e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

Evaluation of Shortest Path and Distance Queries on Road using PRP

Ambati Naga Sai Durga & J. Rajanikanth
[1]M.Tech, Department of Computer Science and Technology, SRKR Engineering College, Bhimavaram, West Godavari, Andhra Pradesh, India.
[2]Assistant Professor, Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, West Godavari, Andhra Pradesh, India.

Abstract-- *We research a scenario for route path in road networks, where the aim to be optimized may change between every shortest path query. Since this invalidates many of the known speedup techniques for road networks that are based on data pre-processing of shortest path structures, we investigate optimizations exploiting simply the topological structure of networks. We experimentally estimate our technique on a large dataset of real-world road networks of various data sources. With lightweight data pre-processing our technique response long distance queries across continental networks significantly faster than previous approaches towards the same problem formulation.*

Keywords:- PRP; preprocessing; road networks;

1. Introduction

On such large networks, Dijkstra's classical shortest path algorithm incurs substantial running times of several seconds even on modern computer hardware. This is too slow for many applications such as navigation, route planning, location-based services, range and trajectory queries, k-nearest-neighbor search, and other queries on spatial network databases. Hence, the past decade has seen numerous research (by both theoretical and applied communities) into techniques that accelerate shortest path queries. For an overview see the recent surveys. Assuming that the graph metric is fixed or does not change too often, these techniques offer very fast queries at considerate preprocessing effort, enabling route planning services that serve millions of users per day. However, if instead costs change for every query, these techniques cease to provide benefit over Dijkstra's algorithm. Yet, in practice, even the same user might prefer a quickest route in the morning but a safe and fuel-efficient route back home.

Here, every arc in the road graph is associated with a vector c of several non-negative numeric costs such as for example travel time, distance, speed, emissions, and energy consumption. The input of a query, in addition to the source and the target node, consists of a cost vector w with non-negative entries. In the search, every arc is associated with the scalar product of w and c. The output consists of the shortest path with respect to this weighted sum of costs. Solving the PRP problem efficiently seems very useful in order to construct route planning services that adapt to the individual needs of every person. Unfortunately, in practice not all routing constraints can be modeled as a linear combination of additive costs. For example, summing up height limitations is not meaningful (i. e., a 3 m high truck will not fit through two consecutive tunnels of 2 m height). A similar observation holds for vehicle weight limitations or the limit on the maximum slope that a vehicle can climb. Further constraints are the avoidance of certain road categories, such as for example highways, city centers, or water conservation zones (which trucks with dangerous goods are not allowed to traverse). In this work, we generalize PRP to also support such restrictions.

2. Related Work

The classic solution to solving shortest path problems on road networks is Dijkstra's algorithm. Slightly faster queries are achieved by employing bidirectional search from both source and target. Heuristic search using easily available bounds (e.g., Euclidean distance) is still a common choice. However, some studies, such as , have come to the conclusion that on road networks, Euclidean distance bounds is not necessarily beneficial over Dijkstra's algorithm; it can even slightly decrease efficiency. We have witnessed similar behavior in preliminary experiments in our specific setting. Many techniques have been proposed for further acceleration. Nearly all of these divide the work into two phases: In a preprocessing phase the graph is augmented with auxiliary data that is then exploited during the query phase for faster shortest path or distance retrieval.

The PRP problem is essentially a high-dimensional, linear multi-criteria search problem, related to the parametric shortest path problem. Extensions of known preprocessing techniques to multi-criteria optimization have been proposed, but were only evaluated experimentally for the bi-criteria and tri-criteria case. Even for the three criteria of travel time, travel distance, and fuel consumption (which are even quite correlated), diminishing returns in terms of query speed over preprocessing effort have been reported.

3. Our Contribution

The primary results of our work are:

- We generalize Personalized Route Planning (PRP) to support a more rich set of restrictions. The generalization allows to model, for example, maximum vehicle heights (e. g., for tunnels) and maximum vehicle weights (e. g., for bridges) as well as user-preferences such as avoidance of highways.
- A new preprocessing-based algorithm for PRP, extending the Bidirectional Dijkstra. While we build on basic and easy to implement concepts, in combination our approach is better at PRP than the state-of-the-art.
- A key ingredient is efficient identification of topologically important core nodes, while preserving all (not just shortest) paths. our construction, which is computed optimally in time linear in the size of the input graph.
- We conduct an extensive experimental study on a large set of real-world road graphs of different data sources.
- Our algorithms achieve significantly faster personalized route planning queries than previous approaches at less preprocessing costs.
- Our query times are well below one second even on the largest instance tested for random long-distance queries. This is fast enough for a wide range of applications. Note that in practice most queries are short-distance that result in even lower query times.
- Our analysis further shows that performance gains significantly vary depending on the data source—as opposed to just the geographical instance considered. While observed before, overall it is surprisingly underreported in the literature on route planning in road networks.
- We conclude that ranking road networks just by node count is not meaningful, and cross comparisons of the performance of route planning techniques are inconclusive without careful consideration of the respective data sources used for experimental evaluation.

4. Dijkstra’s Algorithm

Dijkstra's algorithm is the textbook solution to the shortest path problem, and many modern techniques still use it as a subroutine. Fine-tuning its implementation therefore directly results in better overall running times, but it also tightens the baseline for reporting speedups. (The speedup of a technique, which is used as an indication of machine-independent performance, is measured in terms of its query speed in relation to an implementation of Dijkstra's algorithm.). To ensure reproducibility of our experimental findings, we document details of our implementation and the reasoning behind the choices we made, as much as space allows.

Node Orders:-

Node data is usually stored as a large array and the nodeIDs correspond to the offset in this array. A small IDdifference therefore implies a high likelihood that the data of both nodes is loaded simultaneously into the cache. Dijkstra's algorithm works by accessing the memory attached to the two endpoints of an arc directly after another. If both are in cache, memory access times decreases. To illustrate this influence we consider three node orders as in [6]: (a) random order, (b) input order, and (c) DFS pre-order. A random order performs the worst as it does not have much locality. The quality of the input order solely depends on the data source. Usually it has some locality as nodes often appear in the order that they were added to the dataset and adjacent nodes are often added successively. The DFS pre-order consists of picking a random root node and running a depth first search. Nodes get ordered in the way they are first visited. Every node with pre-order ID i that is not the root or a leaf in the tree (i.e. the vast majority of the nodes) will have two neighbors with directly adjacent node IDs: The parent node has ID $i - 1$ and the first child has ID $i + 1$.

Bidirectional Dijkstra's:-

Dijkstra's algorithm works by visiting all nodes around the source node increasing by distance until the target node is reached. A speedup can be gained by visiting the nodes around the source and the target node simultaneously. The central idea consists of running two instances of Dijkstra's unidirectional algorithm simultaneously. The first search explores the nodes close to the source node, while the other explores the nodes around the target node. Once a node is reached by both searches, a (not necessarily shortest) path is found. Denote by μ the length of the shortest path found so far. Further denote by dF the distance of the next node in the forward instance's queue and by dB the distance of the next node of the backward instance. We abort the search once $\mathbf{d_f} + \mathbf{d_b} \geq \boldsymbol{\mu}$, as any path that we find from that point on, has a distance of at least μ. Several alternation strategies exist that decide from which of the two queues a node should be popped and processed: The strategy alternation (alt) switches each step between forward and backward search. The min-key strategy (mk) picks the forward search if $\mathbf{d_f} \leq \mathbf{d_b}$. The min-queue-size strategy (mq) picks the forward search if the backward queue size is not smaller than the forward queue size. Note that if the considered graph is directed, the backward search must operate on the reversed graph instead of the input graph.

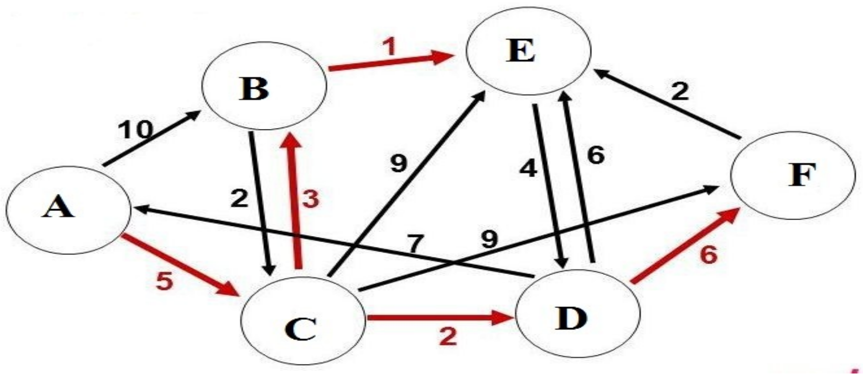

Fig: - Bidirectional Graph

A Bidirectional Dijkstra's is a preprocessing-based technique to accelerate shortest path queries. In the preprocessing phase a core graph $G_C = (V_C, A_C)$ is computed. Think of this core graph as a coarsened sub graph containing all major roads. The query phase is a Bidirectional Dijkstra's algorithm. Conceptually, it first searches locally around the source and the target nodes until the core is reached on both sides. From there on the search is restricted to the core graph. This decreases query times because G_C is smaller than G and therefore only parts of the graph have to be searched. Formally the nodes V_C of G_C are a subset of V and called core nodes.

The arcs of the core are defined as following: For every loop-free path $v_1, v_2 \ldots v_k$ for which only the endpoints v_1 and v_k are in V_C and all intermediate nodes are in $V \setminus V_C$, there exists a shortcut arc (v_1, v_k) ε A_C in the core graph. It is possible that multi-arcs are created by this construction. The cost vector $c(v_1, v_k)$ of the shortcut is defined as the combination of the cost vectors of the arcs within the path, i.e., $c(v1, v_k) = c(v_1, v_2)$ _ . . . _ $c(v_{k-1}, v_k)$.

Given a core graph we compute a forward and a backward search graph as follows:

1) The forward graph G_F is the union of G and G_C without the arcs (u, v) that leave the core, i.e., u ε V_C and v ε V \\V_C.
2) The backward graph G_B is constructed analogously: First compute the union of G and G_C, then reverse the direction of every arc and finally remove the arcs leaving the core. The query phase is a bidirectional of Dijkstra's algorithm. The forward search is run on G_F while the backward search runs on G_B. We abort the search if $\mathbf{d_f} + \mathbf{d_b} \geq \boldsymbol{\mu}$, where μ is the tentative distance, and no queue contains a non-core node.

5. Results analysis

Dir	Time[ms]			Nodes Popped from queue
	Random	Input	DFS	
Uni	470	265	223	1539k
Bi-directional	302	171	143	900k

Table 6: Query running time and number of queue pop-operations for variants of Bidirectional Dijkstra's

Query running time and number of queue pop-operations for Bidirectional Dijkstra's algorithm on the graph for the general PRP problem. "random", "input" and "dfs" are the node orders considered. They vary in terms of running time because of cache-effects but not in terms of pop-operations. "uni" and "bi-directional" are the alternation strategies. The performance of Bidirectional Dijkstra's algorithm in its unidirectional and bidirectional variants and with all three node orders. Overall, bidirectional search with minimum-queue-size alternation strategy yields the best query performance, consistently about 55% faster than unidirectional search. Additionally, DFS-reordered nodes improve query times by 19–23 %, compared to the input order.

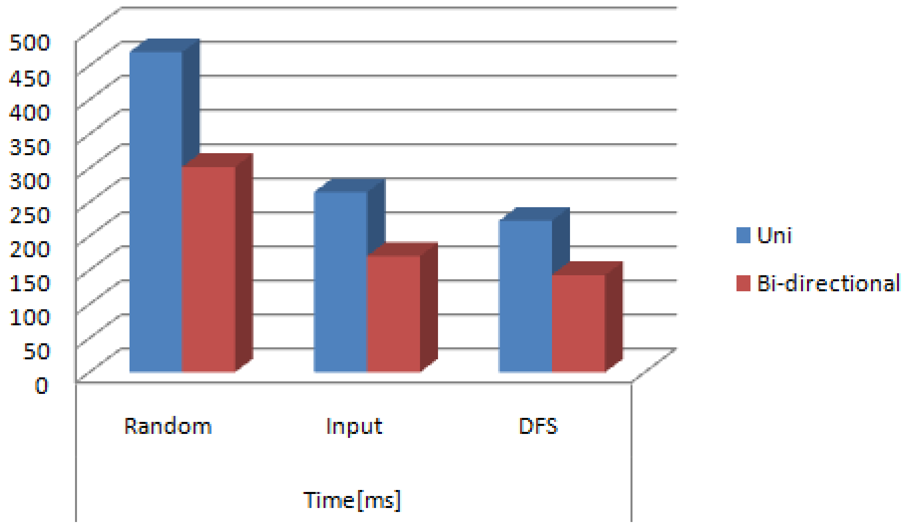

Fig:- Query running time and number of queue pop-operations for variants of Bidirectional Dijkstra's

6. Conclusions

We evaluated a preprocessing-based speedup technique for faster Personalized Route Planning. On all tested instances - which include very large-scale networks with hundreds of millions of nodes - we were able to achieve running times well below a second. This is fast enough for many applications, including web services of moderate user base. The main advantage of the Personalized Route Planning is that costs are individually adjusted for every user and every query in a very flexible way. Rerunning preprocessing is only necessary when roads are build or cost vectors are adjusted (e. g., a new speed limit is posted).

7. References

[1] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. HLDB: Location-based services in databases. In Proceedings of the 20th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems (GIS'12), pages 339–348. ACM Press, 2012. Best Paper Award.

[2] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension and provably efficient shortest path algorithms. 2013.

[3] H. Bast, D. Delling, A. V. Goldberg, M. Müller–Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. Technical Report abs/1504.05140, ArXiv e-prints, 2015.

[4] B. Bresar, F. Kardos, J. Katrenic, and G. Semanisin. Minimum k-path vertex cover. Discrete Applied Mathematics, 159(12):1189–1195, 2011.

[5] G. B. Dantzig. Linear Programming and Extensions. Princeton University Press, 1962.

[6] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck. PHAST: Hardware-accelerated shortest path trees. Journal of Parallel and Distributed Computing, 73(7):940–952, 2013.

[7] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Robust distance queries on massive networks. In Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014.Proceedings, pages 321–333, 2014.

[8] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning in road networks. Transportation Science, 2015.

[9] D. Delling, M. Kobitzsch, and R. F. Werneck. Customizing driving directions with GPUs. In Proceedings of the 20th International Conference on Parallel Processing (Euro-Par 2014), volume 8632 of Lecture Notes in Computer Science, pages 728–739. Springer, 2014.

[10] D. Delling and D. Wagner. Pareto paths with SHARC. In Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09), volume 5526 of Lecture Notes in Computer Science, pages 125–136. Springer, June 2009.