International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 12
April 2018

Efficient Cache-Supported Path Planning on Roads

[1]T. Yashasree, [2]M. Joshna, [3]C. Sravani, [4]Ch. Vyshnavi
[1]Assistant Professor, [2,3,4]B.Tech,
[1,2,3,4]Department of Computer Science and Engineering,
[1,2,3,4]Vignan Institute of technology and Science.

ABSTRACT

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 reducing the overall system workload.

Keywords: Spatial Database, Path Planning, Cache.

INTRODUCTION

Due to advances in big data analytics, there is a growing need for scalable parallel algorithms. These algorithms encompass many domains including graph processing, machine learning, and signal processing. However, one of the most challenging algorithms liein graph processing. Graph algorithms are known to exhibit low locality, data dependence memory accesses, and high memory requirements. Even their parallel versions do not scale seamlessly, with bottlenecks stemming from architectural constraints, such as cache effects and on-chip network traffic. Path Planning algorithms, such as the famous Dijkstra's algorithm, fall in the domain of graph analytics, and exhibit similar issues. These algorithms are given a graph containing many vertices, with some neighboring vertices to ensure connectivity, and are tasked with finding the shortest path from a given source vertex to a destination vertex. Parallel implementations assign a set of vertices or neighboring vertices to threads, depending on the parallelization strategy. These strategies naturally introduce input dependence. Uncertainty in selecting

the subsequent vertex to jump to, results in low locality for data accesses.

Moreover, threads converging onto the same neighboring vertex sequentialize procedures due to synchronization and communication. Partitioned data structures and shared variables ping-pong within on-chip caches, causing coherence bottlenecks. All these mentioned issues make parallel path planning a challenge. Prior works have explored parallel path planning problems from various architectural angles. Path planning algorithms have been implemented in graph frameworks. These distributed settings mostly involve large clusters, and in some cases smaller clusters of CPUs. However, these works mostly optimize workloads across multiple sockets and nodes, and mostly constitute either complete shared memory or message passing (MPI) implementations.

In the case of single node (or single-chip) setup, a great deal of work has been done for GPUs are a few examples to name a few. These works analyze sources of bottlenecks and discuss ways to mitigate them. Summing up these works, we devise that most challenges remain in the fine-grain inner loops of path planning algorithms. We believe that analyzing and scaling path planning on single-chip setup can minimize the fine-grain bottlenecks. Since shared memory is efficient at the hardware level, we proceed with parallelization of the path planning workload for single-chip multi-cores. The single-chip parallel implementations can be scaled up at multiple nodes or clusters granularity, which we discuss.

To meet existing 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). The proposed system consists of three main components: (i) PPattern Detection, (ii) Shortest Path Estimation, and (iii) Cache Management.

Given a path planning query, 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. 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).

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 queried paths in cache, we design a grid-based index for the PPattern Detection module. Based on these detected PPatterns, the Shortest Path Estimation module 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 .Because the unmatched segments are usually only a smaller part of the original query, the server only processes a **"**smaller subquery**"**, with a reduced workload.

Once we return the estimated path to the user, the Cache Management 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 networks. 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.

LITERATURE SURVEY

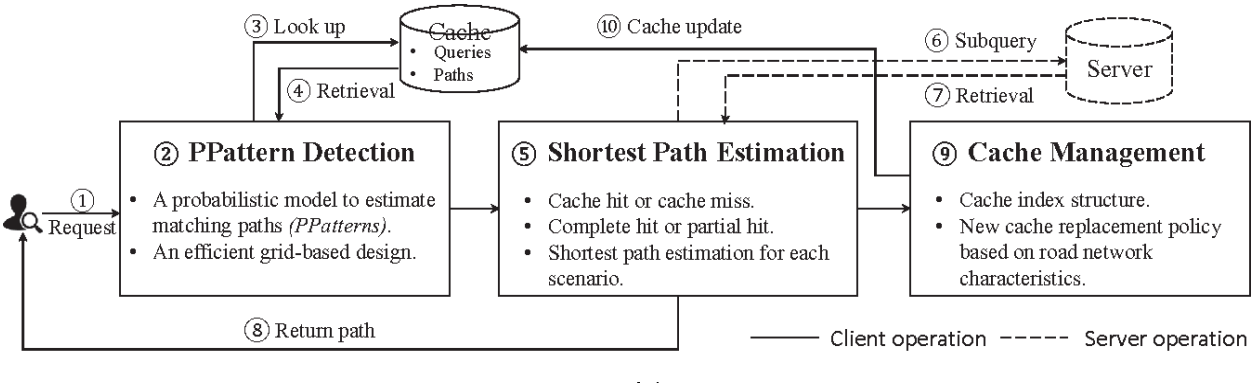
An enhanced version [10] adds easy route curves to lessen vertices from being gone to and utilizes halfway trees to diminish the pre-processing time. This work additionally joins the advantages of the achieve based and ATL ways to deal with decrease the quantity of vertex visits and the pursuit space. The examination demonstrates that the cross breed approach gives a predominant outcome as far as diminishing question preparing time. Jung and Pramanik [11] propose the HiTi diagram model to structure a huge street organize display. HiTi expects to decrease the look space for the briefest way calculation. While HiTi accomplishes superior on street weight overhauls and lessens stockpiling overheads, it brings about higher calculation costs when processing the most brief ways than the HEPV and the Hub Indexing strategies [12][13][14].

To process time-subordinate quick ways, Demiryurek et al. [15] propose the B-TDFP calculation by utilizing in reverse inquiries to diminish the hunt space. It receives a territory level parcel plot which uses a street progressive system to adjust every zone. Be that as it may, a client may incline toward a course with better driving knowledge to the briefest way. Consequently, Gonzalez et al. propose a versatile quick way calculation which uses speed and driving examples to enhance the nature of courses [16]. The algorithm utilizes a road hierarchical partition and pre-computation to enhance the execution of the course calculation. The little street redesign is a novel way to deal with enhancing the nature of the route computation.

In order to enhance the recovery efficiency of the way arranging framework, Thomsen et al. [17] propose another reserve administration arrangement to store the aftereffects of continuous questions for reuse later on. To upgrade the hit proportion, an advantage esteem capacity is utilized to score the ways from the question logs. Thusly, the hit proportion is expanded, henceforth diminishing the execution times. Be that as it may, the cost of developing a store is high, since the framework must compute the advantage values for all sub-ways in a full-way of inquiry results. For on-line, delineate applications, preparing a

substantial number of concurrent way questions is an essential issue. In this paper, we give another system to reusing the already reserved inquiry comes about and a successful calculation for enhancing the question assessment on the server.

SYSTEM ARCHITECTURE



PATH PLANNING ALGORITHMS AND PARALLELIZATIONS

Baseline used in path planning applications. However, several heuristic based variations exist those trade-off parameters such as parallelism and accuracy. Δ-stepping is one example

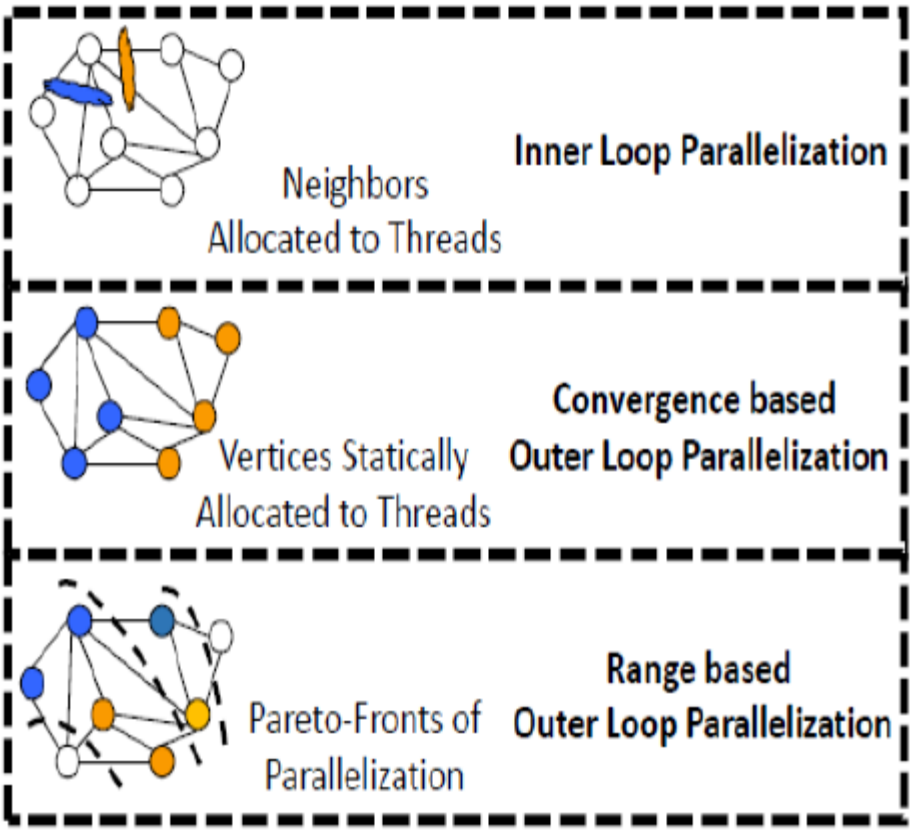


Fig.1. Dijkstra's Algorithm Parallelization's. Vertices Allocated to Threads Shown in Different Colors.

Dijkstra's Algorithm and Structure

Dijkstra's algorithm consists of two main loops, an outer loop that traverses each graph vertex once, and an inner loop that traverses the neighboring vertices of the vertex selected by the outer loop. The most efficient generic implementation of Dijkstra's algorithm utilizes a heap structure, and has a complexity of $O(E + V \log V)$. However, in parallel implementations, queues are used instead of heaps, to reduce overheads associated with re-balancing the heap after each parallel iteration. Algorithm 1 shows the generic pseudo-code skeleton for Dijkstra's algorithm. For each vertex, each neighboring vertex is visited and compared with other neighboring vertices in the context of distance from the source vertex (the starting vertex). The neighboring vertex with the minimum distance cost is selected as the next best vertex for the next outer loop iteration. The distances from the source vertex to the neighboring vertices are then updated in the program data structures, after which the algorithm repeats for the next selected vertex. A larger graph size means more outer loop iterations, while a large graph density means more inner loop iterations. Consequently, these iterations translate into parallelism, with the graph's size and density dictating how much parallelism is exploitable. We discuss the parallelizations in subsequent subsections and show examples in Fig 1.

CONCLUSION

Path Planning by Caching (PPC), 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% of the computational latency on average.

REFERENCES

[1]. H. Mahmud, A.M. Amin, M.E. Ali and T. Hashem, "Shared Executionof Path Queries on road Networks," Clinical OrthopaedicsRelated Research, Vol. abs/1210.6746, **(2012)**.

[2]. L. Zammit, M. Attard, and K. Scerri, "Bayesian HierarchicalModelling of Traffic Flow - With Application to Malta's RoadNetwork," in Proceedings of International IEEE Conference on Intelligent Transportation System, pp. 1376-1381**(2013)**.

[3]. S. Jung and S. Pramanik, "An Efficient path Computation Model for Hierarchically Structured Topographical Road Maps," IEEE Transportation Knowledge Data

Engineering, Vol. 14, No. 5, pp. 1029-1046, September, **(2002)**.

[4]. E.W. Dijkstra, "A Note on Two problems in Connexion with Graphs," Numerical Mathematics, Vol. 1, No. 1, pp. 269-271, **(1959)**.

[5]. U. Zwick, "Exact and Approximate Distances in Graphs – a survey," in Proceedings of 9th Annual European Symposium Algorithms, Vol. 2161, pp. 33-48**(2001)**.

[6]. A.V. Goldberg and C. Silverstein, "Implementations of Dijkstra's Algorithm based on Multi-level Buckets," Network Optimization, Vol. 450, pp. 292-327, **(1997)**.

[7]. P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Transportation System Science and Cybernetics, Vol. SSC-4, No. 2, pp. 100-107, July, **(1968)**.

[8]. A.V. Goldberg and C. Harrelson, "Computing the Shortest path: A Search meets Graph Theory," in Proceedings of ACM Symposium on Discrete Algorithms, pp. 156-165, **(2005)**.

[9]. R. Gutman, "Reach-based routing: A New Approach to Shortest path Algorithms Optimized for Road Networks," in Proceedings of Workshop Algorithm Engineering Experiments, pp. 100-111, **(2004)**.

[10]. A.V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient Point-to-Point shortest path Algorithms," in Proceedings of Workshop Algorithm Engineering Experiments, pp. 129-143, **(2006)**.

[11]. S. Jung and S. Pramanik, "An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps," IEEE Transportation Knowledge Data Engineering, Vol. 14, No. 5, pp. 1029-1046, September, **(2002)**.

[12]. R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina, "Proximity Search in Databases," in Proceedings ofthe International Conference on Very Large Data Bases, pp. 26-37,**(1998)**.

[13]. N. Jing, Y.W. Huang and E.A. Rundensteiner, "Hierarchical Optimization of Optimal Path Finding for Transportation Applications," in Proceedings of ACM Conferenceon Information Knowledge Management, pp. 261-268, **(1996)**.

[14]. N. Jing, Y. Wu Huang and E.A. Rundensteiner, "Hierarchical encoded path views for path query processing: An Optimal Model and its Performance Evaluation," IEEE Transportation Knowledge Data Engineering, Vol. 10, No. 3, pp. 409-432, May/June, **(1998)**.

[15]. U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ranganathan, "Online Computation of Fastest Path in Time-dependent Spatial Networks," in Proceedings of 12th International Conference on Advanced Spatial Temporal Databases, pp. 92-111, **(2011)**.

[16]. H. Gonzalez, J. Han, X. Li, M. Myslinska and J.P. Sondag, "Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach," in

Proceedings of 33rd International Conference on Very Large Data Bases, pp. 794-805, **(2007)**.

[17]. J.R. Thomsen, M.L. Yiu, and C.S. Jensen, "Effective caching of shortest paths for location-based services," in Proceedings of ACM SIGMOD International Conference Management Data, pp. 313-324, **(2012)**.

[18]. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", 3rd ed. Cambridge, MA, USA: MIT Press. **(2009)**.

[19]. E. Markatos, "On Caching Search Engine Query Results," ComputerCommunication, Vol. 24, No. 2, pp. 137-143, **(2001).**

[20]. R. Ozcan, I.S.Altingovde and O. Ulusoy, "A Cost-aware Strategy for Query Result Caching in Web Search Engines," in Proceedings of Advanced Information Retrieval, Vol. 5478, pp. 628–636,**(2009)**.