Efficient Budgeted Graph Search

Jasmeet Kaur[1] , **Nathan R. Sturtevant**[1,2]
[1]Department of Computing Science, University of Alberta, Canada
[2]Alberta Machine Intelligence Institute (Amii)
jasmeet8@ualberta.ca, nathanst@ualberta.ca

Abstract

Iterative Budgeted Exponential Search (IBEX) is a general search algorithm that can limit the number of re-expansions performed in common problems like iterative-deepening tree search and search with inconsistent heuristics. IBEX has been adapted into a specific tree algorithm, Budgeted Tree Search (BTS), which behaves like IDA* when f-cost layers grow exponentially, but keeps the worst-case guarantees when this does not hold. The analogous algorithms on graphs, Budgeted Graph Search (BGS), does not have these same properties. This paper reformulates BGS into Efficient Budgeted Graph Search (BGS$_e$) showing how to implement the algorithm so that it behaves identically to A* when the heuristic is consistent, and retains the best-case performance otherwise. Experimental results validate the performance of BGS$_e$ on a range of theoretical and practical problem instances.

1 Introduction

A^* is a popular search algorithm used to solve the shortest path problem. It uses a heuristic function to guide the search and its behavior depends on the properties of this heuristic function [Hart *et al.*, 1968]. A^* with an admissible and consistent heuristic is optimal in terms of the number of node expansions [Dechter and Pearl, 1985]. In the worst-case scenario, if the heuristic is inconsistent, it can do $O(2^N)$ node expansions of N distinct nodes [Martelli, 1977].

Previous work [Martelli, 1977; Bagchi and Mahanti, 1983; Mero, 1984] addressed this with modified versions of A^* to lower the worst-case performance to $O(N^2)$. Bidirectional pathmax (BPMX) has also been shown to improve the worst-case bound of $O(N^2)$ on some practical problems [Zhang *et al.*, 2009; Felner *et al.*, 2011]. Recent work on Budgeted Graph Search (BGS) [Helmert *et al.*, 2019] shows how to improve this by performing $O(N\log(C^*))$ node expansions in total, where N is the number of necessary expansions required to solve the problem.

The current version of BGS, however, has three flaws. First, BGS often performs $\Theta(N\log(C^*))$ node expansions even when the heuristic is consistent, resulting in very poor performance relative to A*. Second, BGS repeatedly restarts its search from scratch, something that can be avoided in practice to further improve performance. Finally, BGS breaks ties towards low g-cost, which results in poor tie-breaking.

In this paper, we describe BGS$_e$ an efficient reformulation of BGS that addresses all of these issues. It improves the best-case performance of BGS with consistent heuristics. It also provides an efficient way to re-use the data structures by not discarding all of the information about the nodes that the search has seen.

We show that BGS$_e$ performs identical to A^* when using consistent heuristics or when there are only a few node re-expansions. But, when the inconsistency is severe, it still only does $O(N\log(C^*))$ expansions. We validate these theoretical claims experimentally showing performance on a broad range of problems. Finally, we define a class of *conservative* algorithms that only expand states with $f(n) \leq C^*$, and show that, under some assumptions, conservative algorithms perform $\Omega(N^2)$ expansions.

2 Background

This paper considers search problems defined by $\{s_{init}, S_{goal}, succ, c, h\}$. In this definition s_{init} is the initial state and S_{goal} is the set of goal states. The full state-space, S, is defined implicitly by the successor function, $succ$ that maps each state s to a finite set of successor states and a cost function, c that determines the $c(s,s')$ of reaching a successor state s' from state s. The operation of generating successor states for a given state is called a *node expansion*.

Let $h^*(s)$ be the minimal cost path that reaches a goal state from state s. The heuristic function, $h : s \rightarrow [0,\infty]$ is an estimate of $h^*(s)$. A heuristic is said to be admissible if never overestimates the cost of state s to the goal state. In other words, $h(s) \leq h^*(s)$. It is said to be consistent if $h(s) \leq h(s') + c(s,s')$ for all pairs of states (s,s'). A heuristic that does not follow this property is said to be inconsistent [Felner *et al.*, 2011]. The set of problems with admissible heuristics (both consistent and inconsistent) are called I_{AD}.

A path in the state space is a sequence of k states, $\pi = (\pi_i)_1^k$ where $\pi_1 = s_{init}$ and the cost of this path is $g(\pi) = \sum_1^{k-1} c(\pi_i, \pi_{i+1})$. A solution is a path that ends in any one of the goal states. The objective of the search algorithm is to find a path with minimum cost, which is defined as C^*.

In order to discuss various algorithms, we must clearly de-

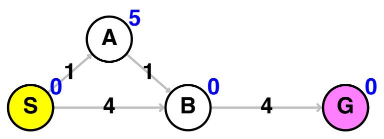

Figure 1: Search problem from S to G where A* re-expands state B twice.

fine which states must be expanded in order to solve a problem. There are some details of this definition that go beyond what can be discussed here; we follow the definition used in recent work on this topic [Helmert *et al.*, 2019]. Let Π be the set of all possible paths and recall that S is all possible states. Then the set of necessarily expanded states is $N = \{s \in S : \exists_{\pi \in \Pi : \pi_k = s}(\max_{i<k}(g(\pi_i) + h(\pi_i))) \leq C^*\}$. That is, N is all states on paths that do not contain a state s with $f(s) > C^*$. When it is clear from context, we also use N as shorthand for $|N|$ for simplicity. In the worst case, all states in N must be expanded to find and prove the optimality of the solution. But, there do exist instances where tie-breaking or heuristic propagation [Mero, 1984] could reduce the number of expansions needed in practice. The proofs later in the paper are not impacted by either tie-breaking or heuristic propagation, so we do not explore this issue further here.

2.1 A* with Inconsistent Heuristics

A* is a best-first search algorithm that orders expansions by $f(n) = g(n)+h(n)$, where $g(n)$ is the current minimum cost path known from the initial state to state n and $h(n)$ is the heuristic estimate of state n. By breaking ties among states with equal f towards states with larger g, A* is often able to achieve much better than worst-case performance.

A* maintains an OPEN list and a CLOSED list. It selects a node n from the OPEN list with the best $f(n)$ value, expands it and adds it into the CLOSED list starting with state s_{init}. This process continues until a goal state from S_{goal} is found or no nodes are available for expansion, at which point the search reports no solution. The complete pseudo-code to A* is in Algorithm 1.

If the complexity of a search algorithm is measured by the number of node expansions performed, A* with an admissible and consistent heuristic has been proven to be optimal [Dechter and Pearl, 1985]. If the heuristic is admissible but not consistent, A* can end up re-expanding nodes that have already been place in CLOSED. Figure 1 shows how an inconsistent heuristic can result in a node re-expansion. In this example, edges are marked with their costs and heuristic values are placed next to nodes. When node S is expanded, nodes B and A are put into the OPEN list with f-cost values of 4 and 6, respectively. Node B is expanded next and placed in CLOSED. When node A is expanded, a better path to node B is discovered, and it is put back into the OPEN list for re-expansion. B is then expanded a second time before the optimal path to G is found. Martelli (1977) showed that inconsistency can lead to $O(2^N)$ node expansions in the worst-case. In general, A* can have poor performance when the shortest paths have higher heuristics, and the longer paths have lower heuristics – meaning that the heuristic is misleading. But, some forms of inconsistency can be helpful [Felner *et al.*, 2011]. Variants of A* such as B [Martelli, 1977], C [Bagchi and Mahanti, 1983], and B' [Mero, 1984] reduce the worst-case to $O(N^2)$ expansions by temporarily ignoring the heuristic and prioritizing states only by g-cost.

Algorithm 1 $A^*(cost_l, re_l, n_l)$

```
 1: while !OPEN$_f$.$empty()$ do
 2:     $curr \leftarrow$ OPEN$_f$.$pop()$
 3:     $currF \leftarrow curr.g + curr.h$
 4:     if $isGoalState(curr)$ then
 5:         solutionPath $\leftarrow ExtractPath(curr)$
 6:         solutionCost $\leftarrow currF$
 7:         return
 8:     end if
 9:     if $isReopened(curr)$ then
10:         $reexpanions \leftarrow reexpanions + 1$
11:         if $reexpanions \geq re_l$ then
12:             return $[f, \infty]$
13:         end if
14:     else
15:         if $nodes \geq n_l$ then
16:             return $[f, \infty]$
17:         end if
18:         $nodes \leftarrow nodes + 1$
19:     end if
20:     CLOSED.$insert(curr)$
21:     for each $succ$ of $curr$ do
22:         $succC \leftarrow curr.g + c(curr, succ)$
23:         if $succC < succ.g$ then
24:             $succ.g \leftarrow succC$
25:             $succ.parent \leftarrow curr$
26:         end if
27:         if $succC + succ.h > cost_l$ then
28:             if $f > succC + succ.h$ then
29:                 $f \leftarrow succC + succ.h$
30:             end if
31:         end if
32:         OPEN$_f$.$insert(succ)$
33:     end for
34: end while
35: return $[f, \infty]$
```

2.2 Breadth-first Heuristic Search

Breadth-first heuristic search (BFHS) [Zhou and Hansen, 2006] takes a different approach from A*. Instead of ordering expansions by f-cost, it orders expansions by g-cost. It then uses a cost limit to delay the expansion states that have f-cost above the cost limit. The cost limit can be iterative raised until an optimal solution is found. BFHS was originally designed to reduce the size of OPEN in cases where there are fewer states with a given g-cost than f-cost. But, because BFHS orders states by g-cost, it will not expand a state more than once per iteration. In the worst case every node could both have a unique f-cost and lead to a shorter path to all previously expanded nodes (excluding s_{init}), requiring $\Theta(N^2)$ total expansions.

If we had an oracle that provided C^* as the initial cost limit,

Algorithm 2 BFHS($cost_l$, re_l, n_l)

```
1:  while !$\text{OPEN}_g.empty()$ do
2:      if $isGoalState(curr)$ then
3:          solutionPath ← $ExtractPath(curr)$
4:          solutionCost ← $currF$
5:          return solutionCost
6:      end if
7:      if $nodes \geq nodeLimit$ then
8:          return $f$
9:      end if
10:     $nodes \leftarrow nodes + 1$
11:     $\text{CLOSED}.insert(curr)$
12:     for each $succ$ of $curr$ do
13:         $succC \leftarrow curr.g + c(curr, succ)$
14:         if $succC < succ.g$ then
15:             $succ.g \leftarrow succC$
16:             $succ.parent \leftarrow curr$
17:         end if
18:         if $succC + succ.h > cost_l$ then
19:             $\text{OPEN}_f.insert(succ)$
20:             if $f > succC + succ.h$ then
21:                 $f = succC + succ.h$
22:             end if
23:         else
24:             $\text{OPEN}_g.insert(succ)$
25:         end if
26:     end for
27: end while
28: return $f$
```

BFHS would perform $\Theta(N)$ expansions. For instance, in Figure 1 if the cost limit was 6, BFHS would expand A, B, and then G. But, with an initial cost limit of 4, BFHS would still expand B prior to A, and thus be forced to later re-expand B.

BFHS can avoid re-expansions in the face of inconsistency, with two limitations to overcome. The first limitation is tie-breaking in the last layer. BFHS will tend to break ties towards expanding the goal last. Practically speaking, if there are many states with $f(s) = C^*$, this penalty could be significant. The second issue is that C^* is not known. Running a BFHS with every consecutive f value until C^* is reached can perform poorly, but Iterative Budgeted Exponential Search (IBEX) [Helmert *et al.*, 2019] improves this significantly. The pseudo-code BFHS is in Algorithm 2.

2.3 IBEX and Budgeted Graph Search

Algorithms like A*, IDA*, and BFHS search from low to high f-costs, and search every f-cost to completion before searching with a larger f-cost. IBEX improves the worst-case performance of these algorithms by growing the search cost exponentially, using a budget to limit the expansions if the f-cost gets too large.

IBEX divides the search into iterations. From one iteration to the next, it requires that the number of nodes expanded grows by a constant factor, giving overall exponential growth. This ensures that the cost of each iteration is amortized into the cost of the final iteration that finds the goal. In an iteration, IBEX asks an oracle to provide the f-cost that will result in a *target* number of node expansions. The oracle is implemented by an exponential search algorithm [Bentley and Yao, 1976]. IBEX performs one or more searches in each iteration. BGS describes IBEX using a budgeted BFHS as the underlying search algorithm, while BTS describes IBEX using a cost-limited DFS as the underlying search algorithm.

Iteration	Step	Curr. f	Target f	Exp.	Target Exp.
2	Init	9	$[9, \infty)$	500	-
3	Init	10	$[10, \infty)$	580	1000-4000
3	Exp.	20	$[11, \infty)$	800	1000-4000
3	Exp.	40	$[21, \infty)$	4000*	1000-4000
3	Bin.	30	$[21, 40)$	3500	1000-4000
4	Init	31	$[31, \infty)$	3800	7000-28000
4	Exp	62	$[32, \infty)$	25000	7000-28000
5	Init	63	$[63, \infty)$	27000	7000-28000

Table 1: BFHS searches performed as part of an IBEX search.

We illustrate the behavior of BGS in Table 1. Each row in the table corresponds to a complete BFHS search, and each row is labeled with the iteration in which the search takes place and the maximum f-cost that will be expanded. To begin, we assume BGS just completed iteration 2, searching with an f-cost of 9 and performing 500 node expansions. In iteration 3, BGS needs to ensure that the node expansions grow by a constant factor. BGS allows a range of growth, in this case 2x-8x. For iteration 3, BGS asks its oracle to find a *target* f-cost which results in 1000-4000 node expansions, which we call the *budgeted* number of expansions.

In this problem we assume all f-costs are integers, so BGS's oracle begins by performing a BFHS with a f-cost limit of 10. BFHS only expands 580 nodes, which is lower than the budget. Next, BGS will begin to increase the f-cost used in the search exponentially. In this example BGS multiplies the previous f-cost by two. Thus, its next search is with f-cost 20, knowing that the *target* f-cost, is in the range $[11, \infty)$. This search requires 800 expansions, which is still below the budget. The target f-cost is then known to be in the range $[21, \infty)$. The next search uses f-cost 40, but does not complete before performing 4000 expansions. Once 4000 expansions have been performed, the budget is exceeded, and no further expansions are needed to know that the f-cost is too large. This is a important mechanism to limits the cost of the search.

At this point the target f-cost is in the range $[21, 40)$, and BGS can perform a binary search to find the f-cost that performs the budgeted number of expansions. A search with f-cost of 30 requires 3500 expansions, which is within the budget. The oracle then returns the target f-cost to BGS. BGS will then request the next target f-cost from the oracle, and the oracle will begin searching with a f-limit of 31. In iteration 4, the search finds a target f-value during the exponential search, and does not need the binary search. These iterations continue until the goal is found.

If the target f-cost for an iteration is t, and the maximum budget is b, each iteration requires $O(\log(t))$ steps, with each step having cost no greater than b. Because the budget grows exponentially, the running time is dominated by the last iter-

Algorithm 3 BGS$_e$()

```
1:  solutionPath ← {}
2:  solutionCost ← $\infty$
3:  OPEN$_g$ ← {}
4:  OPEN$_f$ ← {}
5:  CLOSED← {}
6:  $budget \leftarrow 0$
7:  $i \leftarrow [h(initialState), \infty]$
8:  while $solutionCost > i.lower$ do
9:      $f \leftarrow$ Oracle($budget$)
10:     $i \leftarrow i \cap$ Search($f, \infty, \infty$)
11:     $nextCost \leftarrow i.lower$
12:     $budget \leftarrow$ size of the tree
13: end while
14: return
```

ation which has at most $\log(C^*)$ steps, each with cost $O(N)$, for total worst-case running time $\Theta(N \log(C^*)$.

A naive implementation of BGS improves the worst-case efficiency of A*, but does not maintain the best-case efficiency with consistent heuristics. Thus, in this paper we propose an efficient version of BGS and call it Efficient Budgeted Graph Search, BGS$_e$.

There are three sources of inefficiency in BGS. First, BGS always uses an exponential search to find the target f-cost, even when it is unnecessary (e.g. because no re-expansions will be required). Second, BGS breaks ties towards low g-cost in the BFHS, which can result in poor performance in the last layer. Finally, BGS does not maintain data structures such as OPEN and CLOSED between iterations, resulting in extra overhead. In the appendix, IBEX suggested using A* and reverting to BGS when re-expansions are encountered. Our approach is both more general and robust as it can return to A* if there are only few re-expansions at one point during the search.

3 Efficient BGS

BGS$_e$ improves BGS by only using an exponential search when necessary to avoid the worst-case behavior of previous algorithms. The following changes are made in BGS$_e$: (1) BGS$_e$ begins each iteration by expanding states in the A* order of low to high f-cost. This is like A*, but is limited to searching with a single f-cost. (2) BGS$_e$ explicitly counts re-expansions. When the number of re-expansions exceeds a provided limit it switches to expanding from low to high g-cost. (3) BGS$_e$ maintains all states in data structures between calls to BFHS to avoid unnecessary re-expansions. (4) We implement heuristic propagation (BPMX) [Felner *et al.*, 2011] for undirected domains in BGS$_e$.

This section uses pseudo-code to describe BGS$_e$ in detail. Algorithm 3 is the high-level code for BGS$_e$ using an oracle, which is found in Algorithm 4.

To assist the reader in understanding the structure of the pseudo-code and the approach, we trace the behavior of BGS$_e$ with a consistent and inconsistent heuristic, describing some of the details of why BGS$_e$ works efficiently. For simplification of the pseudo-code, we do not explicitly handle detecting the goal or returning the optimal path.

Algorithm 4 Oracle($budget$)

```
1:  $f_{current} \leftarrow 0$
2:  while $nodesReexpanded \leq k \times budget$ do
3:      $f_{current} \leftarrow i.lower$
4:      $i \leftarrow i \cap$ AStar($i.lower, k \times budget, \infty$)
5:      if $nodesExpanded \geq c1 \times budget$ then
6:          $nodesReexpaned \leftarrow 0$
7:          return $f_{current}$
8:      end if
9:  end while
10: if $nodesExpanded \geq c1 \times budget$ then
11:     $nodesReexpaned \leftarrow 0$
12:     return $f_{current}$
13: end if
14: // Exponential Search
15: $\delta \leftarrow 1$
16: while ($i.upper \neq i.lower$ & !($nodesExpanded \leq c1 \times$
    $budget$) do
17:     $\delta = \delta * \gamma$
18:     $nextCost = i.lower * \delta$
19:     $i \leftarrow i \cap$ Search($nextCost, \infty, c2 \times budget$)
20: end while
21: if $nodesExpanded < c2 \times budget$ then
22:     return $i.lower$
23: end if
24: // Binary Search
25: while ($i.upper \neq i.lower$ & !($c1 \times budget \leq$
    $nodesExpanded \leq c2 \times budget$) do
26:     $nextCost = (i.lower + i.upper)/2$
27:     $i \leftarrow i \cap$ Search($nextCost, \infty, c2 \times budget$)
28: end while
29: if $nodesExpanded \leq c1 \times budget$ then
30:     return $i.upper$
31: else
32:     return $nextCost$
33: end if
```

3.1 BGS$_e$ with (Near-)Consistent Heuristics

In case of a consistent heuristic, BGS$_e$ will begin by calling the oracle (BGS$_e$ line 9), asking for the target f-cost that doubles the number of node expansions.

Because no states are ever re-expanded with a consistent heuristic, the oracle will loop from lines 2 – 9, repeatedly calling A* on line 4, until the number of nodes expanded meets the budget. The variable i contains the current interval $[i.lower, i.higher]$ that bounds the target f-value. When a f-value is found that meets or exceeds the budget, the target f-cost is returned.

BGS$_e$ then passes this f-cost to the search procedure (Algorithm 5). This procedure runs BFHS with the target f-cost. The search procedure takes all states from OPEN$_f$ that have f-cost less than the target f-value, and places them on OPEN$_g$. However, when the heuristic is consistent, all of these states will already have been expanded by A*. Thus, there will be no states in OPEN$_f$ with this property.

BGS$_e$ repeats this process until the goal is found. Because the heuristic is consistent, BGS$_e$ will only call A* through the oracle, and have the same overall performance as A*.

If the heuristic is inconsistent, but does not perform many re-expansions, the bound in line 2 of the Oracle ($k \times budget$ for some constant k) will allow a constant number of re-

Algorithm 5 Search($cost_l$, re_l, n_l)

```
1: $currF \leftarrow 0$
2: while !$\text{OPEN}_f.empty()$ and $currF \leq cost_l$ do
3:     $curr \leftarrow \text{OPEN}_f.front()$
4:     $currF \leftarrow curr.g + curr.h$
5:     if $currF \leq cost_l$ then
6:         $\text{OPEN}_g.insert(\text{OPEN}_f.pop())$
7:     end if
8: end while
9: $next \leftarrow$ BFHS($cost_l$, $re_l$, $n_l$)
10: while !$\text{OPEN}_g.empty()$ do
11:     $\text{OPEN}_f.insert(\text{OPEN}_g.pop())$
12: end while
13: return [$next$, $\infty$]
```

expansions. This parameter for BGS$_e$ will be evaluated in the experimental results.

3.2 BGS with Inconsistency

If there is significant inconsistency in the heuristic, the budget for re-expansions will be exceeded in line 2 of the Oracle, and the code will continue.

There are some subtleties in the implementation. For instance, it is possible that the last call to A* simultaneously exceeded the re-expansion limit, but also found the target f-cost. This is handled in line 10, which returns the target f-cost. This f-cost has not been searched completely, but the remainder of the search will be done with BFHS in BGS$_e$ line 10. Otherwise, the oracle will now use the exact same procedure as IBEX, an exponential search followed by a binary search, to find the target f-value

Also note that it is possible that there is no target f-cost which corresponds to a number of node expansions that is within the budget. In this case the binary search in the Oracle will return the minimum f-cost which is guaranteed to result in node expansions that are above the budget, since the f-cost which is below the budget has already been searched to completion without finding the goal. This f-cost will then be searched in its entirety by the call to BFHS in line 10 of BGS$_e$.

3.3 Algorithmic Complexity

Our formulation of BGS$_e$ means that it has the same performance as A* when the heuristic is consistent. However, we need to ensure that we have not compromised the efficiency with these modifications.

The primary modification is that the oracle begins by repeatedly running A* in lines 2 to 9. At this point in the search, the oracle is looking for the target f-cost that falls within the budget. It can make at most $\log(C^*)$ calls to the the search procedure and the node expansions performed in each call will be bounded by the budget. Within our formulation, BGS$_e$ essentially makes one set of additional searches which are also bounded by the same budget. If these searches are successful (e.g. they do not require a significant number of re-expansions), the $\log$ overhead is avoided. If they are unsuccessful, only a constant overhead is added to the search.

The other point for maintaining the correctness of BGS$_e$ is that no successor will ever be discarded; these are saved for possible later expansions.

4 Theory

IBEX and its variants expand states with $f(n) > C^*$ in order to improve the performance of search with inconsistent heuristics. In unidirectional search with a consistent heuristic, and with the exception of tie-breaking for states with $f(n) = C^*$, there is no admissible algorithm which expands fewer states than A* [Dechter and Pearl, 1985]. In unidirectional search with inconsistent heuristics, however, no single algorithm can have the best performance on all problems [Mero, 1984]. There are some problems where, due to the inconsistency of the heuristic, performing expansions of states with $f(n) > C^*$, could lead to the discovery of heuristic information that, when propagated to other states, reduces the number of states with $f(n) \leq C^*$, thus reducing the total number of expansions required to solve the problem.

Thus, there is a distinction between unidirectional algorithms that expand states with $f(n) > C^*$ and those that do not. While A* may perform $\Theta(2^N)$ expansions of N states, and B and B' may perform $\Theta(N^2)$ expansions of N states, IBEX requires at most $\Theta(N \log C^*)$ expansions.

The question addressed here is whether it is necessary to expand states with $f(n) > C^*$ in order to improve on the previous $\Theta(N^2)$ best case. We show that all unidirectional algorithms which only expand states with $f(n) \leq C^*$ must expand $\Omega(N^2)$ states in the worst case, where N is the number of *necessarily expanded* states, as defined previously.

4.1 Assumptions

The analysis here applies to unidirectional algorithms that meet three properties (admissible, strongly DXBB, and conservative) when run on problem instances in I_{AD}.

Algorithms are *admissible* if they find optimal solutions [Hart *et al.*, 1968; Martelli, 1977; Dechter and Pearl, 1985] when a solution exists. Note that admissibility of an *algorithm* is different that the admissibility of a *heuristic*. For instance, Dijkstra's algorithm is admissible with any heuristic, A* is only admissible with admissible heuristics, and BS* [Kwa, 1989] is only admissible with a consistent heuristic.

Algorithms are deterministic, expansion-based black-box (DXBB) if they are not randomized and can only gain information about the problem being solved through the use of a black-box expansion function [Eckerle *et al.*, 2017]. The results here require a slightly stronger version of DXBB.

Definition 1. *An algorithm is* strongly DXBB *if it is DXBB and additionally, information about shorter paths or better heuristics (e.g. through BPMX) is only propagated to neighboring states (previously expanded or otherwise), and only by expansion operations.*

Strongly DXBB algorithms are excluded from the use of alternate data structures that might be used to substitute different operations for node expansions. This would exclude, for instance, the possibility of dynamically performing contractions [Geisberger *et al.*, 2008] on the graph of expanded states during search. All current algorithms that we are aware of that are DXBB are also strongly DXBB.

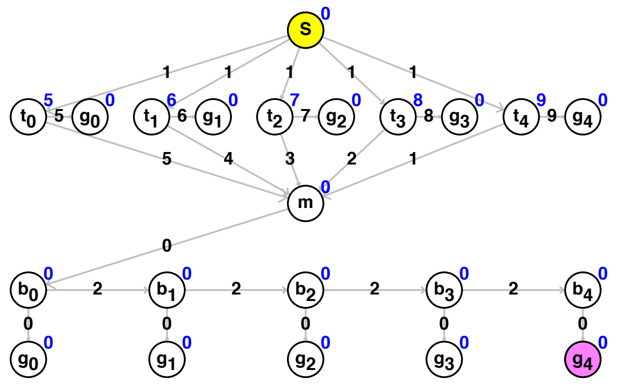

Figure 2: Example demonstrating that $\Omega(N^2)$ expansions are required when states with $f(n) > C^*$ are not expanded.

Definition 2. *An algorithm is* conservative *if it only expands a state* n *if it has proven that* $f(n) = g(n) + h(n) \leq C^*$.

Note that definition of conservative relies on the current $g(n)$, which may not be the optimal g-cost from the start. A* and IDA* are conservative algorithms, but Dijkstra's algorithm is not, because it ignores heuristic values.

The structure of the proof relies on the propagation of information that is required to ensure the algorithm is both admissible and conservative. By identifying the states that must be expanded to prove $f(n) \leq C^*$ for a given state, it is possible to put a lower bound on the expansions required.

4.2 Problem Structure

The proof is based on the problem in Figure 2, which is derived from previous published examples [Mero, 1984]. Previous work showed the performance of algorithms B, and B' on a similar problem, but did not generalize this to other algorithms. Figure 2 is an instance (for $k = 5$) of a general problem that can be scaled for any k to have $4k + 2$ nodes, and requires $\Omega(k^2)$ expansions by any conservative, admissible, and strongly DXBB algorithm.

The structure of the problem is as follows. Every state labeled g_i is a potential goal state, but g_{k-1} is the actual goal. These are states that must be expanded by any admissible algorithm to prove that they are not the goal and to maintain the conservative property. There are two separate states labeled g_i. These can be considered to be the same state reachable by two different directed edges; in fact all g_i could be the same state. But, we draw these explicitly to make the example as straightforward as possible.

There is a start state S at the top, which connects to 5 (k) states at the top, labeled t_0 through t_4 with $h(t_i) = k + i$. Each t_i is connected by a directed edge cost $h(t_i)$ to a g_i and to the middle state, m, with a directed edge cost $k - i$. This middle state connects to a directed chain of bottom states b_0 through b_4. These are connected to g_i by an edge with cost 0 and to their next neighbor on the chain with an edge cost 2. The heuristic from states $t_1 \ldots t_4$ to m is inconsistent.

As a reminder, the f-cost of a state s is defined by the current-best g-cost plus the h-cost of s, as opposed to the shortest path from S to s, which could be smaller. Thus, expanding a state that currently has $f(n) > C^*$ would mean an algorithm is not conservative, even if state n might later be discovered to have $f(n) \leq C^*$ for some alternate path to n. This is because the definition of conservative requires that the algorithm prove that $f(n) \leq C^*$ prior to each expansion.

Lemma 1. *Let $\mathcal{A}$ be any algorithm which is admissible, conservative, and strongly DXBB. $\mathcal{A}$ must perform a minimum of i expansions after expanding t_{i-1} and before expanding t_i on the k-state generalization of Figure 2.*

Proof. We prove this by induction using two properties. First, that there are at least i expansions after t_{i-1} and before t_i, and second that when t_i is about to be expanded $f(b_i) > f(t_i)$. In this proof we will ignore the g_i states, except when showing that the potential existence of these states forces the expansion of a t_i or b_i state.

Base case. For the base case we need to show that at least 1 expansion is performed after expanding t_0 and before expanding t_1. Initially, by definition, $f(t_0) = k + 1$ and $f(t_1) = k + 2$, and all other states that can be expanded by a DXBB algorithm (are generated from the start state) have larger f-cost. Because $\mathcal{A}$ is conservative, it must expand all states with $f = k + 1$ before expanding $f(t_1)$. Thus, it must minimally expand m and b_0, which both have $f = k+1$. This demonstrates the first property.

After expanding b_0, it is the case that $f(b_1) = g(b_0) + 2 = f(t_0) + 2$, and $f(t_1) = f(t_0) + 1$. Thus the second property holds because $f(b_1) > f(t_1)$.

Recursive case. We now assume that after expanding t_{i-1} and before expanding t_i a total of i states have been expanded. We further assume that $f(b_i) > f(t_i)$. Then, we need to show that at least $i+1$ expansions will be performed after the expansion of t_i and before the expansion of t_{i+1}.

First, $\mathcal{A}$ must eventually expand t_i after expanding t_{i-1}. If it does not, we could change the goal to g_i, and then $\mathcal{A}$ would not be admissible. Either because it does not find the solution, or if $\mathcal{A}$ expands g_i through b_i without updating the path through t_i, it would find the solution with suboptimal cost.

After expanding t_i, $\mathcal{A}$ must continue expanding down the chain of states from t_i to b_i, finding a shorter path to m along the way, as all of these states have the same f-cost. It cannot expand a state t_j for $j > i$ because $\mathcal{A}$ is conservative and these states have larger f-cost. It must expand the entire chain of states to propagate the improved g-cost from m to b_i because $\mathcal{A}$ is strongly DXBB. Finally, because it is admissible, it must expand b_i in case the goal is below b_i at g_i. Once this sequence of expansions is complete, $\mathcal{A}$ will have expanded states $b_0 \cdots b_i$ states ($i+1$ total) while propagating the shorter path cost from t_i to b_i. As in the base case, it is the case that $f(b_{i+1}) = g(b_i) + 2 = f(t_i) + 2$ and $f(t_{i+1}) = f(t_i) + 1$, thus $f(b_{i+1}) > f(t_{i+1})$. □

Lemma 2. *Let $\mathcal{A}$ be any algorithm which is conservative, admissible, and strongly DXBB. $\mathcal{A}$ must perform $\Omega(k^2)$ expansions on the k-state generalization of Figure 2.*

Proof. Due to Lemma 1, $\mathcal{A}$ must expand at least $i + 1$ states between expanding t_i and t_{i+1}. Thus, if the goal is placed in state g_k, it must perform a minimum of $\sum_{i=1}^{k} i$ expansions,

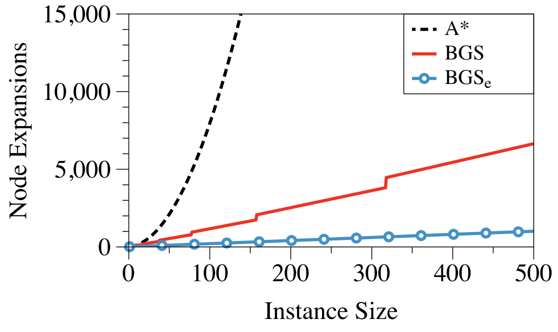

Figure 3: A comparison between A*, BGS, and eBGS on an artificially constructed worst-case graph.

which means that the total number of expansions is $\Omega(k^2)$. □

Theorem 3. *All unidirectional algorithms which are admissible, conservative, and strongly DXBB have worst-case complexity $\Omega(N^2)$ on problems in I_{AD}, where N is the number of* must-expand *states.*

Proof. The k-state generalization of Figure 2 is in I_{AD} and has $N = 3k + 2$ must-expand states. Thus, by Lemma 2, any such algorithm will require $\Omega(N^2)$ expansions to find the optimal solution. □

It follows from this that there are three possible routes to improving the worst-case search performance with an inconsistent heuristic. IBEX and its variants relax the conservative requirement and expand states with $f(n) > C^*$ in order to reduce the worst-case running time. It is an open question whether an algorithm can achieve similar gains by relaxing the strongly DXBB requirement. A bidirectional case an algorithm could improve performance in our example by expanding m in the backwards direction, but this is unlikely to generalize. Because the necessary conditions for bidirectional search are significantly more complex [Eckerle *et al.*, 2017], this is also left as an open question.

5 Experiments

While our improved implementation of BGS maintains the same worst-case performance as BGS, the constant factors in an implementation can have a large impact. The experimental result here show that BGS$_e$ has the same performance as A* when the heuristic is consistent, and much better performance when the heuristic is inconsistent.

We used the C++ code from the open-source HOG2 repository[1] for the baseline algorithms and testing. Our code for BGS$_e$ that is used to run these experiments is available in the same repository. All experiments are run on a 3.4Ghz Intel Core i7 machine with 16GB of RAM.

Worst-Case Evaluation. The first experiment is run on a graph similar to Figure 2, which corresponds to the example used in previous experimental results with BGS [Helmert *et al.*, 2019]. The structure of the problem is the same, but all edge costs from m to the goal are 1, and all g_i nodes are omitted. This experiment is designed to show that, in the worst case, BGS$_e$ maintains its improvements over A*, and also improves over the unoptimized implementation of BGS. The results of this experiment are found in Figure 3. The x-axis is the size of the instance created, and the y-axis is the total number of node expansions performed. The instance sizes tested runs from 3 (8 total nodes) to 500 (1002 total nodes). The re-expansions budget for BGS$_e$ is set to 0*budget, but the results are indistinguishable for 1*budget.

We expect A* to perform $\Theta(k^2)$ expansions on an instance size k, which can be clearly seen from the top black line. The general implementation of BGS improves over this significantly. BGS$_e$ shows improvement over BGS primarily because it does not throw away OPEN and restart from scratch in each iteration.

Algorithm	Avg. Expansions	Avg. Time (ms)
BGS	22,993	10.3
BGS$_e$	6,700	4.1
A*	6,700	3.5

Table 2: Expansions and time required to solve Dragon Age: Origin benchmark problems with a consistent heuristic.

Consistent Heuristic Evaluation. Next, we look at standard pathfinding problems with a consistent heuristic. In these experiments we test problems from Dragon Age: Origins [Sturtevant, 2012] with the octile heuristic to measure the difference between A*, BGS, and BGS$_e$. We tested BGS$_e$ on all maps and show results for all problems in the map `den601d`. We compare the average node expansions and running time to solve these problems in Table 2. The value of parameters c1, c2, and γ are 2, 8, and 2 respectively for all experiments. For BGS$_e$, we used a re-expansion budget of 1*budget in this case. BGS requires 3x the node expansions of A*, while BGS$_e$ performs an identical number of expansions to A*. Our implementation is slightly slower than A* because it is optimized for visualization, debugging, and correctness.

Inconsistent Heuristic Evaluation. In our first experiment we looked at the worst case for an inconsistent heuristic where the heuristic is adversarial, but this is not the typical use case for inconsistent heuristics. Inconsistent heuristics are more likely to be used as a result of compression or randomization [Felner *et al.*, 2011].

So, we experiment with the compressed differential heuristic [Felner *et al.*, 2011; Goldenberg *et al.*, 2011; Goldenberg *et al.*, 2017]. This heuristic is primarily designed to avoid the space overhead of storing many memory-based heuristics [Goldberg and Harrelson, 2005; Sturtevant *et al.*, 2009]. We build a differential heuristic with 10 pivots, but compress it so that each state can only access a single one of these heuristics. This approach works well in practice when combined with bidirectional pathmax (BPMX), which propagates heuristic values between neighboring states during expansion.

The results of running the same problems from Table 2, except with an improved heuristic, are found in Table 3. While

[1]https://github.com/nathansttt/hog2/tree/PDB-refactor

Algorithm	Avg. Expansions	95% Interval
A*	30,788	$\pm 2,169$
BGS	25,452	$\pm 1,295$
BGS$_e$(1*budget)	12,650	± 568
BGS$_e$(5*budget)	21,079	± 1197
BGS$_e$(0*budget)	9,414	± 307
A* [BPMX]	1,829	± 94
BGS$_e$(1*budget) [BPMX]	1,887	± 98
BGS$_e$(5*budget) [BPMX]	1,871	± 97
BGS$_e$(0*budget) [BPMX]	8,496	± 319

Table 3: Expansions and time required to solve Dragon Age: Origin benchmark problems with an inconsistent heuristic.

BGS does better than A*, and BGS$_e$ does better than BGS, it would be more efficient to use the octile heuristic in practice. However, with BPMX this is not the case. The results for BGS$_e$ are shown for three different values of the parameter k (Algorithm 2). Increasing the re-expansion budget allows BGS$_e$ to behave more like A*, and BGS$_e$ with BPMX does only a few node expansions more than A*.

From these experiments as a whole we conclude that BGS$_e$ with a re-expansion budget of 1*budget provides reasonable performance across both inconsistency experiments.

6 Conclusion and Discussion

This paper has introduced BGS$_e$, which keeps the behavior of A* when the heuristic is consistent, but retains the best-case performance of BGS when the heuristic is inconsistent. Thus, unlike the default BGS implementation previously described, BGS$_e$ is suitable for use on all problems. We additionally have shown that broader IBEX framework has an improved worst-case bound because it is not conservative. It is an open question whether there exist bidirectional search or non-DXBB algorithms that could provide similar worst-case performance.

References

[Bagchi and Mahanti, 1983] Amitava Bagchi and Ambuj Mahanti. Search algorithms under different kinds of heuristics—a comparative study. *Journal of the ACM (JACM)*, 30(1):1–21, 1983.

[Bentley and Yao, 1976] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(SLAC-PUB-1679), 1976.

[Dechter and Pearl, 1985] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[Eckerle *et al.*, 2017] Jürgen Eckerle, Jingwei Chen, Nathan Sturtevant, Sandra Zilles, and Robert Holte. Sufficient conditions for node expansion in bidirectional heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

[Felner *et al.*, 2011] Ariel Felner, Uzi Zahavi, Robert Holte, Jonathan Schaeffer, Nathan Sturtevant, and Zhifu Zhang. Inconsistent heuristics in theory and practice. *Artificial Intelligence*, 175(9-10):1570–1603, 2011.

[Geisberger *et al.*, 2008] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 319–333. Springer, 2008.

[Goldberg and Harrelson, 2005] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, pages 156–165. Citeseer, 2005.

[Goldenberg *et al.*, 2011] Meir Goldenberg, Nathan R. Sturtevant, Ariel Felner, and Jonathan Schaeffer. The compressed differential heuristic. In *AAAI Conference on Artificial Intelligence*, pages 24–29, 2011.

[Goldenberg *et al.*, 2017] Meir Goldenberg, Ariel Felner, Alon Palombo, Nathan Sturtevant, and Jonathan Schaeffer. The compressed differential heuristic. *AI Communications*, 30(6):393–418, 2017.

[Hart *et al.*, 1968] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Helmert *et al.*, 2019] Malte Helmert, Tor Lattimore, Levi H. S. Lelis, Laurent Orseau, and Nathan R. Sturtevant. Iterative budgeted exponential search. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[Kwa, 1989] James BH Kwa. Bs: An admissible bidirectional staged heuristic search algorithm. *Artificial Intelligence*, 38(1):95–109, 1989.

[Martelli, 1977] Alberto Martelli. On the complexity of admissible search algorithms. *Artificial Intelligence*, 8(1):1–13, 1977.

[Mero, 1984] Laszlo Mero. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23(1):13–27, 1984.

[Sturtevant *et al.*, 2009] N. R. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch. Memory-based heuristics for explicit state spaces. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 609–614, 2009.

[Sturtevant, 2012] Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[Zhang *et al.*, 2009] Zhifu Zhang, Nathan R Sturtevant, Robert Holte, Jonathan Schaeffer, and Ariel Felner. A* search with inconsistent heuristics. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[Zhou and Hansen, 2006] Rong Zhou and Eric A Hansen. Breadth-first heuristic search. *Artificial Intelligence*, 170(4-5):385–408, 2006.