# Improving MPGAA* for Extended Visibility Ranges

**Carlos Hernández**
Depto. de Ciencias de la Ingeniería
Universidad Andrés Bello
Santiago, Chile

**Jorge A. Baier**
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

**Abstract**

Multipath Generalized Adaptive A* (MPGAA*) is an A*-based incremental search algorithm for dynamic terrain that can outperform D* for the (realistic) case of limited visibility ranges. A first contribution of this paper is a brief analysis studying why MPGAA* has poor performance for extended visibility ranges, which concludes that MPGAA* carries out an excessive number of heuristic updates. Our second contribution is a method to reduce the number of heuristic updates that preserves optimality. Finally, a third contribution is a variant of MPGAA*, MPGAA*-back, which we show outperforms MPGAA* and D* on a wide range of dynamic grid pathfinding scenarios, and visibility ranges.

## Introduction

Incremental Heuristic Search (IHS) is a well-established sub-area of Heuristic Search whose objective is the development of algorithms capable of deciding the actions that lead an agent from a given initial state to a given goal state in a dynamic, changing environment.

Goal directed navigation, the problem of moving an agent from one cell to another over a grid, is an application domain of IHS. Since their proposal, the D* (Stentz 1994) algorithm, and later the D* Lite (Koenig and Likhachev 2002) algorithms were considered the state of the art for this problem. D* is a complex algorithm, and D* Lite, while simpler, is still significantly more complicated than A*.

Recently, Multipath Generalized Adaptive A* (MPGAA*) (Hernández, Asín, and Baier 2015) was proposed as an alternative to D*. MPGAA* is an extension of Generalized Adaptive A* (Sun, Koenig, and Yeoh 2008) that, being based on A*, is simpler to understand and implement than D* and D* Lite. Moreover, MPGAA* was shown to outperform D* Lite over a wide range of benchmarks.

A key feature of MPGAA* is that every action taken by the agent lies on an optimal path to the goal. MPGAA* guarantees this by running a consistency-reestablishing procedure—inherited from GAA*—each time it observes a change in the environment that could make the heuristic inconsistent. By ensuring that A* uses a consistent (and admissible) heuristic, only optimal paths to the goal are found.

In this paper we show that the consistency-reestablishing module of MPGAA* is a source of significant overhead, and propose a remedy: Improved MPGAA*. Our motivation is problems in which agents have extended visibility ranges, that is, problems in which agents can observe changes in the environment that are relatively far away from their current position. Arguably, those situations are not too common in practice, but the question of whether or not MPGAA* still can outperform the D* family of algorithms in such a setting is an interesting one. The question is more relevant given that recent evaluations (Aine and Likhachev 2016) have shown that MPGAA* incurs severe overheads, due to heuristic updating, in some settings.

What enables us to propose Improved MPGAA* is the interesting observation that neither consistency nor admissibility is actually needed to guarantee optimality, but rather a weaker condition. By realizing this, we propose a lazy heuristic update procedure that still guarantees that MPGAA* finds optimal paths. In our empirical evaluation, carried out over a number of benchmarks and visibility ranges, we show that Improved MPGAA* outperforms D* and D* Lite in the majority of the benchmarks, while the original version of MPGAA* usually could not.

## Background

A goal directed navigation problem is a tuple $P = (G, \gamma, s_{start}, s_{goal})$, where $G = (V, E)$ is an undirected graph in which $V$ is a set of states and $E$ is a set of arcs, and where $s_{start}$, the start state, and $s_{goal}$, the goal state, are both in $V$, and where $\gamma = c_0 c_1 \ldots$ is an infinite sequence of cost functions, each of which maps each element in $E$ to a non-negative value in $\mathbb{R} \cup \{\infty\}$. Sequence $\gamma$ is used to model the fact that the terrain may change as the agent moves, and the cost value $\infty$ is used to represent the fact that states may, in practice, become disconnected from its neighbors in $G$.

A *path* over G is a sequence of states $s_0 s_1 \ldots s_n$ such that for all $i \in \{1, \ldots, n\}$, it holds that $(s_{i-1}, s_i) \in E$. The *cost of a path* $s_0 \ldots s_n$ is $\sum_{i=0}^{n-1} c_i(s_i, s_{i+1})$. $\sigma = s_0 \ldots s_n$ is a *solution* to path-planning problem $P$ if $s_0 = s_{start}$, $s_n = s_{goal}$, and the cost of $\sigma$ is finite.

In real-world scenarios, agents have limited visibility and thus can only observe the cost of edges within their *visibility range*. Formally, if the agent is at $s_i$, and its visibility range is $k$, then the arcs by which $c_i$ and $c_{i+1}$ may differ are only those reachable from $s_{i+1}$ via a path of length $k$ or less.

A heuristic function $h_c$ is such that $h_c(s)$ is a non-negative estimate of the cost of a path from $s$ to $s_{goal}$ under cost function $c$. We omit the subscript if the cost function is clear from the context. $h$ is *admissible* iff for every state $s$, $h(s)$ does not overestimate the cost of any path from $s$ to $s_{goal}$. $h$ is *consistent* if for every $(s, t) \in E$ it holds that:

$$h(s) \leq c(s, t) + h(t), \tag{1}$$

and $h(s_{goal}) = 0$. Consistency implies admissibility. Finally, $h^*(s)$ is the cost of an optimal path from $s$ to $s_{goal}$.

## Multipath Generalized Adaptive A*

Multipath Generalized Adaptive A* (MPGAA*) is built on top of the A*-based algorithm, Generalized Adaptive A* (Sun, Koenig, and Yeoh 2008) (GAA*). Like GAA*, its objective is to move an agent to the goal state. In each iteration, it runs an A* search rooted in the current state towards the goal. Once a path is found, it updates the heuristic function and proceeds to move the agent following the path. Each time a step towards the goal is taken by the agent, the algorithm observes the environment and, if relevant changes are detected, it updates the heuristic (if needed), and then runs a complete new A* search towards the goal. The only difference between MPGAA* and GAA* is that the former will stop the A* search earlier, potentially saving significant search time. MPGAA* stops an A* search when expanding a state $s$ if the following conditions hold: (1) $s$ is part of a path returned by a previous A* execution, and (2) it can be verified that the path connecting $s$ and the goal state is still optimal given the current knowledge of the agent.

Algorithm 1 shows a pseudocode of MPGAA*. Procedure `main` runs the main loop described above. There are a number of variables/properties that are used in the pseudocode, the most relevant of which we explain now. Pointer $next(s)$ is set to null at initialization, and, as soon as a path $s_1 s_2 \ldots s_n$ is returned by an A* search, $next(s_i)$ is set to $s_{i+1}$ in the iteration of Line 72. As usual in A*, a parent pointer for $s$, $parent(s)$ is used to keep track of which state was $s$ expanded from. Another relevant property, $search(s)$, is used to store the number of the search iteration at which $s$ was last expanded. Variable $counter$ counts the number of iterations—this important variable is also used by procedure `InitializeSearch`, to set the $g$-value of an unexpanded state in the current iteration. Finally, variable $Q$ is a priority queue used by the procedure that updates the heuristic, a key part of the algorithm that we elaborate on now.

Above we mentioned that $h$ is updated when running MPGAA*. The objective of such updates is twofold. First, updating the heuristic makes it more informed, which potentially saves time in a subsequent search. Second, the heuristics updates also aim at preserving the consistency of the heuristic, which has an important theoretical implication: it guarantees that any path computed by A* is *optimal* with respect to the current knowledge of the agent (Sun, Koenig, and Yeoh 2008; Hernández, Asín, and Baier 2015).

MPGAA* updates the heuristic function in two different sections of the code. First, right after an A* search returns,

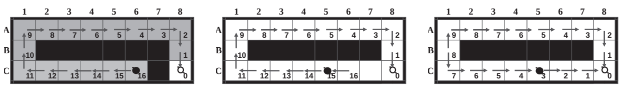
(a) (b) (c)

Figure 1: MPGAA*'s expensive consistency maintainance. Numbers in the cells are $h$-values. The agent is represented by ●, and the goal by ○. Arrows show the $next$ pointer.

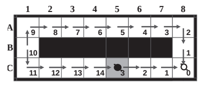
Figure 2: Improved MPGAA*'s heuristic update on the grid-world of Figure 1. The heuristic is updated for three cells, regardless of the width of the grid.

in Lines 70–71. Intuitively this update aims at making $h$ more informed. The second section of the code that updates the heuristic the `Observe` procedure; specifically, in Lines 34–37. Indeed, when an edge decreases its cost, there is potential for the heuristic to become inconsistent, as it may invalidate Inequality (1). Thus whenever an arc $(s, s')$ is observed to decrease its value, `Observe` adds state $s$ to the priority queue $Q$ and then invokes procedure `ReestablishConsistency`. This last procedure is a version of Dijkstra's algorithm: it extracts the state with least $h$-value from $Q$ and updates the $h$-value of its predecessors. The process repeats until $Q$ is empty. Upon termination, both updates guarantee that $h$ is consistent, for every state $s$ in the search graph (Sun, Koenig, and Yeoh 2008).

## Improved MPGAA*

As mentioned above, MPGAA* is optimal because the heuristic function remains consistent and thus admissible, which allows A* to return an optimal path (Hart, Nilsson, and Raphael 1968). However, below we elaborate on two observations that are key to obtaining an improved version of MPGAA*. First, we observe that re-establishing the consistency is a potentially very expensive procedure, requiring some times the expansion of a large portion of the search space. Second, we observe that admissibility of $h$ is not required to guarantee optimality of A* but, rather, a weaker condition. When put together, these two observations lead us to propose an intuitively less ambitious consistency maintenance algorithm which results in improved performance.

We observe first that consistency maintenance is an expensive procedure. We illustrate this with the 4-neighbor grid world of Figure 1. After finding a first path (shown in Figure 1b), the agent updates the heuristic function, moves to $C5$. Upon reaching $C5$, it discovers that $C7$ is no longer an obstacle (depicted in Figure 1b). Because some arcs have decreased their cost `ReestablishConsistency`—which henceforth we abbreviate as `RC`—is executed, which updates the heuristic of 8 cells, $B1$ and $C1..7$, resulting in the situation depicted by Figure 1c. Observe that, had we chosen a similar but wider grid, the number of cells expanded by `RC` would have grown

**Algorithm 1:** MPGAA*

```
1  procedure InitializeState($s$)
2      if $search(s) \neq counter$ then
3          $g(s) \leftarrow \infty$
4      $search(s) \leftarrow counter$
5  procedure A*($s_{init}$)
6      InitializeState($s_{init}$)
7      $parent(s_{init}) \leftarrow$ null
8      $g(s_{init}) \leftarrow 0$
9      Open $\leftarrow \emptyset$
10     insert $s_{init}$ into Open with f-value $g(s_{init}) + h(s_{init})$
11     Closed $\leftarrow \emptyset$
12     while Open $\neq \emptyset$ do
13         remove a state $s$ from Open with the smallest f-value $g(s) + h(s)$
14         if GoalCondition($s$) then
15             return $s$
16         insert $s$ into Closed
17         for each $s' \in succ(s)$ do
18             InitializeState($s'$)
19             if $g(s') > g(s) + c(s, s')$ then
20                 $g(s') \leftarrow g(s) + c(s, s')$
21                 $parent(s') \leftarrow s$
22                 if $s'$ is in Open then
23                     set priority of $s'$ in Open to $g(s') + h(s')$
24                 else
25                     insert $s'$ into Open with priority $g(s') + h(s')$
26     return null
27 function Observe($s$)
28     $E^{inc} \leftarrow$ arcs in the range of visibility from $s$ whose cost has increased
29     $E^{dec} \leftarrow$ arcs in the range of visibility from $s$ whose cost has decreased
30     for each $(t, t') \in E^{inc} \cup E^{dec}$ do
31         $c(t, t') \leftarrow$ new cost of $(t, t')$
32     for each $(t, t') \in E^{inc}$ do
33         $next(t) \leftarrow$ null
34     if $E^{dec} \neq \emptyset$ then
35         for each $(s, s')$ in $E^{dec}$ do
36             InsertState ($s, s', Q$)
37         ReestablishConsitency() // not in Improved MPGAA*
38     return $E^{inc} \cup E^{dec} \neq \emptyset$
39 function GoalCondition($s$)
40     while $next(s) \neq null$ and $h(s) = h(next(s)) + c(s, next(s))$ do
41         $s \leftarrow next(s)$
42     return $s_{goal} = s$
43 procedure InsertState($s, s'$)
44     if $h(s) > c(s, s') + h(s')$ then
45         $h(s) \leftarrow c(s, s') + h(s')$
46         $next(s) \leftarrow$ null
47         $support(s) \leftarrow s'$
48         if $s$ in $Q$ then
49             Update priority of $s$ in $Q$ to $h(s)$
50         else
51             Insert $s$ into $Q$ with priority $h(s)$
52 procedure ReestablishConsitency()
53     while $Q$ is not empty do
54         Extract state $s'$ with lowest $h$-value in $Q$
55         for each $s$ such that $s' \in Succ(s)$ do
56             InsertState ($s, s', Q$)
57 procedure main()
58     $Q \leftarrow$ empty priority queue
59     $counter \leftarrow 0$
60     Observe ($s_{start}$)
61     for each state $s \in S$ do
62         $search(s) \leftarrow 0$
63         $h(s) \leftarrow H(s, s_{goal})$
64         $next(s) \leftarrow$ null
65     while $s_{start} \neq s_{goal}$ do
66         $counter \leftarrow counter + 1$
67         $s \leftarrow$ A*($s_{start}$)
68         if $s =$ null then
69             return "goal is not reachable"
70         for each $s' \in$ Closed do
71             $h(s') \leftarrow g(s) + h(s) - g(s')$
72         while $s \neq s_{start}$ do
73             $next(parent(s)) \leftarrow s$
74             $s \leftarrow parent(s)$
75         repeat
76             $t \leftarrow s_{start}$
77             $s_{start} \leftarrow next(s_{start})$
78             $next(t) \leftarrow$ null
79             Move agent to $s_{start}$
80             $restart \leftarrow$ Observe($s_{start}$)
81         until $s_{start} = s_{goal}$ or $restart =$ true
```

**Algorithm 2:** Improved MPGAA*

```
1  procedure InitializeState($s$)
2      if $search(s) \neq counter$ then
3          $g(s) \leftarrow \infty$
4      $search(s) \leftarrow counter$
5      ReestablishConsitency($s$)
6  procedure ReestablishConsitency($t$)
7      while $Q$ is not empty and $h(top(Q)) < h(t)$ do
8          Extract state $s'$ with lowest $h$-value in $Q$
9          for each $s$ such that $s' \in Succ(s)$ do
10             InsertState ($s, s', Q$)
```

with the size of the grid. In fact, the number of updated states is equal to the width of the grid. Worse even, after making several updates to the heuristics, those new $h$-values may play no role in future searches. In our example, observe that the next search expands *only* one state: C5 (Figure 1c). Note also, that had our grid been wider, this second search would have expanded one cell too.

Our second observation is that it is not necessary to focus on re-establishing consistency of the heuristic. This is because neither consistency nor admissibility is required to guarantee A*'s optimality. Theorem 1 by Hart, Nilsson, and Raphael (1968) states (slightly rephrased): "if $h(s) \leq h^*(s)$, for every node $s$, then A* finds an optimal path". Nevertheless, the requirement "$h(s) \leq h^*(s)$, for every node $s$" is not used in the proof but rather a weaker condition, namely that "$h(s) \leq h^*(s)$, for every node $s$ that ever enters in the open list". So with this observation in hand, we conclude that, to preserve optimality, we only need to guarantee that the $h$-value of each node entering Open does not overestimate the true cost to the goal. This is important because it means we can focus only on having $h$-values of states relevant for the search.

One more observation is needed to produce an improved, optimal MPGAA*. When executing RC, the $h$-values of states extracted from the priority queue $Q$ are non-decreasing. Despite this being an obvious remark, following from the fact that $Q$ is ordered by $h$, the important implication is that we can stop RC sooner and run it only when needed. Indeed, assume that we observed changes in the environment, that we added these states to $Q$ (Lines 35-36, Algorithm 1) but that we do not run RC in Line 37. Assume we now start searching for a new path using A* and that we are about to add $s$ to the open list. Here we would be at risk of generating a suboptimal path if $h(s) > h^*(s)$. If such were the case, running RC before adding $s$ to Open would result in setting the $h$-value of $s$ to a lower value. Thus, before

| | Visibility 50 | | | | | Visibility 100 | | | | | Visibility 200 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bucket | A* | D* Lite | iMPGAA* | iMPGAA*-B | Faster | A* | D* Lite | iMPGAA* | iMPGAA*-B | Faster | A* | D* Lite | iMPGAA* | iMPGAA*-B | Faster |
| 1 | 0.38 | 0.39 | 0.40 | **0.36** | 72% | **0.38** | 0.43 | 0.47 | 0.43 | 46% | **0.38** | 0.44 | 0.59 | 0.57 | 13% |
| 2 | 1.34 | 1.26 | 1.32 | **1.12** | 90% | **1.35** | 1.39 | 1.50 | 1.36 | 75% | **1.40** | 1.52 | 1.86 | 1.73 | 33% |
| 3 | 3.62 | 2.51 | 3.01 | **2.30** | 87% | 3.85 | 2.73 | 3.60 | **2.63** | 76% | 4.13 | **3.00** | 4.68 | 3.40 | 40% |
| 4 | 5.24 | 3.87 | 4.42 | **3.46** | 89% | 6.13 | 4.20 | 5.52 | **4.14** | 76% | 7.36 | **4.80** | 8.07 | 5.57 | 49% |
| 5 | 9.85 | 5.55 | 6.57 | **4.93** | 87% | 10.38 | 5.96 | 8.14 | **5.72** | 78% | 12.50 | **6.90** | 11.84 | 7.72 | 52% |
| 6 | 12.01 | 7.59 | 7.60 | **6.58** | 88% | 14.04 | 8.18 | 10.23 | **7.76** | 80% | 14.66 | **9.37** | 13.94 | 10.01 | 56% |
| 7 | 16.81 | 10.31 | 10.00 | **8.47** | 92% | 19.29 | 10.96 | 13.72 | **9.77** | 84% | 22.61 | **12.82** | 19.52 | 12.98 | 61% |
| 8 | 28.67 | 14.82 | 14.55 | **12.13** | 93% | 30.09 | 15.77 | 17.88 | **13.76** | 86% | 36.05 | 18.11 | 26.25 | **17.28** | 70% |
| 9 | 45.88 | 22.42 | 19.74 | **16.95** | 99% | 51.49 | 23.95 | 25.53 | **18.61** | 96% | 60.95 | 27.44 | 37.10 | **23.93** | 83% |
| 10 | 127.43 | 50.01 | 40.61 | **34.35** | 99% | 140.60 | 51.09 | 52.43 | **36.92** | 97% | 163.12 | 57.98 | 76.13 | **45.66** | 86% |

Table 1: Average runtime of algorithm variants.

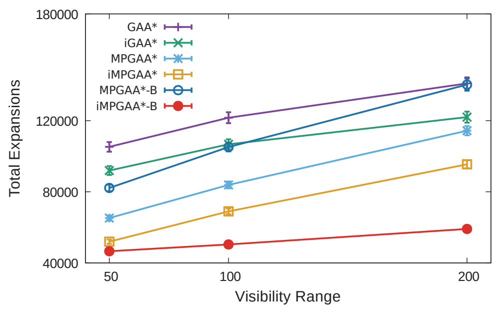

Figure 3: Impact of our technique on expansions.

adding $s$ to Open, assume we run RC. Assume further that eventually the $h$-value at the top of queue $Q$ is greater than $h(s)$. This means that $h(s)$ *will not* decrease if we continue to run RC. We can therefore conclude that $h(s) \leq h^*(s)$, stop the update procedure, and resume search.

Our improved version of MPGAA* replaces two procedures of the original MPGAA*, which are shown in Algorithm 2, and removes Line 37 of Algorithm 1. The rest of the code of MPGAA* remains without modification. The resulting algorithm, Improved MPGAA* is also optimal.

**Theorem 1 (Optimality of Improved MPGAA*)** *If $h$ is initially consistent, the movement performed in Line 79 lies on an optimal path to $s_{goal}$ over graph $G$ with respect to (current) cost function $c$.*

On the practical side, as a result of updating only those states that are needed, Improved MPGAA* may need to update substantially fewer states than MPGAA*. Figure 2 illustrates this for the example described earlier.

**MPGAA*-B** Another improvement we propose comes from the observation that the first search can be done with backward A* (from the goal to the initial state). Afterwards we set $h(s)$ as the $g$-value of $s$ for each state expanded by A*, and resume normal execution of MPGAA*. Besides a more informed heuristic, doing this has the advantage that the backward search creates many paths to the goal state. We call this version MPGAA*-back or MPGAA*-B, for short.

## Experimental Evaluation

The objective of our evaluation was twofold. First, we wanted to investigate the impact of applying our technique over algorithms that use the RC procedure. Second, we wanted to compare with state-of-the-art algorithms. We focus the evaluation on pathfinding tasks in grid-like dynamic terrain. We use eight-neighbor grids with cardinal moves costing 1 and diagonal moves costing $\sqrt{2}$. The heuristic is the octile distance. All algorithms have a common code base and use a standard binary heap for Open and $Q$. Experiments were ran on a 2.60GHz Intel Core i7 under Linux.

We used maps of $512 \times 512$ scaled to $1024 \times 1024$ from the MovingAI repository (Sturtevant 2012). Specifically, we used 4 room maps (Indoor), 4 World of Warcraft III maps (Outdoor) and 4 random maps.[1] In each map we set an additional 5 percent of random obstacles. Like Aine and Likhachev (2016), every 50 movements, 0.5% randomly chosen unblocked cells become blocked and an equal number of the blocked cells are randomly chosen and set as unblocked. We evaluated 3 visibility range values: 50, 100, and 200. For each map we generated 250 random problems.

Figure 3 shows the impact of applying our technique over GAA*, MPGAA* and MPGAA*-B. We evaluate efficiency in terms of total expansions, which includes expansions during search and during updates (the figure also shows standard error bars). Our technique improves every algorithm and more significant improvements are observed for iMPGAA*-B over MPGAA*-B. As the the visibilty ranges increase improvements increase.

Table 1 compares repeated A* (i.e., A* with replanning), D* Lite, iMPGAA* and iMPGAA*-B. We analyzed the efficiency (in average runtime) depending on problem hardness, where we measure the hardness of problem as the runtime that D* Lite needs to solve them. We sort the problems according to hardness, allocating each problem in one of 10 buckets such that each bucket contains the same number of problems (bucket 10 thus contains the hardest problems). Additionally, we show the percentage of instances that iMPGAA*-B is faster than D* Lite in each Bucket. We observe that (1) for small and medium visibility ranges, iMPGAA*-B is faster than D* Lite in all buckets, and (2) for the largest visibility range, D* Lite has a smaller runtime than iMPGAA*-B in some buckets. This may be explained by the fact that iMPGAA*-B needs to carry out more search (farther away from the current state) when the visibil-

[1] 8room_000, 16room_000, 32room_000, 64room_000, blastedlands, dragonfire, duskwood, gardenofwar, random512-10-0, random512-15-0, random512-20-0 and random512-25-0.

ity range is larger. Note that A* can be faster than the other algorithms for simple problems. Overall, we observe that iMPGAA*-B dominates D* Lite because it is faster in most of the problems evaluated.

Due to space, we do not include results per each benchmark, but iMPGAA*-B dominates in all. When aggregating results over visibility ranges we see that cost is similar for all algorithms and that iMPGAA*-B has the best runtime.

## Conclusions

We presented Improved MPGAA*, a variant of MPGAA* that does not maintain the heuristic consistent yet is still optimal. Our exprimental analysis confirms that our techniques improve the efficiency of algorithms that use the `RC` procedure, especially for larger visibility ranges. iMPGAA*-B dominates all algorithms evaluated, including D*Lite.

## Acknowledgements

We acknowledge support from Fondecyt via grants number 1150328 and number 1161526.

## References

Aine, S., and Likhachev, M. 2016. Truncated incremental search. *Artificial Intelligence* 234:49–77.

Hart, P. E.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2).

Hernández, C.; Asín, R.; and Baier, J. A. 2015. Reusing Previously Found A* Paths for Fast Goal-Directed Navigation in Dynamic Terrain. In Bonet, B., and Koenig, S., eds., *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, 1158–1164. AAAI Press.

Koenig, S., and Likhachev, M. 2002. D* lite. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 476–483.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 11th International Conference on Robotics and Automation (ICRA)*, 3310–3317.

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

Sun, X.; Koenig, S.; and Yeoh, W. 2008. Generalized Adaptive A*. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, 469–476.