Rapid Modified To Order in Road Network Using Customer Point of Interest

M. Anbu#1, R. Kanniyarasu #2, S.P. Santhoshkumar #3, V. Anuradha#4
UG Scholar, Dept of CSE, Shree Sathyam College of Engineering and Technology,Sankari, India.
Assistant Professor, Dept of CSE, Shree Sathyam College of Engineering and Technology,Sankari, India
Assistant Professor, Dept of CSE, Shree Sathyam College of Engineering and Technology,Sankari, India
HOD, Dept of CSE, Shree Sathyam College of Engineering and Technology,Sankari, India
[1]anbuaisha848@gmail.com,[2] kanniyarasu.k@gmail.com
[3]spsanthoshkumar16@gmail.com, [4]kavkamanu@gmail.com@gmail.com

***Abstract*— We present a combined structure for dealing with exact point-of-interest (POI) queries in dynamic continental road networks within interactive applications. We show that partition-based algorithms developed for point-to-point shortest path computations can be naturally extended to handle augmented queries such as finding the closest cafe or the best place of work to stop on the way home, always position POIs according to a user-defined cost function. Our solution allows different trade-offs between indexing effort (time and space) and query time. Our most flexible variant allows the road network to change regularly (to account for travel information or modified cost functions) and the set of POIs to be specified at query time. Even in this fully dynamic scenario, our key is fast enough for interactive applications on continental road networks.**

***Keyword*—Points-of-interest, dynamic road networks, and indexing techniques. Continental road network.**

I. INTRODUCTION

A point of interest, or POI, is a specific point location that someone may find useful or interesting. An example is a point on the Earth representing the location of the Space Needle, or a point on Mars representing the location of the mountain, Olympus Mons. Most consumers use the term when referring to hotels, campsites, fuel stations or any other categories used in modern (automotive) navigation systems. In medical fields such as histology/pathology/histopathology, points of interest are selected from the general background in a field of view.

A POI is a human construct, describing what can be found at a Location. As such a POI typically has a fine level of spatial granularity. A POI has the following attributes. A name, A current Location (see the commentary below on the loose coupling of POI and Location),A category and/or type, A unique identifier, A URI, An address, Contact information.

A POI has a loose coupling with a Location; in other words, a POI can move. When this occurs, the loose coupling with the previous location is removed and, providing the POI continues to exist, it is then coupled with its new Location. This can happen when the human activity at the POI relocates, such as when your local coffee shop relocates to a new address. It's still your local coffee shop, it's now found at a different Location.

A POI has temporal boundaries; it starts when the human activity at that Location commences and ends when human activity ceases, such as when a company or organisation goes out of business.

Present map services and other spatial systems must support a wide range of applications. Besides compute optimal (with respect to a cost function such as travel times or time in traffic) point-to-point routes, advanced queries like “find the closest Thai restaurant to my current location” or “what is the best place to shop for groceries on my way home” need to be supported as well. All these location services depend on a location and a set of points-of-interest (POIs) with certain properties (such as open times, category, or personal preferences). Given a location, we want to rank the POIs to decide which ones to report first. If one wants to order them by the actual driving (or walking) time from a given location, all these problems could be solved with one or more calls to a standard graph search algorithm, such as Dijkstra’s [2]. For continental road networks, however, this takes several seconds for long-range queries [3], too slow for interactive applications. For better performance, more sophisticated solutions are needed.

An indexing step can associate POI information with these hubs, allowing for quick queries. These methods work reasonably well, but have major drawbacks, including nontrivial preprocessing effort and excessive space requirements.

Most importantly, hierarchical methods are not robust to changes in the cost function even small changes(such as setting a high cost for making a U-turn, i.e., turning into the opposite direction on the same road segment) can have a significant adverse effect on their performance.

As long as queries are fast enough, considerations such as flexibility the types of queries supported), low space consumption, predictable performance,

realistic modeling, and robustness (with respect to the cost function) are more important for map services than raw speed. CRP excels in this regard because it moves the metric dependent portion of the preprocessing to a metric customization phase, which runs in roughly a second on a continental road network using a standard server. This enables support for real-time traffic and personalized cost functions. Unlike hierarchical approaches, CRP supports realistic modeling (turn costs and restrictions) with little overhead and is much more robust to the choice of the optimization function. Not coincidentally, the routing engine for Bing Maps is based on CRP.

II. PRELIMINARIES

A. Road Networks and Shortest Paths

A road network is usually modeled as a directed graph G = (V; A), where each vertex v 2 V represents an intersection of the road network and each arc (v; w) 2 A represents a road segment. A metric (or cost function) ` : A ! N maps each arc to a positive length (or cost). For a more realistic model, we also take turn costs (and restrictions) into account. We think of each vertex v as having one entry point for each of its incom-ing arcs, and one exit point for each outgoing arc. We extend the concept of metric by also associating a turn table T_v to each vertex v. In this matrix, entry $T_v[i; j]$ specifies the cost of turning from the i-th incoming arc to the j-th outgoing arc. Such modeling is essential for a realistic map service to properly account for turn restrictions and to avoid unnatural routes with frequent U-turns or turns against traffic.

In the point-to-point shortest path problem, we are given a source location s and a target location t, and our goal is to find the minimum-cost path from s to t (considering both arc and turn costs). We denote the length of this path by dist (s; t). As in real-world road networks, s and t are not necessarily vertices, but points located anywhere along the arcs. These can be thought of as addresses within streets.

Without turns, this problem can be solved by Dijk-stra's algorithm [3], which scans vertices in increasing order of distance from s and stops when t is pro-cessed. We can run Dijkstra's algorithm on the turn-aware graph by associating distance labels to entry points instead of vertices [18]. An alterna-tive approach (often used in practice) is to operate on an expanded graph G^0, where each vertex corresponds to an entry point in G, and each arc represents the concatenation of a turn and an arc in G. This allows standard (non-turn-aware) algorithms to be used, but roughly triples the graph size [18]. In contrast, the turn-aware representation is almost as compact as the simplified one (with no turns at all), since identical turn tables can be shared among vertices. For tech-nical reasons, however, hierarchical methods such as contraction hierarchies [14] tend to have much worse performance on this representation [18].

B. Points of Interest

We focus on applications that deal with POIs, such as tourist attractions or store locations. In computational terms, each POI p is simply a location along an arc of the road network. We say that such an arc contains a POI, or simply that it is a POI arc. We denote the set of candidate POIs (for a given query) as P. All problems may also be parameterized by an integer k (with 1 k jPj) indicating the maximum number of POIs that are to be reported in any query.

Fig. 1: Viewing point of interest pointing

We consider two problems with these inputs. In the k-closest POI problem, we are given a source s and must compute the set of k POIs pi from P that minimize dist (s; pi). In the k-best via problem, we are given a source s and a target t, and must compute the set of k POIs pi from P that minimize dist (s; pi)+ dist (pi; t). To solve these problems, we propose algorithms that work in up to four phases, each potentially taking the outputs of previous phases as additional inputs. The first phase is metric-independent preprocess-ing, which takes as input only the graph topology. The second phase, customization, takes as input the metric (cost function) that defines the cost of each arc. The third phase, selection (or indexing), processes the set P of candidate POIs, given k. Finally, the query phase takes as input a source s (and potentially a target t) and computes the best POIs among those in P. Some algorithms may conflate two or more phases into one. Different applications may impose different con-straints on each phase. For example, in the static variant of our problems, the metric (cost function) is known in advance, and does not change often. In the dynamic version, the cost function can change frequently (to account for real-time traffic information or individual user preferences, for example). This en-ables a richer user experience, but restricts the amount of time the customization phase can spend.

Orthogonally, our problem may be offline or online, depending on when the set P of candidate POIs is known to the routing engine. In the offline version, the set P is known in advance. This happens in a typical store locator feature found in websites for large chains: the set of all stores is known in advance, and users just want the closest to their current location.

C. Customizable Route Planning

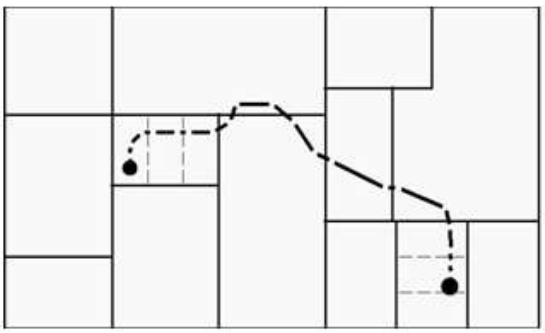
We now explain the multilevel overlays approach for computing point-to-point shortest paths on road networks. It computes multiple nested over-lay graphs, each consisting of a subset of the original vertices with additional arcs created to preserve the distances between them. More concretely, we focus on the customizable route planning (CRP) algorithm [18], [19], the fastest variant of this method.

III. ONLINE QUERIES

1. Generalized Multilevel Dijkstra

Before getting to specific applications, we introduce a general framework to analyze elaborate queries. We consider queries in which distances to several points (not just s and t) are relevant; these are the POIs.

We consider an abstract setting in which each non-trivial cell (on level 1 or higher) is labeled either safe or unsafe. (Level-0 cells, corresponding to individual vertices, are always safe.) We define a safety assignment to be valid if no unsafe cell has a safe super cell. (In other words, all ancestors of an unsafe cell must be unsafe as well.) We say that a cell is active if it is safe but has an unsafe super cell. We say that a vertex is active if it is a boundary vertex of an active cell, and that an arc is active if both of its endpoints are active. The active graph consists of all active vertices and arcs. Note that every vertex in the original graph belongs to exactly one active cell. If the input graph is disconnected, so is the active graph.



In this setting, a generalized MLD search consists of running Dijkstra's algorithm on the active graph. We define forward, backward, and bidirectional versions of this in the natural way.

Fig 2 : Active graph and CRP path for an s–t query using the 2-level partition from Figure 1. All cells and vertices drawn are active; active arcs are omitted

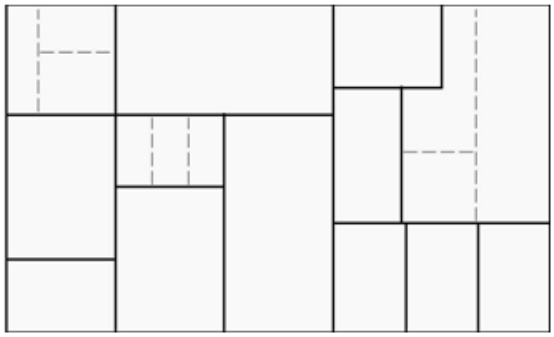


Fig. 3: Active graph for a set of POIs. The 2-level partition is the one from Figure 1. Active cells and vertices are drawn, but active arcs are omitted.

With the notion of the active graph at hand, the standard CRP s–t point-to-point query can be described as a generalized MLD search in which only the nontrivial cells that contain s and t are unsafe. Figure 5 gives an example. To analyze more complex queries (beyond point-to-point), we first show that the active graph is an overlay for all active vertices.

2. One-to-Many Queries

We first consider the one-to-many problem: given a source s and a set of POIs P, compute the distance from s to all vertices in P (not just the closest).

The trivial solution to this problem is to run Dijkstra's algorithm from s (in G) until all vertices in P are scanned. We can use the multilevel approach to accelerate this algorithm, with the help of Theorem 1. During the selection phase, we mark all nontrivial cells containing either a POI or s as unsafe (see Figure 6); we then run a generalized MLD search from s with this assignment. The final distance labels of the POIs are the answer to the one-to-many problem.

3. Finding Closest POIs

One-to-many queries can also be applied to the k-closest POI problem. We simply run the selection phase on the set P of POIs, run a one-to-many query, and then pick the k POIs with the smallest distances.

In practice, we can do better by running a restricted version of the one-to-many query. Since we scan POIs in increasing order of distance from s, we can stop as soon as we are about to scan the k-th POI. As our experiments will show, this technique is

surprisingly effective in practice, and POI queries (for small values of k) tend to be even faster than point-to-point queries, since they are usually local. Although in adversarial POI distributions the pruned search is not much faster than a full one-to-many query, our experiments will show that automatic descent is quite efficient for distributions that occur in practice.

4. Best Via POIs

We now address the k-best via POIs problem. Given a source s, a target t, and a set P of POIs, we must find the k POIs pi 2 P that minimize dist (s; pi)+dist (pi; t).

This can be solved with two one-to-many queries. We use a forward query to find the distances from s to P, and a backward query to find the distances from P to t. This provides all the information necessary to compute dist (s; pi) + dist (pi; t) for every POI pi.

IV. INDEX-BASED APPROACHES

We now consider an index-based approach. Compared to automatic descent, it provides a different trade-off: much faster queries, but worse selection times and space requirements. As our experiments will show, selection is still quite fast (a few seconds sequentially), making the indexing applicable not only offline scenarios, but also in some online scenarios.

Recall that automatic descent may be slow because it must visit all cells that contain POIs, and they may be numerous. The index-based approach avoids this by precomputing (at selection time), for every cell containing a POI, the information that the automatic descent approach would learn at query time. This in-formation (the index) is then associated with elements (arcs or boundary vertices) of the cell itself. A query can then gather all the information it needs without descending into the cell. Since the selection phase no longer needs to mark POI cells as unsafe, the number of vertices visited during the POI query is similar to that of a point-to-point query, regardless of the number (or location) of POIs in the system.

We propose two variants of this approach: single-source indexing for the closest POI problem, and double-source indexing for the best via path problem

A. Single-Source Indexing

We first consider single-source indexing, an acceleration to the closest POI problem. This idea is related (but somewhat different) to the bucket-based approach developed in the context of one-to-many computa-tions using hierarchical speedup techniques (see Sec-tion 5 for more details). This section discusses how our approach as applied to multilevel overlays. To index a cell C, we associate with each entry point v of C a bucket B(v) containing the k POIs p 2 P within cell C minimizing the distance dist (v; p), together with the distances themselves; if C has fewer than k POIs, the bucket includes all. See Figure 4.

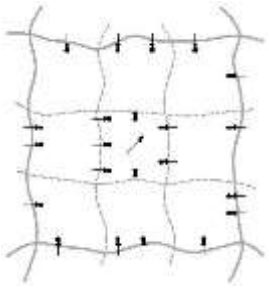


Fig.4. Single-source indexing: we add p to the buckets of the entry vertices of p's cells. Level-1 cells are indicated by dotted lines, solid lines show level-2 cells.

With this index in place (its efficient construction will be discussed shortly), we can accelerate k-closest POI queries from any source s. The query is a forward generalized MLD search from s in which only the cells containing s are marked as unsafe. It is a standard search, with minor adjust-ments. First, we maintain a list L (initially empty) containing the best k candidate POIs found by the al-gorithm so far. Moreover, before scanning each vertex v, we examine each entry (pi; dist (v; pi)) in the associated bucket B(v). We compute the tentative distance from s to pi through v (given by dist (s; v)+dist (v; pi)), and add pi to L if it is among the lowest k seen so far. (This involves adding a new entry or replacing another, possibly associated with pi itself.)

B. Double-Source Indexing

We now discuss double-source indexing, a strategy to accelerate k-best via node queries.

Unlike in the single-source scenario, where we associate buckets with vertices, double-source indexing associates a bucket B(v; w) to each arc (v; w) in the graph. If (v; w) is an original arc in G, B(v; w) contains the POIs assigned it. If (v; w) is a shortcut for a cell C, B(v; w) has the best k POIs pi within C for (v; w), i.e., it contains what would be the k via POIs for v–w if C were the entire graph. In either case, the entry corresponding to POI p in bucket B(v; w) also holds the actual length of the shortest v–p–w path (restricted to the cell). See Figure 8 for an illustration. As Section 5 will explain, this approach is related to the double-hub

indexing strategy [34] introduced for hierarchical speedup techniques.

1. Queries

An indexed s–t via query works as follows. First, we mark only the nontrivial cells containing s and t as unsafe, then execute two simultaneous generalized MLD searches: a forward search from s and a backward search from t (we can alternate between the two searches in any way). For each active vertex v, we will have two distance labels (ds(v) and dt(v)), representing the exact distances from s and to t, re-spectively. These labels are initially infinite, except for ds(s) and dt(t) (which are zero). During the algorithm, we maintain a list L with the k POIs leading to the shortest via paths found so far.

2. Indexing

We now turn our attention to the selection (indexing) phase. For each shortcut (v; w) in the overlay graph, we must build a bucket B(v; w) containing the k best via points between v and w within (v; w)'s cell.

The straightforward approach is POI-based indexing. We initialize all buckets as empty. We then pro-cess each original POI arc (a; b) by running a forward MLD search from b and a backward MLD search from a; both searches can be pruned at the boundary of the level-L cell containing (a; b) (the top level does not need to be visited in full). For each cell C that contains (a; b) (of which there are up to L), we consider all pairs (v; w) of entry and exit points in C, adding (a; b) to B(v; w) with value dist C (v; a) + `(a; b) + dist C (b; w). (Here dist C indicates the distance restricted to cell C.) Since it requires a separate search from each POI arc, POI-based indexing can be costly.

3. Hybrid Approaches

The indexing techniques introduced in Sections 1 and 2 have faster queries than the automatic descent approach introduced in Section 3, at the cost of significant more effort spent at selection time and higher space usage. For a smoother trade-off, we can use a hybrid approach: index only the lower q levels (for some q), and use automatic descent above that. The query algorithm is still generalized MLD, but with POI cells marked as unsafe only if they are above level q (or contain s or t). Indexed cells are safe. This approach works for the k-closest POI and k-best via POI problems.

Finally, the fastest point-to-point algorithm, HL, computes the distance between two random points in well below one microsecond [15], [16]. Since these queries are so fast, it is often feasible to run HL queries from s (and t) to all POIs, and then, like for RPHAST, pick the k best POIs among those. This approach can be accelerated by preselecting a (conservative) set of POIs based on Euclidean distances.

4. Applications

The applications for POI are extensive. As GPS-enabled devices as well as software applications that use digital maps become more available, so too the applications for POI are also expanding. Newer digital cameras for example can automatically tag a photograph using Exif with the GPS location where a picture was taken; these pictures can then be overlaid as POI on a digital map or satellite image such as Google Earth. Geocaching applications are built around POI collections. In Vehicle tracking systems POIs are used to mark destination points and/or offices to those users of GPS tracking software would easily monitor position of vehicles according to POIs.

V. EXPERIMENTS

We now present an experimental evaluation of our algorithms. Our code is written in C++ and compiled with Microsoft Visual C++ 2012. Our test machine runs Windows Server 2008 R2 and has 96 GiB of DDR3-1333 RAM (of which we use less than 16 GiB) and two 6-core Intel Xeon X5680 3.33 GHz CPUs, each with 6 64 KB of L1, 6 256 KB of L2, and 12 MB of shared L3 cache. All runs are single-threaded.

Time Complexity:

Time complexity of topological sorting is O(V+E). After finding topological order, the algorithm process all vertices and for every vertex, it runs a loop for all adjacent vertices. Total adjacent vertices in a graph is O(E). So the inner loop runs O(V+E) times. Therefore, overall time complexity of this algorithm is O(V+E).

TABLE 1

	PREPROCESS		CUSTOM		SELECTION		4-CLOSEST		ALL	
algorithm	space [MiB]	time [s]	space [MiB]	time [s]	space [MiB]	time [s]	#scanned vertices	time [ms]	#scanned vertices	time [ms]
Greedy Algorithm	—	—	—	—	—	—	7098	1.40	29 297 561	13 347.12
Dijkstra	—	—	—	—	—	—	7096	1.95	29 297 531	13 367.92
RPHAST	1 519	8 170.1	—	—	32.18	0.28	651 974	11.98	651 974	11.95
HL	64 396	11 652.3	—	—	—	—	—	7.26	—	7.23
BHL	64 396	11 652.3	—	—	13.72	0.21	—	0.01	—	2.16
CRP no index	3 119	5 190.6	71.0	3.9	0.00	0.01	1 626	0.64	12 319 771	8 081.23
CRP reverse index	3 119	5 190.6	71.0	3.9	33.27	9.12	443	0.17	3 933	6.77
CRP bulk-4 index	3 119	5 190.6	71.0	3.9	5.73	2.20	443	0.17	—	—
CRP reverse index-2	3 119	5 190.6	71.0	3.9	3.53	1.78	451	0.17	198 602	99.31
CRP bulk-4 index-2	3 119	5 190.6	71.0	3.9	3.53	1.69	451	0.17	—	—

REFERENCES

[1] Daniel Delling, Renato F. Werneck, "Customizable Point-of-Interest Queries in Road Networks", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. X, NO. Y, JANUARY 2015.

[2] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, vol. 1, pp. 269–271, 1959.

[3] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck, "PHAST: Hardware-Accelerated Shortest Path Trees," Journal of Par. and Dist. Computing, vol. 73, no. 7, pp. 940–952, 2013.

[4] Z. Chen, H. T. Shen, X. Zhou, and J. X. Yu, "Monitoring Path Nearest Neighbor in Road Networks," in SIGMOD. ACM Press, 2009, pp. 591–602.

[5] H.-J. Cho and C.-W. Chung, "An Efficient and Scalable Ap-proach to CNN Queries in a Road Network," in VLDB, 2005, pp. 865–876.

[6] H. Hu, D. Lee, and J. Xu, "Fast Nearest Neighbor Search on Road Networks," in EDBT, ser. LNCS, vol. 3896. Springer, 2006, pp. 186–203.

[7] C. S. Jensen, J. Kolar,´ T. B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," in SIGSPATIAL GIS. ACM Press, 2003, pp. 1–8.

[8] M. Kolahdouzan and C. Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases," in VLDB, 2004, pp. 840–851.

[9] H. Samet, The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1989.

[10] S. Shekhar and J. S. Yoo, "Processing In-Route Nearest Neigh-bor Queries: A Comparison of Alternative Approaches," in SIGSPATIAL GIS. ACM Press, 2003, pp. 9–16.

[11] S.-H. Shin, S.-C. Lee, S.-W. Kim, J. Lee, and E. G. Lim, "K-Nearest Neighbor Query Processing Methods in Road Net-work Space: Performance Evaluation," in ICINC, 2009, pp. 958–962.

[12] D. Delling, A. V. Goldberg, and R. F. Werneck, "Faster Batched Shortest Paths in Road Networks," in ATMOS, ser. OASIcs, vol. 20, 2011, pp. 52–63.

[13] R. Geisberger, "Advanced Route Planning in Transportation Networks," Ph.D. dissertation, KIT, February 2011.

[14] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact Routing in Large Road Networks Using Contraction Hierar-chies," Transportation Science, vol. 46, no. 3, pp. 388–404, 2012.

[15] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks," in SEA, ser. LNCS, vol. 6630. Springer, 2011, pp. 230–241.

[16] "Hierarchical Hub Labelings for Shortest Paths," in ESA, ser. LNCS, vol. 7501. Springer, 2012, pp. 24–35.

[17] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, "Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm," ACM JEA, vol. 15, no. 2.3, pp. 1–31, 2010.

[18] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, "Customizable Route Planning," in SEA, ser. LNCS, vol. 6630. Springer, 2011, pp. 376–387.

[19] D. Delling and R. F. Werneck, "Faster Customization of Road Networks," in SEA, ser. LNCS, vol. 7933. Springer, 2013, pp. 30–42.

[20] M. Holzer, F. Schulz, and D. Wagner, "Engineering Multi-Level Overlay Graphs for Shortest-Path Queries," ACM JEA, vol. 13, no. 2.5, pp. 1–26, December 2008.

