ERA*: Enhanced Relaxed A* algorithm for Solving the Shortest Path Problem in Regular Grid Maps

Adel Ammar *

Posted Date: 17 August 2023

doi: 10.20944/preprints202308.1215.v1

Keywords: path planning; A*; grid maps; algorithms; shortest path problem

Article

ERA*: Enhanced Relaxed A* algorithm for Solving the Shortest Path Problem in Regular Grid Maps

Adel Ammar

RIOTU Lab, Prince Sultan UNiversity, Rafha Street, Riyadh, 11586, Saudi Arabia; aammar@psu.edu.s a

Abstract: This paper introduces a novel algorithm for solving the point-to-point shortest path problem in a static regular 8-neighbor connectivity (G8) grid. This algorithm can be seen as a generalization of Hadlock algorithm to G8 grids, and is shown to be theoretically equivalent to the relaxed A^* (RA^*) algorithm in terms of the provided solution's path length, but with substantial time and memory savings, due to a completely different computation strategy, based on defining a set of lookup matrices. Through an experimental study on grid maps of various types and sizes (1290 runs on 43 maps), it is proven to be 2.25 times faster than RA^* and 17 times faster than the original A^*, in average. Moreover, it is more memory-efficient, since it does not need to store a G score matrix.

Keywords: path planning; A*; grid maps; algorithms; shortest path problem

1. Introduction

The shortest path problem is a well-established and important problem that finds applications in various fields, including robotics ([1,2]), VLSI design ([3]), wireless sensor networks ([4]), and transportation ([5,6]). This problem can be represented either on a grid or a more general graph, with different techniques employed for each representation. Grid-based approaches are commonly favored in scenarios like extensive VLSI design, particularly when there are numerous obstacles to navigate around.

Traditionally, two types of methods have been used to address the global path planning problem. The first type is exhaustive search, which involves thorough exploration of the entire search space to find the optimal solution. Examples of exact search algorithms commonly used for this purpose include Dijkstra's algorithm (([7,8])) and A^* algorithm ([9]). These algorithms guarantee finding the shortest path but may become impractical when dealing with large grids due to their computational complexity.

The second type is local search, which employs metaheuristic algorithms to explore only a portion of the search space. These algorithms, such as Tabu Search ([10]), Ant Colony Optimization ([11]), Genetic Algorithm ([12]), Particle Swarm Optimization ([13,14]), and their multiple variants, provide approximate solutions and are often used when the exhaustive search is not feasible due to the large size of the grid. By exploring a limited subset of the search space, local search algorithms can find reasonably good solutions efficiently.

Recently, some relaxed alternatives of exhaustive search algorithms have emerged. These approaches aim to find a balance between the constraint of achieving optimality in path length and the need for faster search times. The work being discussed here falls into this category. It is demonstrated that the proposed Enhanced Relaxed A^* (ERA^*) algorithm is theoretically equivalent to the relaxed A^* (RA^*, [15]) algorithm in terms of the path length of the solutions it provides. However, it achieves significant savings in terms of time and memory requirements by employing a completely different computation strategy.

By relaxing the constraint of path length optimality to some extent, this approach manages to expedite the search process while still delivering solutions that are comparable in terms of path length to those obtained through the RA^* algorithm. The key advantage lies in the adoption of a novel computation strategy based on lookup matrices. These matrices store precomputed values that can be

 doi:10.20944/preprints202308.1215.v1

efficiently accessed during the search, eliminating the need for redundant calculations and resulting in substantial time and memory savings.

The remaining of this paper is organized as follows: Section 2 discussed the most relevant related works. Section 3 presents the proposed ERA^* algorithm and proves its equivalence to RA^* in terms of path cost. Section 4 presents the results of the evaluation of ERA^* on a benchmark of representative grid maps, and compares it to A^* and RA^* in terms of path cost and execution time. Finally, section 5 concludes the paper and suggests some future works.

2. Related Works

Shortest path algorithms are a crucial tool in the field of computer science and are used to solve a wide range of problems. In general, point-to-point shortest path algorithms work by starting at a source node and exploring the neighboring nodes until they reach the destination node. Along the way, the algorithm keeps track of the shortest distance from the source to each node, using this information to guide the search and ensure that the shortest path is found. There are a variety of shortest path algorithms that have been developed for finding the shortest path between two points in a regular grid ([16]). These algorithms typically involve searching through the grid to identify the optimal path, taking into account factors such as the cost of moving from one cell to another and any obstacles that may be present in the grid.

One of the most popular and well-known shortest path algorithms for regular grids is Dijkstra's algorithm, which was first proposed by Dutch computer scientist Edsger Dijkstra ([7,8]). Dijkstra's algorithm is a general-purpose algorithm that can be used to find the shortest path between any two points in a weighted, directed graph. It is based on the idea of building a "shortest path tree" from the source point to all other points in the grid, with the shortest path to each point being the minimum sum of the weights of the edges along the path. Although Dijkstra's algorithm is relatively simple and easy to implement, it has a time complexity of $O(|E| + |V|log|V|)$ when using a suitable data structure (Fibonacci heaps), where $|E|$ is the number of edges in the graph and $|V|$ is the number of vertices. This makes it less efficient for larger grids.

Another popular shortest path algorithm for regular grids is A^* [9], which was proposed as an improvement of Dijkstra's algorithm. A^* is a heuristic search algorithm that uses a combination of a best-first search and a cost-estimation function to guide the search towards the goal. Unlike Dijkstra's algorithm, which considers all paths from the source to the goal, A^* only considers paths that are likely to lead to the goal, based on the cost-estimation function. This makes A^* more efficient than Dijkstra's algorithm for many grid-based problems, although the performance of A^* can be sensitive to the choice of cost-estimation function.

[17] proposed the first Maze-solving algorithm that is based on target-directed grid propagation and is memory-efficient. It was extensively employed in the field of printed circuit board design for finding wire paths. However, [18] later revealed that the original claim of the algorithm's generality regarding the path cost function is incorrect.

More recent shortest path algorithms for regular grids include the Jump Point Search (JPS) algorithm, proposed by [19] as an improvement over A^* in grid maps. It is based on a selective node expanding process that specifically identifies and expands certain nodes in the grid map, referred to as jump points. Thus, intermediate nodes along a path between two jump points are not expanded at all, which enhances the speed of A^* search by an order of magnitude.

[20] proposed the Minimum Detour (MD) algorithm in G4 regular grids. Its main idea is to calculate the detour number d(P), defined as the number of nodes on the path P that are directed away from the goal. Then, the path length is calculated as M(S,G) + 2d(P), where M(S,G) is the Manhattan distance between the start S and the goal G nodes. This is because any move opposed to the direction of the goal necessarily needs to be compensated to reach the goal. Since M(S,G) is constant for a given (S,G) pair, minimizing the path length is equivalent to minimizing the detour number. Nodes that have lower detour numbers are given higher priority in the grid expansion process, in a breath-first

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

search style. Since then, no similar algorithm has been proposed in G8 grids, because the detour computation is not as simple as in G4 grids. This paper proposes to bridge this gap by introducing a Hadlock-inspired algorithm applicable to G8 grids.

Relaxed Dijkstra and Relaxed A^* (RA^*) algorithms proposed by [15] exploit the grid-map structure to establish an accurate approximation of the optimal path, without visiting any cell more than once. The path length is approximated in terms of number of moves on the grid. This approach distinguishes itself from previous bounded relaxation algorithms of A^* ([21–24]) primarily by performing relaxation on the exact cost, denoted as g, of the evaluation function f (where f = g + h). This sets it apart from existing relaxations of the A^* algorithm, which typically focus on relaxing the heuristic h. The current work proposes aims to further accelerate the execution of the RA^* algorithm by completely changing its computing paradigm, drawing inspiration from the Hadlock algorithm.

It is worth noting that path planning in a grid map can be substantially accelerated in a completely different way by building connection graphs before applying a search algorithm ([25]). Nevertheless, the fact that the proposed ERA^* algorithm does not construct connection graphs is a significant advantage in many cases. For instance, in large VLSI design problems with a high number of obstacles, the construction of the entire connection graph could be extremely costly.

3. Methodology

The proposed algorithm is based on calculating detour penalties that are propagated from the source node S to the the goal node G. For a given current node C, we define:

- α: the angle between the x-axis and the $\overrightarrow{CG}$ vector.
- $\Delta_x = x_G - x_C$, where x_G and x_C are the abscissas of the goal node and the current node, respectively.
- $\Delta_y = y_G - y_C$, where y_G and y_C are the ordinates of the goal node and the current node, respectively.

Figure 1 clarifies the definition of the variables α, Δ_x, and Δ_y on an example. Figure 2 presents the five first matrices of incremental detour penalty for $\alpha \in [0°, 90°]$. These 3x3 matrices store the incremental detour (penalty) from the goal expressed as:

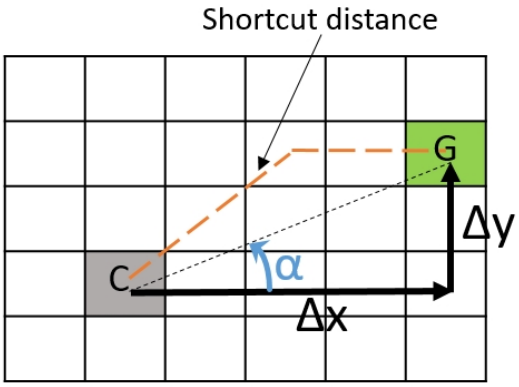

Figure 1. *Example showing the angle α between the current node (C) and the goal node (G), $\Delta_x = x_G - x_C$, $\Delta_y = y_G - y_C$, and the shortcut distance between C and G, which corresponds to the minimum distance in a G8 grid when there are no obstacles between the two nodes.*

$$D(n_i) - D(n_{i-1}) = dist(n_{i-1}, n_i) + h(n_i) - h(n_{i-1}) \quad (1)$$

Where h is the shortcut distance to goal (shortest path assuming there are no obstacles, see figure 1).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

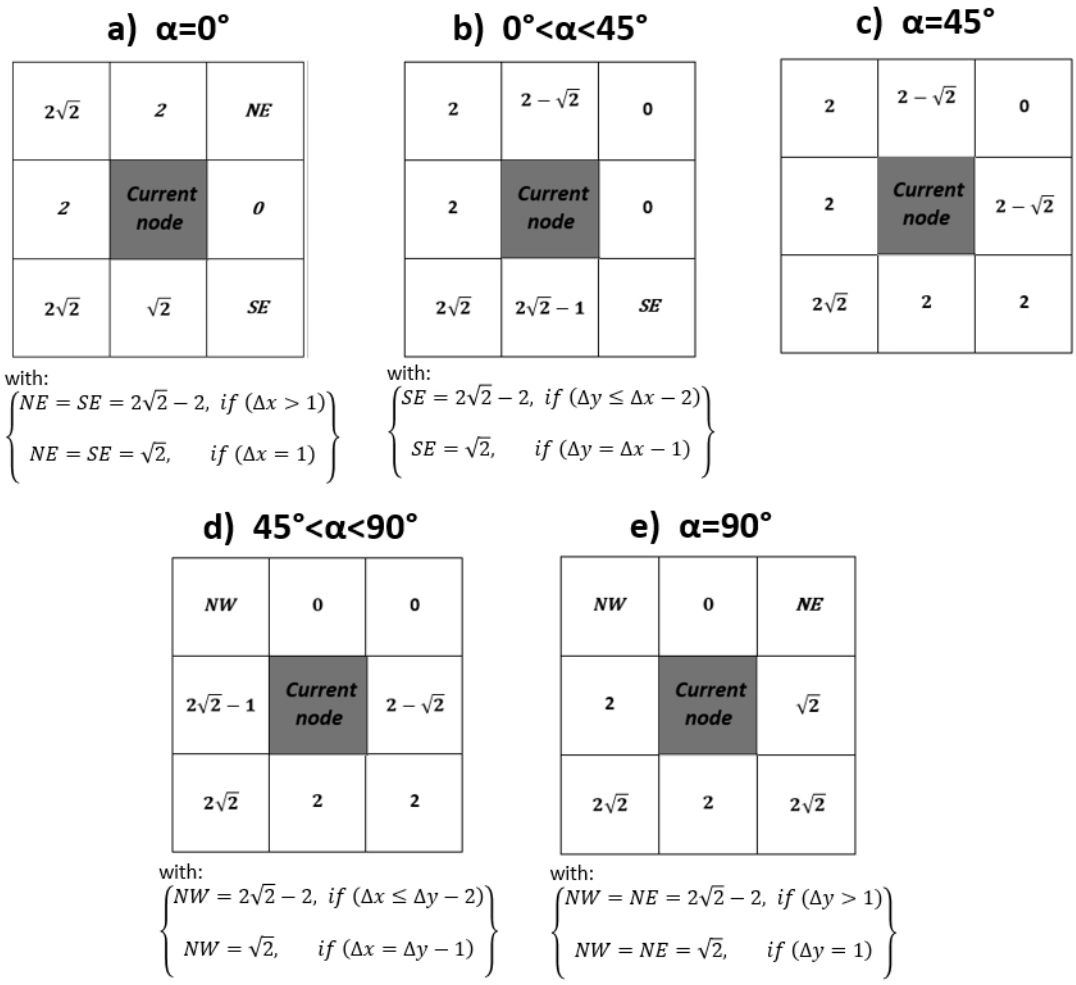

Figure 2. *Incremental detour penalty lookup matrices for angles between the current node and the goal node from 0° to 90°. The remaining cases can be obtained from the above cases, using simple 90° rotations of the 3x3 matrix. NE, SE, and NW stand for Northeast, Southeast, and Northwest, respectively. These incremental penalties are added when propagating the total penalty the source node to the goal node.*

The value of some penalties, which are denoted as NE (North-East), SE (South-East), and NW (North-West), depend on the value of Δ_x and Δ_y. For instance, in case (a), corresponding to $\alpha = 0$ (C and G on the same row, with G on the right, so that the optimal path between them is equal to Δ_x), if $\Delta_x = 1$, it means that the goal is adjacent to the current node on its right. Consequently, if we move towards NE or SE, we will reach the goal with a minimum cost of $\sqrt{2}+1$ instead of just 1 for the optimal path between C and G. That is why the penalty is $\sqrt{2}$. Whereas, for all other values of $\Delta_x > 1$, if we move towards NE, we can return back to the row containing G with a total cost of $2\sqrt{2}$ instead of 2 for the optimal path. That is why the penalty is $2\sqrt{2}-2$. Obviously, the case $\Delta_x < 1$ cannot correspond to angle $\alpha = 0$. The reasoning is similar for the other cases.

Notice that the matrix (e) can be obtained from the matrix (a) by applying a 90° anti-clockwise rotation and substituting Δ_x for Δ_y in the formulas of NE/SE that becomes NW/NE. Likewise:

- (f) $90° < \alpha < 135°$: is obtained from (b) by applying a 90° anti-clockwise rotation.
- (g) $\alpha = 135°$: is obtained from (c) by applying a 90° anti-clockwise rotation.
- (h) $135° < \alpha < 180°$: is obtained from (d) by applying a 90° anti-clockwise rotation.

In total, we have 28 fixed 3×3 matrices (counting the cases of different Δ_x and Δ_y values), that we store before running the search algorithm.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

As in [20], nodes that have lower detour numbers are given higher priority in the grid expansion process. This process proceeds until the goal position is reached. Then the path is reconstructed backwards from the goal cell, by moving from each cell to its parent cell that expanded it, until the starting cell is reached.

Instead of two matrices (actual cost g and combined score f) as in A^* and RA^*, only one matrix of detour penalties (D) is stored in memory. At a give node n_i, the detour value can be calculated recursively, using the euclidean distance $dist$ and the heuristic function h (shortcut distance), as:

$$\begin{aligned} D(n_i) &= D(n_{i-1}) + dist(n_{i-1}, n_i) + h(n_i) - h(n_{i-1}) \\ &= D(n_{i-2}) + dist(n_{i-2}, n_{i-1}) + dist(n_{i-1}, n_i) + h(n_{i-1}) - h(n_{i-2}) + h(n_i) - h(n_{i-1}) \\ &= ... \\ &= D(n_0) + \sum_{k=0}^{i-1} dist(n_k, n_{k+1}) + \sum_{k=1}^{i} h(n_k) - h(n_{k-1}) \end{aligned}$$

With:

$$\begin{aligned} &D(n_0) = 0 \\ &\sum_{k=0}^{i-1} dist(n_k, n_{k+1}) = g(n_i) \\ &\sum_{k=1}^{i} h(n_k) - h(n_{k-1}) = h(n_i) - h(n_0) \end{aligned}$$

Therefore:

$$\begin{aligned} D(n_i) &= g(n_i) + h(n_i) - h(n_0) \\ &= f(n_i) - h(n_0) \end{aligned} \tag{2}$$

Consequently, since $h(n_0)$ is constant, optimizing D is equivalent to optimizing f.

Algorithm 1 presents the main steps for path exploration in ERA^*. The algorithm takes as input a grid represented by a 2D matrix (map) where obstacles are marked with a predefined value, and the coordinates of the start (S) and goal (G) nodes. It calculates the penalty matrix D with a finite value for all explored nodes. The algorithm stops when the node exploration reaches G, or when a predefined maximum number of iterations is reached. We first store the 28 3×3 incremental penalty matrices (see Figure 2), and initialize the priority queue I of expanded nodes to the start node S with associated penalty 0 (line 4). The penalty matrix D and the Predecessor matrix P (used for path reconstruction) are both initialized with default infinity values (lines 6-8), except for the penalty associated with S which is initialized to 0 (line 10). Then, we iterate over the priority queue I by dequeuing each time the node that has the minimum penalty value $D(i, j)$. Based on the value of Δx and Δy between this current node C and the goal node, we choose the corresponding incremental penalty matrix D_i. This choice is accomplished through a series of if statements where the most likely cases (e.g., inequalities such as $\Delta y \geq 1$) are placed before less likely cases (e.g., equalities such as $\Delta x = 0$), in order to optimize execution time. We define J (line 16) as the set of all free (i.e., non-obstacle) neighbor nodes of C for which the penalty value $D(i, j)$ is infinite (i.e., has not been calculated yet). For each node in J, we save its predecessor C in the P matrix (line 17) for later path reconstruction, and we calculate its penalty value $D(i_P, j_P)$ as the sum of the current node's penalty plus the corresponding incremental penalty in the 3×3 D_i matrix (line 18). Then, we enqueue J in I (line 21), where the priorities correspond to the D values. If there is a path between S and G, G will be reached and its penalty value calculated, unless the predefined maximum number of iteration is attained.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

Algorithm 1: ERA^* search algorithm.

input : map: The whole grid with obstacle information
max_nb_iter: maximum number of iterations
(i_S, j_S): coordinates of the Start node S on the grid
(i_G, j_G): coordinates of the Goal node G on the grid
output: D: penalty matrix
nb_iter: number of iterations to find the near-optimal path

1 $nb\_iter = 0$
2 $fail$ = False
3 Store the 28 (3×3) D_i penalty matrices *// See Fig.2 and eq.1*
4 $I = [(i_S, j_S, 0)]$ *// Priority queue of expanded nodes*
5 *// Initialization of the distance matrix D and the predecessors matrix P on the map:*
6 **foreach** *node in map* **do**
7 $D(node) = +\infty$
8 $P(node) = +\infty$
9 **end**
10 $D(i_S, j_S) = 0$
11 **while** $(D(i_G, j_G) = +\infty$ ***and*** $nb\_iter < max\_nb\_iter$ ***and*** $size(I) > 0)$ **do**
12 (i_C, j_C) = dequeueMin(I) *// C will be the current node*
13 $\Delta x = i_G - i_C$
14 $\Delta y = j_G - j_C$
15 Choose the correct (3×3) D_i penalty matrix $matP$ according to the relative value of Δx and Δy
16 $J = \{(i, j, D(i, j)) \mid (i, j) = N \in map$, N neighbor of C, N is not an obstacle, and $D(i, j) = +\infty\}$
17 **foreach** $(i_P, j_P, D(i_P, j_P))$ *in* J **do**
18 $P(i_P, j_P) = (i_C, j_C)$
19 $D(i_P, j_P) = D(i_C, j_C) + matP(i_P - i_C, j_P - j_C)$
20 **end**
21 Enqueue J in I
22 nb_iter++
23 **end**

Once Algorithm 1 reaches the goal G, Algorithm 2 is used to reconstruct the path backward. It takes as input the penalty matrix D and predecessors matrix P that were both generated in Algorithm 1, and the start and goal coordinates on the grid. If $D(i_G, j_G) \neq +\infty$ (line 1), it means that Algorithm 1 has reached the goal and a path from S to G has been found. The path reconstruction algorithm starts from the goal (i_G, j_G) (line 3), and prepends the current node's predecessor to the path at each iteration (line 14), while adding 1 (for horizontal and vertical moves) or $\sqrt{2}$ (for diagonal moves) until it reaches the start node S. Alternatively, we can count the number of diagonal moves, and multiply it by $\sqrt{2}$ only at the end to avoid accumulating numerical inaccuracies.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

Algorithm 2: path reconstruction algorithm.

input : D: penalty matrix *// Output from Algorithm 1*
P: Predecessors matrix *// Output from Algorithm 1*
(i_S, j_S): coordinates of the Start node S on the grid
(i_G, j_G): coordinates of the Goal node G on the grid

output: *path*: near-optimal path between S and G
length: length of the near-optimal path between S and G
fail : True if no path has been found

if $D(i_G, j_G) \neq +\infty$ **then**
  $fail$ = False
  $(i_C, j_C) = (i_G, j_G)$ *// Path reconstruction starts from the Goal*
  $path = [(i_G, j_G)]$
  $length = 0$
  while $(i_C, j_C) \neq (i_S, j_S)$ **do**
    $(i_P, j_P) = P(i_C, j_C)$
    if $i_P = i_C$ ***or*** $j_P = j_C$ **then**
      $length$ += 1
    else
      $length$ += $\sqrt{2}$
    end
    $(i_C, j_C) = (i_P, j_P)$
    $path.prepend((i_C, j_C))$
  end
else
  $fail$ = True
end

4. Evaluation and discussion

4.1. Dataset

The simulation study was performed using Matlab R2012a on a laptop with an Intel Core i7 processor (2.4 GHz) and 16 GB of RAM. The evaluation of the algorithms was based on the same benchmark used in [15], which is composed of four categories of maps:

- 26 maps with randomly placed rectangular obstacles of various obstacle sizes and ratios, ranging from 100 x 100 to 2000 x 2000 in size.
- 6 mazes with passages of different sizes, all 512 x 512 in size, with variable corridor size.
- 4 room maps (512 x 512) filled with random square rooms of variable size.
- 6 maps from video games and 1 real-world map (Willow Garage), ranging from 512 x 512 to 1024 x 1024 in size and selected for their varying levels of difficulty.

We designed and generated the first category of random maps, while the other three categories (mazes, rooms, and video games) were selected from a large set of benchmarking maps provided by [26]. For each map, we conducted 30 runs with randomly selected start and goal nodes each time. Thus, the total number of runs is 1290 (43 maps × 30) for each algorithm.

We compared the proposed ERA^* algorithm to the original A^*_t (A^* using path tie-breaks) and to ${RA^*}_{wot}$ (Relaxed A^* without using path tie-breaks) which was proven in [15] to provide the best tradeoff between path cost and execution time among the tested relaxed variants in G8 grids.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

4.2. Results

Two main performance metrics are considered to evaluate the three global planners:

- The path length: it represents the length of the shortest global path found by the planner.
- The execution time: time spent by an algorithm to find its best (or optimal) solution.

Table 1 shows the percentage of optimal paths (when a path exists) for the three tested algorithms, per environment. The original A^* provides the optimal in all cases, since no relaxation is applied to it. We observe that ERA^* yields a slightly higher rate of optimal paths than RA^*_{wot} for random environments of different sizes, a slightly lower rate for rooms and video games, and a markedly lower rate for mazes, in which path lengths are the longest in average, as shown in table 2. This is due to the numerical issues entailed by the incremental computation. Nevertheless, table 3 shows that the percentage of extra length compared to the optimal path remains low at 2.3% in average, and a maximum of 10.4%, which is slightly lower than the maximum extra length produced by RA^*_{wot}.

Table 1. Percentage of exact optimal paths (when a path exists) per environment, for the three tested algorithms.

Algorithm	100x100	500x500	1000x1000	2000x2000	Mazes (512x512)	Rooms (512x512)	VideoGames (512x512 to 1024x1024)	All
A^*_t	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
RA^*_{wot}	79.6%	85.0%	78.9%	96.7%	21.7%	25.0%	47.2%	62.9%
ERA^*	81.9%	86.0%	80.6%	96.7%	4.4%	23.3%	46.7%	61.1%

Table 2. Average path cost per environment size, in cell units.

Algorithm	100x100	500x500	1000x1000	2000x2000	Mazes (512x512)	Rooms (512x512)	VideoGames (512x512 to 1024x1024)	All
A^*_t	60.4	284.8	631.2	1086.2	1479.1	317.1	375.8	490.3
RA^*_{wot}	60.7	285	632.3	1086.3	1501	321	381.3	494.8
ERA^*	60.7	285.2	633	1086.3	1517	323.6	382.8	497.7

Table 3. Percentage of extra length compared to optimal path, calculated for non-optimal paths over all environments.

Algorithm	Mean	Std	Max
RA^*_{wot}	1.7%	1.6%	10.7%
ERA^*	2.3%	1.8%	10.4%

On the other hand, Table 4 shows the average execution time of each algorithm in each grid environment. ERA^* is 2.25 times faster in average than RA^*_{wot}. This ratio varies from 1.74× to 2.47× depending on the type of grid environment. Figure 3 displays the histogram of this ratio. It shows that RA^*_{wot} is faster than ERA^* in only very few cases.

Table 4. Average execution time (in seconds).

Algorithm	100x100	500x500	1000x1000	2000x2000	Mazes (512x512)	Rooms (512x512)	VideoGames (512x512 to 1024x1024)	All
A^*_t	0.23	3.71	39.27	113.96	113	17.76	29.97	31.35
RA^*_{wot}	0.06	1.15	6.33	24.82	11.04	2.87	3.5	4.12
ERA^*	0.03	0.51	2.71	10.05	4.65	1.3	2.01	1.83

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

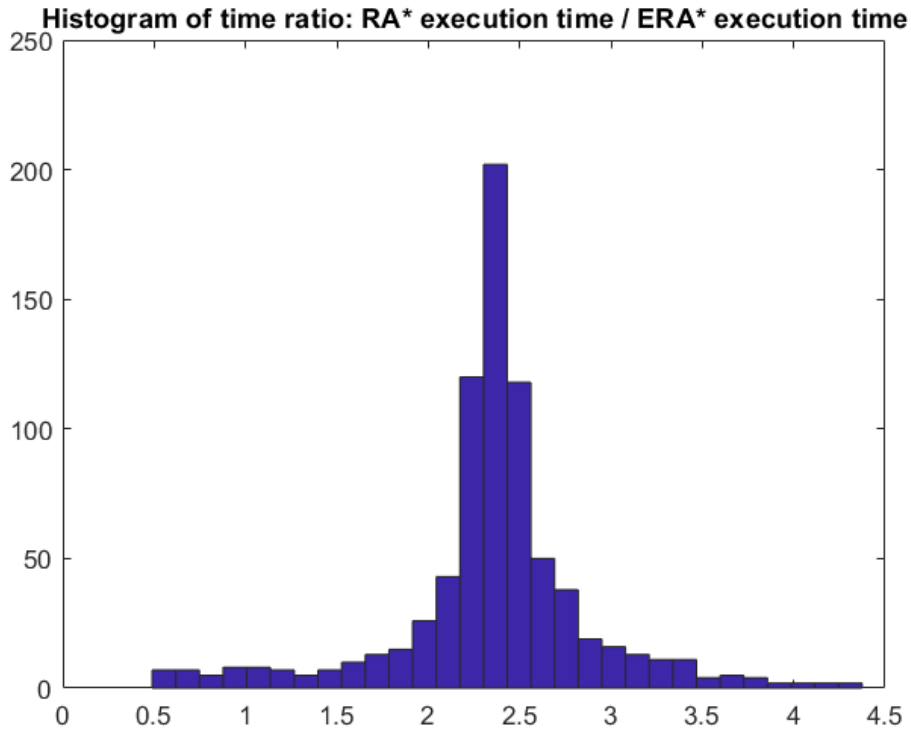

Figure 3. *Histogram of the execution time ratio between RA^*_{wot} and ERA^* algorithms.*

When compared to A^*, ERA^* is 17.13 times faster in average, with a range from 7.27× to 24.3×. Table 5 shows that ERA^* has consistently the fastest execution time among the three algorithms in nearly 90% of cases. This superiority applies in all tested grid environments as can be seen in the scatter plots in Figure 4 (randomly generated environments of various sizes) and Figure 5 (structured environments) which compare ERA^* and RA^*_{wot} in the Cost/Time space. These two figures also show that the range of path length for the two algorithms is similar.

Table 5. Percentage of runs for which each algorithm appears in rank 1 to 3, with regards to the execution time.

Rank	1	2	3
A^*_t	9.0%	5.3%	85.7%
RA^*_{wot}	1.1%	86.6%	12.3%
ERA^*	89.9%	8.1%	2.0%

Figure 6 depicts the box plot of the ratio between the execution time and the optimal path for the three algorithms. It shows a markedly reduced range for ERA^* and RA^* compared to A^*. On a closer scale, Figure 7 shows that ERA^* further reduces this range by almost one half compared to RA^*.

Figure 8 provides a comprehensive evaluation of the performance of the three algorithms by representing them in cost/time space, in terms of average and standard deviation. The aim of this comparison is to evaluate the performance of different algorithms for both the path cost and the execution time. The cost/time space, therefore, shows the trade-off between the cost and time incurred by each algorithm. The evaluation is done based on two metrics: average and standard deviation. The width of each rectangle is proportional to the path cost standard deviation, while the height is proportional to the execution time standard deviation. The average is represented by a star at the center of each rectangle. The figure shows that ERA^* provides the best tradeoff between cost and time, a reduced range of execution time, and a range of path cost that is close to RA^*'s.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

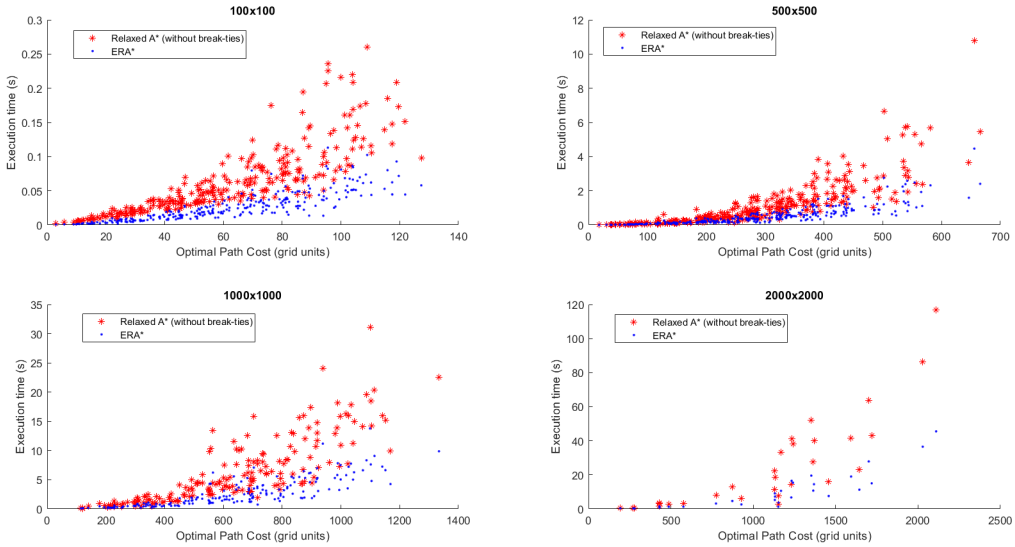

Figure 4. *Comparison of execution time between the proposed ERA^* (in blue) and RA^*_{wot} (in red) algorithms, for different environment sizes (from 100x100 to 2000x2000 nodes), randomly generated.*

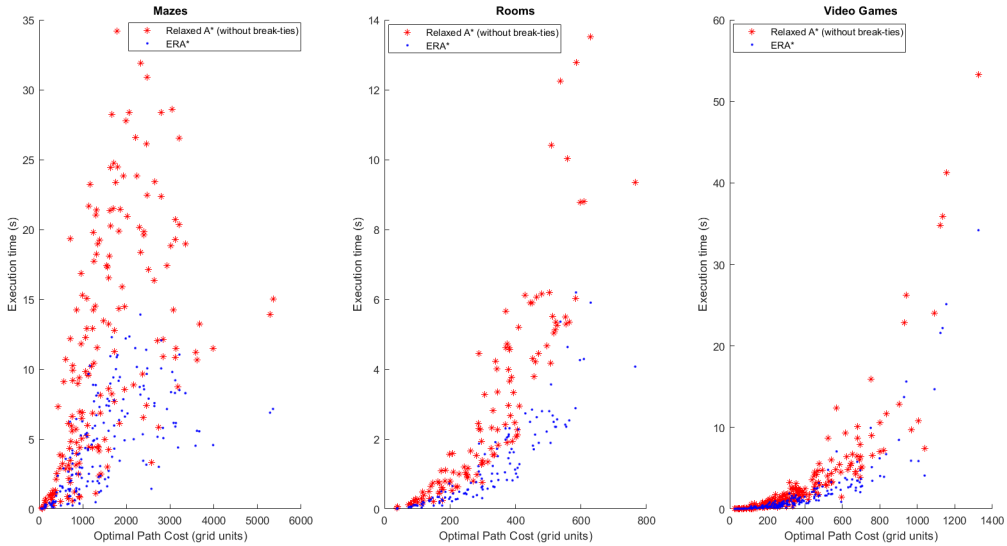

Figure 5. *Comparison of execution time between the proposed ERA^* (in blue) and RA^*_{wot} (in red) algorithms, for different structured environments.*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

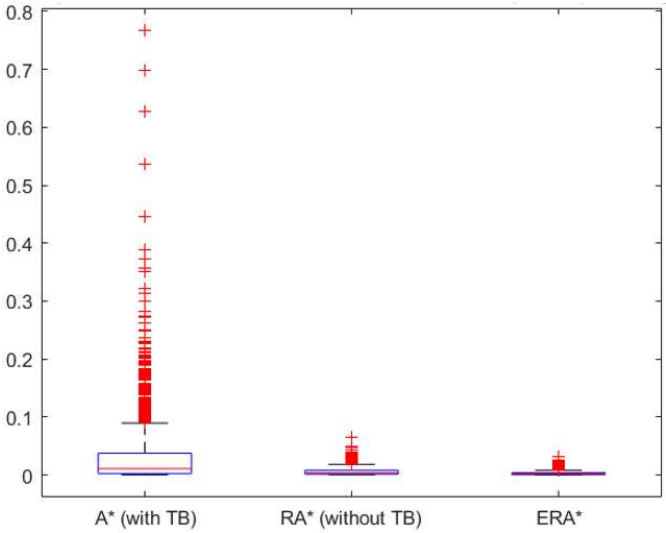

Figure 6. *Box plot of the execution time divided by the optimal path length, for A^*, RA^*_{wot} and ERA^* algorithms.*

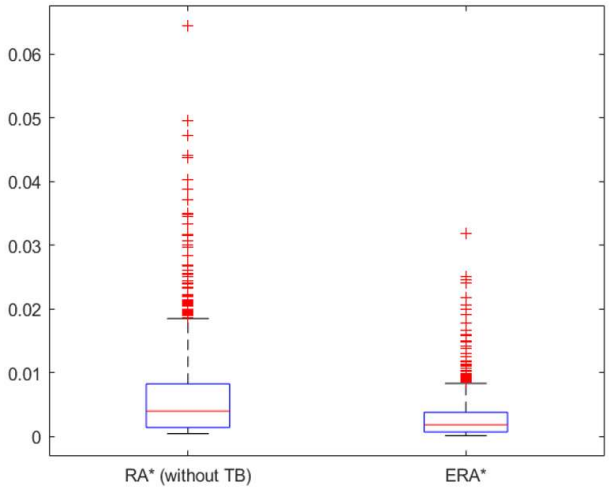

Figure 7. *Box plot of the execution time divided by the optimal path length, for RA^*_{wot} and ERA^* algorithms.*

Furthermore, we conducted an analysis on the cost and time performance of the three algorithms on the described set of experiments, using a T-test. The first comparison was between the ERA^* and RA^*_{wot} algorithms, where the null hypothesis of equal means for time was rejected with a very low p-value of $1.1216e^{-20}$. Whereas, the null hypothesis for cost was accepted with a p-value of 0.91, indicating that the difference in cost between the two algorithms is not statistically significant. The second comparison was between the ERA^* and A^*_t algorithms. Again, the null hypothesis for equal means in time was rejected with a very low p-value of $1.1002e^{-35}$, indicating that there is a statistically significant difference in time performance between the two algorithms. While the null hypothesis for cost was accepted with a p-value of 0.7607, indicating that the cost difference between the two algorithms is not statistically significant.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 August 2023 doi:10.20944/preprints202308.1215.v1

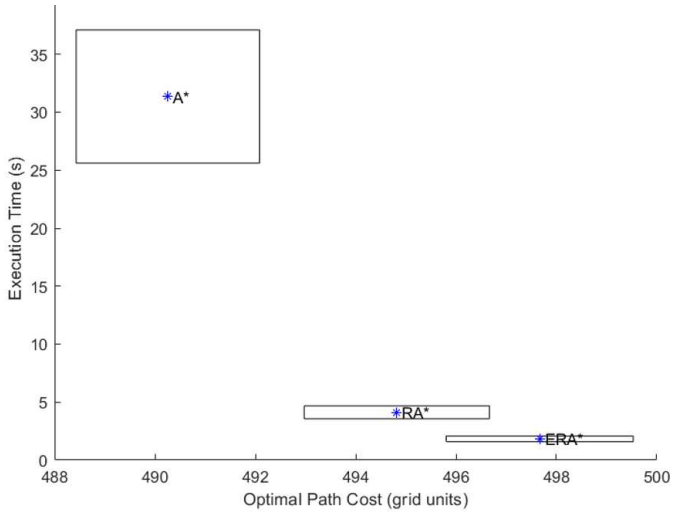

Figure 8. *Comparison between the tested algorithms in cost/time space, in terms of average and standard deviation. The width of each rectangle is proportional to the path cost standard deviation, and its height is proportional to the execution time standard deviation. The average is represented by a star at the center of each rectangle.*

5. Conclusion

In conclusion, this paper introduced a novel algorithm for solving the point-to-point shortest path problem on a static regular 8-neighbor connectivity (G8) grid. The algorithm is a generalized version of the Hadlock algorithm specifically designed for G8 grids. This work falls within the category of relaxed alternative algorithms that balance the trade-off between path length optimality and search time. By relaxing the constraint of path length optimality to some extent, the proposed Enhanced Relaxed A* (ERA*) algorithm accelerates the search process while still providing solutions with similar path lengths to those obtained using RA*. The key advantage lies in the novel computation strategy utilizing lookup matrices, which significantly reduces redundant calculations, resulting in substantial time and memory savings.

Experimental results obtained through extensive testing on various grid maps validate the algorithm's effectiveness. On average, it is 2.25 times faster than RA* and 17 times faster than the original A* algorithm. Additionally, it exhibits improved memory efficiency since it eliminates the need for storing a G score matrix.

Future research can focus on exploring further optimizations and extensions of the ERA* algorithm. This could involve investigating its performance in dynamic or uncertain environments, incorporating additional constraints or cost factors, and exploring possibilities for parallelization or distributed computation. Additionally, the algorithm's applicability to other grid types or graph structures can be explored. The insights gained from this work lay the foundation for potential advancements in efficient path planning algorithms for various domains and applications.

References

1. H.-y. Zhang, W.-m. Lin, A.-x. Chen, Path planning for the mobile robot: A review, Symmetry 10 (2018) 450.
2. A. Koubâa, H. Bennaceur, I. Chaari, S. Trigui, A. Ammar, M.-F. Sriti, M. Alajlan, O. Cheikhrouhou, Y. Javed, Robot path planning and cooperation, volume 772, Springer, 2018.
3. G. Udgirkar, G. Indumathi, Vlsi global routing algorithms: A survey, in: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), IEEE, 2016, pp. 2528–2533.
4. M. Radi, B. Dezfouli, K. A. Bakar, M. Lee, Multipath routing in wireless sensor networks: survey and research challenges, sensors 12 (2012) 650–685.

5. H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, R. F. Werneck, Route planning in transportation networks, Algorithm engineering: Selected results and surveys (2016) 19–80.
6. S. Zhu, D. Levinson, Do people use the shortest path? an empirical test of wardrop's first principle, PloS one 10 (2015) e0134322.
7. E. W. Dijkstra, A note on two problems in connexion with graphs:(numerische mathematik, 1 (1959), p 269-271) (1959).
8. E. W. Dijkstra, A note on two problems in connexion with graphs, in: Edsger Wybe Dijkstra: His Life, Work, and Legacy, 2022, pp. 287–290.
9. P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE transactions on Systems Science and Cybernetics 4 (1968) 100–107.
10. I. Châari, A. Koubâa, H. Bennaceur, A. Ammar, S. Trigui, M. Tounsi, E. Shakshuki, H. Youssef, On the adequacy of tabu search for global robot path planning problem in grid environments, Procedia Computer Science 32 (2014) 604–613.
11. F. H. Ajeil, I. K. Ibraheem, A. T. Azar, A. J. Humaidi, Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments, Sensors 20 (2020) 1880.
12. M. Alajlan, A. Koubaa, I. Chaari, H. Bennaceur, A. Ammar, Global path planning for mobile robots in large-scale grid environments using genetic algorithms, in: 2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR), IEEE, 2013, pp. 1–8.
13. M. D. Phung, Q. P. Ha, Safety-enhanced uav path planning with spherical vector-based particle swarm optimization, Applied Soft Computing 107 (2021) 107376.
14. C. Huang, X. Zhou, X. Ran, J. Wang, H. Chen, W. Deng, Adaptive cylinder vector particle swarm optimization with differential evolution for uav path planning, Engineering Applications of Artificial Intelligence 121 (2023) 105942.
15. A. Ammar, H. Bennaceur, I. Châari, A. Koubâa, M. Alajlan, Relaxed dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments, Soft Computing 20 (2016) 4149–4171.
16. M. Panda, A. Mishra, A survey of shortest-path algorithms, International Journal of Applied Engineering Research 13 (2018) 6817–6820.
17. C. Y. Lee, An algorithm for path connections and its applications, IRE transactions on electronic computers (1961) 346–365.
18. F. Rubin, The lee path connection algorithm, IEEE Transactions on computers 100 (1974) 907–914.
19. D. Harabor, A. Grastien, Online graph pruning for pathfinding on grid maps, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 25, 2011, pp. 1114–1119.
20. F. Hadlock, A shortest path algorithm for grid graphs, Networks 7 (1977) 323–334.
21. I. Pohl, First results on the effect of error in heuristic search, Machine Intelligence 5 (1970) 219–236.
22. J. Pearl, Heuristics: intelligent search strategies for computer problem solving, Addison-Wesley Longman Publishing Co., Inc., 1984.
23. I. Pohl, The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, in: Proceedings of the 3rd international joint conference on Artificial intelligence, 1973, pp. 12–17.
24. A. L. Köll, H. Kaindl, A new approach to dynamic weighting, in: Proceedings of the 10th European conference on Artificial intelligence, 1992, pp. 16–17.
25. S.-Q. Zheng, J. S. Lim, S. S. Iyengar, Finding obstacle-avoiding shortest paths using implicit connection graphs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15 (1996) 103–110.
26. N. R. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Transactions on Computational Intelligence and AI in Games 4 (2012) 144–148.