PROFICIENT PATH PLANNING BY CACHING (PPC)

G.ASWANI [1], Mr. P. VISWANATHA REDDY [2]

[1] *PG Scholar, Dep of CSE, Sir Vishveshwaraiah Institute of Science & Technology, Madanapalle, AP, India.*

[2] *Assistant Professor, Dep of CSE, Sir Vishveshwaraiah Institute of Science & Technology, Madanapalle, AP, India.*

ABSTRACT

Owing to the wide availability of the global positioning system (GPS) and digital mapping of roads, road network navigation services have become a basic application on many mobile devices. Path planning, a fundamental function of road network navigation services, finds a route between the specified start location and destination. The efficiency of this path planning function is critical for mobile users on roads due to various dynamic scenarios, such as a sudden change in driving direction, unexpected traffic conditions, lost or unstable GPS signals, and so on. In these scenarios, the path planning service needs to be delivered in a timely fashion. In this paper, we propose a system, namely, Path Planning by Caching (PPC), to answer a new path planning query in real time by efficiently caching and reusing historical queried-paths. Unlike the conventional cache-based path planning systems, where a queried-path in cache is used only when it matches perfectly with the new query, PPC leverages the partially matched queries to answer part(s) of the new query. As a result, the server only needs to compute the unmatched path segments, thus significantly reducing the overall system workload. Comprehensive experimentation on a real road network database shows that our system outperforms the state-of-the-art path planning techniques by reducing 32 percent of the computation latency on average.

Keyword : - *Spatial database, path planning, cache*

1. INTRODUCTION

WITH the advance of the global positioning system (GPS) and the popularity of mobile devices, we have witnessed a migration of the conventional Internet-based on-line navigation services (e.g., Mapquest) onto mobile platforms (e.g., Google Map). In mobile navigation services, on-road path planning is a basic function that finds a route between a queried start location and a destination. While on roads, a path planning query may be issued due to dynamic factors in various scenarios, such as a sudden change in driving direction, unexpected traffic conditions, or lost of GPS signals. In these scenarios, path planning needs to be delivered in a timely fashion. The requirement of timeliness is even more challenging when an overwhelming number of path planning queries is submitted to the server, e.g., during peak hours. As the response time is critical to user satisfaction with personal navigation services, it is a mandate for the server to efficiently handle the heavy workload of path planning requests. To meet this need, we propose a system, namely, Path Planning by Caching (PPC), that aims to answer a new path planning query efficiently by caching and reusing historically queried paths (queried-paths in short). Unlike conventional cache-based path planning systems where a cached query is returned only when it matches completely with a new query, PPC leverages partially matched queried-paths in cache to answer part(s) of the new query. As a result, the server only needs to compute the unmatched path segments, thus significantly

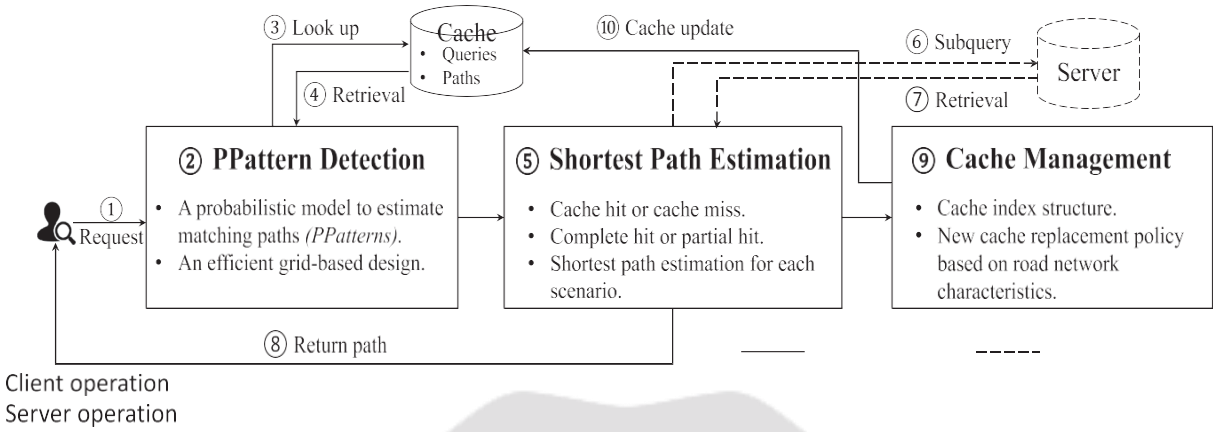


Fig. 1. Overview of the PPC system.

reducing the overall system workload. Fig. 1 provides an overview of the proposed PPC system framework, which consists of three main components (in rectangular boxes, respectively): (i) PPattern Detection, (ii) Shortest Path Estimation, and (iii) Cache Management. Given a path planning query (see Step (1)), which contains a source location and a destination location, PPC firstly determines and retrieves a number of historical paths in cache, called PPatterns, that may match this new query with high probability (see Steps (2)-(4)).1 The idea of PPatterns is based on an observation that similar starting and destination nodes of two queries may result in similar shortest paths (known as the path coherence property [1]). In the component PPatern Detection, we propose a novel probabilistic model to estimate the likelihood for a cached queried-path to be useful for answering the new query by exploring their geospatial characteristics. To facilitate quick detection of PPatterns, instead of exhaustively scanning all the queriedpaths in cache, we design a grid-based index for the PPattern Detection module. Based on these detected PPatterns, the Shortest Path Estimation module (see Steps (5)-(8)) constructs candidate paths for the new query and chooses the best (shortest) one. In this component, if a PPattern perfectly matches the query, we immediately return it to the user; otherwise, the server is asked to compute the unmatched path segments between the PPattern and the query (see Steps (6)-(7)). Because the unmatched segments are usually only a smaller part of the original query, the server only pro-cesses a "smaller subquery", with a reduced workload. Once we return the estimated path to the user, the Cache Manage-ment module is triggered to determine which queried-paths in cache should be evicted if the cache is full. An important part of this module is a new cache replacement policy which takes into account the unique characteristics of road net-works. Through an empirical study, we find that common road segments in various queried-paths usually have road types of higher importance and capacity [2], [3].2 This inspires us to define a usability value for each path by con-sidering both of its road type and historical frequency of use.
The main contributions made in this work are summa-rized as follows:

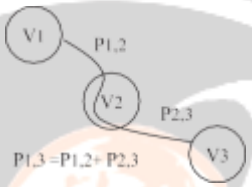
We propose an innovative system, namely, path planning by caching, to efficiently answer a new path planning query by using cached paths to avoid undergoing a time-consuming shortest path computation. On average, we save up to 32 percent of time in comparison with a conventional path planning system (without using cache). _ We introduce the notion of PPattern, i.e., a cached path which shares segments with other paths. PPC supports partial hits between PPatterns and a new query. Our experiments indicate that partial hits constitute up to 92.14 percent of all cache hits on average. _ A novel probabilistic model is proposed to detect the cached paths that are of high probability to be a PPattern for the new query based on the coherency property of the road networks. Our experiments indicate that these PPatterns save retrieval of path nodes by 31.69 percent on average, representing a 10-fold improvement over the 3.04 percent saving achieved by a complete hit. _ We have developed a new cache replacement mechanism by considering the user preference among roads of various types. A usability measure is assigned for each query by addressing both the road type and query popularity. The experimental results show that our new cache replacement policy increases the overall cache hit ratio by 25.02 percent over the state-of-the-art cache replacement policies. The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 analyzes the problem and introduces the notion of PPattern. Section 4 proposes a probabilistic model to detect PPatterns with an efficient detection algorithm. Section 5 discusses how these PPatterns are used to estimate the path. Section 6 details cache management issues and the proposed cache replacement policy. Section 7 reports the evaluation result on a real data set. Finally, Section 8 concludes this paper.

2 RELATED WORK

In this section, we review the related works in the research lines of path planning, shortest path caching and cache management, which are highly relevant to our study.

2.1 Path Planning

The Dijkstra algorithm [4], [5] has been widely used for path planning [6] by computing the shortest distance between two points on a road network. Many algorithms such as A_ [7], ATL [8] have been established to improve its performance by exploring geographical constraints as heuristics. Gutman [9] propose a reach-based approach for computing the shortest paths. An improved version [10] adds shortcut arcs to reduce vertices from being visited and uses partial trees to reduce the preprocessing time. This work further combines the benefits of the reach-based and ATL approaches to reduce the number of vertex visits and the search space. The experiment shows that the hybrid approach provides a superior result in terms of reducing query processing time. Jung and Pramanik [11] propose the HiTi graph model to structure a large road network model. HiTi aims to reduce the search space for the shortest path computation.



Xs,t	Es,t
$X_{1,2} = 2$	$E_{1,2} = 20.3$
$X_{2,3} = 5$	$E_{2,3} = 20.3$
$X_{1,3} = 5$	$E_{1,3} = 40.6$

While HiTi achieves high performance on road weight updates and reduces storage overheads, it incurs higher computation costs when computing the shortest paths than the HEPV and the Hub Indexing methods [12], [13], [14]. To compute time-dependent fast paths, Demiryurek et al. [15] propose the B-TDFP algorithm by leveraging backward searches to reduce the search space. It adopts an area-level partition scheme which utilizes a road hierarchy to balance each area. However, a user may prefer a route with better driving experience to the shortest path. Thus, Gonzalez et al. propose an adaptive fast path algorithm which utilizes speed and driving patterns to improve the quality of routes [16]. The algorithm uses a road hierarchical partition and pre-computation to improve the performance of the route computation. The small road upgrade is a novel approach to improving the quality of the route computation.

2.2 Shortest Path Caching

Thomsen et al. [17] propose an efficient shortest path cache (SPC). Based on the optimal subpath property [18], given a source s and a destination t, the shortest path $p_{s;t}$ contains the shortest path $p_{k;j}$, where s _ k; j _ t, SPC computes a benefit value to score a shortest path to determine whether to preserve it in the cache. The benefit of a path is a summation of the benefit value of each sub-path in the shortest path. The formula of a benefit value considers two features: the popularity of a path and its expense. The popularity of a path p is evaluated based on the number of occurrences of the historical sub paths which overlap p. On the other hand, the expense of a path represents the computational time of the shortest path algorithm. Fig. 2 shows an example to illustrate how to calculate the benefit value for a path using SPC. In this example, a path $p_{1;3}$ contains three subpaths $p_{1;2}$, $p_{2;3}$, and $p_{1;3}$. The popularity and expense values for each path from node s to node t are listed in the table, denoted as $X_{s;t}$ and $E_{s;t}$, separately. Accordingly, the benefit value for path $p_{1;3}$ is calculated as 2 _ 20:3 þ 5 _ 20:3 þ 5_ 40:6 ¼ 345:1 using SPC. Because SPC has to score each sub-path in the shortest path, the time complexity is high. Assuming that a shortest path contains n nodes, a shortest path contains ðn _ ðn_ 1ÞÞ=2 sub-paths. The time complexity for scoring a shortest path is $Oðn_2Þ$. In the above study, each query is answered independently. However, when queries in the current request pool share similar properties, they may be processed as a group. Thus, Mahmud et al. [1] propose a groupbased approach to accelerate the processing by calculating the similarity among a group of queries and send the common part as a query to the server. Therefore, only dissimilar segments for each query are answered by the server individually. However, this work does not explore any cache mechanism in the system.

2.3 Cache Management

Caching techniques have been employed to alleviate the workload of web searches. Since cache size is limited, cache

replacement policies have been a subject of research. The cache replacement policy aims to improve the hit ratio and reduce access latency. Markatos et al. conduct experiments to analyze classical cache replacement approaches on real query logs from the EXCITE search engine [19]. Three important observations are described as follows. First, a small number of queries are frequently re-used. By preserving results of these queries in cache, the system is able to respond to the users without incurring time-consuming computations. Second, while a larger cache size implies a higher hit ratio, significant overheads may be incurred for cache maintenance. Third, static cache replacement has better performance when the cache size is small, and vice versa for dynamic cache replacement. Static cache replacement [19], [20], [21], [22], [23], [24] aims to preserve the results of the most popular and frequent queries, thus incurring a very low workload during query processing. However, the cache content may not be up to date to respond to recent trends in issued queries. Dynamic cache replacement [19], [25], [26], in contrast to a static cache, preserves the results of the most recent queries, but the system incurs an extra workload. In order to improve the retrieval efficiency of the path planning system, Thomsen et al. propose a new cache management policy [17] to cache the results of frequent queries for reuse in the future. To enhance the hit ratio, a benefit value function is used to score the paths from the query logs. Consequently, the hit ratio is increased, hence reducing the execution times. However, the cost of constructing a cache is high, since the system must calculate the benefit values for all sub-paths in a full-path of query results. For on-line, map-based applications, processing a large number of simultaneous path queries is an important issue. In this paper, we provide a new framework for reusing the previously cached query results as well as an effective algorithm for improving the query evaluation on the server.

3 PRELIMINARIES

3.1 Symbols and Definitions

In this section, we first introduce the notations of symbols used throughout the paper. Note that some symbols may be used as prefixes further explained when used.

Definition 1 (Road network). A road network is represented as a directed graph G ¼ ðV;EÞ, where V ¼ fv1; v2; . . . ; vng is a set of nodes denoting road intersections and terminal nodes,and E is a set of edges denoting road segments connecting two nodes in V .

Definition 2 (Path, source node, destination node). A path $p_{s;t}$ is a sequence of nodes fv1; v2; . . . ; vng. The first node is named as the source (node) v_s ¼ v1 and the last node is named as the destination (node) v_t ¼ vn.

Definition 3 (Path planning query). A path planning query $q_{s;t}$, is specified by a source node v_s and a destination node v_t. The system answers the query by returning a path from v_s to v_t that satisfies the query criteria.

By default, an important criteria for a path planning query is the distance, i.e., the minimal distance path is returned. However, due to the requirement of timeliness, the performance of a path planning service is evaluated in terms of both distance and response time, i.e., the path computed by the server is acceptable if it is returned within a tolerable time, with a tolerab tolerable distance deviate from the true shortest distance path.

Definition 4 (Cache). A cache C contains a collection of paths. The cache size jCj is measured as the total number of nodes in the cached paths. Note that jCj < b, where b is the maximal cache size.

Definition 5 (Query logs). A query log is a collection of time stamped queries issued by users in the past.

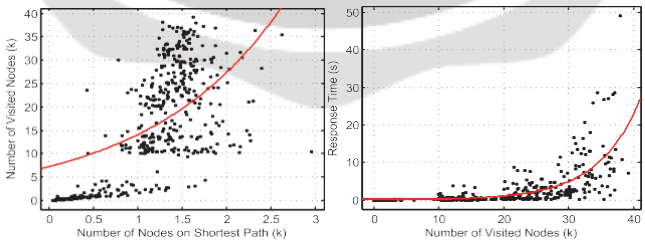


Fig. 3. The relationships between (a) shortest path distance and number of visited nodes, and (b) number of nodes and computational cost (response time).

3.2 Problem Analysis

The main goal in this work is to reduce the server workload by leveraging the queried-paths in cache to answer a new path planning query. An intuitive solution is to check whether there exists a cached queried-path perfectly matching the new query. Here, a perfect match means that the source and destination nodes of the new query are the same as that of a queried-path in cache. Such a match is com monly known as a cache hit; otherwise, it is referred to as a cache miss. When a cache hit occurs, the system retrieves the corresponding path directly from the cache and returns it to the user. As a result, the server workload is signifificantly reduced. However, when there exists a cache miss, the sys tem needs to request the server to compute the path for the new query. In this paper, we aim to reduce the server work load when a conventional cache miss occurs by fully leveraging the cached results. We fifirstly analyze the potential factors influencing the computational cost at the server, which can be measured by the total number of nodes a routing algorithm visits [17]. We randomly generate 500 queries, compute the shortest paths using the Aalgorithm upon a real road network dataset, and display (1) the relationship between the total number of nodes A visits and the total number of nodes on the shortest path (as illustrated in
. These two figures show that the computational cost of A increases exponentially as the length of the shortest path increases. Additionally, we observe that the paths of two queries are very likely to share segments. Thus, we argue that if one is cached, the shared segments can be reused for the other query such that only the unshared route segments need to be computed by the server. As the unshared segment is shorter than the original path, the computational overhead may be significantly reduced (as indicated in Fig. 3). Based on the observation, we introduce below some new notions to be used in this work.

Fig. 4. Shortest path computation, $p_{s;t}$ is a 2-PPattern for path p_s0 ;$t0$ as they share a segment $p_{a;b}$ with each other.

Defifinition 6 (PPattern, Head, Tail). Given a path, Path Pattern (PPattern for short) is another path which shares at least two consecutive nodes. The fifirst and last node of their shared segments is named as head and tail, respectively. Specififically, if the PPattern has k consecutive nodes where k 2, it is termed as k-PPattern.

4.PPATTERN DETECTION

To detect the best PPatterns, an idea is to calculate the estimation distance based on each cached path by Eq. (5), and select the cached path with the shortest distance. However, it faces several challenges. Firstly, the distance estimation in Eq. (3) requires the server to compute the unshared segments (i.e., SDPðs0; aÞ; SDPðb; t0Þ). Therefore, it incurs significant computation to exhaustively examine all cached paths. Secondly, such an exhaustive operation implicitly assumes that each cached path is a PPattern candidate to the query. However, this is not always true. For example, a path in Manhattan does not contribute to a query in London. While we assume that users may accept an approximate path if its error is within a certain tolerable range, exhaustive inspection cannot not be sure that the path with the minimal error is found until all paths are inspected. To address these challenges, we aim to narrow down the inspection scope to only good candidates.

4.1 Probabilistic Model for PPattern Detection

The coherency property of road networks indicates that two paths are very likely to share segments while source nodes (and their destination nodes, respectively) are close to each other [27]. This property has been used in many applications for various purposes, e.g., efficient trajectory lookups as the common segments among multiple paths are queried only once [1]. Notice that this property is mainly attributed to the locality of the path source and destination nodes. We argue that, for two queries, if they satisfy certain spatial constraints, their shortest distance paths are very likely to be the PPatterns to the other. Several existing studies have proposed algorithms to group paths with similar trajectories together [27]. In these studies, paths within a cluster can be taken as the PPatterns to each other. Given a new query, the system checks whether it fits into an existing cluster and directly returns the shortest path if there exists at least one path in that cluster. However, all these studies require a complete knowledge graph computed from the basic road network data, which incurs a heavy workload and distracts from our goal. Differing from the existing studies, we propose a method to

detect the potential PPatterns for an input query using only existing paths in cache. In summary, the coherency property indicates that two queries are more likely to share sub-paths if they meet thefollowing three spatial constraints concurrently: (1) the source nodes of the two queries are close to each other; (2) the destination nodes of the two queries are close to each other; and (3) the source node is distant from the destination node for both queries. Formally, we denote pðqs0;t0; qs;tÞ as the probability for two queries qs0;t0 and qs;t to be PPatterns of each other. Thus, constraints (1) and (2) indicate pðqs0;t0; qs;tÞ / 1
Dðs;s0Þ
and
pðqs0;t0; qs;tÞ / 1

Dðt;t0Þ
respectively. Onthe other hand, constraint
(3) implies pðqs0;t0; qs;tÞ / Dðs0;
t0Þ _ Dðs; tÞ. Thereafter, the final probability can be computed as the product of these three terms. As we would like the three factors to achieve sufficient satisfaction concurrently,the probability will only be computed if each factor is over a threshold as expressed by Eq. (7):
pðqs0;t0; qs;tÞ ¼
Y
x2fs;t;lg
wx uðfxðqs0;t0; qs;tÞ _ DxÞ; (7)
where fsðqs0;t0; qs;tÞ ¼ 1

Dðs;s0Þ

, ftðqs0;t0; qs;tÞ ¼ 1

Dðt;t0Þ

and flðqs0;t0 ;

qs;tÞ ¼ Dðs0; t0Þ _ Dðs; tÞ
respectively indicate the sourcesource, destination-destination and source-destination node distance factors between the two queries of qs0;t0 and qs;t; wx is a weight indicating the contribution from each factor fx; Dx is a threshold controlling the validation scope for fx; and uðÞ is a shifted unit step function: uðx _ DxÞ ¼
1 : x _ Dx
0 : x < Dx:

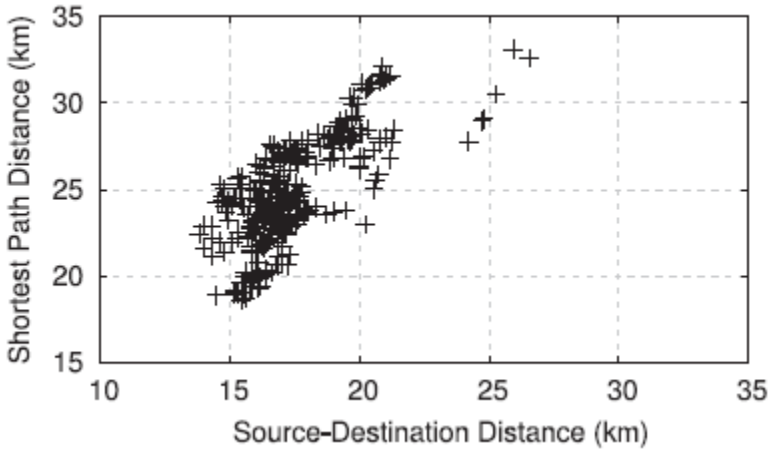


Fig. 5. Correlation between source-destination distance and the shortest path distance.

$$PT(q_{s',t'}, q_{s,t}) = \begin{cases} 1: & \mathbf{p}(q_{s',t'}, q_{s,t}) > \epsilon \\ 0: & otherwise, \end{cases}$$

where _ is a probability threshold. Note that in the above, all distances are Euclidean distances. Based on the above, we select the highest ranked PPatterns for a new query. In the following, we prove that the estimation error is upper-bounded

to a value 2aðDs þ DtÞ where Ds and Dt are source-source and destination- destination distance thresholds and a is a parameter approximating the relation between the shortest path distance of two points on a road network and their true Euclideandistance, i.e., SPDða; bÞ ¼ aDða; bÞ. Factor a can be estimated through an empirical study on the road network. For example, Fig. 5 shows the correlation between the shortest path distance and the Euclidean distance of endpoints for 5k queries on a real road network database. To validate the estimation bound, Fig. 6 summarizes three scenarios where a new query $q_{s;t}$ is estimated from a pattern candidate $q_{s0;t0}$. In the first scenario (see Fig. 6a), there exists at least one common segment between the paths of the two queries. In the other two scenarios (shown in Figs. 6b and 6c), there exist no common segments, but the two queries are similar to each other. We calculate the maximal estimation error for each scenario as follows. Proof. According to Eq. (4) and the assumption SPDða; bÞ ¼ aDða; bÞ, the estimation distance error is represented as:

$$e = \alpha(D(s',s) + D(t,t') + D(s,t) - D(s',t')).$$

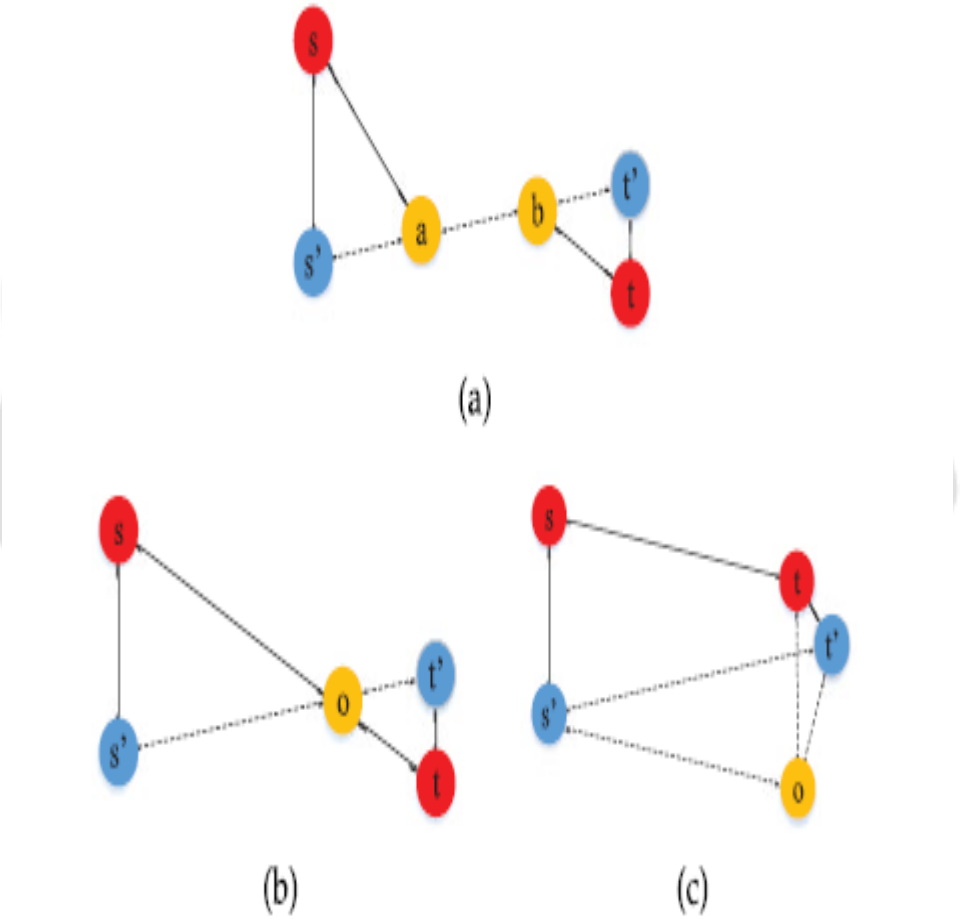


Fig. 6. Three scenarios for error estimation of $q(s', t')$ from $p(s, t)$.

In Scenario 6(b),

$$\begin{aligned} e &= \alpha(D(s', s) + D(t, t') + D(s, o) + D(o, t) \\ &\quad - D(s', o) - D(o, t')) \\ &\leq \alpha(D(s', s) + D(t, t') + D(s, s') + D(t, t')). \end{aligned} \tag{13}$$

In Scenario 6(c), assume $D(s, s') = D(t, o)$ and $D(s, t) = D(s', o)$, so Eq. (10) is written as:

$$\begin{aligned} e &= \alpha(D(s', s) + D(t, t') + D(s', o) - D(s', t')) \\ &\leq \alpha(D(s', s) + D(t, t') + D(o, t')) \\ &\leq \alpha(D(s', s) + D(t, t') + D(t, o) + D(t, t') \\ &= \alpha(D(s', s) + D(t, t') + D(s, s') + D(t, t'). \end{aligned} \tag{14}$$

From Eqs. (12), (13), (14), the maximal error is

$$e \leq 2\alpha(D(s', s) + D(t, t')).$$

As we require the endpoint nodes of two paths to be close within distance D_x, i.e., $D(x', x) < D_x$, we have

$$e \leq 2\alpha(D_s + D_t).$$

In other words, the error is upper bounded by $2\alpha(D_s + D_t)$. □

4.2 An Efficient Grid-Based Solution

In order to retrieve these patterns efficiently, we propose a grid-based solution to further improve the system performance. The main idea is to divide the whole space into equally sized grid cells, where the endpoints of all paths are mapped to the grid cells. As such, the grid index facilitates efficient cache lookups [28], [29]. The distance measures can be approximated by counting the total number of covered grids. Therefore, Eq. (9) is transformed as follows:

$$PT(q_{s',t'}, q_{s,t}) = \begin{cases} 1: & g(s) = g(s'), g(t) = g(t') \\ & D(s, t) \leq D_l \\ 0: & otherwise, \end{cases}$$

5.Cache with Different Mechanisms

Performance comparison. We further compare the performance of our system (PPC) with three other cachesupported systems (LRU, LFU, SPC_) which adopt various cache replacement policies or cache lookup policies. The first two algorithms detect conventional (complete) cache hits when a new query is inserted, but update the cache contents using either the Latest Recent Used algorithm (denoted as LRU) or the Least-Frequently Used replacement policies

(LFU), respectively. The third compared algorithm, namely, the shortest-path-cache (SPC_), is a state-of-the-art cachesupported system specifically designed for path planning as PPC is. SPC_ also detects if any historical queries in the cache match the new query perfectly, but it considers all sub-paths in historical query paths as historical queries as well. We compare these four cache mechanisms by converting the two metrics, number of visited nodes and response time, to saving ratios against non-cache system for better presentation d_{node} ¼ $nodes_{cache}$=$nodes_{noncache}$ _ 100%;
(17) d_{time} ¼ $time_{cache}$=$time_{noncache}$ _ 100%: (18) Visited node saving ratio and Query time saving ratio indicate
how many nodes and how much time an algorithm can save from a non-cache routing algorithm (e.g., A_), respectively. A larger value indicates better performance. In the experiment, we increase the total query number from 1k to 5k and calculate the above two metrics using each cache mechanism with the results shown in Fig. 10. The x-axis represents the total number of queries while the y-axis indicates the metric values in percentages. From these figures, we can see clearly that our cache policy always achieves the best performance among all measurements. On average, LFU, LRU, SPC* and PPC visit 30.47 26.86 27.78 and 34.73 percent fewer nodes than A* algorithm, and reduces the computational time from A* algorithm by 29.83, 26.32, 27.04 and 32.09 percent, respectively

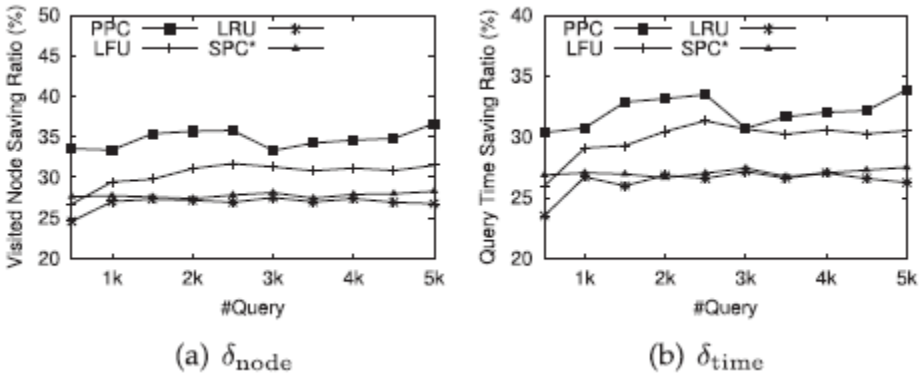


FIG 5:Performance comparison with four cache mechanisms in terms of (a) visited node saving ratio, and (b) query time saving ratio with different numbers of queries.

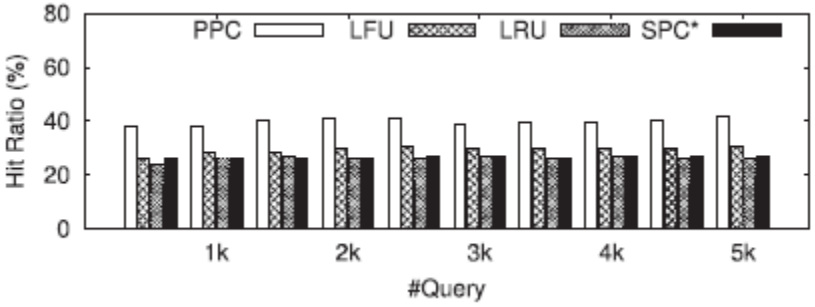


FIG: Performance comparison with four cache replacement mechanisms in terms of hit ratio.

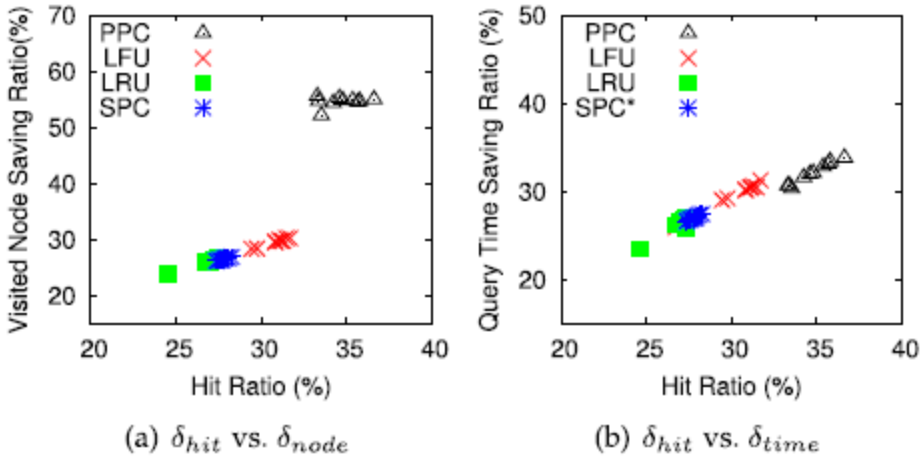


Fig: Correlation between (a) hit ratio and visited node saving ratio, and (b) hit ratio and query time saving ratio.

network graph, i.e., PPC does not always save sub paths if they require complex computations. Because PPC leverages both complete and partial hit queries to answer a new query, we additionally measure the saving node ratio for partial hits. We ran an experiment with 5k queries, and have plotted the results in Fig.. From the figure we can see that partial hits appear evenly along the temporal dimension. On average it achieves a 97.63 percent saving ratio, which is quite close to the complete hit saving ratio (100 percent). Among all cache hits, the take-up percentages of complete and partial hits are illustrated in Fig. 14a and their saving node percentage is shown in Fig. b. The x-axis is the query size and the y-axis is the percentage values. Notice that partial hit does not achieve a 100 percent saving node ratio. However, as partial hits occur much more frequently than complete hits, its overall benefit to the system performance outweighs that of the complete hits. On average, partial hits take up to 92.14 percent of the whole cache hit. The average saving node ratio is 31.67 percent by partial hits, 10 times as much as that from complete hits at 3.04 percent.

5.1 Cache Construction Time

Because both SPC* and PPC are cache-supported systems mulated, so the cache is constructed periodically. PPC is designed as a dynamic cache, i.e., the cache is updated whenever a new query is inserted, i.e., the cache is constructed gradually over time. Therefore, for a fair comparison, we apply our algorithm to a batch of consecutively inserted queries and calculate their total cache update time (Note that the routing time has been removed for both

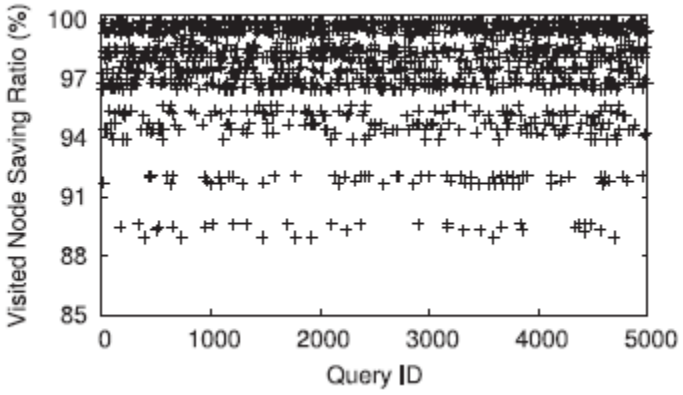


FIG: Visited node saving ratio distribution among partial hit queries with #Q ¼ 5K.

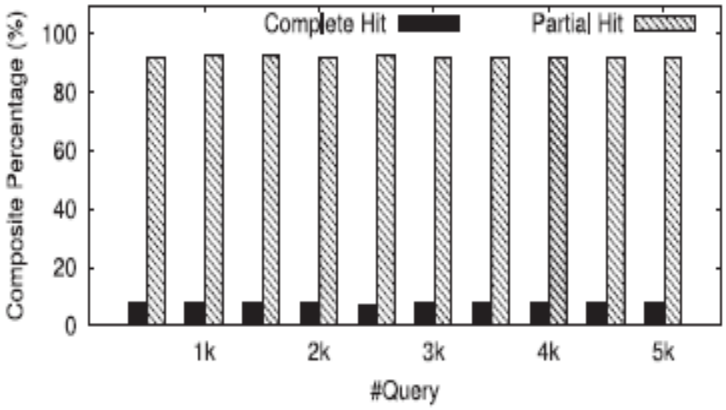


Fig :Comparison between partial hits and complete hits using PPC with different query sets in terms of (a) hit ratio and (b) visited node saving ratio.

systems). The comparison result is illustrated in Table 6. From the statistics in this table, we can see that our algorithm significantly reduces the construction time to 0.01 percent of SPC_ on average. Such significant improvement may be due to the following reason. Let the total size of the log files be n nodes. The time complexity for computing usability values for the paths in the cache is OðnÞ. However, SPC* needs to compute the usability values for all possible paths, resulting in a time complexity of Oðn2Þ.

5.2 Querylog Distributions

Previous experiments are conducted on queries generated with a normal distribution. However, in a realistic scenarios, the queries may appear in different distribution. To investigate whether our algorithm can robustly achieve satisfactory results under different distributions, we generate three more query sets by uniform, directional and central distributions (denoted as $QL_{uniform}$, $QL_{central}$ and $QL_{directional}$, respectively). Under a uniform distribution, each query on the road network appears with equal probability. Central distribution appears when there exists a large-scale event. Directional distribution is used to formulate the driving behavior in which a driver may continuously change directions. Experiments are carried out to measure both saving node ratio and saving response time ratio by the LFU, LRU, SPC*, and PPC algorithms.

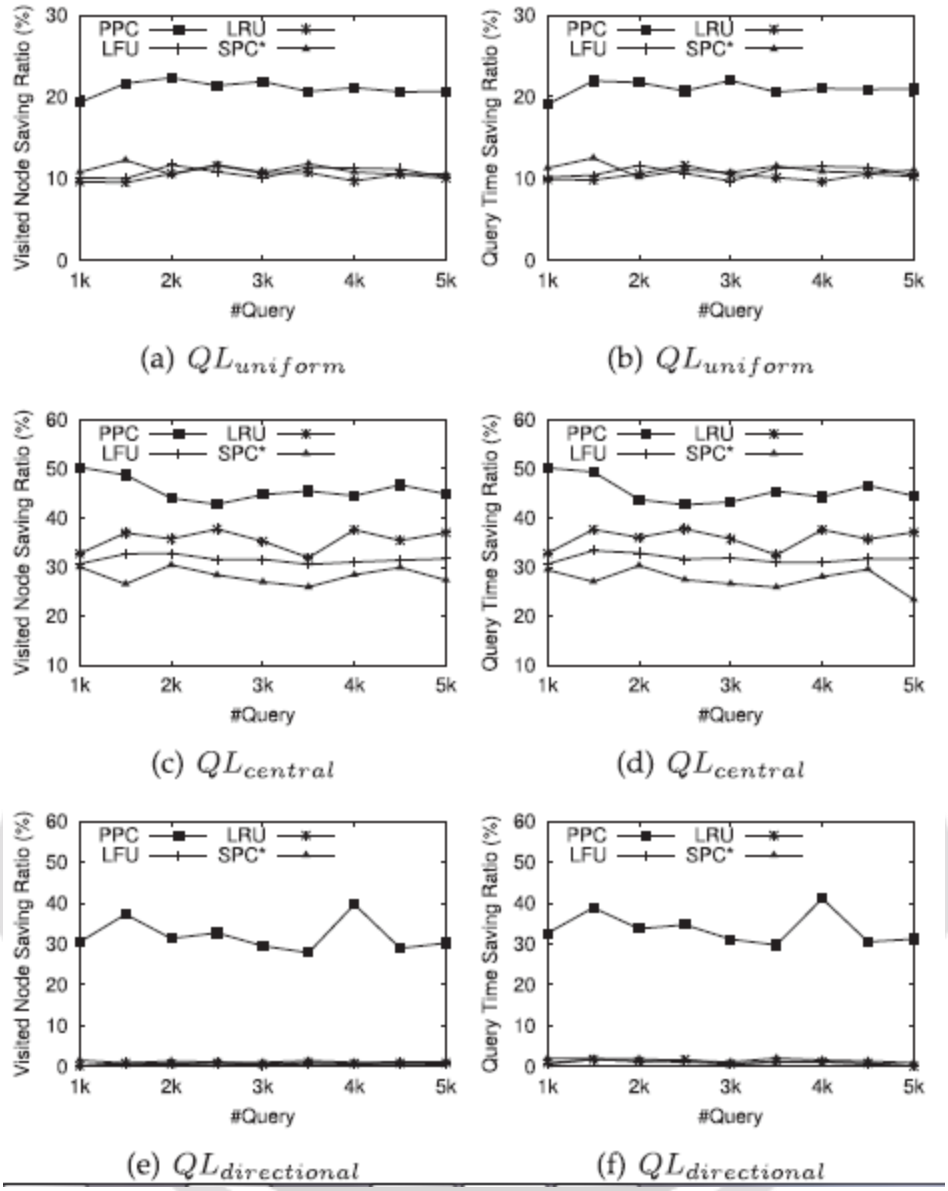


FIG 5.2: Performance comparison among four cache mechanismswith various data distributions for query logs in terms of (a,c,e) visited node saving ratio and (b,d,f) query time saving ratio with different numbers of queries.

5.3 :Effect of Source-Destination Minimal Length

The minimal source and destination node distance (i.e., D¼ js0; t0j) in the PPatterns detection algorithm (Algorithm 1) is another tunable factor. Fig. 18 illustrates the system performance with different distance values. By increasing this distance threshold, the deviation percentage reduces as expected because the coherency property indicates that queries are more likely to share subpaths when the source node is distant from the end node. However, the visited node saving ratio and query time saving ratio decrease because it takes more space to store a longer path.

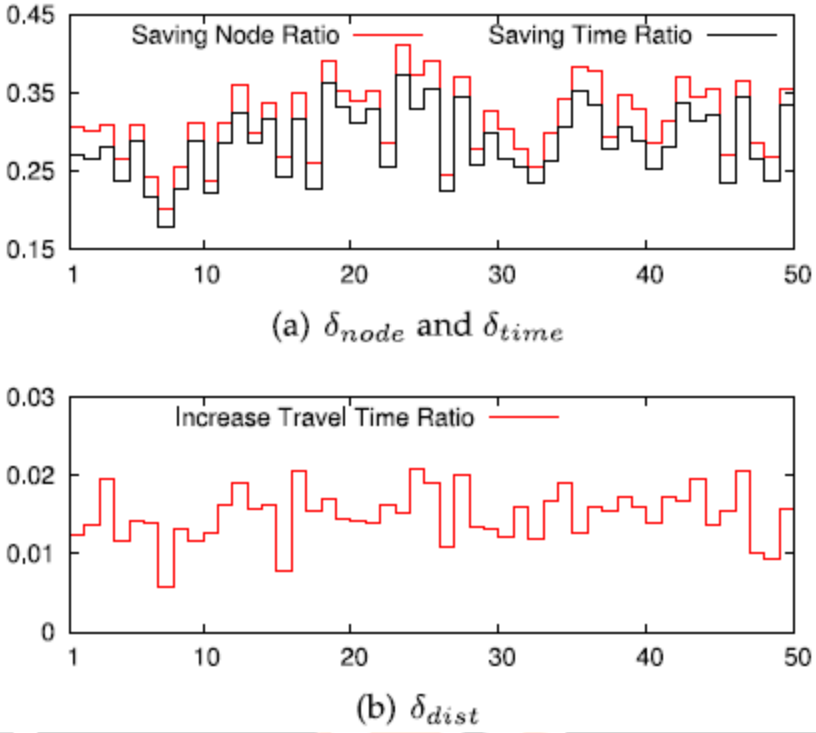


FIG: PPC performance analysis: effect of temporal factor. (a) Visited node saving ratio and query time saving ratio, and (b) average deviation percentage.

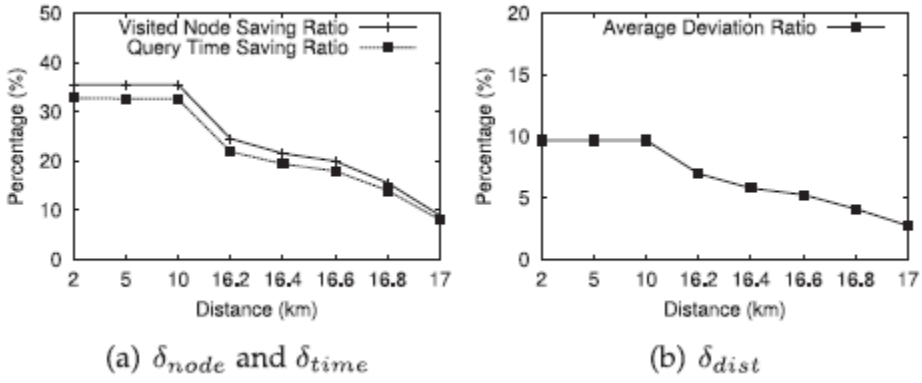


FIG: PPC performance analysis: effect of minimal source-destination distance. (a) Visited node saving ratio and query time saving ratio, and (b) average deviation percentage.

4. CONCLUSIONS

In this paper, we propose a system, namely, Path Planning by Caching, to answer a new path planning query with rapid response by efficiently caching and reusing the historical queried-paths. Unlike the conventional cache-based path planning systems, where a queried-path in cache is used only when it matches perfectly with the new query, PPC leverages the partially matched cached queries to answer part(s) of a new query. As a result, the server only needs to compute the unmatched segments, thus significantly reducing the overall system workload. Comprehensive experimentation on a real road network database shows that our system outperforms the state-of-the-art path planning techniques by reducing 32 percent of the computational latency on average.

5. REFERENCES

[1]. G. Aswani is currently a PG scholar of M.Tech (CSE) in Sir Vishveshwaraiah Institute of Science & Technology, Madanapalle, A.P, affiliated to JNTU Anantapur.
[2]. Mr. P.Viswanatha Reddy working as Assistant Professor, department of CSE in Sir Vishveshwaraiah Institute of Science & Technology, Madanapalle, A.P, affiliated to JNTU Anantapur.