RESEARCH ARTICLE OPEN ACCESS

AN EFFECTIVE AND ROBUSTIVE ON CACHE-SUPPORTED PATH PLANNING ON ROADS

K SRINIVASA RAO*[1], CHINTALA NAGAMANI [2]

*[*1] ASSISTANT PROFESSOR,DEPARTMENT OF MCA, VIGNAN'S LARA INSTITUTE OF TECHNOLOGY&SCIENCE, VADLAMUDI, GUNTUR, ANDHRA PRADESH, INDIA.
[2]MCA STUDENT, DEPARTMENT OF MCA, VIGNAN'S LARA INSTITUTE OF TECHNOLOGY&SCIENCE, VADLAMUDI, GUNTUR, ANDHRA PRADESH, INDIA

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 ofGPS 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.

I. 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 lie in 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.

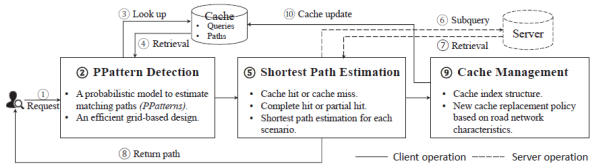


Fig :over view ppc system

Furthermore, programming language variations for large scale processing also cause scalability issues that need to be analyzed effectively so far the most efficient parallel shared memory implementations for graph processing are in C/C++. However, due to security exploits and other potential vulnerabilities, other safe languages are commonly used in mission-deployed applications. Safe languages guarantee dynamic security checks that mitigate vulnerabilities, and provide ease of programming. However, security checks increase memory and performance overheads. Critical sections of code, such as locked data structures, now take more time to process, and hence communication and synchronization overheads exacerbate for parallel implementations. Python is a subtle example of a safe language, and hence we analyze it's overheads in the context of our parallel path planning workloads. This paper makes the following contributions:

- We study sources of bottlenecks arising in parallel path planning workloads, such as input dependence and scalability, in the context of a single node, single chip setup.

- We analyze issues arising from safe languages, in our case Python, and discuss what safe languages need to ensure for seamless scalability.

- We plan to open source all characterized programs with the publication of this paper.

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.

Path Planning

The Dijkstra algorithm [4], [5] has been widely used for path planning [6] by computing the shortest distance be-tween 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.

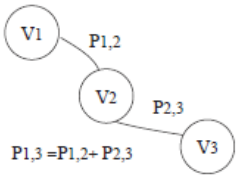
Jung et al. [11] propose the HiTi graph model to struc-ture a large road network model. HiTi aims to reduce the search space for the shortest path computation. While HiTi achieves high performance on road weight updates and reduces storage overheads, it incurs higher computation costs when computing 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 com-putation. 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 $s \leq k$, $j \leq t$, 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 sub-paths $p_{1;2}$, $p_{2;3}$, 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 $2 \times 20.3 + 5 \times 20.3 + 5 \times 40.6 = 345.1$ using SPC.



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

Fig. 2: An example to illustrate the calculation of path benefit value in SPC. The popularity and expense values for each road segment are listed in the table. For the path $p_{1;3}$ containing three sub-paths, its benefit value is calculated as $2 \times 20{:}3 + 5 \times 20{:}3 + 5 \times 40{:}6 = 345{:}1$.

Because SPC has to score each sub-path in the short-est path, the time complexity is high. Assuming that a shortest path contains n nodes, a shortest path contains $(n \times (n - 1))/2$ sub-paths. The time complexity for scoring a shortest path is $O(n^2)$. 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 group-based 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 im-portant observations are described as follows. First, a small number of queries are frequently re-used. By preserving re-sults of these queries in cache, the system is able to respond to the users without incurring time-consuming computa-tions. 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 per-formance 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 man-agement 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 previ-ously cached query results as well as an effective algorithm for improving the query evaluation on the server.

PPATTERN DETECTION

To detect the best PPatterns, an idea is to calculate the esti-mation 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., SDP(s′, a), SDP(b, t′)). 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. How-ever, 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 the following 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 p(qs;t,qs;t) as the probability for two queries qs;t′ and qs;t to be PPatterns of each other. Thus, constraints (1) and (2) indicate $p(qs';t', qs;t) \propto 1$ and $p(qs';t', qs;t) \propto 1, D(s;s')D(t;t')$

respectively. On the other hand, constraint (3) implies $p(qs';t', qs;t) \propto D(s', t') \times D(s, t)$. Thereafter, the final prob-ability can be computed as the product of these three terms. As we would like the three factors to achieve sufficient satis-faction concurrently, the probability will only be computed if each factor is over a threshold as expressed by Eq. (1).

$$\mathbf{p}(q_{s',t'}, q_{s,t}) = \prod_{x \in \{s,t,l\}} w_x \cdot u(f_x(q_{s',t'}, q_{s,t}) - D_x)$$

-------(1)

and $fl(qs';t', qs;t) = D(s', t') \times D(s, t)$ respectively indi-cate the source-source, destination-destination and source-destination node distance factors between the two queries of qs′;t′ and qs;t; wx is a weight indicating the contribution from each factor fx; Dx is a threshold controling the valida-tion scope for fx; and u(•) is a shifted unit step function:

$$u(x - D_x) = \begin{cases} 1: & x \geq D_x \\ 0: & x < D_x \end{cases}$$

------(2)

When the score p(qs′ ;t′ , qs;t) is over a threshold, it is very likely that the two paths will share a path segment. In

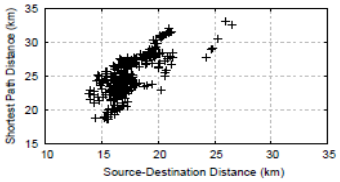


Fig. 3: Correlation between source-destination distance and the sortest path distance.

PPatterns detection, given a query, we check every queried-path in cache and select the ones with top scores as PPattern candidates:

P T (qs′ ;t′ , qs;t) = {1 : p(qs ;t , qs;t) > ϵ
0 : otherwise′ ′
.

where ϵ is a probability threshold. Note that in the above,all distances are Euclidean distances.

Based on the above, we select the highest ranked PPat-terns for a new query. In the following, we prove that the estimation error is upper-bounded to a value 2α(Ds + Dt) where Ds and Dt are source-source and destination- destination distance thresholds and α is a parameter ap-

proximating the relation between the shortest path distance of two points on a road network and their true Euclidean distance, i.e., SP D(a, b) = αD(a, b). Factor α can be esti-mated 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 qs;t is estimated from a pattern candidate qs′ ;t′ .

In the first scenario (see Fig. 6(a)), there exists at least one common segment between the paths of the two queries. In

the other two scenarios (shown in Figs. 6(b) and 6(c)), there exist no common segments, but the two queries are similar to each other

An Efficient Grid-Based Solution

In order to retrieve these patterns efficiently, we propose a grid-based solution to further improve the system per-formance. 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:

P T (qs′ ;t′ , qs;t) = 1 : D(s, t) ⩽ Dl)
g(s) = g(s), g(t) = g(t0 : otherwise.

where g( ) is the grid cell in which a node is located.

Algorithm 1 PPatterns Detection.

Input: $q_{s;t}$: a query ; D_l: distance threshold; D_g: grid cell size,
C: a cache.
Output: All candidate PPatterns $P\,T$.

1. **if** $D(s;\, t) < D_l$ **then**
2. Return $P\,T = \emptyset$.
3. **end if**
4. Divide the target space by size D_g.
5. Determine the start grid g_s and destination grid g_t.
6. $Q_s \leftarrow$ Logged queries whose paths pass g_s.
7. $Q_t \leftarrow$ Logged queries whose paths pass g_t.
8. $Q = Intersect(Q_s;\, Q_t)$.
9. $P\,T \leftarrow$ (Sub)paths from G_s to G_t for each query in Q.
10. Return $P\,T$.

CACHE-SUPPORTED SHORTEST PATH ESTIMA-TION

Based on the PPatterns detected above, we estimate the shortest path for a new query using Note that the detected PPatterns contribute to at least a part of the answer path returned to the users and actually increases the cache utilization.

To detect the estimated shortest path, we propose a heuristic algorithm as shown in Algorithm 2.

Fig. : Illustration of the four cases in *P P C*.

Algorithm 2 Shortest Path Estimation.

Input: query source node s' and destination node t'; all candi-date PPatterns *P T* ; Cache *C*.
Output: Estimated shortest path $p^{\wedge *}$.

```
1:  if isEmpty(P T )) then
2:  p^* ← Calculate path from server and return.
3:  end if
4:  Initialize Estimated Shortest Distance ESD = ∞.
5:  for each path p ∈ P T do
6:  if p is a complete hit then
7:  Return p^*_{s',t'} = p.
8:  end if
9:  s* = arg min_{s∈Vp} D(s'; s).
10: d_s = D(s'; s*).
11: Remove s* from path node-set V_p.
12: t* = arg min_{t∈Vp} D(t; t').
13: d_t = D(t*; t').
14: Let d_r = |SDP(s'; t')|.
15: d = d_s + d_r + d_t
16:         if d < ESD then
17:         ESD = d.
18: Update best PPattern p* = ps*;t* .
19: end if
20: end for
21: if s' is not equal to v_s^{p*} then
22: SDP(s'; v_s^{p*}) ←Compute shortest path SDP(s'; v_s^{p*}).
23: end if
24: if t' is not equal to v_t^{p*} then
25: SDP(v_t^{p*} ; s') ←Compute shortest path SDP(v_t^{p*} ; t').
26: end if
27: Return p^* = SDP(s'; v_s^{p*}) ⊙ p* ⊙ SDP(v_t^{p*} ; t').
```

CACHE MANAGEMENT

In a cache-supported system, it is important to efficiently manage the cache contents to accelerate the path planning. Therefore, in this section, we first discuss the implemen-tation of a grid-based index, followed by describing a dy-namic cache update and replacement policy

Algorithm 3 Cache Construction and Update.

```
Input: a query q, a cache C.
Output: a cache C.
1:  PT ← PPatterns Detection.
2:  p ← Shortest Path Estimation from PT.
3:  if C is not full then
4:  Insert p into C; Return C.
5:  else
6:  {μ} ← Calculate usability for each cached path.
7:  p* ← Path with the minimum usability.
8:  if p*.μ < p.μ then
9:  C ← Replace p* with p.
10: end if
11: end if
12: Return C.
```

CONCLUSION

In this paper, we propose a framework, to be specific, Path Planning by Caching (PPC), to answer another path planning inquiry with quick reaction by proficiently caching and reusing the authentic queried-paths. Not at all like the customary reserve based path planning frameworks, where a queried-path in the store is utilized just when it coordinates impeccably with the new inquiry, PPC influences the somewhat coordinated stored questions to answer part(s) of another inquiry. Subsequently, the server just needs to figure the unmatched fragments, in this manner essentially decreasing the general framework workload. Thorough experimentation on a genuine street organize database demonstrates that our framework outflanks the cutting edge path planning procedures by decreasing 32% of the computational dormancy by and large.

II. REFERENCES

[1] H. Mahmud, A. M. Amin, M. E. Ali, and T. Hashem, "Shared Ex-ecution of Path Queries on Road Networks," Clinical Orthopaedics and Related Research, vol. abs/1210.6746, 2012.

[2] L. Zammit, M. Attard, and K. Scerri, "Bayesian Hierarchical Mod-elling of Traffic Flow - With Application to Malta's Road Net-work," in International IEEE Conference on Intelligent Transportation Systems, 2013, pp. 1376–1381.

[3] S. Jung and S. Pramanik, "An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps," IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1029–1046, 2002.

[4] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[5] U. Zwick, "Exact and approximate distances in graphs – a survey," 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 Optimiza-tion, 1997, vol. 450, pp. 292–327.

[7] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transac-tions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1967.

[8] A. V. Goldberg and C. Harrelson, "Computing the Shortest Path: A Search Meets Graph Theory," in ACM Symposium on Discrete Algorithms, 2005.

[9] R. Gutman, "Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks," in Workshop on Algorithm Engineering and Experiments, 2004.

[10] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient Point-to-Point Shortest Path Algorithms," in Workshop on Algorithm Engineering and Experiments, 2006, pp. 129–143.

[11] S. Jung and S. Pramanik, "An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps," IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1029–1046, 2002.

[12] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina, "Proximity Search in Aatabases," in Interna-tional Conference on Very Large Data Bases, 1998, pp. 26–37.

[13] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, "Hierarchical Optimization of Optimal Path Finding for Transportation Appli-cations," in ACM Conference on Information and Knowledge Manage-ment, 1996.

[14] N. Jing, Y. wu Huang, and E. A. Rundensteiner, "Hierarchical En-coded Path Views for Path Query Processing: An Optimal Model and its Performance Evaluation," IEEE Transactions on Knowledge and Data Engineering, vol. 10, pp. 409–432, 1998.

[15] U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ran-ganathan, "Online Computation of Fastest Path in Time-Dependent Spatial Networks," in International Conference on Ad-vances in Spatial and Temporal Databases, 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 International Conference on Very Large Data Bases, 2007.

[17] J. R. Thomsen, M. L. Yiu, and C. S. Jensen, "Effective caching of shortest paths for location-based services," in ACM International Conference on Management of Data, 2012.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduc-tion to Algorithms, Third Edition, 2009.

[19] E. Markatos, "On Caching Search Engine Query Results," Com-puter Communications, 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 Advances in Information Retrieval, 2009, vol. 5478, pp. 628–636.

[21] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The Impact of Caching on Search Engines," in International ACM Conference on Research and Development in Information Retrieval, 2007.

[22] R. Baeza-Yates and F. Saint-Jean, "A Three Level Search Engine Index Based in Query Log Distribution," in String Processing and Information Retrieval, 2003, vol. 2857, pp. 56–65.

[23] R. Ozcan, I. S. Altingovde, B. B. Cambazoglu, F. P. Junqueira, and zgr Ulusoy, "A Five-Level Static Cache Architecture for Web Search Engines," Information Processing and Management, vol. 48, no. 5, pp. 828 – 840, 2012.

[24] R. Ozcan, I. S. Altingovde, and O. Ulusoy, "Static Query Result Caching Revisited," in International Conference on World Wide Web, 2008.

[25] Q. Gan and T. Suel, "Improved Techniques for Result Caching in Web Search Engines," in International Conference on World Wide Web, 2009.

[26] X. Long and T. Suel, "Three-level Caching for Efficient Query Pro-cessing in Large Web Search Engines," in International Conference on World Wide Web, 2005.

[27] 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.

[28] H. Hu, J. Xu, and D. L. Lee, "A Generic Framework for Moni-toring Continuous Spatial Queries over Moving Objects," in ACM International Conference on Management of Data, 2005.

[29] X. Xiong, M. F. Mokbel, and W. G. Aref, "SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases," in IEEE International Conference on Data Engineering, 2005.

[30] H. Kanoh, "Dynamic Route Planning for Car Navigation Sys-tems Using Virus Genetic Algorithms," International Journal of Knowledge-based and Intelligent Engineering Systems, vol. 11, no. 1, pp. 65–78, 2007.

[31] M. Ali, J. Krumm, T. Rautman, and A. Teredesai, "Acm sigspatial gis cup 2012," in ACM International Conference on Advances in Geographic Information Systems, 2012.

[32] "Cloudmade," downloads.cloudmade.com.