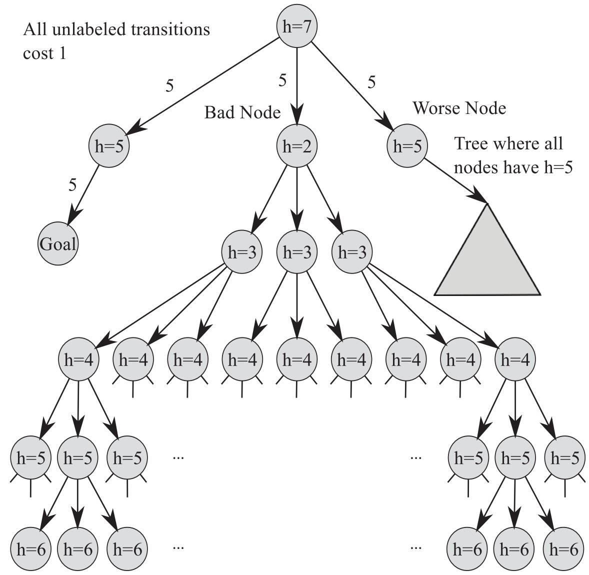

Figure 3: Greedy search and heuristic error

occur. For A*, we use two primary features to predict when this phenomenon will be observed. The first feature is that the ratio of the highest cost operator to the smallest cost operator is high. Since error in h accumulates along a path proportionately to operator cost, the presence of extraordinarily large operators means it is possible for the error to make very large jumps. This allows the ratio of heuristic error to the smallest operator cost to rise quickly, possibly with the application of a single very large operator. The second feature is that the rate at which h can change is bounded, which bounds how quickly the search algorithm can recover from the accumulated error in h when applying low cost operators. These two features combine to introduce an exponential number of nodes onto the open list that A* must expand in order to terminate.

The situation with greedy search is slightly different. With greedy search, the only extra expansions are associated with vacillating. Vacillation can occur whenever the children of a node all have higher h value than the head of the open list, but this principle can be applied recursively to all descendants. As can be seen in Figure 3, greedy search pays by expanding an exponential number of nodes for the error of 3 associated with the node labeled “Bad Node”. This is because “Bad Node” should not be at the front of the open list, but because of a deceptively low h, it is. In order to recover from this error, the descendants of “Bad Node” all have to be expanded in order to be cleared from the open list. Note that the heuristic below “Bad Node” is behaving as favorably as possible, rising as much as possible at each transition. It could also be the case that h stays the same, as is the case with “Worse Node”. Beneath this node there is an arbitrarily large tree where the heuristic does not change. In this example, greedy search can be made to expand an arbitrarily large number of nodes before discovering the goal, establishing that the exponential growth in the size of the heuristic error is a lower bound, subject to modification only by variation in the branching factor, operator applicability or the closed list.

With weighted A* both h and g can contribute to mitigating the error from h. Even in the best case scenario there are still an exponential number of nodes in the tree under the "Bad Node" because with $b^{\epsilon/(2\cdot w+1)\cdot\delta}$ nodes this tree is still exponential with respect to $\frac{\epsilon}{\delta}$, although the error term is mitigated by the weight w.

The fact that this tree is non-unit is crucial. The error in h increases by 5 when traversing the link from the root to "Bad Node", but the only reason the error is able to increase by this much is because the transition cost is 5 and not 1.

A Solution: Search Distance

We have observed that cost-based best-first searches can easily become mired in an irrelevant sub-graph if faced with a high cost operator whose effects were incorrectly tabulated by the heuristic, or if the heuristic happens to take an incorrect value for whatever reason. This is a general problem, but it is exacerbated in domains where the operator costs vary significantly.

Following Cushing, Benton, and Kambhampati (2010), we argue that searches that consider distance to go estimates fare much better because their consideration of distance typically puts a tighter limit on how far they descend into local minima. The algorithm proposed by Benton et al. (2010) uses two quantities, $h_m(n)$ and $h_c(n)$. $h_m(n)$ is an estimate of the remaining makespan, and $h_c(n)$ is an estimate of the time consumed by all actions that must be incorporated into the plan assuming zero parallelism. The problems we are considering do not have makespan, so as originally proposed this algorithm does not apply.

Thayer, Ruml, and Kreis (2009) and Thayer and Ruml (2011) investigate algorithms that exploit $d(n)$ to speed up search. $d(n)$ is a heuristic that provides an estimate of the number of nodes between the current node and the goal node. The exact semantics of $d(n)$ vary. It can either be the number of nodes along the path to the closest goal, or the number of nodes along the path to the cheapest goal. We consider $d(n)$ to be the estimated distance between the current node and the cheapest goal reachable from that node. $d(n)$ can be calculated in exactly the same way as $h(n)$ would be if the domain had unit cost. For example, in the sliding tile puzzle the ordinary Manhattan distance heuristic can serve as $d(n)$.

We consider six algorithms, four of which make use of a $d(n)$ heuristic. The first algorithm we consider is greedy search, where nodes are expanded in $h(n)$ order. Second, we consider weighted A*, where nodes are expanded in $f'(n) = g(n) + w \cdot h(n)$ order. The third algorithm we consider expands nodes in $d(n)$ order. We call this method Speedy. Since we elect to drop duplicate states in order to further speed things up, we call the specific variant used here Speedier (Thayer, Ruml, and Kreis 2009). Fourth, we consider a breadth-first beam search that orders nodes on $d(n)$. Fifth we consider Explicit Estimation Search (EES), which is a bounded suboptimal algorithm that uses inadmissible estimates of $d(n)$ and $h(n)$ to guide search, as well as an admissible $h(n)$ that is used to prove the quality bound. A detailed description of the algorithm is given by Thayer and Ruml (2011). The last algorithm we consider is Skeptical search (Thayer, Dionne, and Ruml 2011), an algorithm that orders nodes on $\hat{f}(n)' = g(n) + w \cdot \hat{h}(n)$, where $\hat{h}(n)$ is an improved, but inadmissible, estimate of $h^*(n)$, as its evaluation function, using the admissible $h(n)$ to prove the bound on the initial solution.

We show that searching on $d(n)$ leads to very fast solutions, but there is a cost: as would be expected, solutions found disregarding all cost information are of very poor quality. Fortunately, the bounded suboptimal searches that leverage both distance and cost information are able to provide high quality solutions very quickly.

Domain	Algorithm	Expansions	Cost
Tiles	Greedy/Speedier	2,117	519
	WA* 3	13,007	74
	$d(n)$ beam (50)	4,207	90
	EES 3	6,816	85
	Skeptical 3	5,620	84
Pancake	Greedy/Speedier	26,174	7,532
	WA* 3	18,225	15
	$d(n)$ beam (50)	11,940	241
	EES 3	12,358	16
	Skeptical 3	15,640	15

Table 4: Algorithms on unit cost domains

Empirical Results

Table 4 shows the results for the unit-cost version of the sliding tile and pancake puzzles. Since h and d are the same in a unit cost domain, greedy and speedier are the same. On the 15 puzzle, we used the same instances as before. For the pancake puzzle, we used 100 randomly generated 14 pancake problems, using a 7 pancake pattern database as a heuristic. We can see that when we make all moves cost the same, weighted A* performs about the same as EES and skeptical, and there is no single pareto dominant algorithm.

Sliding Tiles

We consider the variant of the sliding tile puzzle where the cost of moving a tile is the face value of the tile raised to the third power, and use the same instances used by Korf (1985). CPU time refers to the average CPU time needed to solve a problem. Solution cost refers to the average cost of the solution found by an algorithm. DNF denotes that the algorithm failed to find a solution for one or more instances. As can be seen in Table 5, with this cost function, speedier is the clear winner in terms of time to first solution, but it lags badly in terms of solution quality. EES and skeptical search, on the other hand, are able to find high quality solutions. Interestingly, the most successful algorithm we were able to find for this domain is a breadth-first beam search that orders nodes on $d(n)$ where ties are broken in favor of nodes with small $g(n)$. We believe this is due to the fact that the

Algorithm	Parameter	CPU Time	Solution Cost
Greedy		DNF	DNF
Weighted A*	1-1000	DNF	DNF
Speedier		0.010	485,033
$d(n)$ beam	10	0.228	199,709
$d(n)$ beam	50	0.086	77,223
$d(n)$ beam	100	0.098	65,187
$d(n)$ beam	500	0.315	57,122
EES	100	0.037	87,682
EES	10	0.038	87,682
EES	3	46.836	74,162
Skeptical	100	0.770	85,030
Skeptical	10	1.084	80,027
Skeptical	3	6.917	68,171

Table 5: Solving the 4x4 face^3 sliding tile puzzle

Algorithm	Parameter	CPU Time	Solution Cost
Greedy		DNF	DNF
Weighted A*	1-1000	DNF	DNF
Speedier		0.679	508,131
$d(n)$ beam	10	36.666	100,507
$d(n)$ beam	50	1.810	8,191
$d(n)$ beam	100	1.642	5,184
$d(n)$ beam	500	1.777	1,905
EES	4	18.401	817
EES	10	1.619	934
EES	100	1.619	934
Skeptical	4	3.435	854
Skeptical	10	3.379	889
Skeptical	100	3.233	951

Table 6: Solving 14-pancake (cost = sum) problems

beam searches found short solutions in terms of path length, which happened to also correspond to low cost solutions. EES, Skeptical, speedier, and $d(n)$ beam search all solve the problem, but the most important fact to note from Table 5 is the fact that both weighted A* and greedy search were not able to solve the problem at all, showing that consideration of $d(n)$ is mandatory in this domain.

Sum Pancake Puzzle

For our evaluation on the pancake puzzle, we consider a 14 pancake problem where the pattern database contains information about seven pancakes, and the remaining seven pancakes are abstracted away. We used the same 100 randomly generated instances from the unit cost experiments. The 14 pancake problem was the largest problem we could consider because we were unable to easily generate a pattern database for a pancake problem that was any larger.

The results of running the selected algorithms on this problem can be seen in Table 6. Once again, we observe weighted A* and greedy search are unable to solve all problems. EES and skeptical are able to solve all instances with larger weights. We had to use a weight of 4 because the weight of 3 used in the previous domain proved to be too aggressive. Speedier is able to solve all instances once again, but this comes at a cost: extremely poor quality solutions. Beam search on $d(n)$ is able to provide complete coverage over the instances, but neither solution quality or time to solution are particularly impressive in this domain. Overall, $d(n)$ based searches perform very well in the pancake puzzle with sum costs. Again, the most important thing to note about this domain is that, although there was no clear winner able to pareto dominate the all other algorithms, weighted A* and greedy search were once again unable to solve all problems.

Algorithm	Parameter	Expansions	Solution Cost
Greedy		68,778	2,992,826
Weighted A*	3	148,407	2,556,569
Weighted A*	10	97,002	2,825,369
Weighted A*	100	71,704	2,904,727
Speedier		18,642	2,978,587
$d(n)$ beam	10-500	DNF	DNF
EES	3	113,413	2,721,338
EES	10	108,044	2,936,708
EES	100	108,044	2,936,708
Skeptical	3	165,146	2,514,215
Skeptical	10	127,079	2,603,531
Skeptical	100	117,056	2,630,606

Table 7: Grid Path Planning with Life Costs

Grid Path Planning

For grid path planning, we consider a variant of standard grid path planning where the cost to transition out of a cell is equal to the y coordinate of the cell, which has two benefits. First, there is a clear difference between the shortest solution and the cheapest solution. In addition, it allows us to simulate an environment in which a being in a certain part of the map is undesirable. The boards are 1,200 cells tall and 2,000 cells wide. 35% of the cells are blocked. Blocked cells are distributed uniformly throughout the space. In this domain, the size of a local minimum associated with any given node is bounded very tightly. For any node n in this problem, the descendants of n are all reached by applying an operator that is within 1 of the cost of the operator used to generate n. Given this, even if the heuristic errs in its evaluation of n, very few levels of nodes will have to be expanded to compensate for this heuristic error.

Another factor that makes grid world with life costs a particularly benign example of a non-unit cost domain is the fact that duplicates are so common. In order to observe the worst case exponential number of states described in Corollary 1, we assume the descendants of the node in question are all unique. In grid world with life costs, this assumption is not the case.

These mitigating factors allow cost-based best-first searches to perform very well, despite a very wide range of operator costs. This can be seen in Table 7 where weighted A* performs very well, in stark contrast to the other domains considered where weighted A* and greedy search are unable to solve all problems. It is a known problem that beam searches perform poorly in domains like grid path planning where there are lots of dead ends (Wilt, Thayer, and Ruml 2010), so it is not surprising that beam searches were unable to solve all instances in this domain. In Table 7, the rows for Speedier, EES, Skeptical, and Weighted A* are all pareto optimal, showing that in this domain, there is no clear benefit to using $d(n)$ the way there is in the weighted tiles domain and the heavy pancake domain, where the searches that did not consider $d(n)$ did not finish.

Summary

The examples in the previous section show the practical empirical consequences of Corollary 1. The sliding tile domain is one where low cost operators are often applicable and duplicates are rare, so there is little to mitigate the effect of the high cost ratio. In the sum pancake domain, duplicates are only mildly more common than in the sliding tiles domain, as the minimum cycle in that domain has length 6. Both of these domains prove to be very problematic for best-first searches when the costs are not all the same.

We can contrast this with what we observe in grid path planning. In grid path planning, duplicates are very common, and although operator costs vary across the space, there is very little local variation in operator costs, which places a very strict bound on the number of nodes that are within any single local minimum. Thus, the range of operator costs is only a part of what makes a non-unit cost domain more difficult.

Conclusion

We have shown that some domains with non-unit cost functions are much more difficult to solve using best-first heuristic searches than a domain with precisely the same connectivity but a constant cost function.

We have also shown that in a domain with a consistent heuristic and invertible operators, the amount that $h(n)$ can change from one node to the next is bounded. We then showed that if the rate of change in h is bounded, it can be very time consuming to recover from any kind of heuristic error if there are both large and small cost operators. We have shown this effect can harm the entire family of best-first searches ranging from A* to weighted A* to greedy search.

Lastly, we proposed a solution for problems where cost-based search performs poorly, which is to consider an additional heuristic, $d(n)$, and use this heuristic to help guide the search. We then showed that when best-first search proves impractical, searches that consider $d(n)$ can still find solutions. This demonstrates that algorithms that exploit $d(n)$ represent a fruitful direction for research. These results are significant because real-world domains often exhibit a wide range of operator costs, unlike classic heuristic search benchmarks.

References

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmueller, R.; and Kambhampati, S. 2010. G-value plateuas: A challenge for planning. In *Proceedings of ICAPS 2010*.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost based search considered harmful. In *Symposium on Combinatorial Search*.

Cushing, W.; Benton, J.; and Kambhampati, S. 2011. Cost based satisficing search considered harmful. arXiv:1103.3687v1 [cs.AI]. `http://arxiv.org/abs/1103.3687`, accessed April 10, 2011.

Doran, J. E., and Michie, D. 1966. Experiments with the graph traverser program. In *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 235–259.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artificial Intelligence* 1570–1603.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics* SSC-4(2):100–107.

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *Symposium on Combinatorial Search*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Likhachev, M., and Ferguson, D. 2009. Planning long dynamically feasible maneuvers for autonomous vehicles. *I. J. Robotic Res.* 28(8):933–945.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1:193–204.

Thayer, J., and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the Twenty-Second International Joint Conference on Articial Intelligence (IJCAI-11)*.

Thayer, J. T.; Dionne, A.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of ICAPS 2011*.

Thayer, J.; Ruml, W.; and Kreis, J. 2009. Using distance estimates in heuristic search: A re-evaluation. In *Symposium on Combinatorial Search*.

Wilt, C.; Thayer, J.; and Ruml, W. 2010. A comparison of greedy search algorithms. In *Symposium on Combinatorial Search*.