Minimizing State Exploration While Searching Graphs with Unknown Obstacles

Daniel Koyfman
Ben-Gurion University
Beer Sheva, Israel
koyfdan@post.bgu.ac.il

Shahaf S. Shperberg
Ben-Gurion University
Beer Sheva, Israel
shperbsh@bgu.ac.il

Dor Atzmon
Bar-Ilan University
Ramat Gan, Israel
dor.atzmon@biu.ac.il

Ariel Felner
Ben-Gurion University
Beer Sheva, Israel
felner@bgu.ac.il

ABSTRACT

We address the challenge of finding a shortest path in a graph with unknown obstacles where the exploration cost to detect whether a state is free or blocked is very high (e.g., due to sensor activation for obstacle detection). The main objective is to solve the problem while minimizing the number of explorations. To achieve this, we propose MXA*, a novel heuristic search algorithm based on A*. The key innovation in MXA* lies in modifying the heuristic calculation to avoid obstacles that have already been revealed. Furthermore, this paper makes a noteworthy contribution by introducing the concept of a *dynamic heuristic*. In contrast to the conventional static heuristic, a dynamic heuristic leverages information that emerges during the search process and adapts its estimations accordingly. By employing a dynamic heuristic, we suggest enhancements to MXA* based on real-time information obtained from both the open and closed lists. We demonstrate empirically that MXA* finds the shortest path while significantly reducing the number of explored states compared to traditional A*. The code is available at https://github.com/bernuly1/MXA-Star.

KEYWORDS

A*, Minimizing Exploration, Unknown Obstacles

ACM Reference Format:
Daniel Koyfman, Shahaf S. Shperberg, Dor Atzmon, and Ariel Felner. 2024. Minimizing State Exploration While Searching Graphs with Unknown Obstacles. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

The well-known A^* algorithm [6] optimally solves the shortest path problem by executing a best-first search, guided by a heuristic that estimates the cost to the goal. In this paper, we focus on finding a shortest path in a special type of graphs, denoted as *graphs with unknown obstacles* (GUO). In GUO, the structure of the graph is known, but some states might be blocked. For example, some junctions in a roadmap might be closed due to weather conditions, construction, or accidents. Similarly, some servers/switches/hubs may be unavailable when routing files over a network. Another example is grids (of different dimensions and connectivity) where the size of the grid is known, but it is not known which cells are free and which are blocked. Naturally, all incident edges of a blocked state are also blocked and can all be removed from the graph for our search task. We assume that the status of a state (blocked/free) remains fixed for the entire duration of the search and the execution.

Identifying whether a state is free or blocked in a GUO requires an *exploration* operation, which may come with a cost. For instance, exploring a location using sensors in robotic navigation might be expensive or time-consuming. Additionally, exploration is costly when privacy needs to be preserved, i.e., when an adversary can detect exploration operators. We thus differentiate between an *exploration*, a real-world, possibly very costly, operation, and an *expansion*, a computational operation done in the CPU and only incurs time overhead. Usually, analysis on A* does not differentiate between exploration and expansion, and treats exploration as part of the expansion processes (i.e., when a node is expanded, all neighboring states are generated and explored). Therefore, A* aims to speed up the search by minimizing the number of node expansions. By contrast, this paper aims to find the shortest path while minimizing the number of explorations, even at the price of increasing the number of node expansions. Note that when the exploration operator is very costly (time-consuming) compared to the time of the expansion operator, then reducing the number of explorations is a better way to minimize the CPU time to find a solution than reducing the number of expansions (as done by A*).

In the first part of the paper, we introduce *Minimize Exploration A** (MXA*), a two-level search algorithm capable of searching any GUO. The high level of MXA* runs A* to find an optimal path from $start$ to *goal*. Once it reaches a state for the first time, it *explores* it. Thus, information about which states are free and which are blocked is continuously being collected. The low level calculates a heuristic for the high level by finding the shortest path to *goal* on the *currently known graph*, bypassing known blocked states while assuming that all states that are yet unexplored are free. The resulting heuristic is more informed than classic heuristics, which assume that *all* states are free. Our low-level heuristic reduces the number of nodes expanded by the high-level A* and, as a result, reduces the number

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

of explorations. Naturally, the tradeoff is a large number of low-level expansions. Experimental results show that MXA* reduces the number of explorations by up to an order of magnitude compared to plain A* on a number of common benchmark domains.

The second part of our paper introduces the new notion of a *dynamic heuristic*. Felner et al. [3] showed that, for calculating a heuristic, A* can avoid paths to *goal* that pass through the Closed list, and still return optimal solutions. Thus, a heuristic within A* may treat states in Closed as blocked. We extend this result also to include some of the Open list. Furthermore, we introduce a unifying view defining and analyzing the notion of a *dynamic heuristic*. The idea of dynamic heuristics is general and holds potential for broad applicability. In this paper, as a first step towards using a dynamic heuristic, we exploit this idea by incorporating a dynamic heuristic into the low-level search of MXA*. Our experimental results show that our improved low level significantly reduces the number of expansions at the low level.

2 DEFINITIONS AND BACKGROUND

A *graph with unknown obstacles* (GUO) $G = (V, E, c, EXP)$ consists of a set of states V (or vertices), a set of (directed) edges $E \subseteq V \times V$, a cost function $c(e)$ for traversing an edge $e \in E$ ($c : E \rightarrow \mathbb{R}^+$), and an exploration function $EXP : V \rightarrow \{free, blocked\}$. State $s_2 \in V$ is a neighbor of state $s_1 \in V$ if $(s_1, s_2) \in E$. A neighboring function $N(s)$ receives a state s and returns all its neighbors. Each state in V is either *free* or *blocked*, and we assume that the status of states remains fixed for the entire duration of the search and the execution. Whether a state is free or blocked is not given as input. Instead, the exploration function *EXP* receives a state and returns whether it is free or blocked. Naturally, if a state is blocked, then all its incident edges are also blocked. A state that *EXP* has yet been executed on it is referred to as *unknown*.

The input to a *GUO-pathfinding* problem consists of a GUO $G = (V, E, c, EXP)$, a start state $start \in V$, a goal state $goal \in V$, and a heuristic function $h : V \rightarrow \mathbb{R}^+$. A *valid path* between two states s_1 and s_2 is a sequence of *neighboring free states* that starts with s_1 and ends with s_2. A path's cost is the sum of the costs of its edges. $d(s_1, s_2)$ denotes the cost of the shortest valid path between s_1 and s_2. The heuristic function $h(s)$ estimates $d(s, goal)$ for any given state s. A solution to the problem is a shortest valid path between $start$ and $goal$, whose cost $d(start, goal)$ is denoted by C^*. The problem we solve in the paper is the *minimize exploration shortest-path problem* (MXSP), where the input is a GUO-pathfinding problem, and the task is to find a shortest valid path while minimizing the number of explore operations (i.e., calls to *EXP*).

Some works have previously investigated methods for pathfinding in unknown environments. Zelinsky [18] and Foux et al. [5] suggested planning a robot's path using known information and dynamically replanning when obstacles are encountered during execution. To enhance the local information collected by robots during execution, Chou et al. [1] proposed performing a remote look-ahead verification to acquire additional data while the robot progresses. For the case of planning paths for multiple agents, Shofer et al. [14] computed a plan graph to capture all possible locations of unknown obstacles in advance, before execution. In addition to the unknown environment, other studies assumed that the environment is dynamic and changes over time while the agent moves [10, 12, 19]. This is in contrast to our current paper; we assume a *static* environment, where paths are calculated offline. Moreover, the aim of our paper is to minimize exploration, while the primary aim of all the aforementioned works is to minimize other objectives.

The minimization of state exploration has also been studied, although in a context distinct from ours. *Physical-A** (PHA*) [4] focuses on finding the shortest path, considering that a physical agent must physically reach each state to explore it. Additionally, Stern et al. [16] searched for a k-Clique in unknown graphs. In that scenario, each exploration operation reveals the neighbors of a state and incurs a specific cost. The primary objective was to discover a k-Clique while minimizing exploration. Although these works share the goal of minimizing exploration, neither of them directly applies to our specific problem.

3 SOLVING MXSP WITH A*

The A* algorithm [6] solves the standard pathfinding problem. Here, we adapt A* to fit the GUO-pathfinding problem. The pseudocode is given in Algorithm 1. Lines 9-13 and 28-29 should be omitted; they will be later introduced when describing MXA*. A* maintains two sets of nodes Open and Closed, as well as the set Blocked containing all states revealed as blocked. Each node n in the search is composed of a state s ($n.s$), a g-value ($g(n)$), an f-value ($f(n)$), and a back pointer p ($n.p$) to its predecessor. The g-value is the cost of reaching state s from $start$, and $f(n) = g(n) + h(n.s)$. A heuristic function h is *admissible* if it is a lower bound of the cost of the shortest path ($h(s) \leq d(s, goal)$ for any state s). Given an admissible h, A* is guaranteed to return an optimal solution [6]. h is *consistent* if $h(goal) = 0$ and $h(s_1) - h(s_2) \leq d(s_1, s_2)$ for any two states s_1 and s_2.

Assuming that $start$ is free, A* initializes Open with a *root* node ($root.s = start$) (Lines 2-4) and iteratively extracts from Open node n with the lowest f-value, denoted f_{min} (Lines 5-6). Then, A* performs a goal test on n (Lines 7-8). If $n.s$ is not a goal, n is expanded: each unexplored (unknown) neighbor s' of $n.s$ is explored (by applying *EXP*, Line 18). If s' is blocked, it is added to Blocked (Lines 19-20). Otherwise, a node n' ($n'.s = s'$) is created (Lines 21-22). A* performs *duplicate detection*, keeping only the node with the smallest g-value for a given state (Lines 23-26). If n' is not a duplication, its f-value is calculated (Line 27), and n' is inserted into Open (Line 30). Note that a path from $start$ to $goal$ can be constructed by following the back pointers from each node to its predecessor in the search tree.

The main disadvantage of A* when running on a GUO is that it uses a *static heuristic*, which remains constant throughout the search and does not exploit dynamic information about the graph. For example, assuming that all states are free, the best such heuristic is the *perfect heuristic*, which is the exact cost to *goal* assuming all states are free. This heuristic can be calculated (1) in a preprocessing phase that stores the *all-pairs shortest-path* data, or (2) lazily on the fly. This is similar to the Manhattan Distance heuristic in 4-connected grids. Next, we present the MXA* algorithm, which is built on A* but improves the heuristic by repeatedly calculating

Algorithm 1: A* (MXA*) in GUO

```
1  Main($instance = \langle G = (V, E, c, EXP), start, goal, h\rangle$)
2      Init OPEN, CLOSED, BLOCKED
3      Init $root$; $root.s = start$; $root.p$ =NIL; $g(root) = 0$
4      Insert $root$ into OPEN
5      while OPEN is not empty do
6          Extract $n$ from OPEN // lowest $f(n)$
7          if $n.s = goal$ then
8              return $n$
9          $h_D(n.s)$ = Low-Level
              ($V, E, c, n, goal, h$, OPEN, CLOSED, BLOCKED)
10         if $g(n) + h_D(n.s) > f_{min}$ then
11             $f(n) = g(n) + h_D(n.s)$
12             Insert $n$ into OPEN
13             continue
14         Insert $n$ into CLOSED
15         foreach $s' \in N(n.s)$ do
16             if $s' \in$ BLOCKED then
17                 continue
18             if $EXP(s') = blocked$ then
19                 Insert $s'$ into BLOCKED
20                 continue
21             Init $n'$; $n'.s = s'$; $n'.p = n$
22             $g(n') = g(n) + c((n.s, n'.s))$
23             if $\exists n'' \in$ OPEN $\cup$ CLOSED s.t. $n''.s = s'$ then
24                 if $g(n'') \leq g(n')$ then
25                     continue
26                 Remove $n''$ from OPEN and/or CLOSED
27             $f(n') = g(n') + h(s')$
28             $h_D(s')$ = Low-Level
                  ($V, E, c, n', goal, h$, OPEN, CLOSED, BLOCKED)
29             $f(n') = g(n') + h_D(s')$
30             Insert $n'$ into OPEN
31     return NO SOLUTION
```

Algorithm 2: Low Level of MXA* + CO

```
1  Low-Level($V, E, c, n, h, goal$, OPEN, CLOSED, BLOCKED)
2      Init OPEN$_l$, CLOSED$_l$
3      Init $root_l$; $root_l.s = n.s$; $g_l(root_l) = 0$
4      Insert $root_l$ into OPEN$_l$
5      while OPEN$_l$ is not empty do
6          Extract $n_l$ from OPEN$_l$ // lowest $f_l(n_l)$
7          if $n_l.s = goal$ then
8              return $g_l(n_l)$
9          Insert $n_l$ into CLOSED$_l$
10         foreach $s' \in N(n_l.s)$ do
11             if $s' \in$ BLOCKED then
12                 continue
13             Init $n'_l$; $n'_l.s = s'$
14             $g_l(n'_l) = g_l(n_l) + c((n_l.s, n'_l.s))$
15             if $\exists n''_l \in$ OPEN$_l \cup$ CLOSED$_l$ s.t. $n''_l.s = s'$ then
16                 if $g_l(n''_l) \leq g_l(n'_l)$ then
17                     continue
18                 Remove $n''_l$ from OPEN$_l$ and/or CLOSED$_l$
19             if $\exists n'' \in$ CLOSED s.t. $n''.s = s'$ then
20                 continue
21             if $\exists n'' \in$ OPEN s.t. $n''.s = s'$ then
22                 if $g(n'') \leq g(n) + g_l(n'_l)$ then
23                     continue
24             $f_l(n'_l) = g_l(n'_l) + h(n'_l.s)$
25             Insert $n'_l$ into OPEN$_l$
26     return $\infty$
```

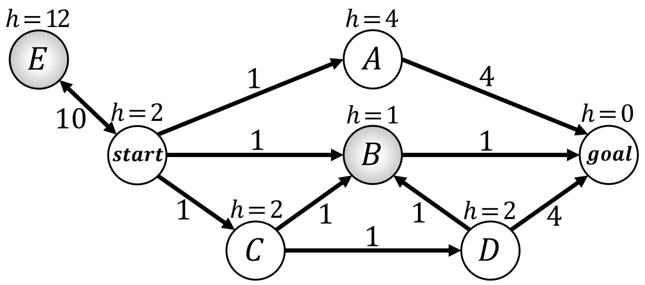
the shortest path to the goal while exploiting information on newly discovered blocked states.

4 MINIMIZE EXPLORATION A* (MXA*)

MXA* is a two-level algorithm. Its *high level* (presented in Algorithm 1) is similar to A* described above, which activates the *EXP* operator on the GUO, but MXA* also includes lines 9-13 and 28-29. After a node n', corresponding to state s', is generated, the *low-level search* is called to calculate a *dynamic* heuristic value for s', $h_D(s')$ (Line 28; we explain lines 9-13 later), and the f-value of n' is recalculated with respect to $h_D(s')$. The low-level search exploits the most updated knowledge on *blocked* states and performs a search strictly in memory to calculate a more informed heuristic. Let n be a newly created node in the high level. The low level is invoked to find the shortest path from $n.s$ to *goal* that does not traverse through any state in BLOCKED. The cost of this path is returned to the high level as $h_D(s)$. Thus, the low-level searches an abstract graph where states that were already explored are treated as *free* or as *blocked* according to the outcome of their *EXP* action. Additionally, we make the *free-space assumption* [11] and treat all *unknown* states as *free*. The cost of the path returned by the low level is clearly a lower bound on the shortest valid path, as the returned path may include *blocked* states (currently unknown and treated as free by the low level). Also, it is clearly more informed than any static heuristic used by A*, which assumes that *all* states are free.

The low-level search is presented in Algorithm 2. Lines 19-23 should be omitted; we will revisit them when discussing MXA* improvements. To differentiate the low-level search from the high-level one, we represent a node in the low-level search's abstract graph as n_l (with subscript l). Its g-value (distance of n_l from the high-level node n from which the low-level search commences) and f-value are g_l and f_l, respectively. Lists OPEN$_l$ and CLOSED$_l$ track the low-level search process. During the low-level search, nodes are not created for states s' that were already revealed as blocked in the high level (Lines 11-12).

Example. Figure 1 illustrates a problem instance, in which states B and E are blocked while all other states are free. Additionally, a table provides the progression of both A* and MXA* on this instance,



	Expand	Explore	Open (f)
A*	*start*	*A*, *B*, *C*, *E*	*C* (3), *A* (5)
	C	*D*	*D* (4), *A* (5)
	D		*A* (5), *goal* (6)
	A		*goal* (5)
	goal		
MXA*	*start*	*A*, *B*, *C*, *E*	*A* (5), *C* (6)
	A		*goal* (5), *C* (6)
	goal		

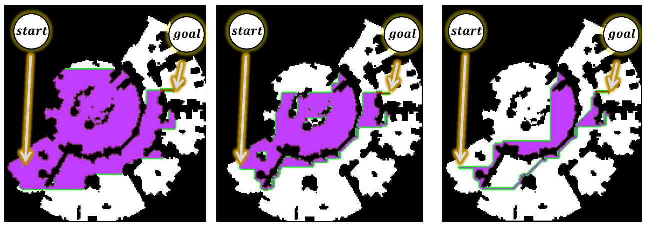
Figure 1: Example of A^* and MXA^* executions.

with each row signifying the expansion of a state. For simplicity, we refer to states and their nodes using similar notations. When *start* is expanded, A, B, C and E are explored but only A and C are added to Open due to B and E being blocked. A^* proceeds to expand C and explore D, leading to its expansion. Only then, it moves to expand A and finds the shortest path. In contrast, MXA^*, upon identifying B and E as blocked, invokes the low-level search on C. It capitalizes on the knowledge of B being blocked while assuming D is unblocked (without explicit exploration). This strategy uncovers a path via D to *goal*, resulting in $f(C) = 6$. Subsequently, the high level expands A ($f(A) = 5$) and finds the shortest valid path. Notably, MXA^*'s high-level search avoids exploring D (and expanding C), exploring fewer states than A^*.

Recalculating the Heuristic. As the high-level search progresses, new states are revealed as blocked and are inserted into Blocked. Therefore, when a node is selected for expansion by the algorithm, its h-value may not be up-to-date with the Blocked list (when its h-value was calculated, Blocked contained fewer states). To remedy this, when a node n is chosen for expansion by the high level, we perform another low-level search and re-calculate $h_D(n.s)$. If $g(n) + h_D(n.s) > f_{min}$, we insert n back to Open. This procedure is shown in Lines 9-13 of Algorithm 1. This is akin to Lazy A^* [7] where a more-informed heuristic is calculated before expansion. Importantly, this re-calculation of the heuristic is not relevant to A^* as its heuristic is static throughout the search.

4.1 Lazy Exploration (LE)

When A^* and MXA^* expand a node, they immediately explore all its yet unexplored neighbors and insert nodes into Open only if the states of the nodes are free. However, this exploration can be delayed. To do so, we first treat the neighboring states of an expanded node as free. Nodes for these states (filtered with duplicate detection) are immediately inserted into Open. Then, when a node is chosen to be expanded, we execute *EXP* on its state and discard it if it turns out to be blocked. We call this approach *Lazy Exploration* (LE). In the example of Figure 1, node E has a g-value of 10. Without LE, A^* and MXA^* explore it immediately and discard it because it is blocked. A^*+LE as well as MXA^*+LE add it to Open but will never explore it.



(a) A* (7,934) (b) MXA* with tb_1 (4,512) (c) MXA* with tb_2 (1,940)

Figure 2: A^* and MXA^* with two tie-breaking rules.

4.2 The Impact of a Tie-Breaking Rule

When multiple nodes in Open have $f = f_{min}$, then A^* uses a tie-breaking rule to determine which of these nodes to expand. The tie-breaking rule only influences which nodes will be expanded among the nodes with $f = C^*$ because any node with $f < C^*$ must be expanded, regardless of the tie-breaking rule [2]. In MXA^*, a tie-breaking rule has a more profound impact. Assume two nodes n and m where $f(n) = f(m) = f_{min}$. If n is chosen for expansion, $n.s$ might be revealed as blocked, increasing $h_D(m.s)$. Thus, $f(m)$ might potentially raise above C^*, causing it never to be expanded.

Figure 2 illustrates the execution of A^* and MXA^*+LE on a 4-connected grid while employing two distinct tie-breaking rules, tb_1 and tb_2, both prioritize nodes with higher g-values. Between nodes with identical f- and g-values, tb_1 resolves ties arbitrarily, while tb_2 goes a step further by resolving ties based on their respective distances from *goal* along the x and y dimensions. Let $\Delta_X(n) = |n.x - goal.x|$ and $\Delta_Y(n) = |n.y - goal.y|$. When two nodes, n_1 and n_2, share the same f- and g-values, tb_2 favors the node with a more balanced spatial distribution between these dimensions, as determined by $\min_{n \in n_1, n_2}(|\Delta_X(n) - \Delta_Y(n)|)$.

In the figure, the purple region contains all expanded nodes n with $f(n) < C^*$, the red region shows all nodes n expanded with $f(n) = C^*$, and the green region presents generated nodes that remained unexpanded. A^* (Figure 2(a)) explored the exact set of states using tb_1 and tb_2, a total of 7,934 states. In fact, the maximal difference in state explorations between any two tie-breaking strategies of A^* on this example is 200 (= 2.5% variance in the overall number of explorations). By contrast, MXA^*+LE with tb_1, shown in Figure 2(b), explored a total of 4,512 states, while MXA^*+LE with tb_2, depicted in Figure 2(c), explored a total of 1,940, a difference of 2,572 states (=57% variation in the overall number of explorations). Evidently, the choice of tie-breaking policy is more significant for MXA^*+LE than for A^*. We found that tb_2 is not always superior to

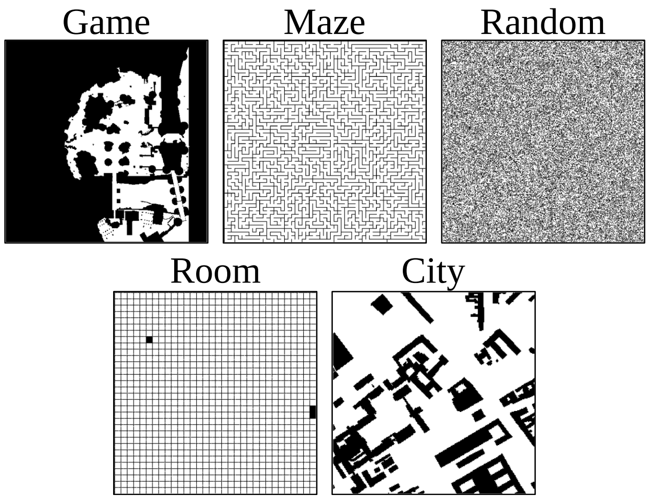


Figure 3: Grid maps.

		A*	A*+LE	MXA*	MXA*+LE
4	Game	9,094	8,766	4,143	**3,602**
	Maze	17,790	17,727	11,458	**10,347**
	Random	12,355	11,560	4,695	**3,330**
	Room	20,318	19,443	6,516	**5,195**
	City	17,569	16,661	4,133	**3,151**
8	Game	7,759	7,366	4,128	**3,536**
	Maze	9,023	8,949	6,197	**5,616**
	Random	8,981	8,047	5,585	**3,447**
	Room	13,128	12,509	6,347	**5,111**
	City	20,129	19,033	7,453	**6,149**

Table 1: Average number of explorations for A* and MXA*, without and with LE, on 4-connected and 8-connected maps.

tb_1 or other tie-breaking policies. Nonetheless, on average tb_2 significantly outperformed tb_1 on representative experiments. Thus, we used tb_2 for both A* and MXA* in our empirical study below.

4.3 Experiments: Comparing Explorations

We empirically compared the number of explorations performed by A* and MXA*, with and without lazy expansion (LE), on two domains: 4-connected grids (with the Manhattan Distance heuristic), and 8-connected grids (with the Octile Distance heuristic). We experimented on five grid domains, representing different topologies, extracted from the *MovingAI* repository [17]: Game, Maze, Random, Room, and City. A map from each domain is shown in Figure 3. Overall, we generated over 36.5k problem instances across all five domains featuring random *start* and *goal* points.

Table 1 presents the average number of explorations for each approach. The results show that MXA* consistently performed fewer explorations than A*. Additionally, enabling LE yielded a reduction in exploration for both A* and MXA*. Notably, in our experiments, MXA*+LE explored the fewest states, outperforming

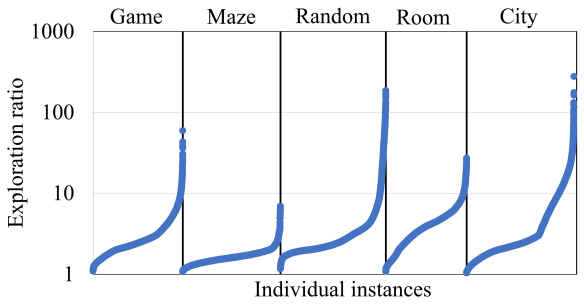


Figure 4: Per-instance exploration ratio of A* / MXA*+LE on 4-connected grids.

A* by a substantial margin ranging from 1.7 (mazes) to 5.6 (cities) in 4-connected grids and from 1.6 (mazes) to 3.3 (cities) in 8-connected grids. Figure 4 shows per-instance outcomes on 4-connected grids of the same maps and problem instances used in the above experiment (Table 1). The y-axis, presented on a logarithmic scale, shows the improvement factor between the states explored by A* and those explored by MXA*+LE for each individual instance. The instances are sorted in increasing order of the improvement factor. The depicted results highlight that the reduction in state exploration achieved by MXA*+LE over A* is exponentially distributed and varies from an improvement factor of close to 1 up to a staggering factor of up to 278. This variability shows that MXA*+LE can significantly enhance exploration in some scenarios while consistently delivering improved performance across a broad spectrum of instances.

5 DYNAMIC HEURISTIC FUNCTION

We now move to the second contribution of this paper, the *dynamic heuristic function* notion. As discussed below, dynamic heuristics offer wide-ranging applicability. We first provide a general definition of dynamic heuristic. However, in this paper, we are confined to the context of our GUO problem and the MXA* algorithm and, therefore, focus on using dynamic heuristics within this context. In particular, until now, our primary aim was to reduce the number of *EXP* actions; the CPU time was of lesser importance. To achieve this, in Section 6, we exploit a dynamic heuristic to improve the search time by reducing the number of expansions in the low-level search.

A study titled *The Closed List is an Obstacle Too* [3] found that states within the Closed list can be considered obstacles (blocked) for calculating a heuristic. Thus, for a given node n, a heuristic can assume that the shortest path from $n.s$ to *goal* must not pass via Closed states. This insight resulted in the BOXA* algorithm, tailored for 4-connected grids, which builds a rectangle around Closed states and computes a heuristic that bypasses this rectangle. We extend this concept to encompass the Closed list and portions of the Open list as obstacles during the low-level search. To achieve this, we first generalize the notion of a *static heuristic function* to a *dynamic heuristic function* that enables heuristic values to be dynamically updated throughout the search and introduce useful properties for such heuristics. Importantly, this broadened definition of heuristics is not confined to GUO-pathfinding; it offers a valuable contribution with broader applicability, as we discuss in Section 7. But, for our purposes, in Section 6 we leverage these definitions and properties to enhance the low-level search of MXA*, while assuring that these enhancements maintain the optimality of the returned solutions.

5.1 Formalizing Dynamic Heuristic Functions

We next introduce and study *dynamic heuristic functions* that exploit information (Denoted I) that became available during the search to dynamically refine their estimations. Formally, let a dynamic heuristic function be denoted as:

$$h : V \times \mathcal{I} \rightarrow \mathbb{R}^+$$

where $\mathcal{I}$ represents space of available information (later, we will define $\mathcal{I}$ in the context of MXA*, w.r.t. the Blocked, Closed and Open lists). Under this formulation, the h-value of state s is denoted by $h(s, I)$. Note that the conventional *static* heuristic definition is a special case wherein only the first parameter (a state $s \in V$) is considered.

It is well known that A*, when coupled with an admissible heuristic, must return an optimal solution [2]. However, this is a sufficient condition but not necessary. As defined by Karpas and Domshlak [8], a heuristic function h is called *path admissible* if there exists an optimal path P from $start$ to $goal$ such that for every state $s \in P$, $h(s) \leq d(n.s, goal)$. A* is guaranteed to return an optimal solution when given a path-admissible heuristic. For example, if there is an optimal path with admissible heuristics, it is sufficient for A* to return it, even if all other states receive a heuristic value of ∞ (which is inadmissible).

When considering a dynamic heuristic, we can further relax the path admissibility assumption while still guaranteeing optimal solutions. Intuitively, the heuristic values of all states along an optimal path do not need to be admissible at all times during the search. Instead, as long as the next unexpanded node with a state in some optimal path has an admissible heuristic, A* is still guaranteed to return an optimal solution.

Definition 1 (PDA). *A dynamic heuristic function h is called* path dynamically-admissible (PDA) *if there exists an optimal path $P = [p_0 = start, ..., p_n = goal]$ such that for every search information I (i.e., at any given time during the search) there exists an index j such that: i) for all $i \in \{0, \dots, j-1\}$ there exists a node $n_i \in$ Closed for which $n_i.s = p_i$, ii) there exists $n_j \in$ Open s.t. $n_j.s = p_j$, and iii) $h(p_j, I) \leq d(p_j, goal)$.*

As mentioned above, this definition is more relaxed than the original path admissibility, as it only requires that a single state (p_j) on an optimal solution would have an admissible heuristic. Nonetheless, we show that this relaxed definition is sufficient to guarantee the optimality of solutions.

Theorem 1. *A* is guaranteed to return an optimal solution when given a PDA heuristic.*

Proof. Assume by contradiction that A* terminated while returning a non-optimal solution on a given problem instance with a PDA heuristic. Thus, at the moment of termination, $f_{min} > C^*$. By definition of PDA, there exists a path $P = [p_0 = start, ..., p_n = goal]$ and an index j such that for all $i \in \{0, \dots, j-1\}$ there exists a node $n_i \in$ Closed s.t. $n_i.s = p_i$, and there exists a node $n_j \in$ Open s.t. $n_j.s = p_j$ and $h(p_j, I) \leq d(p_j, goal)$. Since $\forall i \in \{0, \dots, j-1\} \exists n_i \in$ Closed s.t. $n_i.s = p_i$ (i.e., the prefix of the path is in Closed), then for $n_j \in$ Open s.t. $n_j.s = p_j$ we have $g(n_j) = d(start, p_j)$. Thus, $f(n_j) = d(start, p_j) + h(p_j, I) \leq d(start, p_j) + d(p_j, goal) = C^*$. Thus, there must always exist a node n with $f(n) \leq C^*$. This contradicts the assumption that $f_{min} > C^*$. □

6 USING A DYNAMIC HEURISTIC FOR GUO

We next exploit the notion of dynamic heuristics to further improve the low-level search for MXA*.

6.1 Improved Low-Level Search for MXA*

As defined in Section 4, for high-level node n, the low-level of MXA* finds a shortest path from $n.s$ to $goal$ while pruning away (bypassing) Blocked states. The cost of that path was used as a heuristic for the high-level node n. Given our new definitions, this heuristic is dynamic as more states are added Blocked as the search progresses. We now extend the low level of MXA* to prune more low-level nodes based on the content of Closed and Open (CO; Lines 19-23 in Algorithm 2). We then prove that the resulting heuristic is a *dynamic heuristic* that has the PDA attribute and, thus, MXA* with this heuristic returns the optimal solution.

Given a high-level node n, the new low-level searches for the shortest path from $n.s$ to $goal$. Let n'_l be a low-level node and n'' be a high-level node sharing the same state ($n'_l.s = n''.s$). Node n'_l can be pruned (and treated as an obstacle) by the low-level search if one of the following three conditions is met (where Blocked, Closed and Open refer to the high level):

(i) $n'_l.s \in$ Blocked (Lines 11-12 in Algorithm 2, the original condition of MXA*)
(ii) $n'' \in$ Closed (Lines 19-20)
(iii) $n'' \in$ Open and $g(n'') \leq g(n) + g_l(n'_l)$ (Lines 21-23)

Condition (i) prunes Blocked states as was done in the basic version of the low-level search described in Section 4. Condition (ii) prunes high-level nodes with Closed states and treats them as obstacles during the low-level search for computing the heuristic for nodes n in Open. Intuitively, if a Closed node n'' has already been expanded, the shortest path from $start$ to $n''.s$ was found, and a shorter path from $start$ to $n''.s$ through $n.s$ of node $n \in$ Open cannot exist. Condition (iii) considers two paths from $start$ to the high-level node $n'' \in$ Open. Intuitively, if there is a path P'' from $start$ to $n''.s$, an alternative path P from $start$ to $n''.s$ ($= n'_l.s$) through $n.s$ (note that $g_l(n'_l)$ in condition (iii) refers to distance from n, not from $start$) with the same or higher cost than P'', then P is redundant. Thus, n'_l can be pruned during the low-level search from n.

Example. We have already shown an example of the usage of condition (i) in Section 4. Now, we present a similar example for conditions (ii) and (iii). In Figure 5, after $start$ is expanded, two new nodes, for states A and C are inserted into Open. When executing the low-level search from node C, paths that go through $start$ can be pruned (condition (ii)). Moreover, paths that go through A can

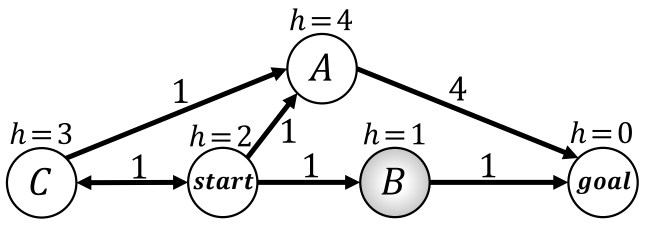


Figure 5: Example for Conditions (ii) and (iii).

be pruned as well (condition (iii)). Therefore, the low-level search sets $h_D(n.s) = \infty$ for $n.s = C$ without expanding $start$ and A.

6.2 Optimality of MXA*

We next prove that MXA* with the improved low-level search, indeed returns optimal solutions. For this, we resort to the definitions from Section 5. The h-value returned by the low-level for a high-level node n is the shortest path from $n.s$ to $goal$ on an abstract graph, denoted as $d_l(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$. B, C and O represent the content of Blocked, Closed, and Open, respectively, which dynamically change during high-level iterations. Consequently, the abstract graph searched by the low-level dynamically changes as more states are considered obstacles due to conditions (i)-(iii). Thus, the heuristic estimations provided by the low-level search for a state s change across high-level iterations. As a result, the low-level search of MXA* can be represented as a dynamic heuristic that depends on Blocked, Closed, and Open, i.e. I = (Blocked, Closed, Open). Under this formulation, the h-value of state s is denoted by $h(s, \text{Blocked}, \text{Closed}, \text{Open})$, or $h(s, \mathrm{B}, \mathrm{C}, \mathrm{O})$ for short.

Since the high-level search is actually A* that uses the values returned by the low-level as a heuristic, it is sufficient to show that $h(s, \mathrm{B}, \mathrm{C}, \mathrm{O})$ is a PDA heuristic in order to ensure the optimality of the solution returned by the high level (Theorem 1). We do this next.

We begin by introducing a new property that is necessary to prove for algorithms that their low-level search returns a PDA heuristic. In order to prune low-level nodes using Open and Closed, we need to ensure that once a node enters Closed, it is via the optimal path and will thus never be reopened. Otherwise, a Closed node n whose g-value is not optimal cannot be treated as an obstacle by the low-level because there might be a shorter path to it.

Definition 2 (OPTEX). Algorithms are said to have the *optimality expansion* property (OPTEX) if once a node n is expanded, it holds that n was discovered via an optimal path, i.e., $\forall n \in$ Closed it hold that $g(n) = d(start, n.s)$.

When given a consistent heuristic, A* exhibits the OPTEX property, as all nodes are expanded with an optimal g-value [2]. However, this property can be attained even in situations where the heuristic is inconsistent. In this context, we next prove that the high-level search of MXA* possesses the OPTEX property (even though the low-level search returns an inconsistent heuristic). Based on this observation, we will conclude that the low-level search within MXA* yields a PDA heuristic.

Theorem 2. *The high-level search of MXA* has the OPTEX property when using the improved low-level search to obtain heuristics.*

Proof. We prove the theorem by induction on the high-level iterations. In the base case, Closed is empty, thus, the property holds by vacuous truth. Assuming the property holds after the first $k - 1$ iterations, we proceed to demonstrate its validity after the k-th iteration.

Assume by contradiction that a node n was expanded in the k-th iteration via a suboptimal path, i.e., $g(n) > d(start, n.s)$. Let P be an optimal path from $start$ to $n.s$, where j is the largest index in P such that p_j was expanded (a node n_j s.t. $n_j.s = p_j$), excluding n. Consequently, let node $n_{j+1} \in$ Open where $n_{j+1}.s = p_{j+1}$ and $g(n_{j+1}) = d(start, p_{j+1})$. We assume w.l.o.g. that there is no node $n_i \neq n$ ($n_i.s = p_i \in P$), where $i > j + 1$, such that $n_i \in$ Open and $g(n_i) \leq g(n_{j+1}) + d(p_{j+1}, p_i)$; otherwise, we would have chosen the path that goes to $n.s$ through p_i as P. As a result, the nodes of all states in P between p_{j+1} and $n.s$ do not fulfill conditions (i)-(iii), and thus $d_l(p_{j+1}, n.s, \mathrm{B}, \mathrm{C}, \mathrm{O}) \leq d(p_{j+1}, n.s)$, where $d_l(x, y, \mathrm{B}, \mathrm{C}, \mathrm{O})$ is the distance from x to y in the abstract graph, constructed by the low-level search. The heuristic of p_{j+1} returned by the low-level search is the cost of the shortest path from p_{j+1} to $goal$ in the abstract graph, $h(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) = d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$. Similarly, the heuristic of $n.s$ is $h(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) = d_l(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$. Due to the triangle inequality, $d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \leq d_l(p_{j+1}, n.s, \mathrm{B}, \mathrm{C}, \mathrm{O}) + d_l(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$. Thus, $d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \leq d(p_{j+1}, n.s) + d_l(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$. As a result,

$$\begin{aligned} f(n_{j+1}) &= g(n_{j+1}) + h(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \\ &= d(start, p_{j+1}) + d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \\ &\leq d(start, p_{j+1}) + d(p_{j+1}, n.s) + d_l(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \\ &= d(start, n.s) + h(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \end{aligned}$$

Since we assumed that n was expanded via a suboptimal path, $g(n) > d(start, n.s)$, and thus:

$$< g(n) + h(n.s, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) = f(n)$$

In contradiction to n being expanded before n_{j+1}. □

Theorem 3. *The low level of MXA* returns a PDA heuristic for the high level.*

Proof. Since the OPTEX property holds, every node $n \in$ Closed has $g(n) = d(start, n.s)$. Let P be an optimal solution, and let n_j with $n_j.s = p_j \in P$ be the node with the largest index in P that was expanded, thus $n_{j+1} \in$ Open. W.l.o.g., we assume that all nodes before p_j in P were expanded; otherwise, we would consider an alternative prefix for P that reaches p_j. In addition, we assume that there is no other node n_i corresponding to state $p_i \in P$ such that $n_i \in$ Open, $i > j + 1$, and $g(n_i) \leq g(n_j) + d(p_j, p_i)$, otherwise, we would have chosen P to be the path that goes to through p_i. As a result, conditions (i)-(iii) are not fulfilled for the nodes of all states in P between p_{j+1} and $goal$. Therefore, $d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O}) \leq d(p_{j+1}, goal)$. Since $d_l(p_{j+1}, goal, \mathrm{B}, \mathrm{C}, \mathrm{O})$ is the heuristic returned by the low-level search for n_{j+1}. $n_{j+1} \in$ Open fulfills the requirement defined for a PDA heuristic, and thus the low level is a PDA heuristic for the high level. □

		A^*	MXA^*+LE	MXA^*+LE + CO	Ratio
4	Game	0.008	333.22	144.69	0.43
	Maze	0.009	1,796.37	88.91	0.05
	Random	0.008	97.64	10.65	0.11
	Room	0.017	126.90	33.25	0.26
	City	0.016	212.20	89.05	0.42
8	Game	0.007	384.52	153.24	0.40
	Maze	0.004	1,063.79	79.88	0.08
	Random	0.006	69.48	4.01	0.06
	Room	0.011	151.04	26.83	0.18
	City	0.018	453.55	138.96	0.31

Table 2: Average number of expansions (in millions) for MXA^* with LE, without and with CO, on different maps. A^*'s high-level expansions are provided in the first column for comparison.

Since the low-level search returns a PDA heuristic and the high-level search is simply A^* that uses the values returned by the low-level as a heuristic, MXA^* is guaranteed to return an optimal solution (Theorem 1).

6.3 Experiments: Comparing Expansions

We evaluated the average number of low-level expansions of MXA^*+LE with and without the pruning of Closed and Open and nodes (CO), on the same maps and problem instances used in the experiments presented in Table 1 and Figure 4. Table 2 presents the average number of expansions (in millions) performed by each algorithm. Notably, the adoption of CO yields a substantial reduction (see the ratio column) in the number of expansions across all maps, resulting in an expansion ratio ranging from 0.05 (Maze) to 0.43 (Game) in 4-connected grids and from 0.06 (Random) to 0.40 (Game) in 8-connected grids. This significant reduction shows the great benefit of exploiting the information on Closed and Open, collected during the search, for pruning nodes.

For comparison, in the first row we also provide the number of nodes expanded by plain A^* that only executes the high-level search with a static heuristic but without any low-level search. Indeed, when considering the number of expansions rather than explorations, this table demonstrates the notable CPU overhead induced by the new MXA^* algorithm. MXA^*+LE +CO performed significantly more expansions than A^*, ranging from approximately 660 times (Random) to around 21,800 times (Game). In practical terms, this implies that if the goal is to minimize the overall runtime, MXA^*+LE +CO will outshine A^* in scenarios where a sensing operation is slower than an in-memory expansion by a comparable factor.

7 DISCUSSION ON THE DYNAMIC HEURISTIC

To stay within the focus of this paper, we exploited the notion of dynamic heuristics by adding it into the MXA^* algorithm. Nevertheless, as mentioned in Section 5, the concept of a dynamic heuristic $h(s, I)$ has broader applicability beyond minimizing exploration in GUO. We next discuss a number of such applications which are left as a challenge for the future.

7.1 General and Standard Graphs.

A dynamic heuristic can be defined for standard graphs (without unknown obstacles) as $h(s, \mathrm{C}, \mathrm{O})$. A specific example of such a heuristic was used by the BOXA* algorithm [3] on 4-connected grid graphs. BOXA* constructs a rectangle encompassing Closed and directly computes a heuristic that avoids traversing this rectangle (without performing a low-level search). However, BOXA* was not substantially efficient in terms of CPU time compared to A^*. Effectively leveraging dynamic heuristics to substantially decrease search times presents a challenge that we defer to future research. This requires a wide future study on using dynamic heuristics on more complex graphs than 4-connected grids. Specifically, this study can be done on the following graphs:

- 8-connected grids (or any of the 2^n- connected family of grids [13]) where the physical structure of the Open and Closed lists is more complex.
- 3D grids of even higher-dimensionality grids.
- Any planner graph or map where lakes or mountains behave as obstacles but are not part of the map, as blocked cells in grids.
- Any general graph, even exponential graphs, such as combinatorial problems, where the shape of the Open and Closed lists cannot be represented in an Euclidean space.

7.2 A* with lookahead

A* with lookahead (AL*) [15] is a hybrid of A* and depth-first search that performs limited DFS lookaheads from the frontier of a best-first search such as the Open list of A*. A direct usage of a dynamic heuristic that uses CO will be to prune away DFS nodes that are also contained on the Closed or Open lists according to our definition above. This might significantly reduce the running time of the DFS phase, and might allow performing deeper DFS lookahead and return better heuristic values.

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed the MXA^* algorithm, designed to minimize exploration while searching graphs with unknown obstacles. We showed that MXA^* dramatically reduces exploration compared to A^*, by avoiding blocked states in the heuristic calculation. We also introduced the novel definition of dynamic heuristic and used it to prove that MXA^* can also exploit dynamic information from the open and closed lists while maintaining its optimality. Looking ahead, our work paves the way for future research to utilize dynamic heuristics in other problem domains. Furthermore, the low-level search introduced in this work could benefit from enhancements through the preservation of search information between iterations, akin to the incremental A^* algorithm proposed by Koenig and Likhachev [9].

ACKNOWLEDGMENTS

This work was supported by the Israel Science Foundation (ISF) grant #909/23 awarded to Shahaf Shperberg and Ariel Felner, by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, awarded to Shahaf Shperberg, and by United States-Israel Binational Science Foundation (BSF) grant #2021643 awarded to Ariel Felner.

REFERENCES

[1] Chih-Chung Chou, Feng-Li Lian, and Chieh-Chih Wang. 2011. Characterizing Indoor Environment for Robot Navigation Using Velocity Space Approach With Region Analysis and Look-Ahead Verification. *IEEE Transactions on Instrumentation and Measurement* 60, 2 (2011), 442–451.

[2] Rina Dechter and Judea Pearl. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM (JACM)* 32, 3 (1985), 505–536.

[3] Ariel Felner, Shahaf S. Shperberg, and Hadar Buzhish. 2021. The Closed List is an Obstacle Too. In *the International Symposium on Combinatorial Search (SOCS)*. 121–125.

[4] Ariel Felner, Roni Stern, Sarit Kraus, Asaph Ben-Yair, and Nathan S. Netanyahu. 2004. PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment. *J. Artif. Intell. Res.* 21 (2004), 631–670.

[5] Guy Foux, Michael Heymann, and Alfred Marcel Bruckstein. 1993. Two-dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Trans. Robotics Autom.* 9 (1993), 96–102.

[6] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2) (1968), 100–107.

[7] Erez Karpas, Oded Betzalel, Solomon Eyal Shimony, David Tolpin, and Ariel Felner. 2018. Rational deployment of multiple heuristics in optimal state-space search. *Artif. Intell.* 256 (2018), 181–210.

[8] Erez Karpas and Carmel Domshlak. 2012. Optimal Search with Inadmissible Heuristics. In *ICAPS*.

[9] Sven Koenig and Maxim Likhachev. 2001. Incremental A*. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1539–1546.

[10] Sven Koenig, Maxim Likhachev, and David Furcy. 2004. Lifelong Planning A*. *Artificial Intelligence* 155, 1 (2004), 93–146.

[11] Sven Koenig and Yury Smirnov. 1997. Sensor-based planning with the freespace assumption. In *Proceedings of International Conference on Robotics and Automation*, Vol. 4. 3540–3545.

[12] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. 2008. Anytime search in dynamic graphs. *Artificial Intelligence* 172, 14 (2008), 1613–1643.

[13] Nicolás Rivera, Carlos Hernández, Nicolás Hormazábal, and Jorge A. Baier. 2020. The 2ˆk Neighborhoods for Grid Path Planning. *J. Artif. Intell. Res.* 67 (2020), 81–113.

[14] Bar Shofer, Guy Shani, and Roni Stern. 2023. Multi Agent Path Finding under Obstacle Uncertainty. In *the International Conference on Automated Planning and Scheduling (ICAPS)*. 402–410.

[15] Roni Stern, Tamar Kulberis, Ariel Felner, and Robert Holte. 2010. Using Lookaheads with Optimal Best-First Search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, Maria Fox and David Poole (Eds.). 185–190.

[16] Roni Tzvi Stern, Meir Kalech, and Ariel Felner. 2010. Searching for a k-Clique in Unknown Graphs. In *the Symposium on Combinatorial Search (SoCS)*, Ariel Felner and Nathan R. Sturtevant (Eds.). 83–89.

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

[18] Alexander Zelinsky. 1992. A mobile robot navigation exploration algorithm. *IEEE Transactions of Robotics and Automation* 8, 6 (1992), 707–717.

[19] Xunyu Zhong, Jun Tian, Huosheng Hu, and Xiafu Peng. 2020. Hybrid Path Planning Based on Safe A* Algorithm and Adaptive Window Approach for Mobile Robot in Large-Scale Dynamic Environment. *Journal of Intelligent & Robotic Systems* 99 (2020), 65–77.