iJRASET
International Journal For Research in
Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 Issue: II Month of publication: February 2018

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue II, February 2018- Available at www.ijraset.com

Path Planning on Road Network by Caching

Saikam Veera Venkata Bala Suryavamsi[1]
[1]Department of computer science and systems engineering, Andhra University college of engineering, Andhra University

Abstract: In road network navigation services, path planning is a basic function, which finds a route between the given source and destination. In case of mobile users on road, they face various dynamic situations, like unstable GPS signals, abrupt change in driving direction and unexpected traffic conditions. In these scenarios, the path planning service needs to provide its outcome as early as possible. This paper proposes a system, namely, Cache Based Path Planning (CBPP), which provide result to new path planning query efficiently by fully utilizing previous historical queried-paths. CBPP utilizes the partially matched queries to answer sub parts of the new query. Hence, the server only needs to compute the paths for unmatched path segments of the query, thus reduces the overall workload of the system. This system decreases almostу30% of the computation cost on average.
Index Terms: Path Planning, Caching, Road network.

I. INTRODUCTION

Road network navigations services have become a basic application on mobile devices due to the wide availability of digital road mappings and global positioning system (GPS).In mobile navigation services, on-road path planning is a fundamental function, which finds a path between given source location and a destination location. In case of mobile users on road, they face various dynamic situations, like unstable GPS signals, abrupt change in driving direction and unexpected traffic conditions. To provide an efficient path planning system in these cases, this proposes a system namely, Cached Based Path Planning (CBPP), which provide result to new path planning query efficiently by using previous historical queried-paths. The proposed CBPP system architecture, which consists of three main components (i) CSubPath Detection, (ii) Building Shortest Path, and (iii) Cache Management.

Our work is summarized as follows:

1) We propose an system, namely, cache based path planning(CBPP), to effectively answer a new path planning query by using cached paths to avoid computing the entire query which is time-consuming computation. On average, we save up to 30% of time in comparison with a conventional path planning system (without using cache).
2) We introduce the notion of *CSubPath*, *i.e.*, a cached path which shares segments with other paths. CBPP supports partial hits between CSubPaths and a new query.
3) 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.

II. RELATEDWORK

In this section, we discuss about the related work in this area. Here mention five research work based on this research domain.

A. *Computing the Shortest Path: A∗ Search Meets Graph Theory:* This work proposed a shortest path computing method that uses A* search algorithm with a new graph-theoretic lower-bounding technique based on landmarks and the triangle inequality [2].
B. *Investigation of the *(star) Search Algorithms: Characteristics, Methods and Approaches*: In this work, a branch of search algorithms that are called *(star) algorithms are analyzed. They are A*, B*, D*, IDA* and SMA*. Features, basic concepts of each algorithm and the different approaches of each type are investigated separately.
C. *Shared execution of path queries on road networks: In* this work, Path queries that find the shortest path between a source and a destination of the user. In particular, they address the problem of finding the shortest paths for a large number of simultaneous path queries in road networks. Conventional systems that consider one query at a time are not suitable for many applications due to high computational and service costs. These systems cannot guarantee required response time in high load conditions. Here propose an efficient group based approach that provides a practical solution with reduced cost. The key concept of this approach is to group queries that share a common travel path and then compute the shortest path for the group [1].
D. *An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps:* This work proposes HiTi (Hierarchical MulTi) graph model for structuring large topographical road maps to speed up the minimum cost route

©IJRASET (UGC Approved Journal): All Rights are Reserved

computation. The HiTi graph model provides a novel approach to abstracting and structuring a topographical road map in a hierarchical fashion. Project propose a new shortest path algorithm named SPAH, which utilizes HiTi graph model of a topographical road map for its computation. Project gives the proof for the optimality of SPAH. Our performance analysis of SPAH on grid graphs showed that it significantly reduces the search space over existing methods. Project also present an in-depth experimental analysis of HiTi graph method by comparing it with other similar works on grid graphs. Within the HiTi graph framework, project also proposes a parallel shortest path algorithm named ISPAH. Experimental results show that inter query shortest path problem provides more opportunity for scalable parallelism than the intra query shortest path problem [9].

E. *Introduction to Algorithms:* This work provides, some of the best shortest path computing algorithms such as, Dijkstra's (solves the single-source shortest path problem), Bellman-Ford algorithm (solves the single-source shortest path problem if edge weights may be negative) and Floyd-Warshall algorithm (solves all pairs shortest paths) [8].

III. PRELIMINARIES

A. *Symbols and Definitions*

In this section, we introduced symbols and definition of the notations that are used in this paper.

1) *Definition 1. Road network:* A road network is represented here as a digraph G = (V,E), where V is the set of all vertices in G ($V=\{v_1,v_2,v_3, \dots , v_n\}$) which denotes the roads terminal or connecting sections, and E is a set of edges which denotes the road segments connecting two nodes in V.
2) *Definition 2. Path Finding Query Q:*A path finding query Q contains source s_q and destination d_q.
3) The system finds the path p form node s_q to d_q then returns the path that satisfies the query criteria
4) *Definition 3. Common Sub Path, CSubPath, CSPat:* Given a path planning query, the CSPath is an another path which shares at least two consecutive nodes. Means some amount path is matching with the new query that is asked.
5) *Definition 4. Cache:* A cache contains the collection of historic paths that are previously asked. The size of cache is measured as the total number of nodes in all the paths of cache.
6) *Definition 5. Complete hit:* When there is a path in cache that exactly matches with new path planning query then it is complete hit.
7) *Definition 6. Partial hit:* When there is a path p in cache that partially matched with new query, some consecutive nodes in p are part of the query. Means p is CSubPath, then it is Partial hit

TABLE I
SYMBOLS & EXPLANATIONS

Symbols	Explanations
D(s,d)	Method to calculate Euclidean distance between node s and d.
C	Represent Cache that stores historic queried paths
SDP(s,d)	Shortest Distance path between node s and d.
Ø.	Empty
μ	Denotes path utilization value
$p_{sq,dq}$	Path from node $s_q$to node d_q

B. *Problem Analysis*

The main aim of this work is to reduce work load of the server by maximum utilizing the queried paths in the cache in-order to answer the new query. A quick solution is check whether there exists a cached path that perfectly matching the new path planning query. Now, here the perfect match means, the source and destination nodes of both cached path and new query are same, which is called as cache hit, otherwise it is cache miss. So, when a cache hit occurs, the system retrieves the cached-path and return it to the user, which reduces the server work load of computing it again. However, when there is a cache miss, the general conventional cache based systems computes the path again, but in this work, unlike the conventional cache based system, we aim reduce the work load of the server when a cache miss occurs, by leveraging the cached paths, using heuristic methods.

©IJRASET (UGC Approved Journal): All Rights are Reserved

Fig. 1 Example illustration of Common Sub Path(CS Path).

In Fig. 1. $p_{s,t}$ is cached path and $p_{s',t'}$ Is new path planning query. They both share a common sub path, that is path from a to b. So, here the Shortest path from s' to t' can be computed as:

$$SDP(s',t') = SDP(s',a)\odot SDP(a,b)\odot SDP(b,t') \quad (1)$$

Now, from Eq. (1), we just need compute shortest path from nodes s' to a and b to t' only, because a to b is already computed and present in cache. We have provided a grid based heuristic method to retrieve this CSubPath and compute the shortest path. We have used A* [2] algorithm to compute shortest path between given two nodes.

IV. CSUBPATH DETECTION

The coherency property of road networks states that two paths are very likely to share sub-paths if they meet the following spatial constraints at same time: (1) the source nodes of two queries are closer to each other; and (2)the destination nodes of two queries are closer to each other; and (3) the source node is distant from the destination in both the queries [3]. Based on this coherency property, we proposed a grid based solution to retrieve common sub paths effectively. The main idea here is to divide the whole space into equally sized grid cells, such that every node belongs to any one the grid cell [4]. Algorithm 1 lists the pseudo code for detecting all CSPaths. First we check whether the Euclidean distance between the given source and destination nodes is at least D_t (Lines 1 to 3). If it is less than D_t then then distance between the source and destination is too small, so the server computes the shortest path directly and returns the path, because it may take longer time for cache lookups and estimation. After that, the target space in divided into equally sized grid cells by an cell size D_g (Line 4). Then, it locate the grid cells (index number of the cell) of source and destination nodes (Line 5).Then, it finds the grid edge which is closer to source, and make a list with source grid and the grid adjacent to the closest edge, similarly with destination also(Line 6 to 7). Then, the system retrieves all the paths in cache, those overlaps the source and destination gird cells (Lines 8 to 9) and add them in to List L (Line 10), a set containing the candidate CSubPaths. This process uses path identifier and grid index to reduce the cache lookup operations. The system returns the List of candidate CSubPaths (Lines 11 to 12).

A. *Algorithm 1 CSubPath Detection*

1) *Input :* Q: a query(which contains source and destination); D_t : Threshold of distance; D_g : cell size of a grid; C : cache.
2) *Output :*All common sub paths in cache : CSPaths
3) *STEP 1:* if(ED(Q.s,Q.d) <D_t) then
4) *STEP 2:* Return CSPaths= ∅.
5) *STEP 3:* end if
6) *STEP 4*: Divide the target space by size D_g.
7) *STEP 5:* Find the grid cell positions of source(G_s) and destination(G_d).
8) *STEP 6:* Find which edge of the grid(G_s), which is closer to source, then find the Grid that is adjacent to that edge, Let G_a. Then make a list {G_s, G_a}.
9) *STEP 7:* Find which edge of the grid(G_d), which is closer to destination, then find the Grid that is adjacent to that edge, Let G_b. Then make a list {G_d, G_b}.
10) *STEP 8:* L_S<- list cache paths, which pass through {G_s, G_a}.
11) *STEP 9*: L_d<- list cache paths, which pass through {G_d, G_b }.
12) *STEP 10:* L <- Intersection(L_s, L_p).
13) *STEP 11:*CSPaths<- Common sub paths from G_s to G_d for each query in L.
14) *STEP 12:* Return CSPaths.

V. BULDING SHORTEST PATH BASED ON CACHE

Based on the CSPaths detected above, we estimate the shortest path for new path planning query using Eq. (1). The CSPaths that are detected provide at least a part of the answer path thus increases the cache utilization. To build shortest path for CSPaths, we

©IJRASET (UGC Approved Journal): All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue II, February 2018- Available at www.ijraset.com

propose a heuristic algorithm as shown in Algorithm 2. If the list CSPaths is empty means no CSPaths exists, then system directly contact server to compute the shortest path and returns it immediately (Lines 1 to 3). If there exists a complete cache hit, the corresponding path is returned directly from cache (Lines 6 to 8). Otherwise, the system calculate the estimated distance from each CSPath in list CSP for the query and selects the one with minimum distance (Lines 9 to 19). The system further uses this sub path to build the shortest path (Lines 21 to 26). Finally, the system combines the newly calculated source-source and/or destination-destination shortest paths and cached segment as a complete path, which is finally returned to the user (Lines 27).

A. Algorithm 2 Building Shortest Path

1) *Input:* Source node s_q and Destination node d_q; all possible Common sub paths CSP; Cache C.
2) *Output:* Computed Shortest path($p_{sq,dq}$).
3) *STEP 1:* if (is Empty(CSP)) then
4) *STEP 2:*$p_{sq,dq}$<- Calculate shortest path from server and return.
5) *STEP 3:* end if
6) *STEP 4:* let Estimated shortest distance Esd= ∞.
7) *STEP 5:* for each path p ϵ CSP do
8) *STEP 6:* if p is a complete hit then
9) *STEP 7:* Return $p_{sq,dq} = p$.
10) *STEP 8:* end if
11) *STEP 9:* $v_s = \min (q\epsilon\ V_p D(q,S))$.
12) *STEP 10*: $d_s = D(v_{s,} S)$.
13) *STEP 11:* Remove v_s from path p node-set V_p.
14) *STEP 12:*$v_d = \min (q\epsilon\ V_p D(q,D))$.
15) *STEP 13:*$d_d = D(v_{d,} S)$.
16) *STEP 14:*let$d_r = D(S,D)$.
17) *STEP 15*: $d = d_s+d_r+d_d$.

VI. CACHE MANAGEMENT

It is a cache based system, it is necessary to effectively manage the cache contents in-order to speed up the path planning process. So, in this section, we first discuss the implementation of grid based index, followed by explaining a dynamic cache update and replacement policy.

A. Grid Structure for Cache

The cache contains two tables for effective cache lookups. The first tables records each grid cell and all the cached paths passing through the corresponding grid cell. This table allows quick detection of CSPaths for the new path planning query. The second table records all nodes of each path in their travelling order. This table is accessed when the final path for new path planning query is determined.

We firstly determine the target space where all the cached paths are overlapped as a minimal bounded rectangular region. Then we divide this space into equally sized grid cells, so that, every node in cached path must mapped into, one of the grid cells. For each grid-cell, we maintain an entry in table1, recording ID,s of all paths which pass through it.

GridID : (pathID) , ... , (pathID).

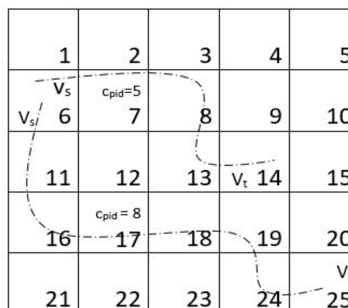

Fig. 2: An example of grid structure.

©IJRASET (UGC Approved Journal): All Rights are Reserved

Fig.2 shows an example for illustration of the grid structure, in which the target space is divided into 25 equally sized grid cells. Each cell with unique ID numbered from 1 to 25. There are two paths denoted as C_{pid} = 5 and 8 in this region. The starting nodes of both the paths are at the grid cell 6, we retain a list for g_6 as <5,8>. When a path is inserted or deleted the list belongs to corresponding cell updated accordingly. This gird based structure increases the search efficiency for cache lookup operations. When we receive a new query $q_{s,t}$ for which the source $v_s$is located at g_6 means cell 6 and the destination node $v_t$located at g_{14}means cell 14, the system directly access the grid cells and find the path with c_{pid}= 5 that has the same source and destination. For each cached path all of its nodes are maintained by second table. These nodes are strictly maintained in travelling order.Table1 shows an example for illustration of the second table.
pathID : (nodeID), ... , (nodeID)

TABLE II
An Example of Cached Paths Table.

C_{pid}	Path
1	$P_{2,6} = \{v_2, v_3, v_6\}$
2	$P_{5,3} = \{v_5, v_4, v_2, v_1, v_3\}$
.	.
.	.
.	.
\	
	$P_{3,5} = \{v_3, v_4, v_2, v_5\}$

B. Cache Construction and Update

Algorithm 3 provides the pseudo code for cache construction and update. Based on the current status of the cache (initially it is empty), when a new path planning query comes, we estimate its path if any CSPath exists (Line 5); otherwise the path is retrieved from the server (Line 3). If cache is not full, then the path is directly inserted into cache (Lines 7 to 9). Otherwise, a replacement is triggered (Lines 10 to 14). We check whether the usability value of the current path p is larger than or equals the minimum usability value in the current cache. If so, we place the current query into the cache. Here, the usability is utilization value, calculated using Eq. (2).

Table iii
Type and weight of each road

Node	Weight W_i	Road type
V_1	1.0	Highway
V_2	0.8	Primary main road
V_3	0.8	Primary main road
V_4	0.6	Tertiary way
V_5	0.3	Street way
V_6	0.1	Goat path

$$\mu(p_{i,j}) = \sum_{k=1}^{n} W_{v_k} \times count(v_k) \qquad (2)$$

Here in Eq. (2), W_{vk} is the weight of the node differ from each node based on the road type. Count(v_k) is total number of times the node previous used in cached paths. For example, the utilization of path $p_{2,6}$ is calculated as $\mu(p_{2,6})$ = 0.8 * 3 + 0.8 * 3 + 0.1 * 1 = 4.9. Similarly, the utilization value for path $p_{3,5}$is $\mu(p_{3,5})$ = 0.8 * 3 +0.6 * 2 + 0.8 * 3 + 0.3 * 2 = 6.6. A cached path with low usability value will be more likely to be deleted first.

©IJRASET (UGC Approved Journal): All Rights are Reserved

Table iv
Three stored paths in the cache.

C_{pid}	Path	μ
1	$P_{2,6} = \{v_2, v_3, v_6\}$	4.9
2	$P_{5,3} = \{v_5, v_4, v_2, v_1, v_3\}$	7.6
3	$P_{3,5} = \{v_3, v_4, v_2, v_5\}$	6.6

VII. EXPERIMENTS

A. Dataset

We conduct a comprehensive performance evaluation of the proposed CBPP system using road network dataset which is generated randomly with 25,600 nodes and 75,000 edges. Next, we randomly select pairs of nodes as the source and destination for path planning queries. Next, we firstly randomly generate a query to be the initial navigational path. Then, we randomly draw a probability to find the chance for a driver to change the direction. Now, when change of direction occurs, this point is treated as a new source. This process is repeated until the required number of queries generated. The parameters that are used in experiments are shown in Table 4.

TABLE 4
PARAMETERS FOR EXPERIMENT.

Parameter	Value
Grid cell size	1 km to 5km
Cache size	Up-to 10 k
Number of Queries	1k to 5k

B. Cache-Based System Performance

The main idea of a CBPP system is to use cache effectively to answer a new path planning query. Thus, we need to know that how much effectiveness does our system shows over a conventional non-cache based system. We have generated a different sizes of queries to analyze the computation cost between CBPP and A* algorithm [2]. The performance is evaluated by the total query time, it is the total time taken by the algorithm to find the shortest path. Table shows the results on the different sized queries. On average, CBPP reduces the computation cost by 31% compared with non-cache A* algorithm.

TABLE V
COMPARISON BETWEEN CBPP AND A* ALGORITHM

#Query	Time(ms)	
	A* with-out cache	CBPP
1k	19,953,526	14,001,791
2k	39,869,232	27,889,826
3k	59,493,098	41,983,994
4k	79,937,416	55,193,522
5k	100,037,168	69,843,515

VIII. CONCLUSION

In this paper, we have implemented a system, namely Cache-Based Path Planning (CBPP), which answers a new path planning query by the maximum utilization of the cache. Unlike the general traditional cache-based path planning systems, where the cached path is used when it exactly matches with new path planning query, CBPP utilizes the partially matched queries to answer sub parts of the new query. As a result, the server only needs to compute the paths for unmatched path segments of the query, thus reduces the overall workload of the system. Thus, reduces the path finding complexity of the system. Experimental results shows that, CBPP reduced the computation cost by 30%.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue II, February 2018- Available at www.ijraset.com

REFERENCES

[1] H. Mahmud, A. M. Amin, M. E. Ali, and T. Hashem, “Shared Execution of Path Queries on Road Networks,” Clinical Orthopaedics and Related Research, vol. abs/1210.6746, 2012
[2] A. V. Goldberg and C. Harrelson, “Computing the Shortest Path: A Search Meets Graph Theory,” in ACM Symposium on Discrete Algorithms, 2005
[3] J. Sankaranarayanan, H. Samet, and H. Alborzi, “Path Oracles for Spatial Networks,” Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 1210–1221, 2009
[4] H. Hu, J. Xu, and D. L. Lee, “A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects,” in ACM International Conference on Management of Data, 2005.
[5] U.Zwick,“Exactandapproximatedistancesingraphs–asurvey,” in Algorithms – ESA 2001, 2001, vol. 2161, pp. 33–48
[6] A. V. Goldberg and C. Silverstein, “Implementations of Dijkstra’s Algorithm Based on Multi-Level Buckets,” in Network Optimization, 1997, vol. 450, pp. 292–327
[7] R. Gutman, “Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks,” in Workshop on Algorithm Engineering and Experiments, 2004.
[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Third Edition, 2009
[9] S. Jung and S. Pramanik, “An Efficient Path Computation Modelfor Hierarchically Structured Topographical Road Maps”, IEEETransactions on Knowledge and Data Engineering, vol. 14, no. 5, pp.1029–1046, 2002.

©IJRASET (UGC Approved Journal): All Rights are Reserved

10.22214/IJRASET

45.98

IMPACT FACTOR:
7.129

IMPACT FACTOR:
7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY