Application of A* Search to Find Safest Egress for Active Shooter Scenarios

Ian Chadwick
Department of Electrical and Computer
Engineering Boston University
8 St. Mary's St, Boston, MA 02215
ianjc@bu.edu

Abstract

Gun violence and active shooter scenarios have become a rising concern amongst many in the field of Risk Management and Occupational Health and Safety for many businesses, places of worship and schools in the United States. The methodology recommended by the U.S. Department of Homeland Security and the FBI to keep individuals safe in these scenarios is, the Run, Hide, Fight- Active Shooter Protocol - where safely escaping the location is the number one priority. In this paper, the active shooter scenario is modeled as a motion planning problem where the A path planning algorithm, with heuristics to prioritize safety of escapees, is applied to determine the best path to avoid the trajectory of a known shooter. Given a building's floor plan, an escapee's initial location, and the initial location of a shooter, the planner will generate the optimal safe path that avoids the shooter as much as possible, while directing the escapee to the nearest building exit as quickly as possible.*

1. Introduction

Active shooter scenarios have become a matter of rising concern on both the individual and societal level in the United States for organizations spanning places of worship, businesses, and schools. The Federal Bureau of Investigation (FBI) defines an active shooter as "…an individual actively engaged in killing or attempting to kill people in a populated area" [1]. The number of active shooter incidents have been increasing in overall frequency from 2000-2009 until reaching an average of approximately 21 incidents per year for the time period of 2010-2018 [2]. The response methodology recommended by the Department of Homeland Security and the Federal Bureau of Investigation for individuals that find themselves in this situation, is the Run, Hide, Fight - Active Shooter Protocol [3]. In the Run, Hide, Fight (RHF) protocol, individuals are encouraged to first attempt to escape the location entirely, hide if escape becomes impossible, and then as a last resort, to fight if they find themselves in direct confrontation with the shooter.

1.1. Problem Definition

In an effort to apply path planning principles to address this growing problem, we propose a model of the active shooter scenario as a combination of a graph-based, shortest path problem and a modified pursuit-evasion problem where, given a known floor plan, and the starting location of a shooter, path planning algorithms can be applied to find the safest and fastest optimal path for an escapee to evacuate - adhering to the first and preferred element of the Run, Hide, Fight methodology. The primary motivation for paper this is to propose a basis for real-world applications of the algorithm, which we will call Safe*, such as implementation on mobile phones or similar devices to provide escapees real-time guidance on the safest path to exit in active shooter scenarios.

1.2. Relation to Prior Work

The fundamental difference between our proposed model and existing pursuit-evasion models, is that many of the pursuit-evasion games and algorithms that have been explored to date model a scenario where the pursuer is the agent under control of the planner, and the goal of the planner is to capture an antagonistic evader, which is not under control of the planner [4]. The exemplary variations of graph-based pursuit-evasion games, such as the cop-robber [5] and hunter-rabbit games [6] provide a general methodology that has been explored extensively and expanded upon in more recent research variations where pursuers and evaders have differing amounts of information [7, 8]. Other pursuit-evasion games model a similar scenario in geometric settings, such as the lion-man game [9] and other visibility-based games [10].

However, the active shooter scenario can be best modeled by the inverse scenario, where the planner controls the evader, and the pursuer is the antagonist, which is not under the planner's control. The need for this inverted model arises from the unique characteristics that are inherent to the active shooter scenario, namely the unpredictability of the shooter's actions, which leads to uncertainty regarding a shooter's location. In analysis of past active shooter incidents, it has been shown that in the

majority of incidents, the shooters' primary objective is simply to cause as much harm as possible before they are captured or eliminated by law enforcement authorities [3]. This means that once they begin their attack, the shooter's targets are chosen opportunistically, and their path through the environment is inherently random and therefore unpredictable.

There has been very little research into path planning for the specific scenario where the evaders are under the control of the planner and the pursuers are the antagonists. To date, there were no papers that attempt to model this problem in this way. The model and Safe* algorithm we propose in this paper are an effort to address that gap in the existing body of knowledge.

In addition, this model also differs from traditional pursuit-evasion games - where the evader's goal is often to maximize the time until capture - because in this case the evader's goal is to escape the environment by the shortest path that also minimizes the chance that it will encounter the pursuer. There are many path planning algorithms that optimize the shortest path to a goal for a mobile robot including variations of potential based functions [11,14] for the continuous case and many variations of A* [12] for the discretized and graph-based cases, including heuristic based approaches [13]. We have chosen to use a modified heuristic A* algorithm that factors in the safety of an escapee as a critical component of the heuristic, to model the escapee's goal adhering to the "Run" component of RHF and avoiding the shooter as much as possible.

1.3. Paper Structure

In the following section, we first outline the methodology and define key features of the active shooter scenario which in turn, define the requirements and framework of the model and application of the Safe* algorithm. Then, the results and algorithm performance are demonstrated through simulations of the algorithm on sample environments of increasing complexity and the runtime measured and compared to A*. Finally, in the last section, we draw conclusions, discuss directions for future work, and propose potential pragmatic applications of the algorithm to the initial problem statement.

2. Methodology

This section is divided into two subsections that first addresses construction of the model and then describes the Safe* implementation. The model's construction and definitions are explained in relation to the unique characteristics of the active shooter scenario, and how these characteristics led to the assumptions and constraints of the model. The following section describes the path planning algorithm and its application, which are based on the expected behaviors of both the shooter(s) and the escapee(s).

2.1. Description of the Model

This section is further divided into a section describing the mapping of the environment, followed by the description of the model used to represent the potential behavior of one or more active shooters.

2.1.1 The Environment

In the active shooter scenario, the environment is typically a place of worship, workplace, or school. This means that the environments have a completely known and fixed floor plan, which contains the locations for all entrances and exits, as well as the locations of all walls and other impassible areas. This allows us to model each floor of the building as a 2D rectangular grid where each grid space has corresponding coordinates (x, y) corresponding to a set of axes each beginning at zero in the top left corner and ranging to the maximum width and height of the rectangle for *x-rows* and *y-columns* respectively.

Each free grid space is then represented as a node n, in the set (N) on a graph (G) where grid locations of walls, holes or any other impassible objects are excluded from the set N, such that N only contains nodes that represent locations that can be occupied.

Every node n_i is represented as an object in a list with attributes: *nodeID,* an integer used to determine the index where it is stored in the list, *coords*, its (x, y) coordinates in the grid, D_{exit}, the distance the node is from the nearest exit, *safety*, a heuristic that is calculated based on the location of the shooter, *neighbors*, a list of the undirected edges to neighboring nodes, *cost* the accumulated path cost, and *backpointer*, the nodeID of the node before it in the path. Each node in N has an undirected edge (E) to each other adjacent node in N, where adjacency is determined by four-point connectivity. Four-point connectivity was a simplification chosen to account for the limitations of movement through doorways and around corners. While diagonal movement is feasible in open space, in most buildings, it is often impossible to move diagonally through a doorway or around a corner in such a way that is represented by eight-point connectivity. Exits are represented as k nodes $\eta_{1\ldots,}\eta_k$ from which an escapee would be able to exit the environment in their next move. The distance from each exit on the grid η_k to each node n_i is determined by the Manhattan distance calculation:

$$D_{exit}(\eta_k, n_i) = | n_i(x) - \eta_k(x) | + | n_i(y) - \eta_k(y) | \qquad (1)$$

And the corresponding D_{exit} attribute for node n_i is updated if D_{exit} is less than the current D_{exit} value, or if the node does not yet have a value.

2.1.2 Active Shooter Behavior

As described in previous sections, the behavior of an

active shooter can be described as mostly opportunistic and random. While the shooter's movements will not be truly random, since their essential motivation is causing harm, the specific order in which they choose to traverse the environment can be considered random. As a result, their actions cannot be well described by any of the existing pursuit-evasion games. However, some elements of pursuit-evasion games can be applied to the model.

The first is that shooters and the escapees move at the same relative speed, which can be modeled as turn based, like the cop-robber [5], rabbit-hunter [6], or lion-man [9] games. Shooters are defined as $S_1 \ldots, S_i$ for i number of shooters with (x, y) coordinates on the grid corresponding to the initial given location of each respective shooter. The subsequent turn-by-turn actual location of each shooter is unknown, but is instead represented by a potential-based wavefront function emanating from each shooter's initial location [14].

The *wavefront* is propagated through each of the nodes n in N with a modified Breadth First Search (BFS) algorithm (see algorithm 1 below), that begins with *safety=0* at a shooter's location and increments the value by one across the wavefront for each node visited, such that every node in the graph will have a value for *safety≥0*. The *safety* value of each node visited is updated with the value of the current *wavefront* number if it either, hasn't yet been initialized, or if it is lower than the current *safety* value of that node. This *safety* value represents how many moves it would take a shooter to reach that node in the worst-case scenario where a shooter chose that direction to walk from its starting location at the beginning of the incident. A lower *safety* value indicates a higher chance of an escapee on that node running into a shooter. This essentially has the same effect as creating a repulsive potential gradient emanating from each shooter location.

Algorithm 1 Shooter Wavefront Algorithm

Input: A graph and location of a shooter
Output: A graph with updated *safety* attributes for each node in the graph with respect to the shooter location.

Enqueue starting node n_{start} to que Q, set $safety(n_{start}) = 0$ ➢ Initialization
Repeat
  Pop the next node n from Q and add to C.
  Set the value of *wavefront* to *safety(n)* + 1.
  for all $n_{nb} \in$ neighbor(n) that are not in C **do** ➢ Expand n
   if $safety(n_{nb}) > wavefront$ **then**
    Set the value of $safety(n_{nb})$ to *wavefront.*
   end if
   if $n_{nb} \notin Q$ **then**
    Enqueue n_{nb}
   end if
  end for
until Q is empty

2.2. Applying the Safe* Algorithm

The escapee's behavior is best modeled by the implementation of the Safe* algorithm. In order to follow the RHF methodology, the escapee's strategy should be to preferentially select moves to nodes that continuously move the escapee away from a shooter toward the nearest exit by the shortest path possible. This is achieved by utilizing the A* algorithm with some modifications to the costs of movement *g(n),* and heuristic *h(n),* to represent the specific behavior unique to the scenario (see algorithm 2).

In the Safe* implementation, the node attributes *safety* and D_{exit} from the graph are used, along with an escapee's starting location, in a modified A* search algorithm where neighboring nodes are first added to a priority queue, where the priority of a neighboring node *f(n)* is determined by:

$$f(n) = g(n) + h(n) \quad (2)$$

Where *g(n)* is the sum of the accumulated move costs from the escapee's starting node to the current node and the cost of the next move, which is determined by the move type, which is summarized in Table 1.

$$g(n) = current\ cost(n) + next\ cost(n) \quad (3)$$

Move in relation to D_{exit} value	**Move in relation to *safety* value**	**Next Cost (n)**
To lower D_{exit}	To higher *safety*	0
To higher D_{exit}	To higher *safety*	1
To lower D_{exit}	To lower *safety*	1.5
To higher D_{exit}	To lower *safety*	2

3. **Table 1:** Movement costs for each move type

An escapee's primary goal is to move away from a shooter and toward an exit, which is represented by a move to a node with higher *safety* and lower D_{exit}, whenever possible and so movement to nodes with these criteria will have a cost of zero. Moving away from an exit but also away from the shooter, represented by a move to a node with higher *safety* but higher D_{exit}, is less preferential, but is still achieving the escapee's primary goal of remaining safe while avoiding a shooter and will have a cost of 1. Moving toward an exit, but also toward a shooter, represented by moving to a node with a lower D_{exit} and a lower *safety*, has a cost of 1.5 because of the two overall objectives, the algorithm should preferentially select paths that prioritize *safety* over the shortest distance to the exit. Moving toward a shooter and away from an exit is the costliest move, represented by a move to a node with a lower *safety* and higher D_{exit}, and has a cost of 2.

The heuristic value, *h(n)* is given by:

$$h(n) = D_{exit}(n) - safety(n) \quad (4)$$

Where $D_{exit}(n)$ is a node attribute calculated from equation (1) and *safety(n)* is a node attribute calculated from the application of the wavefront algorithm described in section 2.1.2. In the general form of the A* algorithm, the heuristic is considered optimistic if $h(n) \leq h^*(n)$ where $h^*(n)$ is the least costly path to the exit. Since D_{exit} is the Manhattan distance from the node to the goal, which is an optimistic heuristic, and it is only ever being reduced by the *safety* value this value for $h(n)$ will also be optimistic.

Algorithm 2 Safe* Algorithm

Input: A graph with the distance D_{exit} and *safety* values calculated for all the nodes representing free space and an escapee's start location.

Output: The shortest path between the start location and a goal(exit) which also maintains an optimized distance from one or more shooters.

1: Add the starting node n_{start} to priority que O, set $g(n_{start}) = 0$, and set the *backpointer* of x to be empty. ➢ Initialization
2: **Repeat**
3: Pick n_{best} from O such that $f(n_{best}) \leq f(n)$ for all $n \in O$.
4: Remove n_{best} from O and add it to C.
5: **if** $n_{best} = q_{goal}$ **then**
6: Exit.
7: **end if**
8: **for** all $x \in$ Star(n_{best}) that are not in C **do** ➢ Expand n_{best}
9: **if** $x \notin O$ **then**
10: Set the value of $g(x)$ to $g(n_{best}) + next\ cost(x)$.
11: Set the *backpointer* of x to n_{best}.
12: Add x to O with the value $f(x)$.
13: **else if** $g(n_{best}) + next\ cost(x) < g(x)$ **then**
14: Update the value of $g(x)$ to $g(n_{best}) + next\ cost(x)$.
15: Update the *backpointer* of x to n_{best}.
16: **end if**
17: **end for**
18: **until** O is empty

This heuristic for $h(n)$ represents the escapee's goal of preferentially picking nodes that are safer and have a higher *safety* rating over nodes that are strictly closer to an exit. As a result, the Safe* algorithm will explore nodes with a preference for nodes with a higher *safety* rating until the overall cost and distance from the exit becomes too great. After which, it will begin to explore nodes that may be closer to a shooter, but closer to the exit. This is the node selection pattern for the traditional, unmodified A* algorithm, which is purely for the shortest path to exit. Safe* will continue this behavior until a safest *and* shortest path to exit is discovered based on the locations of the shooters and exits.

The code used in the implementation can be found at: https://github.com/ianjchadwick/SafeStar

3. Results and Discussion

The Safe* algorithm was tested on a variety of arbitrary grids that could feasibly represent the floor plan of a building, as well as some arbitrarily large grids to measure the runtime performance as the number of nodes in the graph increased. The results are demonstrated in the following two sections. In the first section we discuss the results regarding the performance of the Safe*, A* and Wavefront algorithm in terms of the measured runtimes and path lengths of each test. A demonstrative sample of the test results have been tabulated in Table 2 below. In the second section we discuss the performance of the algorithm in terms of the chosen path routes in relation to the location of the shooters and exits. While the "optimality" of these paths are somewhat subjective, they are arguably the most important metric by which to measure the success of the algorithms for this particular scenario given the fact that the primary objective is to keep the escapee as safe as possible from the shooters.

Test Number	**1** Figure 1	**2** Figure 2	**3** Figure 3	**4** Not Shown	**5** Not Shown	**6** Not Shown
Grid Size	9x9	13x13	13x13	15x15	50x50	100x100
Number of Nodes	55	108	108	199	2500	10000
Number of Exits	2	2	4	3	1	1
Shooter Count	1	1	2	2	2	2
Wavefront Time (s)	$1.78E^{-4}$	$2.39E^{-4}$	$7.50E^{-4}$	$2.81E^{-3}$	$3.98E^{-1}$	6.63
Safe* Time (s)	$4.01E^{-4}$	$3.54E^{-4}$	$2.80E^{-4}$	$8.16E^{-4}$	$1.54E^{-2}$	$5.48E^{-2}$
A* Time (s)	$1.43E^{-4}$	$1.20E^{-4}$	$1.28E^{-4}$	$5.50E^{-4}$	1.28	20.4
Safe* Path Length	14	19	13	25	99	199
A* Path Length	9	10	12	18	99	199

Table 2: Runtime and Path Length for Safe* and A*. Tests 1, 2 and 3 correspond to figures 1, 2 and 3, respectively. Figures were not generated for tests 4, 5 or 6.

3.1. Runtime Analysis

In general, the performance of the Safe* algorithm was comparable to the performance of A* for most grids, considering it was roughly in the same order of magnitude and the path lengths were not too drastically different, and in some cases were even the same. The performance of A* was marginally better for smaller grids (tests 1, 2, 3 and 4), where A* performed roughly between two and three times as fast as Safe*. The very large grids (tests 5 and 6) were designed to push the runtime higher by having a large number of nodes and edges, very few obstacles (test 4), or no obstacles at all (tests 5 and 6) and the greatest possible distance between the start location and exit. With these conditions, A* and other graph search algorithms

generally perform less efficiently. However, somewhat surprisingly, Safe* outperformed our implementation of A* quite significantly in these grids.

We expected the performance trend seen in the smaller grids to continue into the large grids and expected both A* and Safe* to have taken longer to find a path overall. Perplexingly, in our implementation, A* performed several orders of magnitude worse for tests 5 and 6. We have hypothesized about the reasons for these surprising results, but one of the ones we believe is reasonable is that the Manhattan distance representation of the graph structure itself was better suited for the application of the Safe* star cost function. As a result, the cost function for the A* algorithm rendered the effectiveness of its heuristic to be much closer to the more basic and trivial form $h(n)=0$, giving the algorithm similar runtime and node exploration behavior to BFS. Whereas the modified cost function and heuristic for Safe* were very well suited to the representation and structure of the graph and it was able to find the same path much quicker than A*.

The shooter wavefront algorithm's runtime was also measured because we suspected that it may be one of the limiting factors that would hinder a completely online implementation for a real-world system based on the algorithm. The floor plan, including the exits are known in advanced and do not change so they could be used to calculate D_{exit} offline, but the wavefront algorithm requires the shooters' locations to be known before it can be used. We hypothesized that it could be a significant bottleneck to calculate the *safety* of each node in time for it to be useful in a real-world active shooter scenario, where seconds are precious. However, we were pleasantly surprised that the bottleneck was not too significant, with the exception of the very large sparse grid, despite being a relatively inefficient algorithm when compared to A* or Safe*.

The path lengths were about what we expected, with A* generally finding a shorter path than Safe*. However, as we will discuss in the next section, these paths were not always the optimal paths, considering the potential danger that the escapee was exposed to. The following figures demonstrate the behavior of both A*(orange) and Safe*(purple) given obstacles (black), a starting location (blue), shooter locations (red), and exits (green).

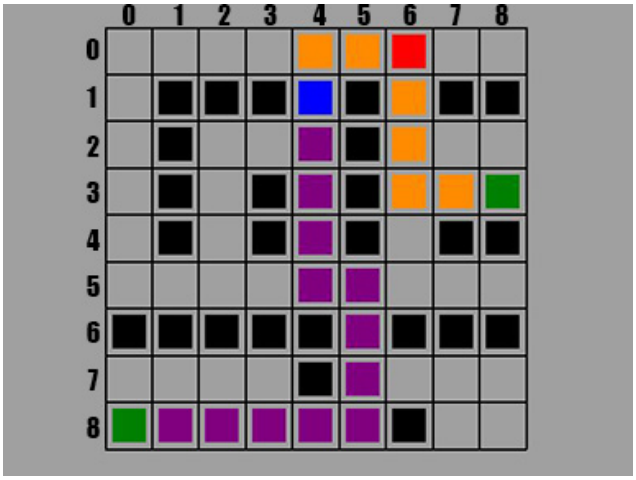

Figure 1: Visualization of test 1 from Table 2.

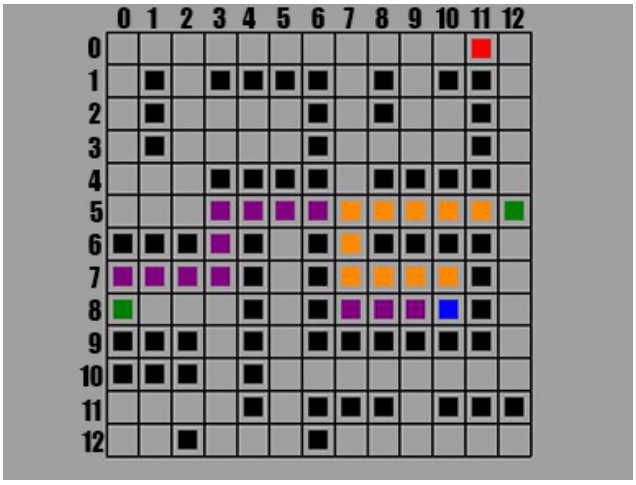

Figure 2: Visualization of test 2 from Table 2.

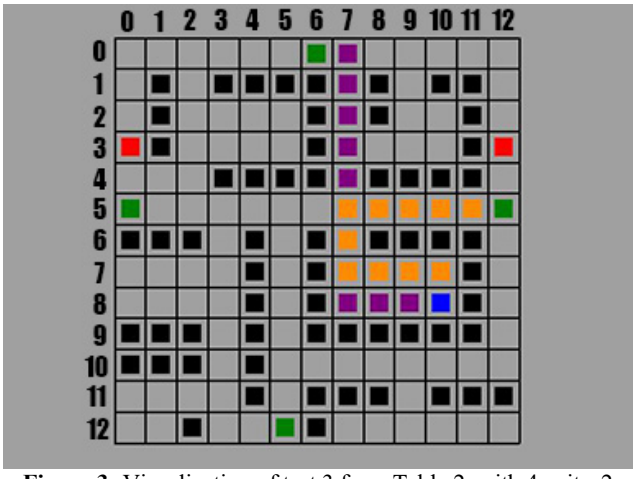

Figure 3: Visualization of test 3 from Table 2, with 4 exits, 2 shooters and the same Start location as figure 2.

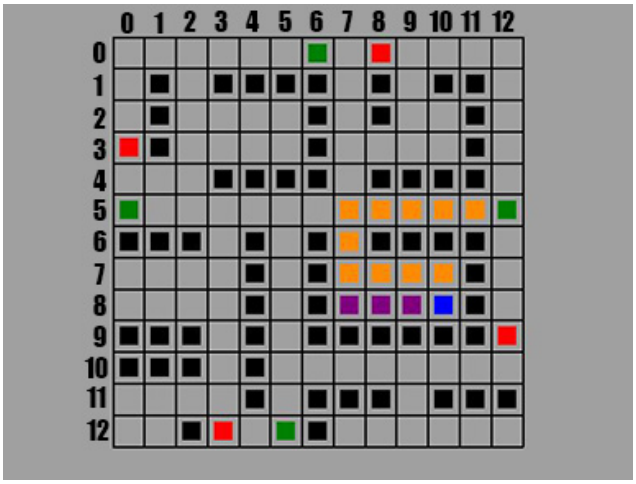

Figure 4: Example where Safe* performs similarly to A* due to too many shooters.
Note: The orange A path is drawn last, and covers over the purple path, but Safe* shares the path from [7,7] to the exit.*

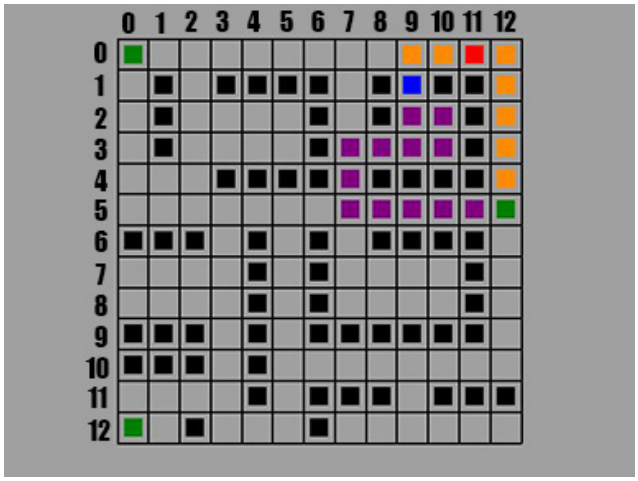

Figure 5: Example where Safe* performs similarly to A* due to the exits being too far away.

In the model of the active shooter scenario that we have created, any direct confrontation with the shooter will likely result in a fatal error for the escapee. Therefore, the overall goal of the Safe* implementation was to avoid the shooter as much as possible while also adhering primarily to the "Run" aspect of RHF and proceeding as quickly as possible to an exit. In that regard, Safe* greatly outperformed A* because while A* generally had a shorter path to exit, it often led an escapee directly through (figure 1) or near (figure 3) a shooter's last known location. Whereas the Safe* path would preferentially lead an escapee to an exit that was far enough away from a shooter to give them the least likely possibility of encountering a shooter given the shooters' last known locations (figures 1, 2 and 3). A* and Safe* generally found very similar paths in large grids, grids with many shooters (figure 4), few exits or exits that were very far from the start location (figure 5).

We believe this was due to the fact that Safe*'s heuristic gave it a preference for selecting nodes for exploration that were safer, but it also has competing pressures pushing it to find the exit. So, in tests where there were many shooters for the grid size, the competing wavefronts emanating from each shooter and the overall distance to the exit effectively made A* and Safe* choose similar paths. At first it might seem like this is a case against Safe* but we would argue that Safe* still is the optimal algorithm for the scenario, because according to the overall objective of RHF, the optimal overall strategy in all cases is to escape by any means necessary. So, in this regard, it would not make sense for Safe* to choose a path that was overly long, because the longer the escapee is in the area, the more likely they are to have an eventual (and potentially fatal) encounter with one of the shooters.

4. Conclusions and Future Directions

Our original objective was to create a model of the active shooter scenario where we could apply a modified A* path planning algorithm, Safe* in a novel way in order to guide an escapee from an active shooter situation. In general, there seems to be a gap in the existing literature for modeling the inverse version of the pursuit-evasion game [4, 5, 6, ,7, 8, 9, 10]. We believe that we developed a good foundation for further research into the area, which can be expanded to model other types of pursuit-evasion strategies from the perspective of the evader. However, there were some limitations to our study that we would like to address and comment on.

One limitation of our study was we made the assumption that given an initial location, the subsequent location of the shooter remained unknown. As a result, we only calculated the wavefront from the shooters' locations once and generated paths based on that information. While this is a reasonable generalization for the scenario, given that in order to find the location of the shooter after an active shooter scenario began, an escapee would either have to come into contact with the shooter, which would be fatal, or determine their location through some other means, such as an update that was pushed to their device, or perhaps from auditory cues or other indirect methods. This would allow us to run the wavefront algorithm, and subsequently Safe*, to update the path with a more accurate path each time the shooters' locations were updated. This would generate a more accurate "real-time" path generator.

A continuously updating *safety* value from updated shooters' locations would allow us to incorporate the "Hide" aspect of the RHF methodology, where the algorithm would choose when it would be more appropriate to hide from a shooter, rather than to always

try to escape. In the current form, where the algorithm is only run once at the start of the incident, the algorithm would simply choose to "hide" whenever there was a shooter within a certain distance threshold, and without any updates it would simply tell the escapee to stop moving. This would not be a very useful component of algorithm given the primary objective of RHF. However, these are very interesting future directions worth exploring that were simply beyond our initial scope.

Another limitation that we would like to address is that the shooter wavefront algorithm tended to be relatively slow on larger grids. A potential solution, and avenue for further exploration, would be to slice the overall floor plan into localized sub-graphs that were updated as needed when the escapee moved to the periphery of each graph, or when the shooters' locations were updated, because the runtime only really started to become untenable when the grid size was greater than 50 x 50.

Despite some of the limitations discussed above, we believe that this algorithm and further research into novel utilization of A* path planning algorithms have great potential for lifesaving real-world applications. An ideal ultimate objective would be to deploy this algorithm as a completely online mobile phone app, where the shooters' locations could be broadcast to each escapee and updated each time the shooters' locations are known. Each time the shooters' locations are updated, a new updated safest path would be generated and safely guide escapees to exits. The Safe* algorithm was our attempt to propose a potential engineering-based solution to a growing societal problem facing Americans, in a time when there seems to be little traction in finding viable alternatives or solutions in the socio-political space.

References

[1] Federal Bureau of Investigation, Office of Partner Engagement, "Active Shooter Resources", fbi.gov, Accessed: Nov. 19, 2021 [Online]. Available: https://www.fbi.gov/about/partnerships/office-of-partner-engagement/active-shooter-resources

[2] Federal Bureau of Investigation, Office of Partner Engagement, "Quick Look: 277 Active Shooter Incidents in the United States From 2000 to 2018", fbi.gov, Accessed: Nov. 19, 2021 [Online]. Available: https://www.fbi.gov/about/partnerships/office-of-partner-engagement/active-shooter-incidents-graphics

[3] U.S. Department of Homeland Security, Active Shooter: How to Respond, October 2008, https://www.dhs.gov/xlibrary/assets/active_shooter_booklet.pdf

[4] T. Chung, G. Hollinger, and V. Isler. Search and Pursuit-evasion in Mobile Robotics. Auton. Robots. 31. 10.1007/s10514-011-9241-4, 2011

[5] M. Aigner, M. Fromme. A game of cops and robbers. Discrete Applied Mathematics, Volume 8, Issue 1, Pages 1-12, ISSN 0166-218X, https://doi.org/10.1016/0166-218X(84)90073-8, 1984.

[6] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized pursuit-evasion in graphs," Combinatorics, Probability and Computing, vol. 12, no. 3, pp. 225–244, 2003.

[7] V. Isler, N. Karnad, The role of information in the cop-robber game. Theoretical Comp. Sci., Volume 399, Issue 3, Pages 179-190, ISSN 0304-3975, https://doi.org/10.1016/j.tcs.2008.02.041, 2008.

[8] A. Kehagias, D. Mitsche, P. Prałat, Cops and invisible robbers: The cost of drunkenness, Theoretical Comp. Sci., Volume 481, 2013, Pages 100-120, ISSN 0304-3975, https://doi.org/10.1016/j.tcs.2013.01.032, 2013.

[9] Bhadauria, D., Klein, K., Isler, V., & Suri, S. (2012). Capturing an evader in polygonal environments with obstacles: The full visibility case. The International Journal of Robotics Research, 31(10), 1176–1189. https://doi.org/10.1177/0278364912452894

[10] Sachs S, LaValle SM, Rajko S. Visibility-Based Pursuit-Evasion in an Unknown Planar Environment. The International Journal of Robotics Research. 2004;23(1):3-26. doi:10.1177/0278364904039610

[11] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," in IEEE Transactions on Robotics and Automation, vol. 16, no. 5, pp. 615-620, Oct. 2000, doi: 10.1109/70.880813.

[12] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, L. Jurišica. Path Planning with Modified a Star Algorithm for a Mobile Robot. Procedia Engineering, Volume 96, 2014, Pages 59-69, ISSN 1877-7058, https://doi.org/10.1016/j.proeng.2014.12.098.

[13] D. Ferguson, M. Likhachev, A. Stentz, (2005). A guide to heuristic-based path planning. In Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS) (pp. 9-18).

[14] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. IEEE Transactions on Man and Cybernetics, 22(2):224–241, Mar/Apr 1992.