Noname manuscript No.
(will be inserted by the editor)

Efficiently Computing Alternative Paths in Game Maps

Lingxiao Li · Muhammad Aamir Cheema · Mohammed Eunus Ali · Hua Lu · Huan Li ·

Received: date / Accepted: date

Abstract Alternative pathfinding requires finding a set of k alternative paths (including the shortest path) between a given source s and a target t. Intuitively, these paths should be significantly different from each other and meaningful/natural (e.g., must not contain loops or unnecessary detours). While finding alternative paths in road networks has been extensively studied, to the best of our knowledge, we are the first to formally study alternative pathfinding in game maps which are typically represented as Euclidean planes containing polygonal obstacles. First, we adapt the existing techniques designed for road networks to find alternative paths in the game maps. Then, based on our web-based system that visualises alternative paths generated by different approaches, we conduct a user study that shows that the existing road network approaches generate high-quality alternative paths when adapted for the game maps. However, these existing approaches are computationally inefficient especially when compared to the state-of-the-art shortest path algorithms. Motivated by this, we propose novel data structures and exploit these to develop an efficient algorithm to compute high-quality alternative paths. Our extensive

L. Li
Faculty of Information Technology, Monash University, Australia
E-mail: lingxiao.li@monash.edu

M. Aamir Cheema
Faculty of Information Technology, Monash University, Australia
E-mail: aamir.cheema@monash.edu

M. Eunus Ali
Bangladesh University of Engineering and Technology, Bangladesh
E-mail: eunus@cse.buet.ac.bd

H. Lu
Department of People and Technology, Roskilde University, Denmark
E-mail: luhua@ruc.dk

H. Li
Department of Computer Science, Aalborg University, Denmark
E-mail: lihuan@cs.aau.dk

experimental study demonstrates that our proposed algorithm is more than an order of magnitude faster than the existing approaches and returns alternative paths of comparable quality. Furthermore, our algorithm is comparable to a state-of-the-art shortest path algorithm in terms of running time.

Keywords Diverse shortest paths · Alternative pathfinding · Game maps

1 Introduction

A shortest path query is one of the fundamental problems with a wide variety of applications in many domains. It requires finding the minimal cost path between two given points, a source *s* and a target *t*. This type of problem is commonly encountered in various contexts, including but not limited to road networks [2], social networks [39], geographical systems [62], indoor spaces [9], and game maps [15], to name a few. In many cases, it is helpful to provide several alternative paths to the users, in addition to the shortest one, as this allows them to select the route that best suits their needs or preferences. For example, popular navigation systems like Google Maps offer several options for traveling from the source to the target, allowing the users to choose the path that they prefer. To be useful for the end users, these alternative paths should be reasonably short and distinct from one another to provide meaningful choices to the user.

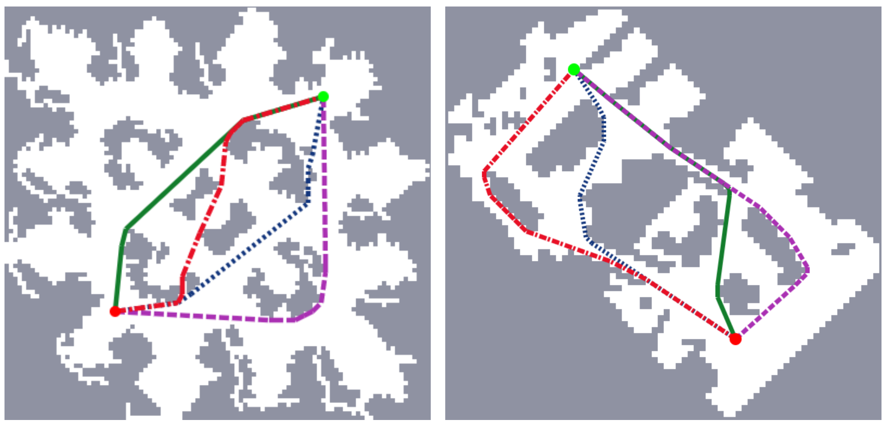

(a) Warcraft III (b) Baldurs Gate II

Fig. 1 Four alternative paths on two different game maps.

There has been a large body of work on finding alternative paths in road networks (e.g., see [37,26,44]). Finding alternative paths in game maps has many useful applications but has not been studied in this context. In real-time strategy (RTS) games, characters typically take the shortest path to reach their destination, but this can make their movements predictable to opponents. To avoid this, it may be beneficial to compute alternative paths and randomly assign one of them to the character. Many RTS games allow players to choose their own waypoints, which they can use to make their characters follow a path different from the shortest path. In such games, a player may be shown several alternative paths so that they can choose a path for their character to take. Figure 1 illustrates two examples where four alternative paths are provided on two different game maps. Alternative paths can also be useful in indoor venues, which are often represented as a Euclidean plane with obstacles. Indoor navigation systems may provide multiple path options for users to choose from.

Some open-source game development projects[1] have attempted to include support for alternative paths in game maps. Existing techniques designed for road networks can also be adapted for the game maps. However, it is not clear how efficient or effective these algorithms are when applied on game maps. In this paper, we aim to fill this gap by formally studying the problem of finding alternative paths in game maps. We adapt the existing road network algorithms for game maps and our experimental study demonstrates that these algorithms are not efficient when applied on game maps. This is mainly because they fail to exploit the properties unique to game maps such as polygonal obstacles and that the source and target can be anywhere in the non-obstacle area in contrast to the road networks where these must be on the road edges/nodes.

Intuitively, the alternative paths must be sufficiently dissimilar to each other and must be meaningful/natural, e.g., should not be non-taut or contain un-necessarily long/short detours etc. Note that computing k shortest paths is not a good solution for alternative pathfinding as these paths are expected to be very similar to each other. In this paper, we make the following contributions.

- To the best of our knowledge, we are the first to study the problem of finding alternative paths in game maps. We adapted three techniques that are commonly used for finding alternative paths in road networks, called Penalty [4,11,32], Plateaus [29] and Dissimilarity [14,40], for use in game maps. However, it was unclear if these techniques would be able to generate high-quality alternative paths in game maps. Therefore, we created a web-based demonstration system and conducted a user study on nine diverse game maps selected from a widely used benchmark. We received a total of 472 responses and found that the adapted techniques generated high-quality alternative paths according to the users. We have made the source code[2] for the web-based demonstration system publicly available for reuse or further development.

[1] For example, see a game development project with Unity3D at `https://arongranberg.com/astar/docs/alternativepath.html`

[2] `https://bitbucket.org/lingxiao29/customized/src/master/`

- We observe that the existing approaches when extended for game maps are computationally expensive especially when compared with the recent shortest pathfinding algorithms for game maps. To address this, we propose an efficient algorithm to compute alternative paths in game maps exploiting a novel compressed via-path database (CVPD) which has certain advantages compared to the traditional compressed path databases (CPDs) [56].
- We conduct an extensive experimental study on a widely used game maps benchmark which shows that our proposed algorithm computes the alternative paths in a time comparable to a state-of-the-art shortest path algorithm, Polyanya [15], (slower for shorter paths but faster for longer paths) and outperforms the existing alternative path approaches by more than an order of magnitude. We also evaluate the quality of alternative paths returned by our algorithm using some well-known quantitative measures such as *path similarity*, *bounded stretch* and *local optimality* [31] . The results show that the paths returned by our algorithm are comparable to those returned by the existing approaches in terms of quality.

This paper is an extended version of our previous work [35]. The main contribution of our previous work was the user study and experiments to evaluate the existing alternative pathfinding approaches. In this extended version, our key contribution is a novel data structured (CVPD) and a new algorithm which computes the alternative paths much more efficiently.

The rest of the paper is organised as follows. In Section 2, we present problem definition, evaluation measures, and related work. In Section 3, we first discuss how existing algorithms for road networks can be extended for alternative pathfinding in game maps and then we present the details of our user study and results. Section 4 presents the details of our efficient algorithm for computing alternative paths. Experimental study is provided in Section 5 followed by conclusions in Section 6.

2 Preliminaries

2.1 Problem Formulation

In this paper, we assume that the input map is a 2D Euclidean plane which contains a set of obstacles. Each obstacle is a polygon and its convex corners are called convex vertices. The set of all convex vertices in the map is denoted as V. We say that two points are co-visible (or are visible to each other) if a straight line connecting them does not pass through any obstacle in the map. A **path** P between a source s and a target t is an ordered set of points $\langle p_1, p_2, \cdots, p_n\rangle$ such that, for each p_i $(i < n)$, p_i and p_{i+1} are co-visible where $p_1 = s$ and $p_n = t$. **Length** of a path P is the cumulative Euclidean distance between every successive pair of points, denoted as $|P|$, i.e., $|P| = \sum_{i=1}^{k-1} EDist(p_i, p_{i+1})$ where $EDist(x, y)$ is the Euclidean distance between x and y. The shortest path $sp(s,t)$ is a path between s and t with the minimum length. The shortest distance between s and t is denoted as $d(s,t)$, i.e., $d(s,t) = |sp(s,t)|$.

Problem Definition. Given a source s, a target t, and a positive integer k, we aim at finding k **alternative paths** (including the shortest path $sp(s,t)$) between s and t such that each alternative path is no longer than $d(s,t) \times \epsilon$ where $\epsilon \geq 1$ is a user-defined parameter.

Intuitively, the k alternative paths must be "significantly different" from each other (e.g., should have small overlap with each other) and each path must be a "reasonable" path, e.g., should not contain unnecessary detours and loops etc. The existing works on finding alternative paths in road networks (e.g., see [31]) have defined several measures to quantify whether a set of alternative paths is "reasonable" or not. "Significantly different" can be quantified by defining a dissimilarity function based on the overlap between paths. We formally define these measures in Section 2.2 which are also used in the experimental study. Like most existing works on alternative paths in road networks, we focus on retrieving k alternative paths with the smallest lengths while guaranteeing that the intra-path dissimilarity between these paths is no less than a user-defined threshold θ. Nevertheless, we remark that our algorithm can be easily generalised to handle other objective functions (e.g., retrieve k paths that minimise a weighted sum defined over several quantiative measures). Please see details in Section 4.2.2.

2.2 Evaluation Measures

Let $P = \langle p_1, p_2, \cdots, p_n \rangle$ be an alternative path between s and t such that $p_1 = s$, $p_n = t$, each p_i $(1 < i < n)$ is a vertex of an obstacle and for each p_i $(i < n)$, p_i and p_{i+1} are visible from each other. We use $P_{x,y}$ where $x < y$ to denote the subpath $\langle p_x, \cdots, p_y \rangle$ of P and denote its length as $d^P(p_x, p_y)$, i.e., $d^P(p_x, p_y) = \sum_{i=x}^{y-1} EDist(p_i, p_{i+1})$. Hereafter, whenever we use x and y, assume $x < y$.

Bounded Stretch [31]. Stretch of a path defines how long is the path compared to the shortest path. Formally, stretch of a subpath $P_{x,y}$ is defined as $S(P_{x,y}) = d^P(p_x, p_y)/d(p_x, p_y)$. For an alternative path P, its bounded stretch is the maximum stretch of any of its subpaths.

$$BS(P) = \max_{\forall (x,y)} \frac{d^P(p_x, p_y)}{d(p_x, p_y)} \tag{1}$$

For example, assume a path P which has a bounded stretch 1.20, i.e., the maximum stretch of any of its subpath is 1.20. This implies that there is no subpath of P which is more than 20% longer than the shortest distance between its end points.

Note that an alternative path P with smaller bounded stretch is better. Also, if P is a shortest path, its bounded stretch is 1. Let $\mathcal{P}$ be a set of alternative paths returned by an algorithm. The bounded stretch of $\mathcal{P}$ is the maximum bounded stretch of any of the paths in $\mathcal{P}$, i.e., $BS(\mathcal{P}) = max_{\forall P \in \mathcal{P}} BS(P)$.

Local Optimality [31]. We say that a subpath $P_{x,y}$ is suboptimal if it is longer than the shortest distance between p_x and p_y, i.e., $d^P(p_x, p_y) > d(p_x, p_y)$. Given an alternative path P between s and t, we use $minL(P)$ to denote the length of the shortest suboptimal subpath of P (if all subpaths are optimal, $minL(P)$ is assumed to be infinity). Note that any subpath of P which is shorter than $minL(P)$ must be optimal. Thus, $minL(P)$ is a measure of optimality. The local optimality $LO(P)$ normalises this measure w.r.t. the shortest distance $d(s, t)$ between s and t.

$$LO(P) = \frac{minL(P)}{d(s,t)} = \min_{\forall (x,y): d^P(p_x,p_y) > d(p_x,p_y)} \frac{d^P(p_x, p_y)}{d(s,t)} \tag{2}$$

Consider an alternative path P between s and t and assume that its shortest suboptimal path has length 20 and $d(s, t) = 100$. Its local suboptimality is $20/100 = 0.2$. This implies that every subpath of P which is shorter than 20% of the shortest path between s and t is guaranteed to be an optimal path. A path P with higher local optimality is better. Also, if P is a shortest path, its local optimality is 1. Let $\mathcal{P}$ be a set of alternative paths returned by an algorithm. The local optimality of $\mathcal{P}$ is $LO(\mathcal{P}) = min_{\forall P \in \mathcal{P}} LO(P)$.
Similarity [37]. Similarity $Sim(\mathcal{P})$ of a set of paths $\mathcal{P}$ is

$$Sim(\mathcal{P}) = \max_{\forall (P_i,P_j) \in \mathcal{P} \times \mathcal{P}: i \neq j} \frac{|P_i \cap P_j|}{|P_i \cup P_j|} \tag{3}$$

where $|P_i \cap P_j|$ (resp. $|P_i \cup P_j|$) denotes the total length of the overlap (resp. union) of two paths P_i and P_j. Dissimilarity is $1 - Sim(\mathcal{P})$.

2.3 Related Work

Graphs are commonly used to model many real-world problems in a wide variety of application domains such as social networks [28,45], recommendation systems [52,49], health informatics [19,22], transportation networks [59, 6], the Internet of Things [27,38], and information security [25,61]. Shortest path queries [17,47,2] are one of the most fundamental and frequently used operations conducted on graphs. In this work, our focus is on path queries on graphs representing physical spaces such as game maps or road networks. In Section 2.3.1, we present existing work related to computing shortest paths in such graphs whereas Section 2.3.2 covers the related works on computing alternative paths in such graphs.

2.3.1 Computing Shortest Paths

Shortest path queries [17,47,2,24,5] and related queries [58,23,10,33,1] have been very well-studied in road networks. Below, we discuss two of the most popular shortest path algorithms for road networks namely contraction hierarchies and hub labeling.

Contraction Hierarchies (CH) [24] is a graph indexing technique that can answer shortest path query orders of magnitude faster than Dijkstra's algorithm. As a successor of Highway Hierarchies [50] and Highway Node Routing [51], CH implements the idea of *shortcuts* to exploit the road network hierarchy. These shortcuts allow efficient shortest path retrieval during the search process which employs a bidirectional search from source and target on the constructed hierarchy. Hub labeling [2] is another indexing technique which assigns, during preprocessing phase, labels to each node in the graph a set of labels. These labels are assigned such that for any source s and t, the labels of s and t are guaranteed to contain a hub node on the shortest path between s and t. During the query processing, the labels of s and t are joined to find the common hub nodes and the hub node that gives the smallest distance from s to the hub node and to t is used to recover the shortest path.

In game maps, shortest pathfinding has also been extensively studied [54, 15, 20]. Typically, these approaches construct a *visibility* graph by connecting the obstacle vertices that are co-visible. At query time, source s and target t are connected to the visibility graph and it can be guaranteed that the shortest path in this graph is the shortest path between the source and target. Polyanya [15] is the state-of-the-art online shortest path algorithm. It employs a navigation mesh [30] which divides the traversable space into a disjoint set of convex polygons and uses an algorithm similar to A* algorithm with subtle differences (see details in [15]). End Point Search [53] (EPS) employs a Compressed Path Database (CPD) [7] and Polyanya. Specifically, given a $V \times V$ table where each row $R(u)$ of a convex vertex $u \in V$ stores, for every $v \in V$, the first vertex f on the shortest path from u to v. Each row $R(u)$ is then compressed using the run-length encoding (RLE) [55]. Given this CPD, for any pair of vertices u and v, the first move (i.e., the first vertex) on the shortest path from u to v can be accessed from the CPD using a binary search on the compressed row $R(u)$. The shortest path from u to v can be obtained by recursively extracting the first moves towards v until v is reached. Regarding EPS, which employs Polyanya to incrementally find vertices visible from s and t, respectively. Then, the CPD is used to obtain the pair-wise paths between these visible vertices efficiently. Several pruning techniques and optimisations are proposed to speed up the computation.

2.3.2 Computing Alternative Paths

Existing studies have proposed a variety of approaches to answer alternative pathfinding queries in road networks [36, 37, 26, 44, 42]. Some existing works focus on finding k-shortest paths [60, 21]. However, these do not reflect good alternative paths as most of the k shortest paths have a very high level of overlap with each other. To address this, several existing works use user-defined parameters to obtain k paths that are significantly different from each other and are not too long compared to the shortest path [40, 12, 13]. These do not guarantee the quality of alternative paths (e.g., paths may have local detours etc.), Furthermore, the problem as defined in these papers is NP-Hard making them computationally challenging without providing any path quality guarantee. The *penalty* based approaches [11,4] compute the alternative paths by increasing the edge weights on the paths already found (to avoid selecting the edges while finding additional paths). Plateau-based approach [41,3,43,18,48,16] is arguably the most popular approach to return the promising alternative paths as it naturally provides several guarantees such as local optimality and smaller overlaps with the other paths, etc. In some works (e.g.,[3]), the author proposed several constraints to formally define the alternative paths, which should be locally optimal, limited sharing, and uniformly bounded stretch.

Indeed, the shortest pathfinding in game maps has been well understood in existing studies, e.g., see [15,53] and references therein. However, alternative pathfinding queries have only been solved in road networks. Next, we briefly explain the alternative pathfinding approaches in road networks, as in [37]. The majority of alternative pathfinding algorithms fall into three broad categories, which are Penalty [4], Plateaus [29], and Dissimilarity [13]. For ease of presentation, we assume that road networks are undirected graphs.

Penalty: Algorithms [4,11,32] in this category iteratively calculate the shortest paths between source and target. Specifically, once the current iteration is finished and the shortest path $P = \langle p_1,p_2, \cdots, p_n\rangle$ is found, the algorithm increases the weight of each edge on the current path P by a certain penalty factor (e.g., multiplying the edge weight by 1.5). Since the edge weights on the shortest path have been increased, in the next iteration, the algorithm is likely to find a different shortest path [37]. The algorithm terminates once k different paths are returned or when the length of the path found in the current iteration is longer than $d(s,t) \times \epsilon$. One major issue with this approach is its slow query processing time because it needs multiple traversals over the graph to find the k alternative paths. Furthermore, there is no guarantee that the k alternative paths are significantly different because despite the increase in edge weights, the paths may still have significant overlap with each other.

Plateaus: Cotares Limited designed this algorithm [29] for their routing engine called Choice Routing. First, it creates a shortest path tree T_s rooted at the source s and another shortest path tree T_t rooted at the target t. Next, T_s and T_t are *joined* and common branches in both trees are found. Each of the common branches is called a plateau. Consider one branch $\langle s, \cdots, u_1, u_2, \cdots, u_n, \cdots, y\rangle$ in T_s and another branch $\langle t, \cdots u_n, u_{n-1}, \cdots, u_1, \cdots, x\rangle$ in T_t. When these branches are joined, the common part $\langle u_1, \cdots, u_n\rangle$ is found, i.e., $\langle u_1, \cdots, u_n\rangle$ is a plateau, which we denote as $pl(u_1, u_n)$ using the end points of the branch. We remark that the shortest path between s and t is always a plateau and its length is $d(s,t)$. Let $pl(u,v)$ be a plateau such that u is the end closer to s and v is the end closer to t. This plateau can be used to retrieve an alternative path $sp(s,u) \oplus pl(u,v) \oplus sp(v,t)$ where $\oplus$ is the concatenation operation. It was observed in [29] that longer plateaus typically generate better alternative paths. Therefore, the algorithm picks k longest plateaus and generates alternative paths based on each of the plateaus. We give a detailed example of how Plateaus works in Section 3.

The alternative paths generated using plateaus have some natural/useful characteristics which make them attractive for both research and commercial solutions. Firstly, an alternative path generated using a longer plateau avoids unnecessary/unnatural detours (e.g., leaving a motorway and entering it again shortly afterward without reducing the traveling cost). This is due to *local optimality* provided by plateaus. Specifically, as noted in [3], a path that is *local optimality* does not have undesirable local detours. Secondly, plateaus do not overlap with each other, therefore, the alternative paths generated using longer plateaus tend to have smaller overlaps with each other. Thirdly, plateaus are generated using the shortest path trees and seamlessly capture the intrinsic properties of the underlying road networks.
Dissimilarity: This group of techniques follows a function to compute dissimilarity between two paths. Specifically, Given a list of the shortest paths, this function returns the shortest alternative paths with a dissimilarity value between any two paths that are less than or equal to the given threshold θ. This problem is NP-hard [14], to which several approximate algorithms [14, 40] have been proposed. Now, we explain an approach [13] shown in [37] to generate high-quality alternative paths in road networks.

Given a vertex v in the road network, a via-path $sp(s, v, t)$ is the concatenation of path $sp(s, v)$ and $sp(v, t)$. Namely, $sp(s, v, t) = sp(s, v) \oplus sp(v, t)$. To efficiently retrieve the paths from any vertex v to either source s or target t, two shortest path trees T_s and T_t are computed and stored. With these shortest path trees, the algorithm iteratively evaluates vertices in the road networks in ascending order of their via-path lengths. If the dissimilarity value between the current via-path and any added alternative path is at least θ, then add this via-path to the result set. Once the result set has k alternative paths or the current via-path is longer than $d(s, t) \times \epsilon$, the algorithm stops.

3 User Study

A recent research study [37] was conducted on road networks in three different cities, Melbourne, Dhaka, and Copenhagen. The study found that using certain techniques, called Penalty, Dissimilarity, and Plateaus, can create alternative routes that are of similar quality to those generated by Google Maps. In this work, we adapt these techniques for generating alternative paths in video game maps (Section 3.1). However, first, we want to determine if these techniques would be effective in generating high-quality alternative paths in game maps. To answer this question, we conducted a user study and present the results later in Section 3.2.

3.1 Adapting Existing Techniques for Game Maps

In this section, we describe our adaptation of the existing techniques, Penalty, Dissimilarity and Plateaus, for the game maps. First, a visibility graph $G =$

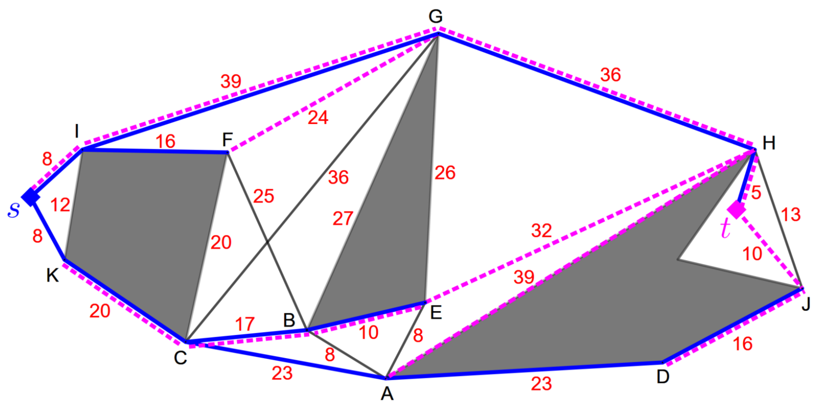
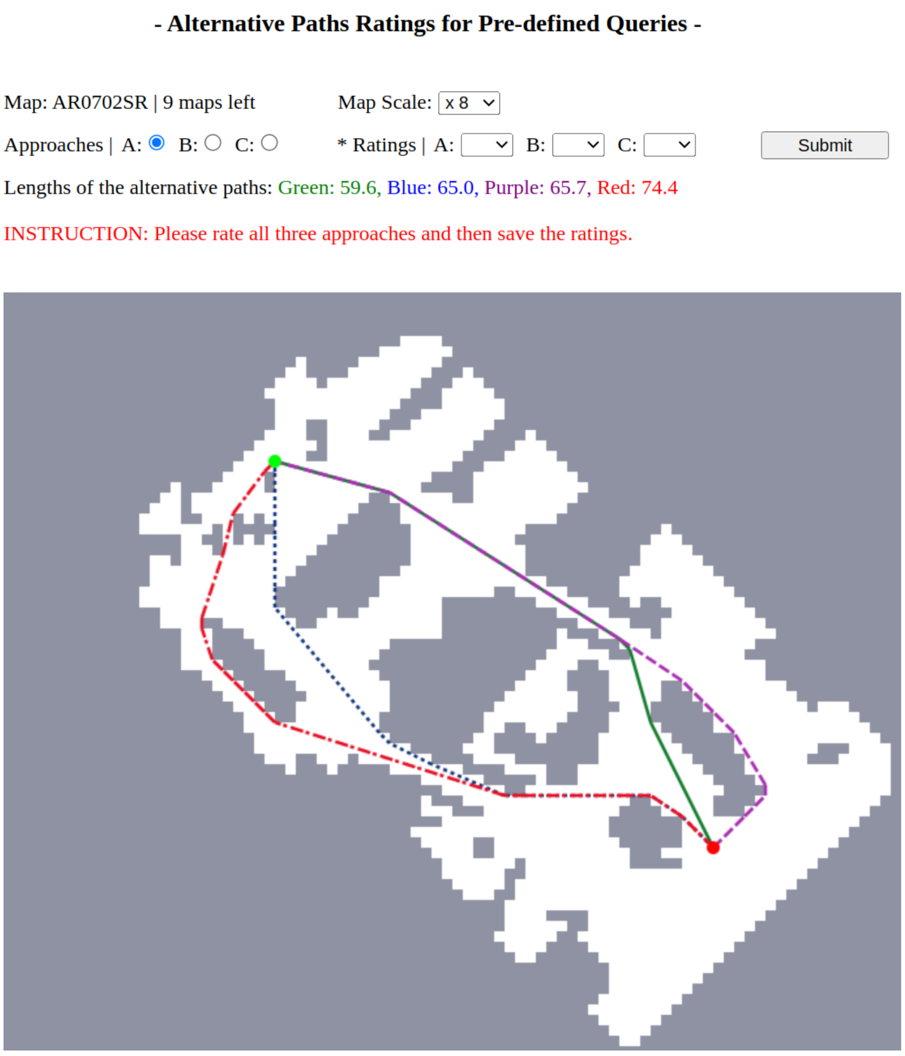


Fig. 2 Three alternative paths generated by Plateaus are $\langle s, I, G, H, t\rangle$, $\langle s, K, C, B, E, H, t\rangle$ and $\langle s, K, C, A, D, J, t\rangle$ with lengths 88, 92 and 100, respectively.

$\{V, E\}$ is created where V is the set containing convex corners/vertices in the game map and E is the set of edges connecting every pair of vertices (u, v) that are visible to each other. The weight of each edge corresponds to the Euclidean distance $EDist(u, v)$ between the pair of vertices. To compute the shortest path between a source s and a target t, the source and target are added to the graph G by inserting some new edges between s (resp. t) and the vertices which are visible from s (resp. t). Figure 2 provides an example. There are three obstacles (grey polygons). The edges shown in the figure correspond to the visibility graph G. Note that s and t have been added to G by connecting them to the vertices visible from them, respectively. After the visibility graph is constructed, the existing approaches can be directly applied on this graph. The paths that are *non-taut* are filtered. A taut path is a path which, when treated as a string, cannot be made "tighter" by pulling on its ends [46]. For example, the path $\langle s, I, F, G\rangle$ is non-taut because string-pulling results in a shorter path $\langle s, I, G\rangle$.

Example 1 Consider the example in Figure 2. We briefly describe how Plateaus generates three alternative paths. In this example, A to K are the convex vertices. Visibility graph includes the edges that connect source and target to their respective visible convex vertices (I and K for s and H and J for t). Then, Plateaus computes the forward shortest path tree T_s rooted at s (see the tree shown in blue edges) and the backward shortest path tree T_t rooted at t (see the pink edges shown in broken lines). These two trees are joined and the branches which are common in the two trees are the plateaus. The algorithm then chooses three longest plateaus: $\langle s, I, G, H, t\rangle$, $\langle K, C, B, E\rangle$ and $\langle D, J\rangle$ with lengths 88, 47 and 16, respectively. Using these three plateaus, three alternative paths are generated connecting s and t to the end of each plateau closer to them. The three alternative paths are $\langle s, I, G, H, t\rangle$, $\langle s, K, C, B, E, H, t\rangle$ and $\langle s, K, C, A, D, J, t\rangle$ with lengths 88, 92 and 100, respectively.

3.2 User Study and Results



- Alternative Paths Ratings for Pre-defined Queries -

Map: AR0702SR | 9 maps left Map Scale: x 8

Approaches | [✓ A:] B: C: * Ratings | A: B: C:

Lengths of the alternative paths: Green: 59.6, Blue: 65.0, Purple: 65.7, Red: 74.4

INSTRUCTION: Please rate all three approaches and then save the ratings.

Fig. 3 The web-based system for the user study: Pre-defined Queries.

To conduct the user study, we extend our previous work [34] to develop a web-based system which helps visualising the alternative paths generated by different approaches. We select 9 diverse maps from a well-known game maps benchmark[3]. Participants in the user study are sent a webpage[4] which

[3] `https://movingai.com/benchmarks/grids.html`
[4] `http://aamircheema.com/paths_games/`

contains the instructions as well as links to the web-based system. We sent two different types of surveys to the participants:

– **Pre-defined:** In this type of survey, for each of the nine game maps, the participant were shown three queries (source-target pairs) which were pre-selected by us. This was to ensure that different participants provide rating for the same set of queries.
– **User-selected:** In this type of survey, the participants were given the freedom to choose any source-target pair by clicking on the map. The participants were required to choose at least one source-target pair for each of the nine maps. This was to ensure that we get user-selected queries from different participants for each of the maps.

Based on the source and target of each query, our web-based system produces up to 4 alternative paths for each approach. We anonymise these approaches and display Plateaus, Dissmilarity and Penalty as A, B and C, respectively. This is to cater for any potential preconceived biases. The participants can view the paths generated by these approaches by clicking on the radio buttons (see Fig. 3). For each of the approaches, the system asks the participants to give a rating at a scale of 1 to 5 where higher is better. All the participants were given a quick overview of the alternative paths in road networks and game maps. We asked them to rate these paths based on their impression of how good the paths generated by these approaches were especially taking into account that the paths should be substantially different from each other but meaningful at the same time (e.g., should not have strange detours or loops). We remark that, due to the popularity of navigation services, most of the people have experience with alternative paths in road networks, however, most people have not necessarily seen alternative paths in video games. Hence, we carefully chose the participants who had background either in game maps pathfinding or alternative path computation in road networks.

The results of the user study are shown in Table 1. In total, we got 472 responses from 9 different participants. As it can be seen, the ratings given by the participants to different approaches are quite similar on average. Also, the ratings are usually typically high (e.g., around 4 on a scale of 1 to 5 where higher is better). To test the statistical significance of the results, we conducted one-way repeated measures ANOVA test. Given a null hypothesis of no statistically significant difference in mean ratings of the three approaches, the results suggest that, at $p < 0.05$ level, there is no evidence that the null hypothesis is false, i.e., there is no credible evidence that the three approaches received different ratings on average.

Later, in the experiments section, we also consider the measures discussed in Section 2.2 and compare these approaches against those measures. Our user study and the experimental study demonstrate that these approaches are able to generate good-quality alternative paths. However, one major limitation of these techniques (as shown later in the experimental study) is that their computation time is quite high especially when compared to the shortest path

		Average Rating		
	#Responses	Plateaus	Dissimilarity	Penalty
All	472	**4.028**	3.998	3.852
Pre-defined	243	4.016	**4.025**	3.938
User-selected	229	**4.039**	3.969	3.760

Table 1 Average user rating for Plateaus, Dissimilarity (shown as Dissim.) and Penalty. Best values for each category are shown in bold.

algorithms in game maps. To fill this gap, in the next section, we present an efficient algorithm to compute alternative paths in game maps.

4 Efficient Alternative Pathfinding Algorithm

4.1 Offline Preprocessing

Before we present our offline preprocessing, we first describe Compressed Path Database (CPD) [7]. Given a $V \times V$ table where each row $R(u)$ of a convex vertex $u \in V$ stores, for every $v \in V$, the first vertex f on the shortest path from u to v. Each row $R(u)$ is then compressed using run-length encoding (RLE) [55]. Given this CPD, for any pair of vertices u and v, the first move (i.e., the first vertex) on the shortest path from u to v can be accessed from the CPD using a binary search on the compressed row $R(u)$. The shortest path from u to v can be obtained by recursively extracting first moves towards v until v is reached.

Example 2 For our running example in Figure 4, Table 2 shows two uncompressed rows of the CPD containing first moves *from* I and K, respectively, *to* the other vertices. Consider the row of K. The first move from K to each of A, B, C, D, E, H and J is C. Therefore, the corresponding cells contain C. A wildcard symbol "*" is stored for the cell K because the path from K to K is not needed. The wildcard symbol can be compressed with any other symbol. Run-length encoding (RLE) is used to compress the row of K as $[1C{:}6I{:}8C{:}9I{:}10C]$ (the compressed row indicates that the value in this row for indices $[1, 6)$ is C, the value is I for indices $[6, 8)$ and so on). To recover the shortest path from K to G, a binary search is conducted on the RLE string of K to extract the first move I on the shortest path from K to G. Next, the algorithm conducts a binary search on the RLE string of I to extract the first move G from I to G. The algorithm stops since G is reached.

Using the CPD, the shortest path can be obtained in $O(e)$ first move extractions where e is the number of edges on the shortest path. Each first move extraction takes $O(\log r)$ where r is the size of the compressed row. Thus, the total cost to obtain the shortest path/distance using a CPD is $O(e \log r)$. Next, we present the details of our proposed data structure called CVPD+DL which allows computing distance between any two vertices $u \in V$ and $v \in V$

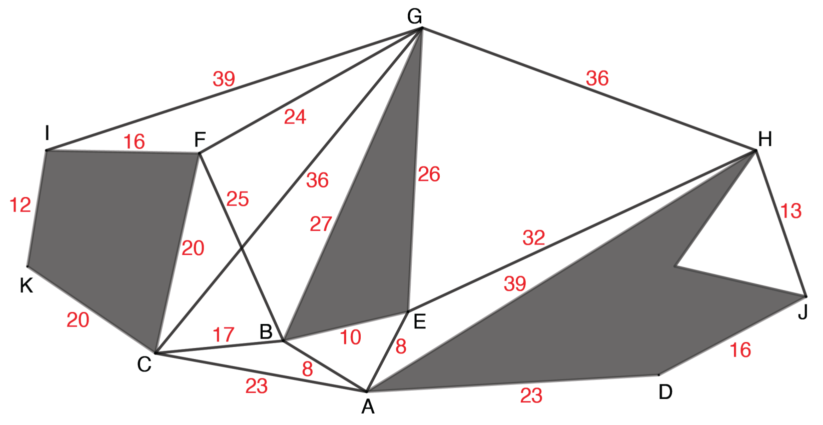


Fig. 4 An example showing three polygonal obstacles

	A	B	C	D	E	F	G	H	I	J	K
I	F	F	K	F	F	F	G	G	*	F	K
K	C	C	C	C	C	I	I	C	I	C	*

Table 2 Rows I and K of a Compressed Path Database (CPD)

in logarithmic time. Once the distance $d(u,v)$ is computed, the shortest path between u and v can be computed in time linear to the number of edges on the shortest path, i.e., $O(e)$.

4.1.1 CVPD+DL:

The proposed index, CVPD+DL, consists of two new indexes namely Compressed Via-Path Database (CVPD) and Distance Labels (DL).
Compressed Via-Path Database (CVPD): We impose a strict total order on all vertices in V. Although any ordering can be used, in this paper, we use betweenness centrality score [8] to order the nodes where the betweenness score of a vertex v is the number of shortest paths passing through v. The betweenness scores of vertices are computed by constructing all shortest path trees. We break the ties arbitrarily but consistently, e.g., using node IDs. Ranks of nodes are their positions in this order, e.g, the highest ranked node is the node with the highest betweenness score. We use $u <_B v$ to denote that u ranks higher than v.

While a traditional CPD stores the first move f on the shortest path from $u \in V$ to $v \in V$, a CVPD stores the highest ranked vertex h on the shortest path between u and v. This vertex is called the highest via node on the shortest path and is denoted as $via(u,v)$. Since $sp(u,v) = sp(v,u)$, we have $via(u,v) = via(v,u)$. Therefore, we only need to store the highest via node $via(u,v)$ only once in the CVPD. Specifically, we store $via(u,v)$ in the row of a vertex u only if $u <_B v$. Otherwise, we store a wildcard symbol "*" to achieve better compression. Note that, unlike CVPD, the traditional CPDs are not symmetric and need to store first move from u to v as well as the first move from v to u.

Example 3 Table 3 shows CVPD for the example shown in Figure 4. The alphabetical order of vertices in Figure 4 represents the betweenness ranks, i.e., A is the highest ranked node and K is the lowest ranked node. Consider the row for vertex F. It stores wildcard symbols for nodes A to F as the rank of F is not higher than the rank of each of these nodes. The shortest path between F and J is $\langle F, B, A, D, J\rangle$ and the highest ranked via node A on this path is stored in the CVPD. For the remaining nodes in this row (i.e., G, H, I, and K), the highest via node on the shortest path from F to these nodes is F itself. Therefore, F is stored for these nodes. The RLE compression of this row $R(F)$ gives [1F:10A:11F]. We remark that, in practice, the columns of the CVPD (and CPD) are ordered following a Depth-First Search (DFS) order to achieve better compression. However, for the sake of simplicity, in our examples, we show the columns ordered according to their betweenness ranks (A to K).

Given the CVPD, for any pair of vertices u and v, we can obtain the highest ranked node on the shortest path between u and v using the compressed row of u (if $u <_B v$) or using the compressed row of v (if $v <_B u$).

Distance Labels (DL): For each vertex u, we store a list of distance labels denoted as $DL(u)$. Specifically, $DL(u)$ contains a distance label for every $v \in V$ for which: 1) $u <_B v$; **and** 2) u is the highest ranked vertex on the shortest path between u and v. Each distance label in $DL(u)$ is a triplet $(v, d(u, v), p)$ where p is the first vertex on the shortest path *from v to u*. The labels in each $DL(u)$ are sorted according to the betweenness ranks of vertices v which allows finding the label of v in $DL(u)$ in logarithmic time. To reduce the number of distance labels, if u and v are co-visible, we do not store the distance label for v in $DL(u)$. For such u and v, at query time, a binary search can be conducted on $DL(u)$ and if v was expected to be in $DL(u)$ but is not found, it implies that u and v are co-visible and the Euclidean distance between them can be computed on-the-fly (as we show later in Example 5).

Example 4 Table 4 shows the distance labels for all vertices in Figure 4. As shown in Table 3, F is the highest ranked via node on the shortest path from F to each of G, H, I and K (see row F in Table 3). Since G and I are visible from F, the distance labels to them are not added. Therefore, we add in $DL(F)$ the distance labels to H and K (see row F in Table 4). The label $(H, 60, G)$ in $DL(F)$ indicates that the distance between F and H is 60 and the first vertex on the shortest path from H to F is G.

Given the CVPD+DL, the shortest distance $d(u, v)$ can be efficiently computed as follows. Without loss of generality, assume $u <_B v$. First, the highest via node h on the shortest path between u and v is found using a binary search on the compressed row of u in the CVPD. Then, the distances $d(u, h)$

	A	B	C	D	E	F	G	H	I	J	K
A	*	A	A	A	A	A	A	A	A	A	A
B	*	*	B	A	B	B	B	B	B	A	B
C	*	*	*	A	B	C	C	B	C	A	C
D	*	*	*	*	A	A	A	D	A	D	A
E	*	*	*	*	*	B	E	E	B	E	B
F	*	*	*	*	*	*	F	F	F	A	F
G	*	*	*	*	*	*	*	G	G	G	G
H	*	*	*	*	*	*	*	*	G	H	B
I	*	*	*	*	*	*	*	*	*	A	I
J	*	*	*	*	*	*	*	*	*	*	A
K	*	*	*	*	*	*	*	*	*	*	*

Table 3 Compressed Via-Path Database (CVPD)

	Distance labels
A	(F, 33, B), (G, 34, E), (I, 49, F), (J, 39, D), (K, 43, C)
B	(H, 42, E), (I, 41, F), (K, 37, C)
C	(I, 32, K)
D	(H, 29, J)
E	(J, 45, H)
F	(H, 60, G), (K, 28, I)
G	(J, 49, H), (K, 51, I)
H	
I	
J	
K	

Table 4 Distance labels

and $d(v, h)$ are obtained from $DL(h)$ using two binary searches[5] to find the labels $(u, d(u, h), p)$ and $(v, d(v, h), p')$, and $d(u, v) = d(u, h) + d(v, h)$. Note that this requires conducting at most three binary searches (one on the compressed row of u and two on $DL(h)$). To obtain the shortest path, the first moves can be used to recursively recover the path. For example, to obtain the path from u to h, the first vertex p is obtained from the label $(u, d(u, h), p)$. Then, the label of p is found in $DL(h)$ and this process continues until h is reached. The label of p can be obtained using another binary search. Alternatively, with each label $(u, d(u, h), p)$ we can store an additional value p_i which corresponds to the position (i.e., index) of the label of p in $DL(h)$ (or -1 if p is not in $DL(h)$ because it is visible from h). This allows obtaining each successive label in $O(1)$ resulting in $O(e)$ time to recover the whole path.

Example 5 Consider our running example and assume that $d(F, J)$ is to be computed. We conduct a binary search on CVPD on row F and find the

[5] For some highest ranked nodes, the distance labels can be stored using $O(V)$ space so that the labels can be found in $O(1)$ instead of logarithmic time.

	A	B	C	D	E	F	G	H	I	J	K
A	*	-	-	-	-	-	-	D	C	-	F
B	*	*	-	-	-	-	-	D	C	E	F
C	*	*	*	-	-	I	I	D	-	E	-
D	*	*	*	*	-	-	H	-	C	-	F
E	*	*	*	*	*	-	-	-	C	A	F
F	*	*	*	*	*	*	-	D	-	G	-
G	*	*	*	*	*	*	*	-	-	A	-
H	*	*	*	*	*	*	*	*	D	-	G
I	*	*	*	*	*	*	*	*	*	G	-
J	*	*	*	*	*	*	*	*	*	*	E
K	*	*	*	*	*	*	*	*	*	*	*

Table 5 Compressed 2nd Via-Path Database (CVPD^2)

highest via node A on $sp(F, J)$. Then, we conduct two binary searches on $DL(A)$ to find the labels corresponding to F and J: $(F, 33, B)$; and $(J, 39, D)$. Here, $d(F, J) = d(F, A) + d(J, A) = 33 + 39 = 72$. The shortest paths from F to A and J to A are obtained to recover the path. For example, the label $(F, 33, B)$ indicates that the first vertex on the shortest path from F to A is B. Next, the label of B is searched in $DL(A)$ which is not found indicating B and A are co-visible. Note that with the label $(F, 33, B)$ we can store the index of B in $DL(A)$ (i.e., -1 in this case indicating that the label of B is not present) which helps avoiding a binary search. The shortest path from J to A is recovered similarly.

4.1.2 Compressed i-th Via-Path Database (CVPD^i):

Now, we present a generalisation of the CVPD called Compressed i-th Via-Path Database (denoted as CVPD^i). While a CVPD records the highest ranked node on the shortest path between u and v, a CVPD^i stores the highest ranked node on the i-th longest plateau between u and v. We ignore the zero length plateaus (i.e., consisting of only a single vertex), and store a special symbol "-" in CVPD^i for such cases. Since i-th longest plateau between u and v is the same as the i-th longest plateau between v and u, we store the highest ranked via node in the row of u only if $u <_B v$. It can be shown that the shortest path between u and v is the longest plateau between u and v. Therefore, CVPD is sometimes denoted as CVPD^1 hereafter. In total, we create m CVPD^{i}s denoted as CVPD^1,$\cdots$, CVPD^m. In our experiments, we evaluate the effect of m on preprocessing time, storage cost and query performance.

Example 6 Table 5 shows CVPD^2 for our running example. Consider the row F. Similar to CVPD, we store "*" for nodes A to F. The second longest plateau between F and H is $\langle D, J\rangle$ and CVPD^2 stores the highest ranked node D on this plateau. Similarly, the second longest plateau between F and J is $\langle G, H\rangle$ and we store the highest ranked node G on this plateau. For other nodes, we store "-" as the second longest plateaus between F and these nodes contain only a single vertex each.

Given a CVPD^i, we can obtain the highest via node – the highest ranked node n on the i-th plateau between u and v – using a binary search on CVPD^i. The via-path between u and v passing through n (i.e., $sp(u, n) \oplus sp(n, v)$) and its length can then be obtained using CVPD+DL as described earlier.

4.1.3 Advantages of CVPD^{i}s and DL:

A major advantage of the proposed CVPD^{i}s in the context of alternative pathfinding is that these can be used to efficiently obtain the highest ranked nodes on i longest plateaus between each pair of vertices visible from s and t. Such vertices are likely to be on high-quality alternative paths between s and t because longer plateaus typically generate better quality alternative paths [29]. CVPD+DL can then be used to efficiently recover these paths and/or find their lengths. While traditional CPDs can be used to recover the paths, CVPD+DL has some advantages. Firstly, the traditional CPD needs to recover the whole shortest path in order to find the distance $d(u, v)$. On the other hand, CVPD+DL can find $d(u, v)$ in at most three binary searches. Secondly, the CPD recovers the shortest path by recursively finding first moves which requires $O(e \log r)$ whereas CVPD+DL recovers the shortest path in $O(e)$ once the shortest distance has been computed in logarithmic time. Furthermore, to obtain each successive first move, CPDs need to do binary look up in different rows of the CPD. In contrast, CVPD+DL can recover the shortest path using a single row $DL(h)$ resulting in fewer cache misses.

4.1.4 Advantages Compared to Hub Labels:

Traditional hub labeling approaches [39] store a set of hub labels for each vertex v such that $d(u, v)$ can be computed by finding the common hub nodes in the labels of u and v. Finding all common hub nodes requires linear search on both the labels of u and v with complexity $O(|HL(u)| + |HL(v)|)$ where $|HL(x)|$ is the number of hub labels stored for a vertex x. In contrast, CVPD+DL does not need to find the common hub labels. Instead, the highest via node is found using a binary search on the compressed row of u assuming $u <_B v$. Also, to recover the shortest paths, the hub labeling approaches need to recursively obtain successors which requires accessing hub labels for different nodes (resulting in potentially more cache misses) wheres CVPD+DL can recover the shortest path using a single row $DL(h)$. In the context of alternative pathfinding which is our main focus, traditional hub labeling cannot be used (or trivially extended) because the number of common hub nodes between two nodes may be less than k (and, even if there are at least k common hub nodes, they may not generate high-quality alternative paths as the hub nodes are not selected with an aim to generate alternative paths).

Algorithm 1: Efficient Alternative Pathfinding

```
Input  : $s$; $t$; $\epsilon$; $\theta$: dissimilarity threshold
Output: a set $\mathcal{P}$ containing up to $k$ alternative paths
1  $\mathcal{P} \leftarrow \emptyset$;
2  if $s$ and $t$ are co-visible then
3      $\mathcal{P} \leftarrow \mathcal{P} \cup \langle s, t \rangle$;
4  $V_s \leftarrow$ get vertices visible from $s$ ;
5  $V_t \leftarrow$ get vertices visible from $t$ ;
6  $ViaNodes \leftarrow \emptyset$;
7  for each $u \in V_s$ do
8      for each $v \in V_t$ do
9          for $i$ in 1 to $m$ do
10             $n \leftarrow \text{CVPD}^i(u, v)$;
11             $ViaNodes \leftarrow ViaNodes \cup n$;
12 $VPaths \leftarrow \emptyset$;
13 for each $n \in ViaNodes$ do
14     use CVPD+DL, $V_s$ and $V_t$ to get $sp(s, n)$ and $sp(n, t)$;
15     $vp \leftarrow sp(s, n) \oplus sp(n, t)$;
16     if $vp$ is taut and $|vp| \leq d(s, t) \times \epsilon$ then
17         Add $vp$ to $VPaths$ ;
18 for each $vp \in VPaths$ in asc. order of length do
19     if $1 - Sim(\mathcal{P} \cup vp) \geq \theta$ then
20         $\mathcal{P} \leftarrow \mathcal{P} \cup vp$;
21         return $\mathcal{P}$ if it contains $k$ paths;
22 return $\mathcal{P}$;
```

4.2 Online Query Processing

4.2.1 Algorithm

Details of our query processing algorithm are shown in Algorithm 1. If s and t are co-visible (which can be checked using ray shooting or Polyanya [15]), the path $\langle s, t \rangle$ is added to the set of alternative paths $\mathcal{P}$. Then, Polyanya is employed (as described in [53]) to retrieve the sets of convex vertices visible from s and t, denoted as V_s and V_t, respectively. The algorithm then utilises the CVPD^{i}s to obtain a set of via nodes denoted as $ViaNodes$ (lines 6 to 11). Specifically, for each pair of visible vertices u and v, the algorithm uses the CVPD^{i}s to obtain the highest i-th via nodes and adds these to $ViaNodes$. In the experiments, we evaluate the effect of m, the number of CVPD^{i}s used by the algorithm. We remark that although the maximum number of nodes in $ViaNodes$ is $|V_s| \times |V_t| \times m$, in practice, the number of nodes in $ViaNodes$ is very small (less than 10 in most cases in our experiments) because the highest ranked nodes usually are the same between many pairs of visible vertices.

Once $ViaNodes$ are found, the algorithm accesses each via node n and uses CVPD+DL to obtain the via-path $sp(s, n) \oplus sp(n, t)$. If the via-path is taut and its length is not greater than $d(s, t) \times \epsilon$, it is added to a list of via paths $VPaths$ (the candidate alternative paths). Finally, the algorithm accesses each via path in ascending order of their lengths, and if its dissimilarity with the existing paths in $\mathcal{P}$ is at least θ, it is added to $\mathcal{P}$. Although any dissimilarity function can be employed, we compute the dissimilarity of a set of paths $\mathcal{P}$ as $1 - Sim(\mathcal{P})$ (see Eq. (3) in Section 2). The algorithm returns $\mathcal{P}$ when it contains k paths or when the algorithm terminates.

We include many optimisations to the basic algorithm described above. Specifically, we remove the *dead-end* and non-taut vertices from V_s and V_t as discussed in [53]. Also, the vertices in V_s (resp. V_t) are accessed in ascending order of $d(s,u)+EDist(u,t)$ (resp. $d(t,v)+EDist(v,s)$) to obtain the via nodes between more promising nodes earlier and, instead of accessing all nodes in V_s and V_t, we stop after accessing at most N nodes each from V_s and V_t. We evaluate the effect of N in experiments.

4.2.2 Extending to other objective functions

Similar to the existing works in road networks, Algorithm 1 primarily aims to minimise the path lengths while satisfying a certain dissimilarity criterion. However, different users may have different requirements, e.g., one user may define a multi-objective function as weighted sum of different measures (e.g., similarity, length, bounded stretch) and may want to retrieve alternative paths with the smallest weighted sum. On the other hand, another user may want to minimise one particular measure while imposing constraints on the other measures, e.g., return alternative paths with the smallest total length such that path similarity is at most 0.5 and bounded stretch is at most 0.3. Algorithm 1 can be easily adapted to meet the demands of these different users. Specifically, at line 18, the algorithm currently accesses the candidate paths $VPaths$ in ascending order of lengths and, at line 19, filters a path if adding it will violate the path similarity constraint. If the objective function is weighted sum, the algorithm can process the candidate paths in ascending order of their weighted sum at line 18, and apply additional filters at line 19 if needed (e.g., if constraints on bounded stretch are required by the user, the path that violates the bounded stretch constraint can be filtered).

5 Experiments

5.1 Settings

Following the experimental settings of previous works on pathfinding in game maps, we conduct experiments on the widely used benchmarks[6] with a total of 298 game maps, as described in [57]. The benchmarks contain 67 maps from Dragon Age II (DA), 156 maps from Dragon Age Origins (DAO), and 75 maps from Baldur's Gate II (BG) (see Table 6 which also shows the total number of queries for each benchmark, average number of vertices in each map and average number of convex vertices in each map).

Algorithm 1 is shown as CVPD(N) in the experimental results where the value of N indicates the maximum number of vertices processed from each of

[6] `https://github.com/nathansttt/hog2`

	Benchmark Stats				Build Time	Memory (MB)			
	#M	**#Q**	**#V**	**#CV**		**CVPD^1**	**CVPD^2**	**CVPD^3**	**DL**
DA	67	68K	1183	611	0.020	0.436	0.058	0.055	0.132
DAO	156	159K	1728	927	0.209	1.744	0.272	0.312	0.387
BG	75	93K	1295	668	0.074	0.978	0.138	0.153	0.217

Table 6 Benchmark stats include total # of maps (**#M**) and total # of queries (**#Q**) in each benchmark, and average # of vertices (**#V**) and convex vertices (**#CV**) per map. For each benchmark, we also show average build time (mins) per map for constructing three CVPD^{i}s and DL as well as average memory (MB) per map for the three CVPD^{i}s and DL.

V_s and V_t at lines 7 and 8. We construct three CVPD^{i}s and evaluate the effect of using different numbers of CVPD^{i}s in the experiments. We evaluate our algorithm, CVPD(N), against some of the most well-known existing techniques: Plateaus, Dissimilarity, and Penalty which are shown as Pla, Dissim, and Pen, respectively, in the results.

We set the penalty factor for the Penalty approach to 1.4. The dissimilarity threshold θ for Dissimilarity and CVPD(N) was set to 0.6. The penalty factor and dissimilarity threshold were selected after trying different values and choosing the best values. ϵ was set to 1.5 for each approach. The default value of k was set to 3.

In addition to comparison with the three alternative pathfinding algorithms mentioned above, we also included Polyanya [15] (displayed as Poly) as a competitor. Polyanya is the state-of-the-art online shortest path algorithm in game maps. Please note that Polyanya only finds the shortest path and not the alternative paths. However, we included Polyanya in the experimental study to demonstrate the overhead cost of finding the alternative paths compared to only finding the shortest path. The source code of Polyanya was provided by its authors[7].

All algorithms were implemented in C++ and compiled with GNU GCC 4.8. We conduct experiments on a Linux (64-bit) dedicated NeCTAR server m1.xxlarge instance with Intel Core Processor 2.9GHz 16-core CPUs and 16GB DDR4-1866 memory.

We evaluate the algorithms considering the pre-processing cost (build time and memory required), query runtimes, and the quality of the alternative paths found by each algorithm. In order to evaluate the quality of the returned alternative paths $\mathcal{P}$, we use bounded stretch $BS(\mathcal{P})$ (lower the better), local optimality $LO(\mathcal{P})$ (higher the better), and similarity $Sim(\mathcal{P})$ (lower the better) as defined in Section 2.2. In our experiments, we report average values over all queries for each of these measures. In addition, we report the maximum of $BS(\mathcal{P})$ and $Sim(\mathcal{P})$. Note that maximum of $BS(\mathcal{P})$ and $Sim(\mathcal{P})$ correspond to the worst-case bounded stretch and similarity for an algorithm across all queries. Furthermore, we report the minimum of $LO(\mathcal{P})$ considering all queries, which represents the worst-case for local optimality. In some cases, an approach may only be able to return less than k alternative paths.

[7] `https://bitbucket.org/mlcui1/polyanya`

Such an approach may get better quantitative scores only because it generated less than k alternative paths which is unfair. For this reason, we only take into consideration the queries that return exactly k alternative paths for all approaches.

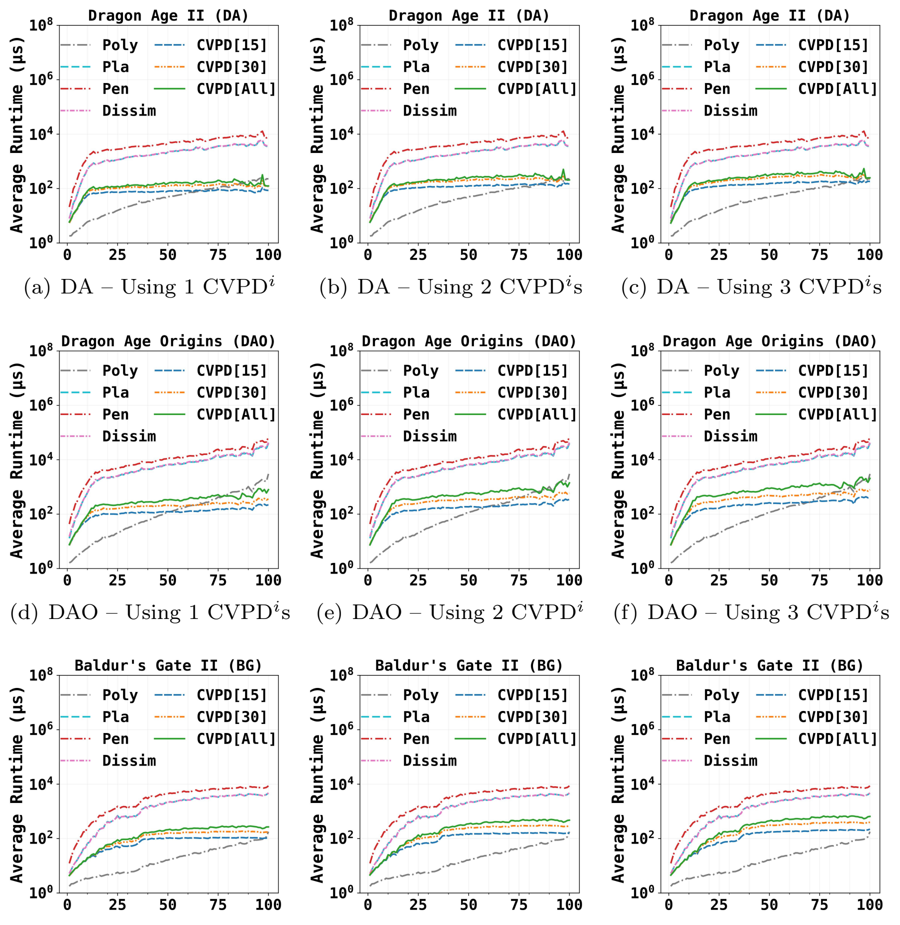
5.2 Results

5.2.1 Preprocessing Time and Memory

Table 6 shows the average build time (in minutes) per map for constructing three CVPD^{i}s and the distance labels (DL). It also shows the memory required by each CVPD^i and DL (in MB). The results show that the build time and the memory required by the CVPDs and DL are quite small. CVPD^1 consumes significantly more memory than CVPD^2 and CVPD^3. This is because for CVPD^2 and CVPD^3, for many pairs of vertices especially those that are close to each other, the 2nd (or 3rd) plateaus do not exist resulting in many cells having the special symbol "-" which leads to better compression.

5.2.2 Query runtimes

Figure 5 demonstrates query runtimes for all algorithms on DA, DAO, and BG maps (please note log-scale on y-axis). In each figure, the x-axis ranks the queries roughly in the order of their difficulty. Specifically, following the existing shortest pathfinding approaches, we sort the queries based on the number of nodes expansion required by the standard A* search to solve them (which serves as a proxy of how challenging a query is). The x-axis denotes the percentile ranks of queries in this order. As Figure 5 shows, Plateaus and Dissimilarity exhibit almost equal query times because both need to compute the shortest paths trees rooted at the source and target which is the dominant cost. The penalty is more expensive given the fact that it requires the iterative computation of k unique paths. We show the performance of CVPD for $N = 15$, $N = 30$ and $N =$ All, which refers to the case when it processes all visible vertices from both ends. CVPD is more than an order of magnitude faster than Plateaus, Dissimilarity, and Penalty. Furthermore, its query time is comparable to (or even better than) Polyanya for the more challenging queries. We also show the performance of our algorithm when it uses one, two, or three CVPD^{i}s. For example, Figures 5(a), 5(b) and 5(c) show the runtimes of CVPD when it uses one, two and three CVPD^{i}s, respectively. As expected, the query processing times increase as the number of CVPD^{i}s used by the algorithms increases (please note that y-axis is in log-scale). However, in all cases, CVPD significantly outperforms the existing alternative pathfinding algorithms.



(a) DA – Using 1 CVPD^i (b) DA – Using 2 CVPD^{i}s (c) DA – Using 3 CVPD^{i}s

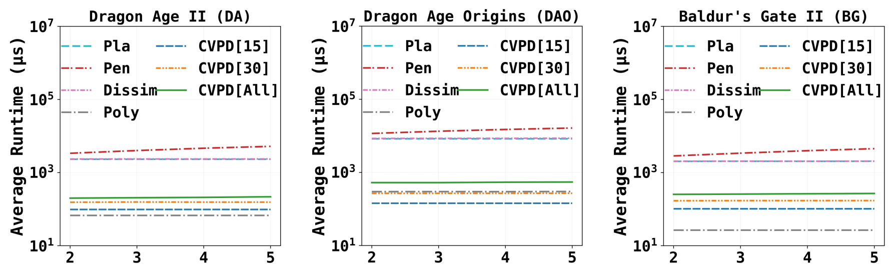
(d) DAO – Using 1 CVPD^{i}s (e) DAO – Using 2 CVPD^i (f) DAO – Using 3 CVPD^{i}s

(g) BG – Using 1 CVPD^i (h) BG – Using 2 CVPD^i (i) BG – Using 3 CVPD^i

Fig. 5 x-axis shows the percentile ranks of queries in number of node expansions needed by A* search to solve them.

5.2.3 Varying K

Figures 6(a), 6(b) and 6(c) display the average query processing time when k is varied from 2 to 5. The cost of Plateaus and Dissimilarity is independent of k. The reason is that the dominant cost of those two algorithms is constructing the forward and backward shortest path trees and this construction cost does not depend on the value of k. The cost of Penalty is positively associated with k. The reason lies in the fact that the algorithm involves k iterations to get the result. We only run Polyanya for $k = 1$ as the algorithm only generates the shortest path (and therefore is expected to be faster than the other algorithms that generate k alternative paths). We show the performance of CVPD for $N = 15$, $N = 30$, and $N =$ All when it uses two CVPD^{i}s. Again,



(a) DA (b) DAO (c) BG

Fig. 6 Effect of varying k.

CVPD outperforms the other alternative pathfinding algorithms by more than an order of magnitude and has performance comparable to Polyanya. Interestingly, CVPD outperforms Polyanya when $N = 15$ and $N = 30$ on the DAO benchmark.

5.2.4 Quality of alternative paths

Table 7 compares the quality of alternative paths that different algorithms generate. As the table shows, the quality of alternative paths generated by our approach is comparable to the existing approaches. However, as our experimental study showed, our approach is significantly faster than the existing approaches, i.e., by more than an order of magnitude. The average bounded stretch produced by our approach outperforms Plateaus and is similar to that of Dissmiliarity and Penalty. With regard to average similarity, our approach sits at the second rank, following Penalty which performs the best. However, Penalty underperforms in terms of the worst-case (i.e., max) similarity (with maximum similarity around 0.9), whereas our approach and Dissimilarity ensure a maximum similarity level at no more than 0.4. In terms of local optimality, our algorithm outperforms the other algorithms. Dissimilarity is the best approach in terms of average path length followed by our approach. We note that using more CVPD^{i}s reduces the average path length of our approach but does not have a clear benefit in terms of the other measures.

6 Conclusions

To the best of our knowledge, we present the first comprehensive study on computing alternative pathfinding in game maps. First, we adapt the previous works, specifically designed for finding alternative paths in road networks, to find the alternative paths in game maps. Then, based on a web-based system that allows users visualise the paths generated by different approaches, we conduct a user study which demonstrates that these previous approaches are capable of generating high-quality alternative paths in game maps. One

Algorithm	$BS(\mathcal{P})$		$Sim(\mathcal{P})$		$LO(\mathcal{P})$		**Length**
	AVG	**MAX**	**AVG**	**MAX**	**AVG**	**MIN**	
DA							
Dissimilarity	**1.086**	**2.087**	0.188	0.400	0.352	0.008	**121.5**
Plateau	1.216	3.335	0.232	0.882	0.261	0.008	125.4
Penalty	1.157	7.000	**0.129**	0.953	0.354	0.008	124.1
1-CVPD(15)	1.096	2.396	0.151	**0.397**	0.398	0.008	123.1
1-CVPD(30)	1.091	2.396	0.153	**0.397**	**0.401**	0.008	123.0
1-CVPD(All)	1.091	2.396	0.153	**0.397**	**0.401**	0.008	123.0
2-CVPD(15)	1.099	7.000	0.158	**0.397**	0.376	0.008	122.6
2-CVPD(30)	1.094	7.000	0.160	**0.397**	0.378	0.008	122.5
2-CVPD(All)	1.094	7.000	0.160	**0.397**	0.378	0.008	122.5
3-CVPD(15)	1.099	7.000	0.164	0.398	0.361	0.008	122.3
3-CVPD(30)	1.096	7.000	0.164	0.398	0.364	0.008	122.2
3-CVPD(All)	1.096	7.000	0.164	0.398	0.363	0.008	122.2
DAO							
Dissimilarity	**1.050**	**3.481**	0.162	**0.400**	0.336	0.005	**124.9**
Plateau	1.207	6.859	0.184	0.931	0.190	0.005	132.7
Penalty	1.103	19.00	**0.083**	0.931	0.335	0.005	126.9
1-CVPD(15)	1.065	5.472	0.125	**0.400**	0.347	0.005	126.4
1-CVPD(30)	1.059	5.472	0.129	**0.400**	**0.351**	0.005	126.2
1-CVPD(All)	1.058	5.472	0.130	**0.400**	**0.351**	0.005	126.2
2-CVPD(15)	1.063	5.472	0.131	**0.400**	0.337	0.005	126.1
2-CVPD(30)	1.058	5.472	0.133	**0.400**	0.343	0.005	125.8
2-CVPD(All)	1.057	5.472	0.135	**0.400**	0.342	0.005	125.8
3-CVPD(15)	1.063	5.472	0.136	**0.400**	0.333	0.005	125.8
3-CVPD(30)	1.059	5.472	0.137	**0.400**	0.340	0.005	125.6
3-CVPD(All)	1.057	5.472	0.139	**0.400**	0.340	0.005	125.6
BG							
Dissimilarity	1.092	2.903	0.136	0.400	0.342	0.009	**282.1**
Plateau	1.224	4.454	0.137	0.876	0.252	0.009	298.0
Penalty	**1.088**	5.333	**0.089**	0.831	0.343	0.009	286.7
1-CVPD(15)	1.106	4.106	0.114	**0.398**	0.335	0.009	285.5
1-CVPD(30)	1.098	4.106	0.116	**0.398**	0.350	0.009	284.9
1-CVPD(All)	1.096	4.106	0.117	**0.398**	**0.351**	0.009	284.8
2-CVPD(15)	1.100	3.672	0.121	**0.398**	0.337	0.009	284.2
2-CVPD(30)	1.094	**2.399**	0.122	0.399	0.350	0.009	283.8
2-CVPD(All)	1.093	**2.399**	0.123	0.399	**0.351**	0.009	283.8
3-CVPD(15)	1.098	3.672	0.125	0.400	0.337	0.009	283.5
3-CVPD(30)	1.092	**2.399**	0.126	0.399	0.349	0.009	283.2
3-CVPD(All)	1.091	**2.399**	0.127	0.399	0.349	0.009	283.2

Table 7 Quality of alternative paths on DA, DAO and BG maps. We show $BS(\mathcal{P})$ (smaller the better), $Sim(\mathcal{P})$ (smaller the better), $LO(\mathcal{P})$ (larger the better) and average path length. Best values for each column are shown in bold. x-CVPD(N) corresponds to the case when x CVPD^{i}s are used by our algorithm and N vertices are processed from each of V_s and V_t.

limitation of the existing techniques is their high query runtimes. To address this, we propose an efficient algorithm to generate alternative paths using a novel variation of CPDs called Compressed Via-Path Database (CVPD). Our experimental study shows that the proposed approach is more than an order of magnitude faster than the existing alternative pathfinding approaches and is capable of generating high-quality paths of similar quality.

Declarations

Ethical approval. The research was considered by the Monash University Human Research Ethics Committee. The Committee was satisfied that the research meets the requirements of the National Statement on Ethical Conduct in Human Research and granted approval.
Competing interests. The authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.
Authors' contributions. All authors contributed to the algorithm development, problem formulation and experiments design. L.L. implemented the algorithms and the web-based system. L.L. conducted the user study and the experiments. L.L. and M.C. wrote the main manuscript text. L.L. prepared all the figures. All authors reviewed the manuscript.
Funding. Muhammad Aamir Cheema is supported by Australian Research Council (ARC) FT180100140. Hua Lu is supported by Independent Research Fund Denmark (No. 8022-00366B).
Availability of data and materials. The web-based system is publicly accessible at `http://aamircheema.com/paths_games/`. The game maps benchmark is also publicly accessible as noted in Section 5.

References

1. Abeywickrama, T., Cheema, M.A., Taniar, D.: K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation. PVLDB pp. 492–503 (2016)
2. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A hub-based labeling algorithm for shortest paths in road networks. In: International Symposium on Experimental Algorithms. Springer (2011)
3. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Alternative routes in road networks. Journal of Experimental Algorithmics (JEA) **18**, 1–3 (2013)
4. Akgün, V., Erkut, E., Batta, R.: On finding dissimilar paths. European Journal of Operational Research **121**(2), 232–246 (2000)
5. Akiba, T., Iwata, Y., Kawarabayashi, K.i., Kawata, Y.: Fast shortest-path distance queries on road networks by pruned highway labeling. In: ALENEX (2014)
6. Banavar, J.R., Maritan, A., Rinaldo, A.: Size and form in efficient transportation networks. Nature **399**(6732), 130–132 (1999)
7. Botea, A.: Fast, optimal pathfinding with compressed path databases. In: SOCS (2012)
8. Brandes, U., Pich, C.: Centrality estimation in large networks. International Journal of Bifurcation and Chaos **17**(07), 2303–2318 (2007)
9. Cheema, M.A.: Indoor location-based services: challenges and opportunities. SIGSPATIAL Special **10**(2), 10–17 (2018)

10. Chen, B.Y., Lam, W.H., Sumalee, A., Li, Q., Shao, H., Fang, Z.: Finding reliable shortest paths in road networks under uncertainty. Networks and spatial economics **13**(2), 123–148 (2013)
11. Chen, Y., Bell, M.G., Bogenberger, K.: Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. IEEE Transactions on Intelligent Transportation Systems (2007)
12. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Exact and approximate algorithms for finding k-shortest paths with limited overlap. In: 20th International Conference on Extending Database Technology: EDBT 2017, pp. 414–425 (2017)
13. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U., Blumenthal, D.B.: Finding k-dissimilar paths with minimum collective length. In: SIGSPATIAL (2018)
14. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U., Blumenthal, D.B.: Finding k-shortest paths with limited overlap. The VLDB Journal pp. 1–25 (2020)
15. Cui, M., Harabor, D.D., Grastien, A.: Compromise-free pathfinding on a navigation mesh. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pp. 496–502. ijcai.org (2017)
16. Dees, J., Geisberger, R., Sanders, P., Bader, R.: Defining and computing alternative routes in road networks. CoRR **abs/1002.4330** (2010). URL `http://arxiv.org/abs/1002.4330`
17. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269–271 (1959)
18. Döbler, H., Scheuermann, B.: On computation and application of k most locally-optimal paths in road networks (2016)
19. Du, J., Michalska, S., Subramani, S., Wang, H., Zhang, Y.: Neural attention with character embeddings for hay fever detection from twitter. Health information science and systems **7**, 1–7 (2019)
20. Du, J., Shen, B., Cheema, M.A.: Ultrafast euclidean shortest path computation using hub labeling. In: AAAI (2023)
21. Eppstein, D.: Finding the k shortest paths. SIAM Journal on computing **28**(2), 652–673 (1998)
22. Ernst, P., Meng, C., Siu, A., Weikum, G.: Knowlife: a knowledge graph for health and life sciences. In: 2014 IEEE 30th International Conference on Data Engineering, pp. 1254–1257. IEEE (2014)
23. Funke, S., Nusser, A., Storandt, S.: On k-path covers and their applications. The VLDB Journal **25**(1), 103–123 (2016)
24. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: International Workshop on Experimental and Efficient Algorithms, pp. 319–333. Springer (2008)
25. Grusho, A.A., Abaev, P.O., Shorgin, S.Y., Timonina, E.E.: Graphs for information security control in software defined networks. In: AIP Conference Proceedings, vol. 1863, p. 090002. AIP Publishing LLC (2017)
26. Häcker, C., Bouros, P., Chondrogiannis, T., Althaus, E.: Most diverse near-shortest paths. In: Proceedings of the 29th International Conference on Advances in Geographic Information Systems, pp. 229–239 (2021)
27. He, J., Rong, J., Sun, L., Wang, H., Zhang, Y., Ma, J.: A framework for cardiac arrhythmia detection from iot-based ecgs. World Wide Web **23**, 2835–2850 (2020)
28. Islam, M.R., Kabir, M.A., Ahmed, A., Kamal, A.R.M., Wang, H., Ulhaq, A.: Depression detection from social network data using machine learning techniques. Health Inf. Sci. Syst. **6**(1), 8 (2018)
29. Jones, A.H.: Method of and apparatus for generating routes (2012). US Patent 8,249,810
30. Kallmann, M., Kapadia, M.: Navigation meshes and realtime dynamic planning for virtual worlds. In: ACM SIGGRAPH 2014 Courses, p. 3. ACM Press (2014)
31. Kobitzsch, M.: An alternative approach to alternative routes: Hidar. In: European Symposium on Algorithms, pp. 613–624. Springer (2013)
32. Kobitzsch, M., Radermacher, M., Schieferdecker, D.: Evolution and evaluation of the penalty method for alternative graphs. In: ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013, vol. 33, pp. 94–107. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik (2013)

33. Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., Teng, S.H.: On trip planning queries in spatial databases. In: International symposium on spatial and temporal databases, pp. 273–290. Springer (2005)
34. Li, L., Cheema, M.A.: Alternative pathfinding in game maps and indoor venues. ICAPS (2021)
35. Li, L., Cheema, M.A., Ali, M.E., Lu, H., Li, H.: Diverse shortest paths in game maps: A comparative user study and experiments. In: Australasian Database Conference, pp. 76–88. Springer (2022)
36. Li, L., Cheema, M.A., Ali, M.E., Lu, H., Taniar, D.: Continuously monitoring alternative shortest paths on road networks. Proceedings of the VLDB Endowment **13**(12), 2243–2255 (2020)
37. Li, L., Cheema, M.A., Lu, H., Ali, M.E., Toosi, A.N.: Comparing alternative route planning techniques: A comparative user study on melbourne, dhaka and copenhagen road networks. IEEE Transactions on Knowledge and Data Engineering (2021)
38. Li, Y., Cao, B., Peng, M., Zhang, L., Zhang, L., Feng, D., Yu, J.: Direct acyclic graph-based ledger for internet of things: Performance and security analysis. IEEE/ACM Transactions on Networking **28**(4), 1643–1656 (2020)
39. Li, Y., Yiu, M.L., Kou, N.M., et al.: An experimental study on hub labeling based shortest path algorithms. Proceedings of the VLDB Endowment **11**(4), 445–457 (2017)
40. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k shortest paths with diversity. IEEE Transactions on Knowledge and Data Engineering (2017)
41. Ltd, C.V.I.T.: Choice Routing. `http://www.camvit.com` (2005)
42. Luo, Z., Li, L., Zhang, M., Hua, W., Xu, Y., Zhou, X.: Diversified top-k route planning in road network. Proceedings of the VLDB Endowment **15**(11), 3199–3212 (2022)
43. Luxen, D., Schieferdecker, D.: Candidate sets for alternative routes in road networks. In: International Symposium on Experimental Algorithms, pp. 260–270. Springer (2012)
44. Moghanni, A., Pascoal, M., Godinho, M.T.: Finding shortest and dissimilar paths. International Transactions in Operational Research **29**(3), 1573–1601 (2022)
45. Nettleton, D.F.: Data mining of social networks represented as graphs. Computer Science Review **7**, 1–34 (2013)
46. Oh, S., Leong, H.W.: Edge n-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths. In: Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA, pp. 64–72. AAAI Press (2017)
47. Ouyang, D., Yuan, L., Qin, L., Chang, L., Zhang, Y., Lin, X.: Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. Proceedings of the VLDB Endowment (2020)
48. Paraskevopoulos, A., Zaroliagis, C.D.: Improved alternative route planning. In: 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2013, September 5, 2013, Sophia Antipolis, France, pp. 108–122 (2013)
49. Peng, M., Zeng, G., Sun, Z., Huang, J., Wang, H., Tian, G.: Personalized app recommendation based on app permissions. World Wide Web **21**(1), 89–104 (2018)
50. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: European Symposium on Algorithms, pp. 568–579. Springer (2005)
51. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: International Workshop on Experimental and Efficient Algorithms, pp. 66–79. Springer (2007)
52. Shaikh, S., Rathi, S., Janrao, P.: Recommendation system in e-commerce websites: a graph based approached. In: 2017 IEEE 7th International Advance Computing Conference (IACC), pp. 931–934. IEEE (2017)
53. Shen, B., Cheema, M.A., Harabor, D., Stuckey, P.J.: Euclidean pathfinding with compressed path databases. In: IJCAI, pp. 4229–4235 (2020)
54. Shen, B., Cheema, M.A., Harabor, D.D., Stuckey, P.J.: Fast optimal and bounded suboptimal euclidean pathfinding. Artificial Intelligence p. 103624 (2021)
55. Strasser, B., Botea, A., Harabor, D.: Compressing optimal paths with run length encoding. Journal of Artificial Intelligence Research **54**, 593–629 (2015)
56. Strasser, B., Harabor, D., Botea, A.: Fast first-move queries through run-length encoding. In: Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014. AAAI Press (2014)

57. Sturtevant, N.R.: Benchmarks for grid-based pathfinding. IEEE Transactions on Computational Intelligence and AI in Games **4**(2), 144–148 (2012)
58. Tao, Y., Sheng, C., Pei, J.: On k-skip shortest paths. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 421–432 (2011)
59. Von Ferber, C., Holovatch, T., Holovatch, Y., Palchykov, V.: Public transport networks: empirical analysis and modeling. The European Physical Journal B **68**, 261–275 (2009)
60. Yen, J.Y.: Finding the k shortest loopless paths in a network. management Science **17**(11), 712–716 (1971)
61. You, M., Yin, J., Wang, H., Cao, J., Wang, K., Miao, Y., Bertino, E.: A knowledge graph empowered online learning framework for access control decision-making. World Wide Web pp. 1–22 (2022)
62. Yu, Y., Wang, C., Zhang, L., Gao, R., Wang, H.: Geographical proximity boosted recommendation algorithms for real estate. In: WISE (2), *Lecture Notes in Computer Science*, vol. 11234, pp. 51–66. Springer (2018)