Received May 28, 2020, accepted June 11, 2020, date of publication June 18, 2020, date of current version July 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3003344

Multipath Adaptive A*: Factors That Influence Performance in Goal-Directed Navigation in Unknown Terrain

CARLOS HERNÁNDEZ ULLOA[1], JORGE A. BAIER[2], AND ROBERTO ASÍN-ACHÁ[3]

[1]Departamento de Ciencias de la Ingeniería, Universidad Andrés Bello, Santiago 8370146, Chile
[2]Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Santiago 8331150, Chile
[3]Departamento de Ingeniería Informática y Ciencias de la Computación, Universidad de Concepción, Concepción 4070386, Chile

Corresponding author: Carlos Hernández Ulloa (carlos.hernandez.u@unab.cl)

This work was supported in part by Fondecyt under Grant 1150328 and Grant 1161526.

ABSTRACT Incremental heuristic search algorithms are a class of heuristic search algorithms applicable to the problem of goal-directed navigation. D* and D*Lite are among the most well-known algorithms for this problem. Recently, two new algorithms have been shown to outperform D*Lite in relevant benchmarks: Multi-Path Adaptive A* (MPAA*) and D*ExtraLite. Existing empirical evaluations, unfortunately, do not allow to obtain meaningful conclusions regarding the strengths and weaknesses of these algorithms. Indeed, in the paper introducing D*ExtraLite, it is shown that D*Lite outperforms MPAA* in benchmarks in which the authors of MPAA* claim superiority over D*Lite. The existence of published contradictory data unfortunately does not allow practitioners to make decisions over which algorithm to use given a specific application. In this paper, we analyze two factors that significantly influence the performance of MPAA*, explaining why it is possible to obtain very different results depending on such factors. We identify a configuration of MPAA* which, in the majority of the benchmark problems we use, exhibits superior performance when compared to both D*Lite and D*ExtraLite. We conclude that MPAA* should be the algorithm of choice in goal-directed navigation scenarios in which the heuristic is accurate, whereas D*ExtraLite should be preferred when the heuristic is inaccurate.

INDEX TERMS MPAA*, D*, D*Lite, D*ExtraLite, incremental heuristic search, goal-directed navigation.

I. INTRODUCTION

Goal-directed navigation, the problem of leading an autonomous agent from an initial location to a goal location over partially known terrain, is an important problem in AI with recognized applications in robotics [1]. The algorithms to solve this problem can by classified in at least two classes: those that represent the terrain as a subset of a continuous two-dimensional space (i.e., sampling-based algorithms [2], [3]), and those that represent the terrain as a graph. While both classes of algorithms are at the core of deployed applications, our focus on this paper is on recently developed incremental heuristic-search algorithms [4]–[6], which fall into the latter class.

Salient among graph-based algorithms for goal-directed navigation are the D* [4] and D*Lite [5] algorithms, which have recently been applied to Transportation Networks [7], Multi-Agent Systems [8], and Multiple Mission Points [9]. Both of them are *incremental heuristic search* algorithms [10], which means that (1) they use heuristic function to guide search at their core, and (2) reuse information gathered in a search episode in subsequent search episodes. D*Lite is simpler to describe and understand than its algorithmically equivalent D*. For this reason, D*Lite is preferred to D* in many applications.

Multi-Path Adaptive A* (MPAA*) [11] and D*ExtraLite [12] are incremental heuristic search algorithms for goal-directed navigation which have been recently proposed. A common factor of both algorithms is that they are simple to understand. MPAA* is a forward-search, A*-based algorithm that replans each time an obstacle is found blocking the way of the current (optimal) path. In each search episode it saves the path towards the goal, and whenever search encounters a state of a previously found path to the goal, search may

The associate editor coordinating the review of this manuscript and approving it for publication was Lubin Chang.

stop early, potentially saving significant search effort. On the other hand D*ExtraLite, like D*Lite, is a backward-search algorithm which searches from the goal state to the initial state, storing in memory the search tree generated in the process. Upon execution, when an obstacle is found to block the current (optimal) path to the goal, D*ExtraLite efficiently prunes its search tree and then extends the remaining search tree, until the current state is reached. This saves time since the search tree does not need to be re-generated. Both algorithms were shown to outperform D*Lite in several standard problem benchmarks.

Both MPAA* and D*ExtraLite were shown—respectively in [11] and [12]—to outperform D*Lite in several problem benchmarks. Indeed, [12] showed that D*ExtraLite was superior to MPAA* in several benchmarks, but most surprisingly showed that MPAA* was outperformed by D*Lite in benchmarks that [11] had shown exactly the opposite. Since MPAA* and D*ExtraLite are representatives of the state of the art, it is currently very hard for a practitioner to decide which algorithm to run, since published data is inconsistent. In a preliminary investigation, we set out to investigate the reason for this inconsistency, and we discovered that while MPAA*'s implementation of [12] is correct, there are factors that influence the performance of MPAA* quite dramatically. These factors explain the apparent inconsistent results published so far. This paper explains what are these factors, and provides a more clear picture to practitioners that may be willing to implement and apply these algorithms.

Specifically, this paper shows an empirical analysis in which vary the two parameters that we found had the most impact on MPAA*'s performance: (1) the heuristic used and (2) the tie-breaking policy for the open list. In our evaluation we use random maps, game maps (which resemble outdoor navigation), room maps (which resemble indoor navigation), and real-city maps. We found that in 4 out of the 7 different map types, MPAA* outperforms all other algorithms. In the remaining 3, D*ExtraLite is the front runner. We conclude that MPAA* performs best when the heuristic function is relatively accurate, and that D*ExtraLite excels when the opposite holds. Also, the tie-braking policy of MPAA* should be set to prefer to expand a node with a higher g-value when there are ties on the f-values, that is, precisely the opposite rule that was used by the implementation of [12].

II. GOAL-DIRECTED NAVIGATION

A goal-directed navigation problem is a tuple $P = (G, C, s_{start}, s_{goal})$, 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. Finally, $C : E \to \mathbb{R}_0^+ \cup \{\infty\}$ is a cost function that associates each arc in G with a non-negative number. To represent the fact that an arc e in E is not traversable we associate a cost ∞ with e.

The objective of the navigation problem is to find a path from s_{start} to s_{goal}. A *path* is a sequence $\sigma = s_0 s_1 \ldots s_n$ where for all $i \in \{1, \ldots, n\}$, it holds that $(s_{i-1}, s_i) \in E$ and $C(s_{i-1}, s_i) < \infty$. The *cost* of a path $s_0 \ldots s_n$ over G is $\sum_{i=0}^{n-1} C(s_i, s_{i+1})$. A path σ is a *solution* to goal-directed navigation problem P if it starts at s_{start} and ends at s_{goal}.

In the rest of the paper we focus on goal-directed navigation over *grids*. An $N \times M$ grid is defined by a pair (C, O), where $C = \{1, \ldots, N\} \times \{1, \ldots, M\}$ is a set of cells, and $O \subseteq C$ is a set of *obstacles*. In 8-neighbor grids, a cell (x, y) is a neighbor of a cell $(x + d_x, y + d_y)$ iff $d_x, d_y \in \{0, 1, -1\}$ and $|d_x| + |d_y| \neq 0$.

A search graph $G = (V, E)$, where V is a set of *states* and E is set of arcs, is constructed from a grid (C, O) by creating a state per each cell (that is, $V = C$), and creating an arc between two vertices iff they are neighbors in the grid. Furthermore, $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ is defined as the Euclidean distance between (x_1, y_1) and (x_2, y_2). The octile distance [13] between (x_1, y_1) and (x_2, y_2) is $\sqrt{2} \max\{\Delta_x, \Delta_y\} + \min\{\Delta_x, \Delta_y\}$, where $\Delta_x = |x_1 - x_2|$ and $\Delta_y = |y_1 - y_2|$.

A. PARTIAL OBSERVABILITY AND LIMITED VISIBILITY RANGE

In the rest of the paper, we focus on goal-directed navigation over *unknown terrain*. When planning in an unknown terrain we assume there is a set of obstacle cells which is unknown by the agent at the outset. We assume the presence of obstacles can only be revealed as the agent becomes close enough to the obstacles. In this setting, it is standard to take a *free space assumption* [14], which means that the agent initially assumes the grid is obstacle-free.

Formally, in the presence of unknown terrain we assume the search graph G is known by the agent but that the cost function C is partially observable. The algorithm keeps a cost function c which stores the agent's version of cost function for the search graph. Because of the free space assumption, $c(s, s')$ is initially defined as the Euclidean distance between s and s' for every arc $\{s, s'\}$ in the search graph. An important observation is that a consequence of using the free space assumption is that function c, during execution, is non-deceasing: only the cost of arcs that touch obstacles may increase as (previously unknown) obstacles are discovered by the agent during execution.

Sensors used by navigating agents may not be able to detect every change in the environment. We say an agent has *visibility* k if the cost function c maintained by the agent is always correct up to visibility k: that is, every arc e contained in a path of k or less arcs that starts from the current state is such that $c(e) = C(e)$.

B. HEURISTIC FUNCTIONS AND A*

The algorithms we describe below are based on heuristic search. Heuristic-search approaches to solving path-planning problems use a heuristic function h. Given a graph $G = (V, E)$ and a goal state s_{goal}, $h : V \to \mathbb{R}$ is such that $h(s)$ is a non-negative estimate of the cost of a path from s to s_{goal}, and such that $h(s_{goal}) = 0$. Function h is *admissible* iff for every state s, $h(s)$ does not overestimate the cost of any path from s to s_{goal}, with respect to a fixed cost function c. Furthermore, we say h is *consistent* iff $h(s_{goal}) = 0$ and for every $(s, s') \in E$ it holds that $h(s) \leq c(s, s') + h(s')$. It is easy to prove that consistency implies admissibility.

A* search can be used to find an optimal path connecting two states of a search graph. It uses a priority queue, called Open, which is initialized to contain the start state. The priority function used for a state s in *Open* is $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of a path found by A* from the start state to s, and $h(s)$ is a heuristic function. At each iteration A* does the following. It extracts a state s from Open. If s is a goal state, then it returns s. Otherwise it *expands* s, generating all the successors of s. For every successor t of s, unless t has been found before via a lower cost path, it sets $parent(t)$ to s, sets $g(t)$ as $g(s) + c(s, t)$, and $f(t) = g(t) + h(t)$, and inserts it into Open if t is not in Open, or reorders Open if t was already there. The path connecting the start state and the goal state is guaranteed to be optimal if h is admissible.

Note that, during and after execution, A*'s attribute *parent* defines a subtree of the search graph which is rooted in the start state. Henceforth we call this tree the *search tree*, which, more precisely is the subgraph of G in which we keep arc (u, v) iff $parent(v) = u$. A pseudocode of A* can be found in Algorithm 1.

III. D*Lite AND D*ExtraLite

D* Lite [15] is one of the most well-known algorithms for goal-directed navigation. Because of its simplicity is preferred to D*. D* Extra Lite [12] is recently proposed variant of D* that is simpler to describe and was shown to be faster than D*Lite [12].

D* Lite and D* Extra Lite are related algorithms proposed for Dynamic Terrain Navigation, of which, unknown terrain navigation is a particular case. In dynamic terrain navigation, obstacles may appear or disappear as the agent moves. Due to these changes, the search-space considered by the agent may become *inconsistent*. For unknown terrain navigation, inconsistent states are those for which the cost of a path to the goal is underestimated. Both D* Lite and D* Extra Lite do a backward search (i.e. finding a path from the goal state to the start state) breaking f-value ties toward smaller g-values and re-plan when a change in the map is detected [12]. In such a re-planning procedure, D*Lite detects, reinitializes, and re-expands states that are inconsistent with respect to the agent's current position. In contrast, D*ExtraLite first prunes the entire branch of the search tree that becomes inconsistent with a efficient recursive procedure, and then reinitializes and re-expands states that are neighbor of the pruned branch. For the case of goal-directed navigation in unknown terrain D*ExtraLite can be described as follows.

1) Let s_{start} be the agent's position.
2) Initialize the Open list with the goal state.
3) Run (backwards) A*, mark as "visited" all states that have been generated. Iterate until s_{start} is at the top of Open. When this happens, do not extract s_{start} from Open and stop search.
4) Perform an action, following the parent of s_{start} in the search tree, and update s_{start} accordingly.

Algorithm 1 Multi-Path Adaptive A* (MPAA*)

```
procedure InitializeState (s)
    if search(s) ≠ counter then
        g(s) ← ∞
    search(s) ← counter
function GoalCondition (s)
    while next(s) ≠ null and h(s) = h(next(s)) + c(s, next(s)) do
        s ← next(s)
    return s_goal = s
function A* (s_init)
    InitializeState (s_init)
    parent(s_init) ← null
    g(s_init) ← 0
    Open ← ∅
    insert s_init into Open with f-value g(s_init) + h(s_init)
    Closed ← ∅
    while Open ≠ ∅ do
        remove a state s from Open with the smallest f-value
        g(s) + h(s)
        if GoalCondition (s) then
            return s
        insert s into Closed
        for each s' ∈ succ(s) do
            InitializeState (s')
            if g(s') > g(s) + c(s, s') then
                g(s') ← g(s) + c(s, s')
                parent(s') ← s
                if s' is in Open then
                    set priority of s' in Open to g(s') + h(s')
                else
                    insert s' into Open with priority g(s') + h(s')
    return null
procedure BuildPath (s)
    while s ≠ s_start do
        next(parent(s)) ← s
        s ← parent(s)
function Observe (s)
    T ←
    arcs in the range of visibility from s whose cost just has changed
    for each (t, t') in T do
        c(t, t') ← new cost of (t, t')
    return T ≠ ∅
procedure main ()
    counter ← 0
    Observe (s_start)
    for each state s ∈ S do
        search(s) ← 0
        h(s) ← H(s, s_goal)
        next(s) ← null
    while s_start ≠ s_goal do
        counter ← counter + 1
        s ← A* (s_start)
        if s = null then
            return "goal is not reachable"
        for each s' ∈ Closed do
            h(s') ← g(s) + h(s) − g(s')
        BuildPath (s)
        repeat
            t ← s_start
            s_start ← next(s_start)
            next(t) ← null // Only necessary in MPGAA*
            Move agent to s_start
            restart ← Observe (s_start)
        until s_start = s_goal or restart = true
```

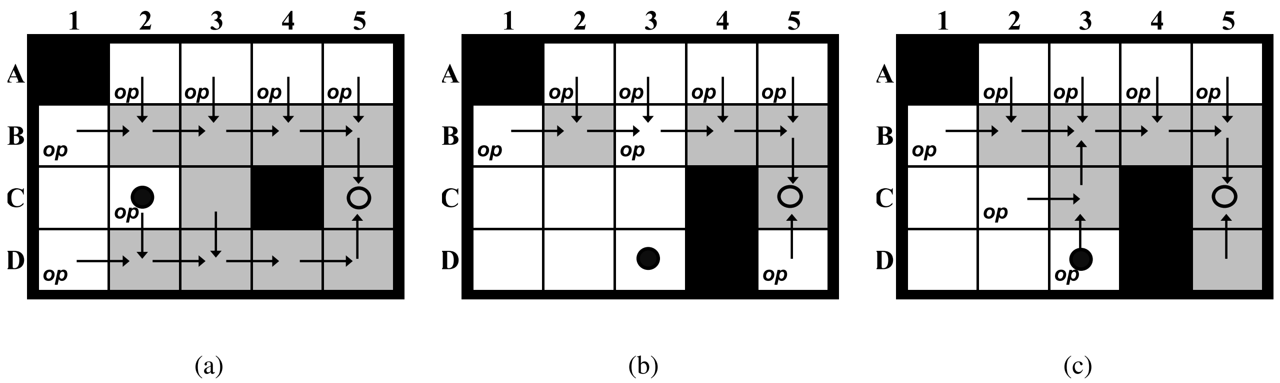

FIGURE 1. Example execution of D*ExtraLite on a 4-connected grid in which the filled circle represents the initial cell and the unfilled circle shows the goal cell. In the initial phase, a backwards search is carried out until the goal state is found. In (a) states in the open list are labeled with *op*, and the arrows represent the search tree of the backwards search. When a new obstacle appears in cell D4, first D*ExtraLite prunes the search tree (b), and then restarts search until the initial state appears at the top of Open.

5) If s_{start} is the goal state, end now.
6) Observe the map. If the cost of an arc (u, v) in the search graph has increased:
    a) Cut all branches starting in u. More precisely, let $s_1, \ldots, s_n$ be a branch of the search tree, where $s_1 = v$. For each $i \in \{1, \ldots, n\}$, assign $parent(s_i)$ to *null*, and remove s_i from Open if s_i was in Open, mark it as not visited. For any visited successor s' of s_i such that $s' \neq s_{i+1}$, add s' to Open.
    b) Go back to Step 3.
7) Go back to Step 4.

In the description above we have omitted the way the priority of nodes is updated between different search episodes. For the full description of D*ExtraLite refer to [12] Figure 1 shows an example of a partial execution D*ExtraLite.

IV. MULTI-PATH ADAPTIVE A*

*Multi-Path Adaptive A** (MPAA*) is an algorithm that builds on *Adaptive A** (AA*) [16], which in turn is an extension of *Repeated A** [17]. We describe these two algorithms before explaining MPAA*.

Repeated A* is straightforward way of using the well-known A* algorithm for goal-directed navigation. In unknown terrain, it runs as follows:

1) Run forward A* to find path from the current state to the goal,
2) Follow the path returned by A*, updating the cost function c with every move, until the goal has been reached or until the path being followed does not reach the goal anymore.
3) If the goal has not been reached go back to Step 1.

Observe that Repeated A* runs a search from scratch each time an obstacle blocks the path to the goal, and does not exploit any information gathered during search in subsequent search episodes. GAA*, instead, uses information gathered over a search episode in subsequent episodes. It turns out that after search has stopped (Step 1 of Repeated A*), it is possible to update the heuristic function h to make it more informed. AA* modifies Repeated A* by adding a *heuristic update* immediately after Step 1 that is such that $h(s) := f^* - g(s)$, for every state s in A*'s *Closed* list, where f^* is the cost of the solution found by A*. This update makes h more informed, and guarantees that if h was previously consistent, it remains consistent (and admissible).

MPAA* further extends AA* by exploiting, during search, even more information from previous search episodes. It turns out that each search episode computes a path to the goal. If in a subsequent search episode a state s is selected for expansion, and such an s was part of a path to the goal found in a previous search episode, then MPAA* (a) runs a quick test for optimality of the remaining path (b) if such a test succeeds, it stops search immediately returning the path to s concatenated with the (previously found) path from s to s_{goal}.

The pseudocode of MPAA* is presented in Algorithm 1. Now we describe further details. Procedure `main()` first initializes relevant variables (Lines 41–46). Variable *counter* keeps track of how many calls to A* have been carried out so far. Variable *search*(s), for each state s, stores the number of the last A* call that generated s. It is equal to 0 when s has not been generated. Variable $h(s)$ contains the h-value of state s, and it is initialized with $H(s, s_{goal})$, which should be a value that does not overestimate the cost of an optimal path between states s and s_{goal} with respect to the initial cost function c. $H(s, s')$ corresponds to the octile distance between s and s'. Finally, variable $next(s)$ points to the next state of a path found by A* from s to s_{goal}.

After initialization, the main loop (Lines 47–61) runs until the current state—which is kept in variable s_{start}—becomes equal to the goal state. It calls A* (Line 49) to find a new path from the current state to the goal and, unless unsuccessful, it builds a path to the goal by calling procedure `BuildPath` (Line 54).

A* (Lines 9–30), is a standard pseudocode for A* over grids, except for the use of function `GoalCondition` to stop search. `GoalCondition` is the core of MPAA*.

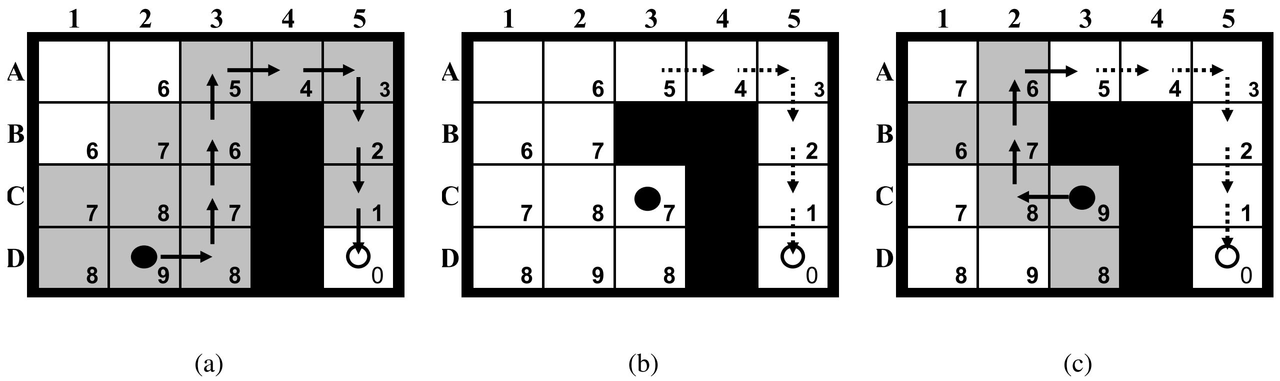

FIGURE 2. Example partial execution of MPAA* on a 4-connected grid in which the filled circle represents the initial cell and the unfilled circle shows the goal cell. In the initial phase, a forward search is carried out until the goal state is found. In (a) states that were expanded are shaded in gray, and the arrows represent the branch of the search tree that reaches the goal. In (b) dotted arrows represent the path to the goal that is kept in memory. When a new obstacle appears in cell B3, MPAA* carries out an A* search that stops as soon as A3 is extracted from Open; only 6 nodes are expanded (shown in gray).

It traverses the path from the argument state s to the goal, returning true iff these two conditions hold: (1) the goal is reachable from s via the $next(s)$ pointers, and (2) the heuristic on such a path is still perfect for the current cost function c.

After each A* search, MPAA* *updates* the h-values of all states in A*'s closed list, making them more informed (Lines 52–53). Specifically, for each state s' in the closed list it sets $h(s')$ to $g(s) + h(s) - g(s')$, where s is the state returned by the previous A* search. It is not hard to prove that such an update cannot decrease $h(s')$ (and usually increases it) if the heuristic is initially consistent [18]. Furthermore, this update has a desirable property: if the heuristic was consistent, it remains consistent after the update [11]. Consequently, optimality is guaranteed in subsequent searches. This is the same update procedure used by AA*.

Finally, the loop of Lines 55–60, is a movement phase, in which the agent moves following the path previously built. This loop exits as soon as the goal is reached or as soon as variable *restart* becomes true. This variable is set by the procedure `Observe`, which, in this version of Repeated A*, returns true if and only if an arc in the search graph changed its cost.

Figure 2 shows an example partial execution.

V. EXPERIMENTAL EVALUATION

Our experimental evaluation had two objectives: first we wanted to understand the factors that influence the performance of MPAA* on goal-directed navigation tasks. Second, We also wanted compare the various configurations of MPAA* to D*Lite and D*ExtraLite.

To achieve our objectives, we designed two sets of experiments. In the first set, we evaluate the impact of using the Euclidean distance versus the octile distance as a heuristic. In the second set, we evaluate the impact of changing the strategy used to break ties in the Open list. Specifically, we consider the following variants of MPAA*:

- MPAA*+g. This variant breaks ties toward larger g-values (MPAA*+g); that is, when the Open list contains two states with the same f-value, the state with larger g-value is preferred for expansion.
- MPAA*-g. Here, MPAA* breaks ties toward smaller g-values (MPAA*-g)
- MPAA*-FIFO: breaks ties according to the order of insertion in the Open. If two states have the same f-value then the state that was inserted first to the Open list is preferred for expansion.

In our study we do not consider different tie breaking rules for D*Lite or D*ExtraLite. The reason is that these algorithms were specifically designed to break ties towards smaller g-values. The version of D*Lite that we use corresponds to the optimized D*Lite algorithm presented in [15], which breaks ties toward smaller g-values. D*ExtraLite, as described by their authors [12], also breaks ties towards smaller g-values. In addition we believe that changing the tie breaking strategy for these algorithms may affect the algorithms' theoretical properties. The implementation of the three algorithms use the same C++ code base of D*ExtraLite in [12], which is available at https://bitbucket.org/maciej_przybylski/heuristic_search.

The terrain is represented as an eight-neighbor grid. We use eight-neighbor grids because this is the connectivity used by [12] and because they these grids are often preferred by the search-base path planning community (e.g., [15], [19]). The cost of orthogonal moves is 1 and the cost of diagonal moves is $\sqrt{2}$.

To test the algorithms we used pathfinding problem instances in Nathan Sturtevant's repository [20]. In particular we considered 10,000 instances in each of the following sets of maps.

- *Real-world maps*. This set contains maps for real cities. The size of each map is between 256×256 and 1024×1024. To the best of our knowledge, this is the first time

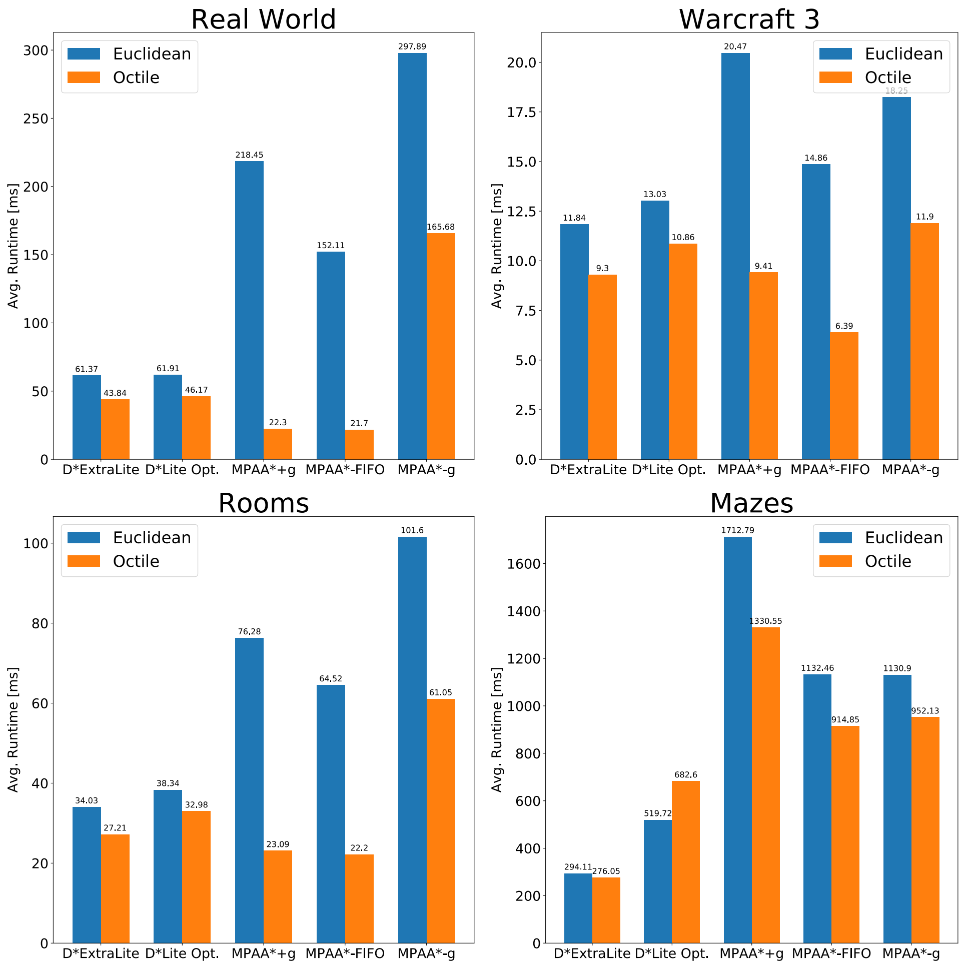

FIGURE 3. **Average runtime with different heuristics for four different scenarios.**

that this set of maps is used to compare these 3 algorithms.

- Random maps. This set contains maps with randomly placed obstacles, which are usually considered when evaluating algorithms for goal-directed navigation (e.g., [15], [17]). We used the 512 × 512 grids with 10% and 40% obstacle ratio.
- Warcraft III. Game maps of size 512 × 512 that can be regarded as good simulation scenarios for outdoor navigation.
- Room maps of size 512 × 512, which can be regarded as good simulation scenarios for indoor navigation.
- Starcraft maps of sizes ranging from 512 × 512 to 1024 × 1024.
- Mazes maps of size 512 × 512. Mazes constitute challenging scenarios for path finding algorithms, even more if the scenario is unknown.

Since the terrain is assumed to be unknown, we use the free-space assumption [14] to compute initial paths. In addition, we set the visibility range of the agent to 10, unless otherwise specified.

Figure 3 shows the average runtime of the algorithms using both the Euclidean and the octile distance, in the four scenarios. For all algorithms, we observe runtime is improved when using the octile distance instead of the Euclidean distance. The only exception is maze maps with D*Lite, where the Euclidean distance leads to best performance. We also observe that the runtime improvement is larger for the MPAA* variants than for D*Lite and D*ExtraLite. MPAA*+FIFO obtains superior performance in real-world maps.

Since best results across most configurations are obtained when using the octile distance as heuristic, in the rest of our evaluation we use the octile distance as heuristic.

Table 1 shows the average runtime, average number of searches and average solution cost of all algorithms using the Octile distance. In contrast to the results reported in [12], the variants of MPAA*, when used with the octile heuristic, outperform the other algorithms in four of the scenarios: Random 10%, Real World, Rooms and Warcraft 3 maps. In Mazes and Random 40% maps, D*ExtraLite shows superior average time. Finally, D*ExtraLite and MPAA*-FIFO show similar behavior on the Starcraft maps.

TABLE 1. **Average runtime, number of searches and costs comparison on standard benchmarks.**

	Algorithm	Runtime	Searches	Cost
Rand. 10%	DExtraLite	11.48	21056	348.89
	DLiteOpt	10.74	21174	348.89
	MPAA*+g	**3.16**	**3158**	**347.82**
	MPAA*-FIFO	4.99	11619	348.27
	MPAA*-g	23.46	93275	348.31
Rand. 40%	DExtraLite	**91.63**	**165077**	3620.77
	DLiteOpt	134.45	251779	3620.78
	MPAA*+g	128.06	321516	3603.50
	MPAA*-FIFO	100.35	282366	3611.83
	MPAA*-g	135.51	491543	**3597.40**
Mazes	DExtraLite	**276.05**	**490533**	12568.22
	DLiteOpt	682.60	919867	12568.22
	MPAA*+g	1330.55	3485724	12577.71
	MPAA*-FIFO	914.85	2788419	12547.68
	MPAA*-g	952.13	3535190	**12502.02**
Starcraft	DExtraLite	**59.32**	**121121**	1232.63
	DLiteOpt	78.95	150424	1232.63
	MPAA*+g	77.79	192747	1234.56
	MPAA*-FIFO	60.72	182371	1234.28
	MPAA*-g	179.58	680469	**1228.53**
R. World	DExtraLite	43.84	95756	732.46
	DLiteOpt	46.17	101765	732.46
	MPAA*+g	22.30	**42681**	**731.07**
	MPAA*-FIFO	**21.70**	55091	738.16
	MPAA*-g	165.68	5822735	731.49
Rooms	DExtraLite	27.21	60789	649.81
	DLiteOpt	32.98	71580	649.81
	MPAA*+g	23.09	**59643**	658.47
	MPAA*-FIFO	**22.20**	72437	647.35
	MPAA*-g	61.05	263795	**628.34**
Warcraft 3	DExtraLite	9.30	20017	313.82
	DLiteOpt	10.86	22405	313.82
	MPAA*+g	9.41	23171	317.99
	MPAA*-FIFO	**6.39**	**18897**	313.51
	MPAA*-g	11.90	46155	**312.27**

TABLE 2. **Average normalized runtime, number of searches and costs comparison on standard benchmarks.**

	Algorithm	Norm. Runtime	Norm. Searches	Norm. Cost	Perc. Wins
Rand. 10%	DExtraLite	3.31	6.09	**1.01**	0.12
	DLiteOpt	3.34	6.00	**1.01**	0.00
	MPAA*+g	**1.00**	**1.00**	**1.01**	**96.58**
	MPAA*-FIFO	1.61	3.89	**1.01**	5.02
	MPAA*-g	5.70	23.46	**1.01**	2.72
Rand. 40%	DExtraLite	2.12	2.60	1.06	16.91
	DLiteOpt	2.71	3.40	1.06	0.01
	MPAA*+g	1.35	**1.60**	**1.05**	**59.19**
	MPAA*-FIFO	**1.32**	2.02	1.06	24.65
	MPAA*-g	2.13	4.89	**1.05**	4.06
Mazes	DExtraLite	**1.39**	**1.43**	1.14	**60.41**
	DLiteOpt	2.76	2.24	1.14	0.00
	MPAA*+g	5.52	6.78	1.15	15.33
	MPAA*-FIFO	3.84	5.78	**1.13**	21.50
	MPAA*-g	4.22	8.07	**1.13**	4.71
Starcraft	DExtraLite	3.00	5.92	**1.17**	11.97
	DLiteOpt	3.41	6.33	**1.17**	0.78
	MPAA*+g	**1.95**	**2.12**	**1.17**	**69.68**
	MPAA*-FIFO	2.25	5.21	1.19	18.60
	MPAA*-g	6.81	21.40	1.18	2.64
R. World	DExtraLite	3.66	11.75	1.08	3.83
	DLiteOpt	3.85	11.98	1.08	0.72
	MPAA*+g	**1.20**	**1.21**	**1.07**	**78.23**
	MPAA*-FIFO	1.68	4.91	1.08	20.04
	MPAA*-g	9.03	41.15	**1.07**	2.05
Rooms	DExtraLite	2.45	3.07	1.09	9.79
	DLiteOpt	2.90	3.49	1.09	0.10
	MPAA*+g	**1.29**	**1.30**	1.10	**70.35**
	MPAA*-FIFO	1.51	2.42	1.08	20.28
	MPAA*-g	3.97	9.28	**1.06**	4.26
Warcraft 3	DExtraLite	3.28	7.65	**1.11**	5.29
	DLiteOpt	3.68	7.94	**1.11**	0.80
	MPAA*+g	**1.39**	**1.45**	1.12	**83.22**
	MPAA*-FIFO	1.83	4.71	1.12	14.40
	MPAA*-g	3.73	14.79	1.12	6.40

Algorithms that perform best in terms of runtime almost always are the ones which perform fewer searches. An interesting exception are MPAA*+g and MPAA*+FIFO in the Real World scenario, where the former replans less on average, but is outperfomed by the latter in terms of runtime. Regarding solution cost, one interesting observation is that, although consistently slower than the others, MPAA*-g usually finds shortest paths. For instance in rooms MPAA*-g is 3.5% shorter than D*ExtraLite and D*Lite, and 4.8% and 3.0% shorter than MPAA*+g and MPAA*-FIFO, respectively.

This difference in performance across different benchmarks may be attributed to the accuracy of the heuristic. Indeed in the Mazes and Random 40%, the octile distance is quite inaccurate.

Table 1 shows averaged magnitudes over different experiments. Problems that require higher runtimes and number of searches may dominate the averages and lead us to wrong conclusions. In our benchmarks, such problems are those where the start and goal states are far apart, problems which belong to larger maps and/or problems where the search task is harder.

To obtain a better comparison, Table 2 shows the normalized runtime, searches and path cost. To compute normalized data, for each problem instance we calculate the minimum runtime, minimum number of searches and minimum path costs. Then, for each algorithm, we divide the achieved number by the corresponding minimum. Then we average these numbers across all instances of the same problem set. For an algorithm A, this average indicates by what factor A is outperformed by the other algorithms. As such, when the average is 1, this means A outperforms all other algorithms in the selected metric. Table 2 also includes ''percentage of wins'', which indicates the percentage of times that the algorithm runs faster than the others. For this indicator we declare as winner all algorithms that are 1% above the minimum runtime; therefore, we may declare more than one winner for a single instance (therefore percentages do not add up to 100%).

Contrasting the data in Tables 1 and 2 for the Starcraft scenario, when we consider average runtime (Table 1), the best performing algorithms are: D*ExtraLite, followed by MPAA*-FIFO, followed by MPAA*+g. In contrast, when considering normalized data, the order changes to: MPAA*+g, followed by MPAA*-FIFO and D*ExtraLite. This can be explained partially because MPAA*+g is the best performing algorithm on up to 69.68% of the cases, followed by MPAA*-FIFO with 18.60% and D*ExtraLite with

TABLE 3. Heuristic accuracy for each map type, as defined and computed in [20].

Map Type	Heuristic Accuracy
Random 10%	0.99
Warcraft 3 (512)	0.84
Rooms (avg. across all sizes)	0.82
Starcraft	0.78
Random 40%	0.46
Mazes (avg. across all corridors)	0.32
Real World	N/A

TABLE 4. Average normalized runtime comparison on standard benchmarks with different visibility ranges.

	Algorithm	Norm. Runtime Vis. 2	Norm. Runtime Vis. 10	Norm. Runtime Vis. 20
	DExtraLite	3.4	3.31	3.23
	DLiteOp t	3.14	3.34	3.66
Rand. 10%	MPAA*+g	**1.01**	**1**	**1**
	MPAA*-FIFO	1.63	1.61	1.62
	MPAA*-g	6.79	5.7	4.76
	DExtraLite	1.74	2.12	2.14
	DLiteOpt	2.13	2.71	2.77
Rand. 40%	MPAA*+g	2.09	1.35	1.39
	MPAA*-FIFO	**1.62**	**1.32**	**1.33**
	MPAA*-g	2.77	2.13	2.06
	DExtraLite	**1.39**	**1.39**	**1.42**
	DLiteOpt	2.59	2.76	2.9
Mazes	MPAA*+g	12.87	5.52	3.69
	MPAA*-FIFO	7.97	3.84	2.76
	MPAA*-g	7.34	4.22	3.33
	DExtraLite	2.7	3	3.11
	DLiteOpt	2.96	3.41	3.62
Starcraft	MPAA*+g	**2.45**	**1.95**	**1.53**
	MPAA*-FIFO	2.68	2.25	1.91
	MPAA*-g	10.43	6.81	5.52
	DExtraLite	3.64	3.66	3.65
	DLiteOpt	3.63	3.85	3.97
R. World	MPAA*+g	**1.33**	**1.2**	**1.13**
	MPAA*-FIFO	1.8	1.68	1.65
	MPAA*-g	12.53	9.03	8.06
	DExtraLite	2.36	2.45	2.48
	DLiteOpt	2.53	2.9	3.18
Rooms	MPAA*+g	1.66	**1.29**	**1.13**
	MPAA*-FIFO	**1.5**	1.51	1.45
	MPAA*-g	4.93	3.97	3.46
	DExtraLite	3.01	3.28	3.32
	DLiteOpt	3.2	3.68	3.86
Warcraft 3	MPAA*+g	**1.64**	**1.39**	**1.26**
	MPAA*-FIFO	2.07	1.83	1.72
	MPAA*-g	5.53	3.73	3.26

11.97%. This happens because across the 10, 000 instances used in this scenario the distance between the start and goal states may greatly differ.

Integrating the information provided by Tables 1 and 2, we observe that for the large majority of the experiments, MPAA*+g tends to perform better. Nevertheless, on those experiments where it does not, the algorithm may exhibit a substantial performance degradation. For example, in mazes MPGAA*+g's average performance is 552% worse than that of the best performing algorithm, which is usually D*ExtraLite.

Table 3 contains the *heuristic accuracy* measure obtained for our benchmark maps by [20]. This measure is obtained experimentally by averaging the ratio between the heuristic value and the actual cost of a path for a number of random planning instances of medium difficulty. We observe that the scenarios in which MPGAA+g performs best are those in which the heuristic is more accurate. This finding is consistent with that observed before: that a better heuristic has a very important impact on the performance of MPGAA+g. Performance of MPGAA+g, relative to D*Lite and D*Extralite, degrades significantly as the heuristic is less accurate. While breaking ties towards states with larger g values leads to best performance most times, the MPGAA*'s FIFO policy seems more robust to performance degradation due to heuristic inaccuracy.

On our experiments, we also wanted to explore the influence of the visibility range of the agent on the different algorithms. Table 4 shows the normalized runtimes for visibility values of 2, 10 and 20. In general, the trend seen on the previous results prevail. Again, we can observe that MPAA*+g is the best suited algorithm for most of the cases, increasing its advantage as the visibility range grows. On the opposite side, if the search problem is hard and the visibility range is small, as is the case in scenarios like mazes with a visibility range of 2, its performance sees a very important misbehavior, being up to, on average, 12.87 times slower than the best performing algorithm.

VI. SUMMARY AND CONCLUSIONS

We presented an empirical study of the effect of varying two search parameters over the performance of MPAA*, and compared the resulting configurations with D*Lite, and D*ExtraLite over a number of goal-directed navigation benchmark problems over grids. Our conclusions are as follows.

- While the choice of a more accurate heuristic improves the performance of all algorithms, we observed that MPAA* was more sensitive to this parameter since its search performance improved more substantially when the octile distance is used compared to the Euclidean distance. This explains the apparently contradictory results reported by [11] and [12]. The experiments on the latter paper used the Euclidean distance as heuristic, whereas the former used the octile distance as a heuristic. As we have seen, the choice of the heuristic impacts the performance of MPAA* so heavily that it can either outperform all other algorithms or be significantly slower.
- The tie-breaking policy used with MPAA* has an important impact on its runtime. In most of the grid maps evaluated, MPAA*+g has the best runtime. Only in maps where the heuristic values are inaccurate (maze, random 40%, and starcraft maps) MPAA*-FIFO is faster than the other versions of MPAA*.
- Our results showed that D*ExtraLite with the Octile distance is the best choice when the heuristic is very inaccurate (maze, random 40% maps). On the hand, in most of the grid maps, including Real-Word maps, MPAA*+g with the Octile distance is the best choice. These results suggests that D*ExtraLite should be preferred

in scenarios in which the heuristic is inaccurate, while MPGAA*+g is the right choice for environments in which the heuristic is more accurate.

REFERENCES

[1] S. Edelkamp and S. Schrödl, *Heuristic Search: Theory and Applications*. San Mateo, CA, USA: Morgan Kaufmann, 2011.

[2] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001, doi: 10.1177/02783640122067453.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011, doi: 10.1177/0278364911406761.

[4] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Proc. 14th Int. Joint Conf. AI IJCAI*, Aug. 1995, pp. 1652–1659.

[5] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proc. 15th Int. Conf. Automated Planning Scheduling ICAPS*, 2005, pp. 262–271.

[6] D. Ferguson and A. Stentz, "Field D*: An interpolation-based path planner and replanner," in *Robotics Research*. Berlin, Germany: Springer, 2007, pp. 239–253.

[7] J. Hartley and W. Alhoula, "Fast replanning incremental shortest path algorithm for dynamic transportation networks," in *Proc. Intell. Comput. Comput. Conf.* Cham, Switzerland: Springer, 2019, pp. 22–43.

[8] K. M. Harvey, "Performance of A* based search algorithms in grid-based stochastic multi-agent systems with static obstacles and non-static goal states," Portland State Univ., Portland, OR, USA, Tech. Rep., 2019, doi: 10.15760/honors.737.

[9] H. Lee, W. Chang, and H. Jang, "Optimal path planning algorithm for visiting multiple mission points in dynamic environments," *J. Korean Soc. Aeronaut. Space Sci.*, vol. 47, no. 5, pp. 379–387, May 2019.

[10] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in AI," *AI Mag.*, vol. 25, no. 2, p. 99, 2004.

[11] C. Hernández, J. A. Baier, and R. Asín, "Making A* run faster than D*-lite for path-planning in partially known terrain," in *Proc. 24th Int. Conf. Automated Planning Scheduling ICAPS*, 2014, pp. 504–508.

[12] M. Przybylski and B. Putz, "D* extra lite: A dynamic A* with search-tree cutting and frontier-gap repairing," *Appl. Math. Comput. Sci.*, vol. 27, no. 2, p. 273, 2017.

[13] N. R. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement," in *Proc. 20th Nat. Conf. AI AAAI*, 2005, pp. 1392–1397.

[14] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Autom.*, vol. 8, no. 6, pp. 707–717, Dec. 1992.

[15] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.

[16] S. Koenig and M. Likhachev, "A new principle for incremental heuristic search: Theoretical results," in *Proc. 16th Int. Conf. Automated Planning Scheduling ICAPS*, 2006, pp. 402–405.

[17] S. Koenig and M. Likhachev, "D* lite," in *Proc. 18th Nat. Conf. AI AAAI*, 2002, pp. 476–483.

[18] S. Koenig and M. Likhachev, "Real-time adaptive A*," in *Proc. 5th Int. Joint Conf. Auto. Agents Multi Agent Syst. AAMAS*, 2006, pp. 281–288.

[19] V. Bulitko, Y. Björnsson, N. Sturtevant, and R. Lawrence, "Real-time heuristic search for game pathfinding," in *Applied Research in Artificial Intelligence for Computer Games*. New York, NY, USA: Springer, 2011.

[20] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, Jun. 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf

CARLOS HERNÁNDEZ ULLOA received the bachelor's degree from the Universidad de Concepción, Chile, and the Ph.D. degree in computer science from the Universidad Autónoma de Barcelona, Spain. He is currently a Full Professor with the Department of Engineering Sciences, Universidad Andrés Bello de Chile (UNAB). His research interests include heuristic search, automated planning, and knowledge representation, with a focus on real-time, on-line, and multiobjective problems. He is also the President of the Chilean Association of Computer Science (SCCC), for the duration of 2019–2020.

JORGE A. BAIER received the bachelor's and master's degrees from the Pontificia Universidad Católica de Chile (PUC), Chile, and the Ph.D. degree in computer science from the University of Toronto, Canada. He is currently an Associate Professor with the Department of Computer Science, PUC. He is also an Associate Dean of engineering education with the School of Engineering, PUC. His research interests include automated planning, heuristic search, and knowledge representation, with a focus on the use of learning techniques in these areas. He is a member of the Association for the Advancement of Artificial Intelligence (AAAI).

ROBERTO ASÍN-ACHÁ was born in Cochabamba, Bolivia, in 1979. He received the B.S. degree in software engineering from the Catholic University of Bolivia, Cochabamba, in 2003, and the Ph.D. degree in computer science from the Technical University of Catalonia, Barcelona, Spain, in 2010. From 2011 to 2015, he was an Assistant Professor with the Catholic University of Concepción, Chile. During 2016, he was a Visiting Scholar with the IEOR Department, University of California at Berkeley, Berkeley, CA, USA. He has been an Assistant Professor with the University of Concepción, Concepción, Chile, since 2016. Up to date, he has coauthored several conference and journal articles in the areas of SAT, constraint satisfaction solving, timetabling, and combinatorial optimization in general.