Expert Systems With Applications 249 (2024) 123719

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Guards: Benchmarks for weighted grid-based pathfinding

Sajjad Kardani Moghadam [a], Morteza Ebrahimi [a,*], Daniel D. Harabor [b]

[a] Department of Data Science and Technology, School of Intelligent Systems, University of Tehran, Tehran, Iran
[b] Department of Data Science and Artificial Intelligence, Monash University, Melbourne, Australia

ARTICLE INFO

Keywords:
Path finding
JPSW
Benchmark
Grid map
Video games
Path planning

ABSTRACT

The primary objective of this paper is to aid game developers in finding the most suitable pathfinding algorithm for their games. Despite recent advancements in this field, there are few available studies that can be compared due to the absence of a standard benchmark set for weighted environments. This paper presents a new dataset for pathfinding in weighted environments. Furthermore, an investigation was conducted into the impact of node weights on pathfinding speed, and a correlation between them was identified. The complexity added to the maps due to node weights was defined as weight complexity, and two metrics were introduced to estimate it. The weight correlation factor has been identified as the most effective metric for estimating the weight complexity of the map. Another contribution of this paper pertains to the development of a model for estimating the pathfinding speed of algorithms based on weight complexity. This was accomplished through the utilization of the non-linear least squares method, which was applied to create a model for each algorithm, considering both its average search time and weight correlation factor values associated with the map. Finally an overall score metric was developed by using the integral of the models, enabling the evaluation of different algorithms in various scenarios.

1. Introduction

Pathfinding in grid maps is a fundamental problem in various fields such as artificial intelligence, robotics, and computer games. A large number of algorithms have been proposed to tackle this problem. However, their evaluation has been primarily based on traditional benchmarks which usually do not support weighted nodes. In the field of pathfinding, benchmarking is a crucial task to evaluate the performance and effectiveness of different algorithms. Currently, standardized datasets are commonly used for benchmarking, which consist of grid maps(usually with uniform cost nodes) with pre-defined start and target locations. These datasets are designed to provide a variety of conditions and challenges, such as different map topologies, varying densities of obstacles, and multiple targets. The algorithms are evaluated based on the quality of the solution, including path length and optimality, as well as the computational resources required to find the solution, such as runtime and memory usage. pathfinding speed is often considered the most important metric. In particular, real-time applications such as video games and robotics require pathfinding algorithms that can quickly generate high quality paths. However, it is important to note that the majority of existing datasets only consider nodes as either passable or impassable, and do not support node weights in the grid map. However, in numerous scenarios nodes may have varying weights that can change during the search process. Therefore, it is crucial to develop new datasets that can serve as benchmarks for pathfinding algorithms capable of effectively accounting the node weights in the environments. These datasets will enable researchers to evaluate and compare the performance of pathfinding algorithms in realistic scenarios, leading to more efficient and effective solutions for complex pathfinding problems. In this regard, a novel dataset is proposed for evaluating pathfinding algorithms in gridmaps where nodes possess weights. In this research, it was observed that adding weights to grid cells can slow down grid pathfinding algorithms, a phenomenon referred to as weight complexity. This term is introduced in this paper to characterize the observed impact on algorithmic performance, providing a novel framework for understanding the distribution of weights across gridmaps. By utilizing these metrics, researchers can gain valuable insights into the factors that affect the performance of pathfinding algorithms and develop more effective approaches for solving the pathfinding problem in weighted gridmaps. Finally, a scoring system is presented to rank algorithms based on their performance on the proposed dataset. Our method paves the way for further exploration in this field by offering a comprehensive and realistic evaluation of pathfinding algorithms in weighted grid maps.

This paper presents several contributions in the field of pathfinding. Firstly, it introduces a novel approach for generating a weighted map dataset based on existing uniform cost maps. This approach enables

* Corresponding author.
E-mail addresses: sajjad.moghadam@ut.ac.ir (S.K. Moghadam), mo.ebrahimi@ut.ac.ir (M. Ebrahimi), Daniel.Harabor@monash.edua (D.D. Harabor).

https://doi.org/10.1016/j.eswa.2024.123719
Received 17 July 2023; Received in revised form 23 December 2023; Accepted 17 March 2024
Available online 19 March 2024

the evaluation of pathfinding algorithms in various weighted scenarios. Secondly, it proposes a metric to quantify the complexity of a given map by considering its characteristics, such as the number of obstacles and the distribution of weights. Thirdly, the paper presents a novel approach to develop a mathematical model for evaluating the performance of algorithms across varying levels of map weight complexity. Fourthly, it performs a comparative analysis of multiple existing algorithms using the generated dataset and utilizes the developed models. Finally, the paper proposes an overall scoring system that facilitates easy and effective comparison of different algorithms in this domain. Together, these contributions provide a comprehensive framework for evaluating and comparing pathfinding algorithms in gridmaps with weighted nodes.

2. Related works

In the field of path finding, benchmarking refers to the process of evaluating and comparing the performance of different algorithms or techniques for finding optimal paths in graphs or maps. It involves running these algorithms on various test cases or datasets and measuring their efficiency in terms of factors like runtime, memory usage, solution quality, or other relevant metrics. Our objective is to establish a standardized dataset to assess the performance of pathfinding algorithms and develop a comprehensive scoring system that can be utilized for evaluating the quality of using pathfinding algorithms in various computer games. Having a standard dataset is important for comparing pathfinding algorithms that can work in weighted environments. However, the number of maps in existing datasets with multiple types of land is limited and not diverse enough to enable a comprehensive evaluation. To provide an example, Stuartvent's dataset includes maps taken from popular games such as Warcraft and Starcraft, which feature up to four different terrain types (Sturtevant, 2012). This enables the testing of pathfinding algorithms in weighted conditions by assigning varying weights to different terrain types. Although these maps can be used in weighted conditions, the number of such maps is very small and 2–4 types of weights are not enough for a thorough comparison. There are numerous applications in which incorporating weight into nodes can prove to be beneficial. In video games, for instance, there could be various terrain types, and the speed of an agent traversing each of them could differ. Additionally, if a map features varying altitudes, ascending to a higher altitude may cause more difficulty for the agent. In such cases, including weight into the nodes during pathfinding can help. Furthermore, a common issue that arises in many games is that agents tend to move alongside walls in narrow corridors, which is not realistic. Assigning a higher weight to nodes situated near the sides of the corridor, using influence maps and potential fields (Jong, Kwon, Goo, & Lee, 2015), can help the agent move away from the walls and towards the center of the corridor. The use of a weights for nodes in a game can potentially improve pathfinding and enhance artificial intelligence, thereby contributing to a more realistic and immersive gaming experience. Another dataset developed for the DTA paper includes 16 terrain types (Sturtevant, Sigurdson, Taylor, & Gibson, 2019), but it only contains 20 maps and, on average, uses six different weights in each map and all pieces of terrain less than 80 pixels have been removed during the dataset's pre-processing (Sturtevant et al., 2019). Additionally, the weight of each terrain type in the dataset was not specified and was randomly assigned for each test. This inconsistency in weight allocation can make it difficult and inappropriate to compare benchmarks. The lack of precise weight information can result in misleading and unreliable results. Therefore, it is important to establish a consistent weight distribution to ensure the accuracy and validity of the results and to facilitate benchmark comparison across different studies. Furthermore, One significant limitation of this dataset is that it considers all map nodes as passable and does not include any impassable nodes, which can be a major drawback for evaluating the performance of pathfinding algorithms in realistic scenarios where obstacles and restricted areas need to be considered (Sturtevant et al., 2019). Stern et al. aimed to establish a standardized benchmark for multi-agent pathfinding problems, To accomplish this, they gathered a collection of maps from different trusted sources (Stern et al., 2019). They used Dragon Origin maps (Sturtevant, 2012), $N \times N$ grids with random obstacles (Standley, 2010) and Warehouse grids (Cohen et al., 2018). Iron Harvest is another benchmark in pathfinding field (Harabor, Hechenberger, & Jahn, 2022). It presents a novel benchmark for evaluating pathfinding algorithms by providing multiple map representations, including grid, mesh, and obstacle-set for 35 different maps. This allows for comparison of algorithms that work on different types of maps, and provides a comprehensive evaluation of their performance. By offering a common set of challenging instances, the benchmark helps researchers and practitioners better understand the strengths and weaknesses of different pathfinding techniques. It is worth noting that Iron Harvest's benchmark is limited to uniform cost gridmaps, which may not be suitable for scenarios where the weight of the nodes is an important factor in pathfinding. To create a standard dataset that can be used to analyze the weight complexity of the maps, it is essential to include maps with the same main skeleton (i.e., passable and impassable points) and scenarios in different versions, each with different weight distribution. This way, the impact of the weight complexity on pathfinding speed can be appropriately evaluated, which will be further explained in detail in the following sections. The table presented in 1 provides a comparative analysis of the attributes associated with well-known pathfinding benchmarks.

To test our dataset, the A* (Hart, Nilsson, & Raphael, 1968) and JPSW algorithms are selected. These two algorithms were selected due to their inherent lack of limitations, which makes them well-suited to be used in various types of games without imposing any restrictions. In modern games, the maps often incorporate weights on the nodes, and these weights can undergo changes over time, such as alterations caused by enemy movements. Hence, our primary focus during the testing phase of this research is on evaluating algorithms that can effectively manage node weights and adapt to dynamic changes. The A* is the oldest intelligent path-finding algorithm, and is our baseline with which to compare other algorithms, and has served as the only option to find optimal paths in dynamically weighted gridmaps for more than five decades. The JPSW is the state-of-the-art algorithm, and we believe it is the only algorithm after A* that can take into account weight and dynamic changes and guarantee the optimal path in the grid map. The JPSW algorithm is a more advanced version of the JPS algorithm that operates in a weighted environment. This modification involves the introduction of revised jumping rules to accommodate the weight of the environment and the application of two novel pruning methods to improve pathfinding efficiency (Carlson, Moghadam, Harabor, Stuckey, & Ebrahimi, 2023). The only other modern option is the Dynamic Terrain Abstraction (DTA) algorithm which is a hierarchical path-finding algorithm and can find sub-optimal paths in dynamic conditions (Sturtevant et al., 2019). There are many other fast algorithms in the pathfinding literature but the focus of this research is to help game developers to find best possible algorithm for their games. A major branch of algorithms developed in this field only work in the uniform cost conditions where there are only passable and impassable nodes (Harabor & Grastien, 2011; Uras, Koenig, & Hernandez, 2013; Yap, Burch, Holte, & Schaeffer, 2011). In the 2014 competition, most of these algorithms were compared to one another (Sturtevant et al., 2015). The weight of the environment cannot be handled by these methods. In modern games, it is necessary to use varying weights for different nodes. For example, if the speed of the agent differs between river, road, forest, and other terrain types, then the correct weight must be set for nodes with different terrain types in order to find the most optimal possible path for the agent. A significant number of algorithms aim to make pathfinding faster by performing offline pre-processing on maps. These algorithms typically require a substantial amount of pre-processing, which makes

Table 1
Pathfinding benchmarks attributes.

Benchmark	Number of maps	Support weight	Specified weight	Contain impassable nodes	Grid based maps	Mesh based maps	Support multi agent	Publish year
Benchmarks for Grid-Based Pathfinding (Sturtevant, 2012)	608	No	Yes	Yes	Yes	No	No	2012
DTA (Sturtevant et al., 2019)	20	Yes	No	No	Yes	No	No	2019
Grid-based MAPF (Stern et al., 2019)	24	No	Yes	Yes	Yes	No	Yes	2019
Iron Harvest (Harabor et al., 2022)	105	No	Yes	Yes	Yes	Yes	No	2021
Guards	4032	Yes	Yes	Yes	Yes	No	No	2023

them unsuitable for dynamic environments as any changes to the map invalidate the pre-computations (Botea, 2012; Cohen et al., 2017; Geisberger, Sanders, Schultes, & Delling, 2008; Sturtevant, Felner, Barrer, Schaeffer, & Burch, 2009). Within the literature, there exists a body of work that attempts to work in dynamic situation. For example Dibbelt et al. try to repair preprocessed data (Dibbelt, Strasser, & Wagner, 2016). Bono et al. use compressed path database as a lower bound heuristic for path planning to achieve bounded sub-optimal path faster in dynamic environments (Bono, Gerevini, Harabor, & Stuckey, 2019). However, these methods have their limitations. They are effective only when costs consistently increase, causing lower-bounding heuristics to weaken without becoming inadmissible. The costs can also decrease. In spite of that, as long as they never fall below a certain minimum threshold, the estimates remain admissible (Mahéo et al., 2021). Nevertheless, dynamic changes are necessary in many modern applications, Therefore, it is essential for pathfinding algorithms used in such scenarios to be able to handle dynamic changes in real-time, without requiring a costly recomputation of the path.

This research focuses on the regular grid representation of maps and the corresponding pathfinding algorithms. However, there are alternative terrain representations, including hierarchical grids that incorporate methodologies such as Probabilistic Roadmap (Rohrmuller, Althoff, Wollherr, & Buss, 2008), quad trees (Brondani, Silva, Zacarias, & de Freitas, 2019), and octrees (Zhao et al., 2022). Nevertheless, this paper does not explore a detailed discussion of these alternative types.

3. Generating dataset

In producing this benchmark, efforts are made to draw inspiration from video games. The foundation of the Guards concept lies in the idea that the risk associated with agent passage between nodes can be converted into node weights. This approach enables artificial intelligence to make more informed decisions when selecting a path to its destination. With this concept, guards can be strategically placed at random locations on a weightless map, where they influence the weight of neighboring nodes. This process results in the creation of a weighted map. Assume that an agent must travel from one location to another where there is a large number of enemy guards in the environment. In the context of our system, it is defined that each guard has a specific range, which can cause detrimental effects to an agent passing through it. This range, commonly referred to as the guard's field of view (FOV), is characterized by a specific range within which an agent is subject to damage per unit time. Shi and Crawfis had similar considerations in their works (Shi & Crawfis, 2013, 2014). They discusses a computational method for determining the optimal placement of cover against static enemy positions for level design in games. The authors propose a mathematical model that takes into account factors such as the number and location of enemy positions, the range of weapons, and the terrain. The risk of passing from the guards' field can be added to grid nodes' weight. Jong et al. stored the risk of crossing each node in an influence map and built their heuristic function according to the weight of the risk relative to the weight of distance (Jong et al., 2015). Since our intention was to create a public dataset for this study, all data of each node is preferable to be stored in one character, therefore, the following approach is used to determine the weight of each node.

Fig. 1. The line of sight of guards, represented by the red area, cannot penetrate impassable nodes represented by the black area in the maps.

Each terrain type initially carries a baseline weight denoted as 'T'. However, when it enters the line of sight of 'g' guards, its weight is subject to adjustment. In this study, the terrain's weight is calculated by multiplying it by 2^g to determine the updated node weight. Each guard has a range (based on the terrain type), and their sight cannot pass through impassable nodes. Weight of nodes can be limited to a specific value. In the current generated dataset the weight of nodes are limited to 2^{20}. Impassable nodes prevent guards' lines of sight from passing through them. This is demonstrated in Fig. 1, where the lines of sight for five guards are highlighted in red on the map. The weight of a node in our system is determined by the number of guards that can observe it. During our testing, we found that for our tower defence game an exponential function was the most effective way to set the weight based on the number of guards. However, depending on the specific application, other functions such as linear weighting may be more appropriate. Our initial motivation for developing this system was to create a "tower defence" game, where the artificial intelligence would have to decide

which base to attack, each of which is protected by a different number of guards. Through our experiments, it was determined that attacking a farther base with fewer guards was often the better decision. Therefore, the node weights were set exponentially based on the number of guards, guiding the artificial intelligence towards better decisions. Algorithm 1 show how to calculate the weight of each cell.

Algorithm 1 Calculate guards map cells weight

```
Let c be a cell of grid map grid.
Let g be a guard in Guards.
for c ∈ grid do
  for g ∈ Guards do
    if IsInGuardRange(c, g) then
      if IsVisibleToGuard(c, g) then
        c.weight = min(2^20, c.weight * 2)
      end if
    end if
  end for
end for
```

The guards dataset is derived from Sturtevant's popular benchmark maps (Sturtevant, 2012). In this dataset, a total of 669 maps have been categorized into game benchmarks (Dragon Age, Warcraft, Baldur's Gate II, and Starcraft), real-world benchmarks, and artificial benchmarks (Mazes, Random maps, and Square Rooms). The Guards dataset is generated by adding (0, 64, 128, 256, 512, 1024, 2048) guards to each map. To ensure that up to 2048 guards could be placed over each map, maps with a passable node count of less than 2048 were excluded. As a result, 27 maps from the Dragon Age 2009 dataset and 5 maps from the Dragon Age 2 dataset were removed due to their lower node count. Therefore, a total of 4459 maps were generated using 637 baseline uniform cost maps and categorized accordingly. The original dataset presents each map with a problem set. Each problem set contains a number of scenario instances. Each scenario consists of a start and an end point. It has been ensured that the starting and ending points of the scenarios are passable and there is definitely a feasible path to get from the starting point to the ending point. All seven guard maps were generated using the same problem set, and no new scenario instances were created in this work. The new dataset are publicly available for download from GitHub repository: https://github.com/Sajjad-moghadam/Guards

The format of the Guards dataset files are described in Section 3.1.

3.1. Files format

A Windows application was developed for generating weighted maps based on the concept of Guards from weightless maps. This application is publicly available for download from the mentioned repository.

The application processes a folder containing map files in the Stuartvent 2012 file format, as well as corresponding scenario files located in an inner folder named "scen". The presence of a Metadata.txt file is essential for generating new maps, which should contain data lines about terrain types, including their default weight, FoV coefficient, and default color, separated by the tab character. The default characters used for terrain types include '.' for normal passable nodes, 'T' for tree nodes, 'S' for shallow water, '@' for impassable nodes, and 'W' for water. These default values should be choose and set by the user. Visualization of the map in png format is one of the application outputs. The default colors will be used to generate this image. The weight 16 777 216 or 2^{24} is used as the symbol of impassable nodes. Table 2 lists the default values used to generate Guard dataset.

Second line of Table 2 indicated that T (Tree) nodes have a base weight of 2, and in this node type, the default FoV of guards should be reduced to half. The last parameter is the default color of this terrain type. If other researchers wish to use this application to generate weighted maps based on a other map that contains different characters, they must define the corresponding values for those characters in the metadata.txt file.

Table 2
The default values which are used to generate Guards dataset.

Char	Weight	FOV	Color
.	1	1	#ffffff
T	2	0.5	#267928
S	4	1.25	#5eadd2
W	16 777 216	0	#0019d4
@	16 777 216	0	#2d2d2d

Five new files are generated for each baseline map to create the Guards dataset, as described below:

- A new scenario file (problem set) is almost the same as the input scenario file.
- An image of the new map in png format.
- An image of new map edges in png format.
- A new map file stores impassable nodes with the '@' character. The number of guards who have that node in their range of view is stored for the other nodes. For numbers higher than 9, the characters a, b, c, ... are used.
- An extra data file which only contain weight correlation factor value of the map at the moment.

For the Guards dataset, 4032 maps and 20 160 files are generated in general.

4. Experimental results

The Warthog pathfinding research library is utilized to evaluate the speed of pathfinding algorithms on the Guards dataset. The library is implemented in C++ and contains the A* and JPSW algorithms. The full source code of the Warthog library and JPSW implementation are available at the following links:

- Warthog library: bitbucket.org/dharabor/pathfinding/src/master/
- JPSW implementation: bitbucket.org/mcar0024/pathfinding

The implemented A* algorithm in this library is the fastest implementation in our experience. The experiments were conducted on an Ubuntu 20.04 operating system, utilizing an Intel Core i7 4710HQ processor and 16 GB of 1600 MHz DDR3 memory. During the testing phase, the system was exclusively dedicated to the execution of the experiment without any concurrent processes. In our implementation, only a single core of the CPU running at a clock speed of 3.5 gigahertz (GHz) is employed. Both A* and JPSW need to determine which of their nodes to expand on each iteration of their main loop. It does so based on the cost of the path up to the current node. In addition, it estimates the cost required to extend the path all the way to the goal. Specifically, they selects the path that minimizes:

$$f(n) = g(n) + h(n) \tag{1}$$

where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. By adding a W multiplier to h(n) where $W > 1$ the priority of expanding nodes further away from the target will be reduced. The A* with $W > 1$ usually find the path faster than A* but at the cost of losing optimality. The purpose of this comparison is to identify the best algorithm for pathfinding in computer games, and some sub-optimality is acceptable in most games. Both A* and JPSW algorithms are also tested with $W = 2$. The theoretical guarantee of A* search algorithm with a consistent heuristic is that it will not expand any state more than once. However,

Table 3
Algorithms average search time and suboptimality.

Algorithm	Average time	Suboptimality
A*	7.56	0%
JPSW	3.84	0%
A*($W = 2$)	3.57	5.08%
JPSW($W = 2$)	1.80	4.57%

by setting $w > 1$ in weighted A* search (Pohl, 1970), consistency may be violated, resulting in re-expansion of states multiple times during the search process. In our implementation of the algorithms with $W = 2$, A policy to avoid node re-expansion was enforced to optimize pathfinding speed. This was achieved by disallowing the reopening of nodes during the search process, ensuring that each node is expanded exactly once. Likhachev et al. demonstrate that this will not invalidate A*'s theoretical suboptimality bounds guarantees (Likhachev, Gordon, & Thrun, 2003). In our tests, it causes an average of less than 1% to be added to the sub-optimality of the paths. But it greatly increases performance. In the case of JPSW, in rare cases when $W > 1$ it may occur that the algorithm is unable to find the path. Therefore, in these cases, an extra search is done with $W = 1$ so that the algorithm is complete. Table 3 presents the results of running these algorithms over all 4032 Guards dataset maps for more than twenty million scenarios.

As mentioned earlier, the level of complexity of the map has a huge effect on the pathfinding speed and also it has different effect on reducing the speed of different algorithms. To investigate this issue, it is necessary to design parameters that quantify the complexity of the map and the sensitivity of the algorithms to this complexity. Therefore, in the next section, an attempt has been made to devise suitable parameters for valuing these items. Therefore, to design a more efficient scoring system, it is necessary to take into the account the complexity of the maps. Also, with this information, game developers can choose a more suitable algorithm according to the complexity of their game map.

5. Weight complexity of maps

In the pathfinding field, there are several commonly used metrics for evaluating the complexity of gridmaps. These metrics typically focus on factors such as obstacle density, connectivity, and overall size of the environment (Kavraki, Svestka, Latombe, & Overmars, 1996; Sturtevant, 2012). While these metrics offer a fundamental understanding of the challenges posed by different environments, none of them specifically analyze the impact of nodes weight distribution on the difficulty of pathfinding. This effect is important for us because the algorithms, which can handle dynamic changes in the environment and can be used generally in all games (like JPSW and DTA), achieve their speed improvements by exploiting the symmetry of the gridmap, which inversely correlates with the weight distribution. Consequently, the performance of these algorithms varies significantly depending on the weight distribution of the gridmaps. Therefore, it is essential to develop a new metric that directly correlates with the weight distribution of gridmaps.

The previous sections introduced a new benchmark for comparing online path-finding algorithms in a weighted environment. Fig. 6 in practical way shows that increasing the number of guards significantly reduces the speed of pathfinding algorithms on the same map with the same scenarios. Fig. 6 illustrates how increasing the number of guards significantly influences the speed of pathfinding algorithms on identical maps under the same scenarios. This reduction is due to the decrease in symmetry and increase in weight complexity of the maps. Symmetrical paths in the context of pathfinding refers to the property of a map where certain paths can be transformed into equivalent paths by reordering the moves in a different way. Specifically, if a new path

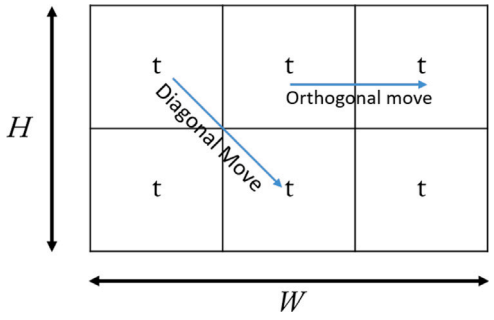

Fig. 2. Types of movement over grid maps.

and its permutation are both valid, and they have the same cost, It can be said that they are symmetric.

Consider a grid map consisting of $H \times W$ cells, where each cell is assigned a terrain cost $t \in \mathbb{R}^+$. A move from one grid cell to another adjacent grid cell is represented by a vector $\vec{m}$, which has a corresponding direction and magnitude. Each move can be represented by a tuple of direction and magnitude $\vec{m} = (\vec{g}, |\vec{m}|)$ where $\vec{g} \in (\vec{N}, \vec{E}, \vec{S}, \vec{W}, \vec{NE}, \vec{NW}, \vec{SE}, \vec{SW})$ and $|\vec{m}| \in \mathbb{R}^+$. These moves can be categorized into two groups , as shown in Fig. 2. the first group comprises orthogonal moves, including directions $(\vec{N}, \vec{E}, \vec{S}, \vec{W})$; the second group consists of diagonal moves, with directions $(\vec{NE}, \vec{NW}, \vec{SE}, \vec{SW})$. The magnitude of each move will be determined by the cost model. In this work, a cost model is adopted in which the cost of moving from one grid cell to another adjacent grid cell is equal to the magnitude of the corresponding move vector multiplied by the weighted average terrain cost, considering all the tiles intersected during the move action. Specifically, the cost of an orthogonal move is equal to the average terrain cost of the cells being moved from and moved to. The cost of a diagonal move is the weighted average of the four cells that the diagonal move intersects, i.e., the cells that the move vector touches. These cells are the source cell, the target cell, and the two cells diagonally adjacent to the target cell. The weighted average is multiplied by $\sqrt{2}$, as any non-point agent will intersect all four cells. Other cost models are possible and have been considered elsewhere, including multiplying the magnitude of the move vector by the terrain type of the destination cell or taking the average of the terrain costs of the source and destination cells. The concepts presented in this work are applicable to these other cost models. The uniform cost model is also considered as a special case, which is a grid where each cell is either passable with unit terrain cost or impassable with infinite cost (see Fig. 2). Mathematically, Symmetrical paths is defined in Definition 1:

Definition 1 (*Symmetrical Paths*)**.** Let $G = (V, E)$ be a graph representing a pathfinding map, where V is the set of vertices (grid cells) and E is the set of edges (connections between adjacent cells). A path P from vertex $s \in V$ to vertex $t \in V$ is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $v_0 = s$, $v_k = t$, and $(v_i, v_{i+1}) \in E$ for $0 \leq i \leq k$ and path $P' = (v_0, v'_i, \ldots, v'_{k-1}, v_k)$. The path P can be represented as a sequence of orthogonal and diagonal moves as $P = (\vec{m}_1, \ldots, \vec{m}_{k-1}, \vec{m}_k)$ where each move $\vec{m}_i$ corresponds to the edge (v_{i-1}, v_i). Two paths P and P' are symmetric if there exists a permutation σ such that $P' = \sigma(\vec{m}_1, \ldots, \vec{m}_{k-1}, \vec{m}_k)$.

Based on Definition 1 the cost of path P is equal to the cost of path P'.

Definition 2 (*Optimal Paths*)**.** An optimal path from vertex $s \in V$ to $t \in V$ is a path with the minimum possible cost among all paths from s to t.

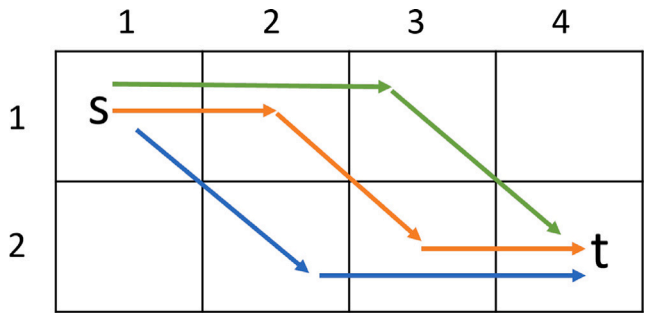

Fig. 3. Paths 1 and 2 are optimal and symmetrical, while Paths 3 and 4 are only symmetrical.

Definition 3 (*Optimal Symmetrical paths*)**.** Optimal symmetrical paths refer to the number of paths between points $s \in V$ and $t \in V$, where path P is not only optimal but also possesses at least one symmetrical counterpart, path P'.

For example, Paths 1 and 2 in Fig. 3 are both optimal and symmetrical. Path 1 can be defined as $P1 = (v_{(1,1)}, v_{(2,2)}, v_{(3,2)})$. which can be represented as a sequence of moves $P1 = (\vec{m}_{(1,1)=>(2,2)}, \vec{m}_{(2,2)=>(3,2)})$ equal to $P1 = ((\vec{SE}, \sqrt{2}), (\vec{E}, 1))$. Path 2 can be defined as $P2 = (v_{(1,1)}, v_{(2,1)}, v_{(3,2)})$. which can be represented as a sequence of moves $P2 = (\vec{m}_{(1,1)=>(2,1)}, \vec{m}_{(2,1)=>(3,2)})$ equal to $P2 = ((\vec{E}, 1), (\vec{SE}, \sqrt{2}))$. As evident, the path P2 is a permutation of the path P1. Fig. 3 also illustrates paths P3 and P4, which, while not optimal, exhibit symmetry.

Definition 4 (*Gridmap Symmetry*)**.**

- Let N be the total number of vertices in V.
- Let TP be the total number of possible pairs of vertices in the gridmap. If all nodes are passable, the total number of pairs of nodes would be $\frac{N \cdot (N-1)}{2}$.
- Let SP_Count be the count of Optimal symmetrical paths between $s \in V$ and $t \in V$.
- The symmetry of the gridmap, denoted as $SymGridmap$, can be calculated as:

$$SymGridmap = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} SP\_Count_{i,j}}{TP}$$

In two gridmaps with the same vertex count, the map with a higher $SymGridmap$ is more symmetrical.

Definition 5 (*Base Gridmap Symmetry*)**.** If the weight of nodes in a gridmap is not considered, the nodes can only be classified as impassable or passable. When the Gridmap Symmetry is calculated under this condition, it is referred to as the Base Gridmap Symmetry.

Based on Definition 1, it can be concluded when the diversity of weights in a gridmap is high, it is likely that the number of symmetrical path count become lower.

So the weight complexity of the map is defined as follows:

Definition 6 (*Weight Complexity*)**.** Let $Base\_Sym$ be the Base Gridmap Symmetry calculated when the weights of nodes are not considered. Let $Grid\_Sym$ be the Gridmap Symmetry calculated when the weights of nodes are taken into account. Weight Complexity, denoted as $Weight\_Complexity$, can be defined as:

$$Weight\_Complexity = \frac{Base\_Sym}{Grid\_Sym}$$

The calculation of the symmetry and weight complexity of a gridmap using this approach is computationally expensive, particularly for large gridmaps, as it involves checking all optimal path permutations for all possible pairs of nodes. For example, in a grid without obstacles, there are 4 symmetrical paths from node (1,1) to (2,4), 32 symmetrical paths to (4,8), 6434 symmetrical paths to (8,16), and 300,540,194 symmetrical paths to (16,32). This value increases exponentially, making the calculation of weight complexity based on its definition quite challenging. For instance, computing it for a 512×512 map could require months of computation on a standard CPU. So another approaches is used to estimate the weight complexity of the maps. The first approach is chunked weight complexity. In this method, the map is divided into 16×16 chunks, and the weight complexity is calculated separately for each chunk. Subsequently, the results are aggregated by addition. Computing chunked weight complexity for a 512×512 map can be done in one day on a standard CPU.

The Weight Correlation Factor (WCF) represents the second approach introduced to model this complexity. This approach is based on the observation that when a node has a weight different from its neighbors on the map, it **sjd**: reduces symmetrical paths while increasing the overall complexity for pathfinding. This complexity can be quantified by assessing the frequency of these weight differences with neighboring nodes. Identifying these differences in weight with neighboring nodes can be managed in a manner akin to the edge detection process used in image processing. For this purpose, the Sobel operator is used in this research (Kanopoulos, Vasanthavada, & Baker, 1988). The Sobel filter is an edge detection filter that utilizes image derivatives. Gridmaps can be thought of as an image, where each node is a pixel. Image derivatives are mathematical operations used to calculate the rate of change of pixel values in an image. They are essential in various image processing tasks, including edge detection, feature extraction, and image enhancement. Derivatives provide information about the intensity variations in an image, indicating regions of significant change in pixel values. Edges and boundaries between objects can be identified by computing derivatives, which are important for image analysis, object recognition, and computer vision applications.

The Sobel operator uses two 3×3 kernels which will convolved with the map to calculate approximations of the derivatives. One for horizontal changes, and one for vertical. A kernel or convolution matrix is a small matrix used in image processing for edge detection and other image related operations like blurring and sharpening. This is accomplished by doing a convolution between the kernel and an image or more simply, when each pixel in the output image is a function of the nearby pixels (including itself) in the input image, the kernel is that function.

Convolution of f and g is expressed as $f * g$, denoting the operator with the symbol $*$. This function is defined as the integral of the product of two functions after one of them has been reflected about the y-axis and shifted.

In order to define the convolution of two finite sequences, the sequences must be extended to finitely supported functions on the set of integers. The coefficients of the ordinary product of two polynomials are the convolution of the original two sequences when the sequences are the coefficients of two polynomials. This is called the Cauchy product of the coefficients of sequences in mathematics.

$$(f * g)[n] = \sum_{m=-M}^{M} f[m]g[n-m]. \tag{2}$$

If M is defined as the source map, and G_x and G_y are two kernel which at each point contain the horizontal and vertical derivative approximations respectively, the $*$ here represents the 2-dimensional signal processing convolution operation. the computations are as follows:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * M$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * M$$

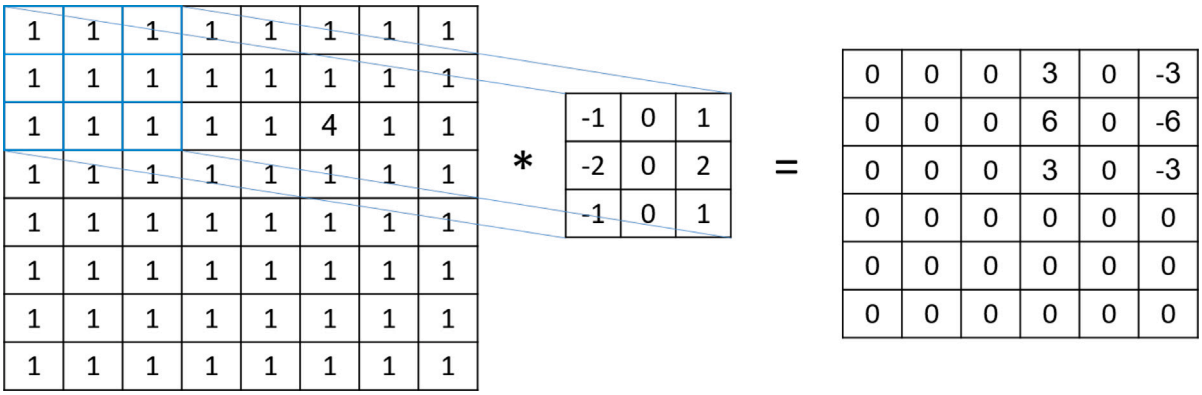

Fig. 4. A 6 × 6 image, padded by one row and one column on each side, is convolved with a horizontal Sobel filter kernel. The kernel slide over the image and producing a new 6 × 6 image.

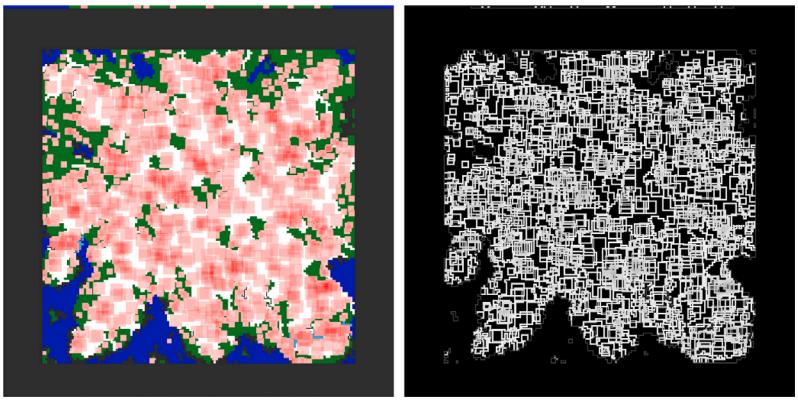

Fig. 5. Left: a sample map with 1024 guards added to it. Right: A visualization of C values.

By combining the gradient approximations at each point in the map, the gradient magnitude can be calculated as follows:

$$G = \sqrt{G_x^2 + G_y^2}$$

The Sobel filter is sensitive to the amount of changes and assigns a higher value to greater changes. However, in our application, minor changes create a little more complexity for the algorithms because higher weights cause that node to go down in the priority queue of path-finding algorithms and help to prune that branch. Therefore, a change from weight 1 to weight 1024 creates a little less complexity for the algorithms than changing from 1 to 2. It is worth mentioning that this assumption may not hold true for well-informed algorithms. However, these types of algorithms typically struggle with handling dynamic changes and fall outside the scope of our discussion. To address this fact, Eq. (3) is presented to calculate how much complexity will be added to the map by each edge based on the Sobel filter result G. In this equation G_i is refer to value of a pixel at position i in the output image G and C_i is the converted value which will indicate the amount of complexity added to the map by that pixel (see Fig. 4):

$$C_i = \begin{cases} 0 & \text{if } G_i \text{ is } 0 \\ 1 - 0.01 * log2(G_i) & \text{otherwise} \end{cases} \tag{3}$$

The multiple of 0.01 in Eq. (3) is based on our experience which reduces the amount of complexity that is added to the map by larger changes. Fig. 5 is a visualization of C values which are extracted from the source map.

By defining the number of passable nodes of the map as t_M To calculate the weight correlation factor, the total value of C is divided by t_M, which will be a numerical result between 0 and 1. As this number approaches zero, the map becomes more symmetrical, and as it approaches one, the map becomes more complex. Eq. (4) shows how to calculate WCF:

$$WCF = \frac{\sum_i^n C_i}{t_M}. \tag{4}$$

Algorithms 2, 3, and 4 present the pseudocode for calculating WCF. The Table 4 displays the WCF values for the overall guards dataset, as well as for each subfolder with varying numbers of guards.

Table 4
This table presents the WCF values for both the overall guards dataset and for each subfolder within the dataset.

Guards	0	64	128	256	512	1024	2048
Overal	0.26	0.31	0.35	0.41	0.49	0.59	0.69
Warcraft	0.14	0.17	0.20	0.26	0.36	0.48	0.65
Starcraft	0.09	0.11	0.14	0.18	0.26	0.36	0.50
Rooms	0.22	0.24	0.25	0.28	0.33	0.42	0.53
Random	0.76	0.77	0.78	0.79	0.80	0.83	0.86
Maze	0.43	0.44	0.45	0.47	0.50	0.56	0.64
DA2009	0.32	0.40	0.46	0.54	0.64	0.75	0.83
DA2011	0.30	0.44	0.53	0.63	0.74	0.85	0.86
City	0.07	0.10	0.14	0.20	0.31	0.44	0.59
BG	0.08	0.17	0.25	0.37	0.49	0.64	0.78

Algorithm 2 Calculate WCF

1: Let t_M be the number of passable nodes
2: $CValues \leftarrow CalculateCValues()$
3: $WCF \leftarrow \frac{CValues}{t_M}$
4: **return** WCF

Fig. 6 displays the Minimum, First Quarter, Second Quarter (Average), Third Quarter, and Maximum search times for JPSW and A* algorithms on seven generated Guard maps, derived from the Bootybay map with varying guard numbers. The tested scenarios on all seven maps are the same. The only difference will be the number of guards on the map (which will change the complexity and WCF of the map). As it can be seen the JPSW is more sensitive to the WCF of the map than A*. Therefore, it seems necessary to provide a parameter to consider this sensitivity in the algorithm scoring system.

Algorithm 3 Calculate C values from sobel filter results

Let $height$ be the height of grid map.
Let $width$ be the width of grid map.
for $y = 1$ **to** $height - 2$ **do**
  for $x = 1$ **to** $width - 2$ **do**
    if $map[y, x]$ **is** $impassable$ **then**
      $CValue[y, x] = 0$
      continue with next iteration
    end if
    $G_{y,x} \leftarrow CalculateSobelFilter(y, x)$
    if $G_{y,x}$ is 0 **then**
      $CValue[y, x] \leftarrow 0$
    else
      $CValue[y, x] \leftarrow 1 - (0.01 * \log_2(G_{y,x}))$
    end if
  end for
end for
return $Cvalue$

Algorithm 4 Calculate Sobel filter for each node

$gx \leftarrow 0$
$gx \leftarrow gx - map[y - 1, x - 1]$
$gx \leftarrow gx - map[y, x - 1] * 2$
$gx \leftarrow gx - map[y + 1, x - 1]$
$gx \leftarrow gx + map[y - 1, x + 1]$
$gx \leftarrow gx + map[y, x + 1] * 2$
$gx \leftarrow gx + map[y + 1, x + 1]$
$gy \leftarrow 0$
$gy \leftarrow gy - map[y - 1, x - 1]$
$gy \leftarrow gy - map[y - 1, x] * 2$
$gy \leftarrow gy - map[y - 1, x + 1]$
$gy \leftarrow gy + map[y + 1, x - 1]$
$gy \leftarrow gy + map[y + 1, x] * 2$
$gy \leftarrow gy + map[y + 1, x + 1]$
$g \leftarrow sqrt((gx * gx) + (gy * gy))$
return g

Table 5 illustrates the WCF values, Chunked Symmetrical Path Count, Chunked Weight Complexity, and algorithm search times across the Warcraft Battleground Map with varying guard numbers. To assess the effectiveness of these metrics in estimating the complexity introduced by node weights on the map, Their correlation with algorithm search times is calculated. The formula presented in Eq. (5) details the method for computing the Pearson Correlation.

$$\rho = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{5}$$

Table 6 displays the correlation analysis between Chunked Weight Complexity and WCF metrics concerning the search time of the A* and JPSW algorithms. Remarkably, both metrics demonstrate a robust correlation with the search speed of both algorithms, with WCF surpassing Chunked Weight Complexity. Additionally, a mutual correlation of 0.78 is observed between these two metrics. Another significant advantage of WCF is its efficiency in computation, allowing for near-instantaneous calculation on a 512 × 512 map, in contrast to the lengthy day-long computation required for Chunked Weight Complexity.

As mentioned before, calculating weight complexity is computationally expensive. However to illustrate the correlation between Weight Complexity and algorithm search time, experiments were conducted on two small maps, namely “den900d” and “den403”, from the Guards dataset. The results, presented in Table 7, highlight a robust correlation between both metrics and algorithms search time across these compact maps.

6. Evaluation results

The average search time of the algorithms for all maps of the guards dataset is shown in Fig. 9. The maps were divided into seven groups based on the number of guards (0-64-128-256-512-1024-2048). The average WCF of each group is shown on the horizontal axis and the average search time for each group is indicated in the chart (see Fig. 7).

As the WCF value increases, the search time of the algorithms also increases, but this increase is unique. For example Fig. 8 show average search time of JPSW algorithm for 252 Warcraft maps in the Guards dataset based on WCF value of the maps.

The data points in Fig. 9 demonstrate that the average search time of each algorithm, as indicated by the WCF value, follows an upward trend and based on our experience can be accurately represented by an exponential mathematical function of the form $A\exp^{Bx} + C$.

This model has the potential to greatly aid in the comparison and evaluation of algorithms. The non-linear least squares method is employed to determine the optimal coefficients for the exponential model for each algorithm. non-linear least squares (NLS) is a statistical method used for fitting a non-linear function to a set of data points. The goal of NLS is to find the best-fitting curve that describes the relationship between the independent and dependent variables in a non-linear manner. In NLS, the parameters of the non-linear function are estimated by minimizing the sum of the squared differences between the observed and predicted values. This approach allows for the estimation of complex relationships that cannot be captured by linear regression. The optimal coefficients for our models are determined using the NLS method in the R programming language. The models were fitted to the points with a high degree of accuracy, as demonstrated by the average residual standard error of less than 0.03. This means that, on average, the predicted values generated by the models were very close to the actual observed values, with a minimal amount of error. The visualization in Fig. 10 confirms that the model estimations based on data points are extremely precise. In this figure, the markers represent the average search time for each algorithm, calculated from benchmark runs over the guards dataset. The maps are categorized based on the count of guards, and the averages are computed accordingly. The lines in the figure correspond to the fitted models for each algorithm.

Fitted model for algorithms are shown in Table 8. By utilizing this model and the WCF value of a map, a well-informed decision can be made regarding the most appropriate search algorithm to use for a given map. Furthermore, if the WCF of a map changes dynamically over time, the use of this model allows for real-time switching to the best search algorithm, ensuring optimal performance and efficient processing.

One of the characteristics of this search time predication model is that its gradient indicates the algorithm’s sensitivity to map complexity. Thus, evaluating the derivative of the model at each point provides insight into the algorithm’s sensitivity to complexity.

7. Algorithms overall score

The search time predication model offers a quantitative way to evaluate the performance of the algorithms being studied. The amount of area under the model line in the diagram corresponds to the search time and serves as a performance indicator. To assess the performance

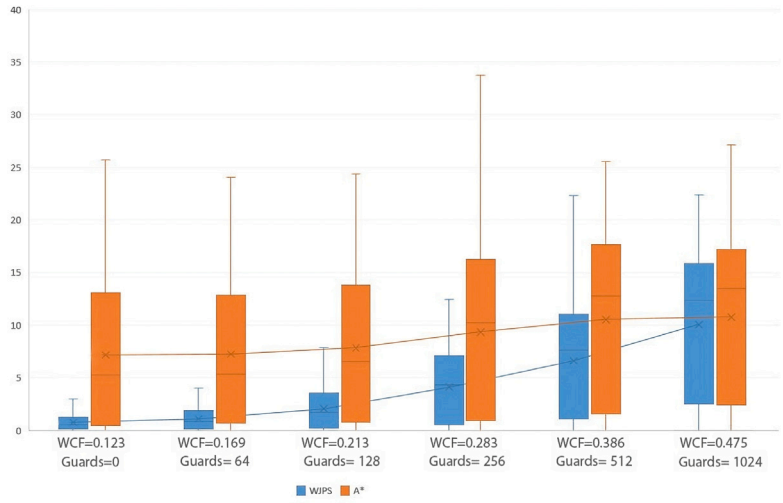

Fig. 6. WCF and search time varies based on the map complexity.

Table 5
Calculation of WCF, Chunked weight complexity, and algorithm search times for the Warcraft Battleground Map with varying guard quantity.

Guards	WCF	WCF Ratio	Chunked symmetrical path count	Chunked weight complexity	A* Average time	JPSW Average time
0	0.14	1.00	2,590,937,828	1.00	1.96	0.47
64	0.18	1.27	2,225,571,974	1.16	2.35	0.70
128	0.21	1.47	1,921,203,002	1.35	2.68	0.91
256	0.27	1.92	1,499,800,270	1.73	3.86	1.60
512	0.37	2.67	886,047,510	2.92	6.17	3.63
1024	0.50	3.55	439,012,586	5.90	8.42	6.11
2048	0.66	4.69	24,893,266	104.08	13.77	13.38

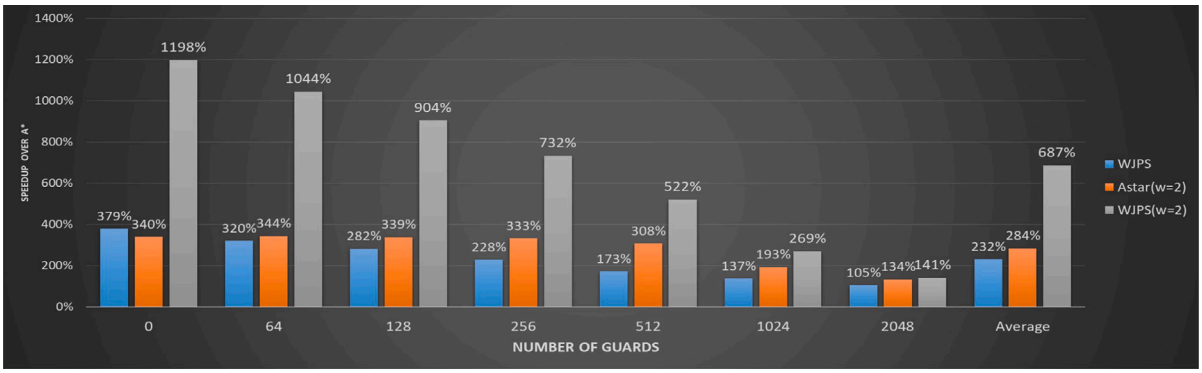

Fig. 7. Algorithms overall speedup over A* which are categorized based on the number of guards on the map.

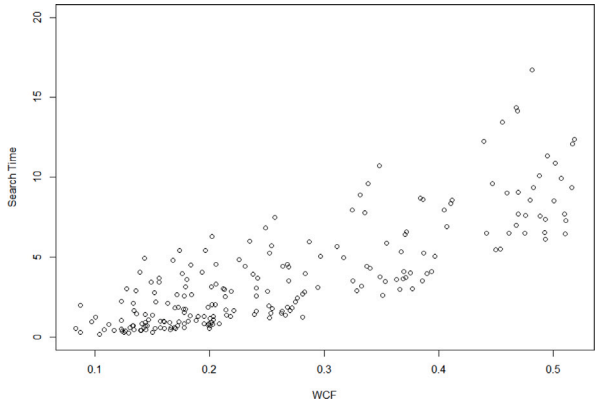

Fig. 8. The average search time of the JPSW algorithm varies across different maps and is influenced by the WCF.

Table 6
Analyzing correlations between WCF, chunked weight complexity, and algorithms search time for the Warcraft Battleground Map.

Metric	Chunked weight complexity	WCF Ratio
A* Search time	0.862	0.990
JPSW Search time	0.919	0.963

Table 7
Examining relationships among WCF, weight complexity, and algorithms search time for two small maps.

Metric	Weight complexity	WCF ratio
A* Search time	0.8945	0.8764
JPSW Search time	0.8870	0.8941

of each algorithm more precisely, we can calculate the integral of each model from the minimum to the maximum WCF value obtained from the average WCF of 0 guards to 2048 guards. Therefore, it is necessary

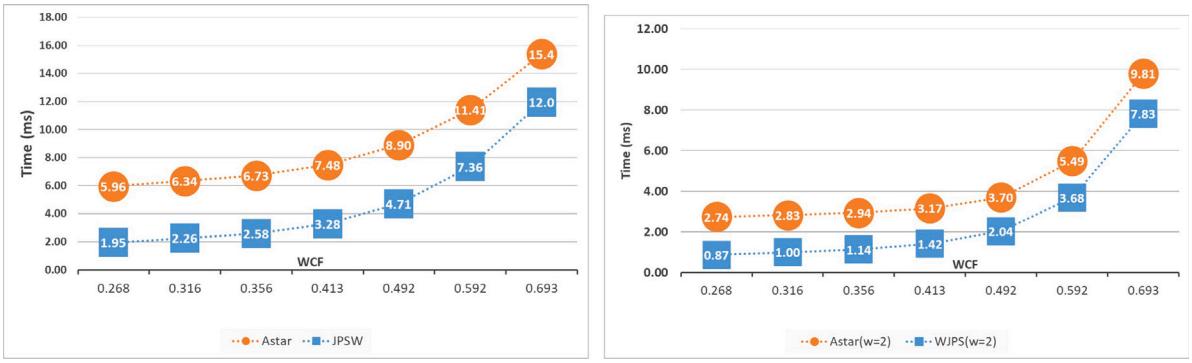

Fig. 9. The JPSW average search time of all the scenarios in the Guards dataset based on WCF value of the maps.

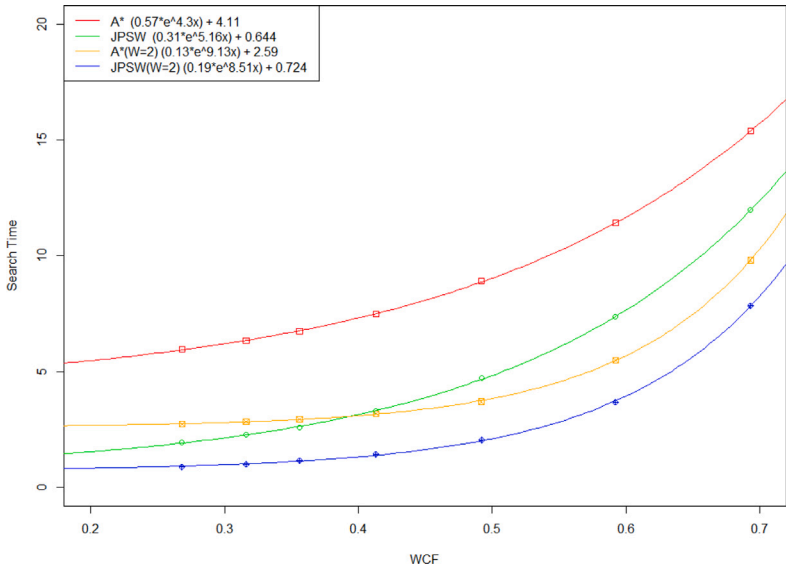

Fig. 10. The fitted model for each algorithm.

Table 8
Fitted models for various algorithms.

Alg	Model	Integral	Derivative
A*	$0.57\exp^{4.3X}+4.1$	$0.132\exp^{4.3X}+4.1X$	$2.45\exp^{4.3X}$
JPSW	$0.3\exp^{5.16X}+0.64$	$0.058\exp^{5.16X}+0.64X$	$1.54\exp^{5.16X}$
A*(W = 2)	$0.012\exp^{9.13X}+2.6$	$0.0013\exp^{9.13X}+2.6X$	$0.11\exp^{9.13X}$
JPSW(W = 2)	$0.019\exp^{8.5X}+0.72$	$0.0022\exp^{8.5X}+0.72X$	$0.16\exp^{8.5X}$

to compute the integral of the model between the WCF values of 0.268 and 0.693. This yields a raw score for each algorithm, providing a comprehensive evaluation of their performance. This raw score represents the algorithm's quality and efficiency, allowing for informed decision-making in the general selection of the optimal algorithm. The calculated score provides a means of comparing the performance of different algorithms and facilitates the selection of the best suited algorithm for a given scenario. It is referred to as the raw score because it only takes into account the search speed and does not factor in the optimality of the found path. The raw score provides a useful measure of an algorithm's efficiency and performance, but other factors such as the optimality of the solution must also be considered when making an informed decision. Eq. (6) shows how to calculate row score of each algorithm:

$$RawScore_i = \int_{0.268}^{0.693} f(x)_i \, dx \tag{6}$$

The significance of finding the optimum path varies across different applications. However, for computer games, which are a critical application of these algorithms, a path that is close to optimum is often sufficient. Based on our experience, it is proposed to incorporate the level of optimality into the overall score of the algorithms through Eq. (7).

$$OS_i = \int_{0.268}^{0.693} f(x)_i \, dx * \left(\frac{\sum_1^n Len_i}{\sum_1^n Len_{optimal}}\right)^e \tag{7}$$

This equation decreases the score of the algorithms exponentially based on their sub-optimality, meaning that a small deviation from the optimum results in a relatively minor penalty. To create this penalty model, linear, power of 2 and 3, and exponential penalty were considered. After consulting with numerous experts in game development, the conclusion was reached that designing a penalty model that results in approximately 30% penalty for 10% suboptimality, 65% penalty for 20% suboptimality, and 100% penalty for 30% suboptimality would be ideal. The exponential function provided results that were closest to our expert opinion, assigning penalty values of 29.5%, 64.1%, and 104% for 10%, 20%, and 30% suboptimality, respectively. Incorporating

Table 9
Raw Score (RS), suboptimality (SO) and overall score (OS) of algorithms (lower score is better).

Alg	RS	SO	OS
A*	3.92	0%	3.92
JPSW	2.08	0%	2.08
A*(W = 2)	1.81	5.08%	2.07
JPSW(w = 2)	1.07	4.57%	1.2

a penalty for memory usage was considered in the algorithms, but after careful evaluation, we concluded that the memory requirements of these algorithms are negligible on modern hardware. Hence, the decision was made not to implement such a penalty.

This way, the overall score of the ith algorithm takes into consideration not only the algorithm's efficiency (reflected by the raw score), but also the degree of optimality of the paths generated by the algorithm (represented by the sub-optimality penalty factor). The overall score provides a comprehensive evaluation of the performance of the ith algorithm, enabling informed decision-making in its selection for specific applications. In the field of pathfinding algorithms, numerous factors can be compared to determine their effectiveness, with varying levels of importance depending on the specific application. Typically, these comparisons involve showcasing the pareto frontier, which can be challenging to interpret and comprehend quickly. Consequently, there is a need for an overall score that facilitates easy and efficient comparison of different algorithms. Table 9 presents the overall scores of various algorithms, where a lower score is deemed to be more favorable. As per the results, JPSW with a value of w = 2 emerges as the best choice for game developers in dynamic environments that incorporate weights. In instances where the game can tolerate higher sub-optimality, a higher value of W can be selected to attain faster pathfinding performance.

8. Conclusions

This paper presents a new benchmark to enhance the evaluation of pathfinding algorithms on weighted grid-based environments. Novel parameters were aimed to be introduced in order to facilitate comparisons between maps and algorithms. Weight complexity is introduced as a term in this research, signifying the observed phenomenon where adding weights to grid cells slows down grid pathfinding algorithms. The paper utilizes this term to characterize the impact on algorithmic performance. By following a standard procedure, researchers can obtain an overall score for their algorithm that can be compared to other algorithms. Our findings indicate that the JPSW algorithm currently performs best in weighted pathfinding in dynamic environments. Overall, this research provides valuable insights into the pathfinding algorithms and offers a new framework for evaluating their performance in weighted grid-based environments.

The focus of this research during the testing phase is on pathfinding algorithms that can handle dynamic weight changes, but the designed dataset does not incorporate dynamic changes to simplify the evaluation process for all researchers. Future work could include developing a separate standard procedure to measure the sensitivity and adaptability of algorithms to dynamic changes. Additionally, The current WCF calculation is tailored for a range of map types, especially those featuring exponential weight. Nevertheless, investigating alternative methods for WCF computation holds the potential to improve its correlation with algorithmic search times across various maps and datasets. This avenue could be explored in future work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2024.123719.

References

Bono, M., Gerevini, A. E., Harabor, D. D., & Stuckey, P. J. (2019). Path planning with cpd heuristics. In *IJCAI* (pp. 1199–1205).

Botea, A. (2012). Fast, optimal pathfinding with compressed path databases. In *Proceedings of the international symposium on combinatorial search, Vol. 3* (pp. 204–205).

Brondani, J. R., Silva, L. A. d. L., Zacarias, E., & de Freitas, E. P. (2019). Pathfinding in hierarchical representation of large realistic virtual terrains for simulation systems. *Expert Systems with Applications*, *138*, Article 112812. http://dx.doi.org/10.1016/j.eswa.2019.07.029.

Carlson, M., Moghadam, S. K., Harabor, D. D., Stuckey, P. J., & Ebrahimi, M. (2023). Optimal pathfinding on weighted grid maps. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 37* (pp. 12373–12380).

Cohen, L., Greco, M., Ma, H., Hernández, C., Felner, A., Kumar, T. S., et al. (2018). Anytime focal search with applications. In *IJCAI* (pp. 1434–1441).

Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., & Kumar, T. (2017). The fastmap algorithm for shortest path computations. arXiv preprint arXiv:1706.02792.

Dibbelt, J., Strasser, B., & Wagner, D. (2016). Customizable contraction hierarchies. *Journal of Experimental Algorithmics (JEA)*, *21*, 1–49.

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental algorithms: 7th international workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1 2008 proceedings 7* (pp. 319–333). Springer.

Harabor, D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 25* (pp. 1114–1119).

Harabor, D., Hechenberger, R., & Jahn, T. (2022). Benchmarks for pathfinding search: Iron harvest. In *Proceedings of the international symposium on combinatorial search, Vol. 15* (pp. 218–222).

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*, 100–107. http://dx.doi.org/10.1109/tssc.1968.300136.

Jong, D., Kwon, I., Goo, D., & Lee, D. (2015). Safe pathfinding using abstract hierarchical graph and influence map. In *IEEE 27th international conference on tools with artificial intelligence* (pp. 72–79).

Kanopoulos, N., Vasanthavada, N., & Baker, R. L. (1988). Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, *23*, 358–367.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*, 566–580.

Likhachev, M., Gordon, G. J., & Thrun, S. (2003). Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, *16*.

Mahéo, A., Zhao, S., Hassan, A., Harabor, D. D., Stuckey, P. J., & Wallace, M. (2021). Customised shortest paths using a distributed reverse oracle. In *Proceedings of the international symposium on combinatorial search, Vol. 12* (pp. 79–87).

Pohl, I. (1970). First results on the effect of error in heuristic search. *Machine Intelligence*, *5*, 219–236.

Rohrmuller, F., Althoff, M., Wollherr, D., & Buss, M. (2008). Probabilistic mapping of dynamic obstacles using markov chains for replanning in dynamic environments. In *2008 IEEE/RSJ international conference on intelligent robots and systems* (pp. 2504–2510). IEEE.

Shi, Y., & Crawfis, R. (2013). Optimal cover placement against static enemy positions. *Foundations of Digital Games*.

Shi, Y., & Crawfis, R. (2014). Group tactics utilizing suppression and shelter. In *2014 computer games: AI, animation, mobile, multimedia, educational and serious games* (pp. 1–8). IEEE.

Standley, T. (2010). Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 24* (pp. 173–178).

Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the international symposium on combinatorial search, Vol. 10* (pp. 151–158).

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

Sturtevant, N. R., Felner, A., Barrer, M., Schaeffer, J., & Burch, N. (2009). Memory-based heuristics for explicit state spaces. In *Twenty-first international joint conference on artificial intelligence* (pp. 194–202).

Sturtevant, N. R., Sigurdson, D., Taylor, B., & Gibson, T. (2019). Pathfinding and abstraction with dynamic terrain costs. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, Vol. 15* (pp. 80–86).

Sturtevant, N. R., Traish, J., Tulip, J., Uras, T., Koenig, S., Strasser, B., et al. (2015). The grid-based path planning competition: 2014 entries and results. In *International symposium on combinatorial search*.

Uras, T., Koenig, S., & Hernandez, C. (2013). Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Proceedings of the twenty-third international conference on automated planning and scheduling* (pp. 211–218).

Yap, P., Burch, N., Holte, R., & Schaeffer, J. (2011). Block a*: Database-driven search with applications in any-angle path-planning. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 25* (pp. 120–125).

Zhao, J., Xu, Q., Zlatanova, S., Liu, L., Ye, C., & Feng, T. (2022). Weighted octree-based 3d indoor pathfinding for multiple locomotion types. *International Journal of Applied Earth Observation and Geoinformation*, *112*, Article 102900. http://dx.doi.org/10.1016/j.jag.2022.102900.