h e g

Haute école de gestion de Genève
CRAG - Centre de Recherche Appliquée en Gestion
Cahier de Recherche

On a many-to-one shortest paths for a taxi service

Sacha Varone

Cahier : N° HES-SO/HEG-GE/C–14/1/1-CH

2014

On a many-to-one shortest paths for a taxi service

Sacha Varone

Cahier de recherche

March 2014

Résumé

Nous décrivons un algorithme de type Dijkstra appliqué à des cartes géographiques, nécessaire à une application à temps réel pour une compagnie de taxis. Le problème qui est résolu consiste en une demande de client reu par une société de taxi, qui doit répondre à cette demande en affectant l'un de ses taxis, tel qu'un temps d'attente maximal ne soit pas dépassé pour prendre en charge le client. La sélection du taxi est basée sur des plus courts chemins, calculés à partir de l'emplacement de la demande jusqu'au taxis. La méthode peut être appliquée à d'autres algorithmes de plus courts chemins que l'algorithme de Dijkstra. Les avantages de cette approche comprennent la possibilité d'intégrer facilement des informations de trafic en temps réel, ainsi que la rapidité d'exécution de l'algorithme.

Mots-clés

Plus courts chemins, cartes géographiques, application temps réel

Summary

Real application involving routing algorithms for moving vehicles needs accurate information, especially for micro mobility. In this note, we describe a reverse bounding Dijkstra algorithm using geographical maps, geographical coordinates of vehicles and pick-up point for a taxi fleet application. The problem that is solved consists in a customer request received by a taxi company, which has to satisfy this request with one of his nearby taxi, restricted to a maximal waiting time for the customer. The selection of the taxi to be assigned to this request is based on many-to-one shortest paths, computed reversly from the location of the request. The methodology can be applied to other shortest paths algorithms than the Dijkstra algorithm. Benefits from this approach include real time information associated with the underlying geographical data to get accurate estimated time to pick-up.

Keywords

Shortest paths, geographical maps, real-time application

1 Introduction

Shortest paths algorithms are extensively studied, since they are used in almost all real cases of transportation problems. Efficient algorithms are especially needed when geographical information systems (GIS) are involved. There exists a special dedicated group, called GIS-T.

A very famous shortest path algorithm is the Dijkstra algorithm [1]. [2] note that it can in theory be used for finding the shortest paths on road networks but for large networks it would be far too slow. Different speed-up techniques, such as bidirectional search, goal direction, etc., can be used to improve performance. An overview of such techniques is available in [2]. Most of shortest paths research is is focused on long distance shortest paths computations, on a map of continental size.

Efficient ones are based on preprocessing, alike the so-called Highway Hierarchy procedure, first published in [3] and [4], which iteratively compact the based graph: neighborhood are defined around origin and destination points. Then, only arcs whose endpoints are not within a single neighborhood are kept, which results in a new compact graph. And this process is iterated several times. The resulting hierarchy of highway networks is then used in a Dijkstra-like bidirectional query algorithm to considerably reduce the search space size without losing exactness.

Contraction hierarchies is another method of simplifying shortest-path routing by first creating precomputed contracted versions of the connection graph. It can be regarded as a special case of highway-node routing. Is is discussed in [5] and [6]. [6] states that contraction hierarchies rely entirely on bypassing, that is contraction of, nodes. Contrary to similar speed-up techniques, such as highway hierarchies, which only bypass a subset of the nodes, contraction hierarchies contract the whole graph. The resulting hierarchy contains many shortcut edges bypassing nodes. A bi-directional search then makes use of those shortcut edges, skipping most nodes in the graph in the process. This results in one of the most efficient speed up techniques. [6] implements a fully functioning mobile routing application called MoNav[1] which is based on contraction hierarchies using the freely available data from OpenStreetMap[2] (OSM). Hub-labelling [7] algorithm is another theoretically very efficient method. Recently, [8] give the first rigorous proofs of efficiency for some of the most efficient shortest path algorithms.

Another service based on contraction hierarchies is the Open Source Routing Machine (OSRM), a C++ implementation of a high-performance routing engine for shortest paths in road networks using data from OSM. It boasts very fast query times, usually below 1 ms for data sets like Europe. Since it is designed with OpenStreetMap compatibility in mind, OSM data files can be easily imported.

In this paper, we solve the following problem: a fleet of taxis belongs to a company which centralises all the customers' requests. A customer calls a unique number, and a dispatcher selects a taxi from those located in the neighbourhood of this customer, and orders him to pick-up the customer. The precise rule of assignment depends on internal rules of the taxi company, but often it is based on a quickest possible pick-up, so as to minimize the customer's waiting time until it is served.

Using the aforementioned methods of highway hierarchies or contraction hierarchies is not

[1] http://code.google.com/p/monav/
[2] http://www.openstreetmap.de/

suitable for micro-mobility: since shortest paths are searched through a compact graph resulting from shrinking actions on the original graph, it is not practically possible to incorporate other useful information such as temporary road restrictions, traffic jam, temporal speed limitations, ... Moreover, in our considered application to taxis management and optimization, a special request is to restrict the search paths to points reachable within a maximum time duration.

Although this situation might be handelt using n runs of Dijkstra algorithm, one for each of the n taxis in the fleet to the customer, a more efficient way to solve the problem is to start from the customer itself and propagate shortest paths search through an expansion algorithm. The advantage of starting from the customer to construct shortest paths is the ability to avoid multiple runs of shortest path algorithms, if starting from vehicles and applying for each of them the Dijkstra algorithm.

In this note, we explain in section 2 how to bound the search for a shortest path algorithm so that the duration of the path is no longer than a predefined value. We then explain how to solve the many-to-one shortest path problem using only one run of a shortest path algorithm. Section 3 explains how to use road maps and geographical coordinates from a database so that real applications can be solved on real distances and real time. Indeed, taxi locations and customer locations can now be given by GPS coordinates through smartphones or GPS devices, or simply orally. It is therefore useful to work directly with up-to-date coordinates and maps instead of using predefined distance metric spaces, in order to compute the relative positions of taxis and customers. Section 4 explains how we applies the concepts defined previously and which tools we use to build our prototype. Last section 5 mentions some generalization, the limitations as well as possible extensions to our method.

Geospatial database query The problem we consider is close to the k-Nearest Neighbours (k-NN) problem, where the goal is to find the k nearest objects closest trajectories to a query object. Our problem is not exactly the same, since we do not know *a priori* how many objects will be returned. Quite a few solutions for the k-NN involve the use of R-tree structure, like in [9, 10]. In continuous k-NN query, we mention for example the results of [11, 12], or [13] who claim to have very good result with a set of moving object trajectories indexed in a 3D-R-tree structure. Probabilistic search is done in [14], which also show that some queries may not be solved in polynomial time. Several authors propose an index method to facilitate the search for nearest taxis. Those methods have in common the use of an updated spatio-temporal list of nearby taxis, as well as a grid partition of the road network. The computation of a distance metric between all elements of the grid might done as a pre-computing step. In [15], the approach avoid predictive queries, but instead use grid indices for objects and queries. They claim that their approach outperforms R-tree based solutions, along with the advantage of being scalable. Several authors have since followed this approach for k-nearest neighbor queries, e.g. [16, 17]. To have an overview on k-NN queries, the reader is refered to [18].

2 A reverse bounded Dijkstra algorithm

In this section, we explain how to find shortest paths within a maximum length from a set of points to a single source. Would the source not being reachable from some of the points in the set, then those points are not visited, and hence marked with an infinite estimated value. The

application of this concept is to know which taxis from a fleet (the set of vertices) are within a maximal waiting time for a customer (the source vertex) before being eventually served.

In the following, we adopt the formulation available in [19]. We use a graph $G(V, A)$ where V is a set of vertices, A is a set of arcs, w is a positive weight function on A. The weight function might be a distance, a duration, a cost, etc. In our case, the weight function is ht time needed to traverse an arc. We associate for each $v \in V$ a shortest-path estimate $v.d$, which is an upper bound on the shortest path from a source s to v, and $v.\pi$, which is a predecessor of v on a shortest path from s to v.

Algorithm 1 Initialize-single-source(G, s)

Require: Graph $G = (V, A)$, source s
Ensure: Initialization of the directed graph G
  for all $v \in V$ **do**
    $v.d = \infty$
    $v.\pi = nil$
  end for
  $s.d = 0$

Algorithm 2 Relax(u, v, w)

Require: Vertices u, v and a weight w
Ensure: Update of d and π
  if $v.d > u.d + w(u, v)$ **then**
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
  end if

Algorithm 3 Dijkstra(G, s)

Require: Graph $G = (V, A, w >= 0)$, source s
Ensure: The single-source s shortest-paths problem on a weighted, directed graph G
  Initialize-single-source(G, s) from Algorithm 1
  $S = \emptyset$
  $Q = V$
  while $Q \neq \emptyset$ **do**
    $u =$ extract-min(Q) from Algorithm 2
    $S = S \cup \{u\}$
    for all v such that $(u, v) \in A$ **do**
      Relax(u, v, w)
    end for
  end while

2.1 A bounded shortest path algorithm

A first modification of a one-to-many algorithm like the Dijkstra algorithm 3 consists in giving an upper-bound on how far away from the source node s should be any shortest paths. Such

limitation is particularly useful in case of huge graph resulting from a map covering a large portion of territory. Indeed, instead of computing shortest paths from s to all points, and then using only some points at the vicinity of s, savings of computation time by limiting the spread of a shortest path algorithm allows to very quickly find those points. Algorithm 4 is based on the Dijkstra algorithm 3, and add a stopping criterion at line 6 so that only shortest paths from s to vertices that are at most at a given distance L are computed.

Theorem 2.1 *The Bounded Dijkstra algorithm 4 works correctly*

Proof It has to be shown that all vertices within L distance from s are visited. Equivalently, we will show that all non visited vertices are at a distance greater than L from s. Dijkstra algorithm 3 works correctly.
Line 6 stops it before achievement.
At this point, vertex u is the vertex closer to s, that has still not been visited.
Therefore, among all points that remains in Q, there is no one whose shortest path from s is less than $u.d$.
Since the Relax algorithm 2 never decreases $v.d$ for all vertices $v \in Q$, $v.d \geq u.d$ and can not be shorter than $u.d$ in any future iteration.
Therefore all vertices lying at most at distance L from the source s are already visited.

Algorithm 4 BoundedDijkstra(G, s, L)

Require: Graph $G = (V, A, w >= 0)$, source s, length L
Ensure: The single-source s shortest-paths problem on a weighted, directed graph G, restricted to an upper bound length L.

```
1: Initialize-single-source(G, s) from Algorithm 1
2: S = ∅
3: Q = V
4: while Q ≠ ∅ do
5:    u = extract-min(Q)
6:    if u.d > L then
7:       STOP
8:    end if
9:    S = S ∪ {u}
10:   for all v such that (u, v) ∈ A do
11:      Relax(u, v, w) from Algorithm 2
12:   end for
13: end while
```

2.2 A reverse bounded shortest path algorithm

In order to consider many-to-one shortest paths, we redefine the notations as follows:
We associate for each $v \in V$ a shortest-path estimate $v.d$, which is an upper bound on the shortest path from v to s, and $v.\pi$, which is a successor of v on a shortest path from v to s. The trick is to take all arcs in the opposite sense, so that shortest path from any vertices to s

are computed. We also restrict the search to vertices that are at most at a given distance L to s. Therefore, the Reverse Bounded Dijkstra (RBD) algorithm 6 is based on the Bounded Dijkstra algorithm 4, using the opposite directions of all arcs.

Algorithm 5 ReverseRelax(u, v, w)

Require: Vertices u, v and a weight w
Ensure: Update of d and π
  if $v.d > u.d + w(v, u)$ **then**
    $v.d = u.d + w(v, u)$
    $v.\pi = u$
  end if

Algorithm 6 ReverseBoundedDijkstra(G, s, L)

Require: Graph $G = (V, E, w >= 0)$, source s, length L
Ensure: The single-source s shortest-paths problem on a weighted, directed graph G, restricted to an upper bound length L.
  Initialize-single-source(G, s) from Algorithm 1
  $S = \emptyset$
  $Q = V$
  while $Q \neq \emptyset$ **do**
    $u =$ extract-min(Q)
    if $u.d > L$ **then**
      STOP
    end if
    $S = S \cup \{u\}$
    for all v such that $(v, u) \in A$ **do**
      ReverseRelax(u, v, w) from Algorithm 5
    end for
  end while

3 Database and maps

The RBD algorithm 6 allows to know which taxis are within a given vicinity of a request location, but suppose to know the geographical map, and the positions of all taxis on the map. Among all taxis, only a subset, possibly empty, results from the RBB algorithm, since only those not farther than a predefined quantity L are selected.

Our problem do not need to continuously know the relative positions of the taxis, but only when a new request is sent. We do not maintain an updated list of grid indices, neither use a specific R-tree structure, but only use the map and the last database recorded GPS coordinates of the taxis. One way to solve this problem uses a database spatial extensions, preferably one following the specification of the Open Geospatial Consortium (see http://www.opengeospatial.org/ for more information). As we do not use such an extension, our method proceeds mainly in three steps as follows:

a) Taxi closeness estimate: estimate the distance between the request location and the taxis.

b) Taxi candidates: locate taxis within $L \cdot (1+\lambda)$ units from the request location to the map ($\lambda > 0$)

c) Taxi list: find the taxis within L units from the request location using RBD Algorithm 6.

3.1 Taxi closeness estimate

Given the GPS coordinates of two objects A and B, e.g. latitude and longitude, find an estimate of the distance between A and B. In our application, one point is the request location A and the other object is a taxi location B. This problem is known under the name of Great-circle distance, which is the shortest distance between two points on a surface of a sphere. Since our problem is practically defined on a city sized map, small distances have to be estimated. To calculate distances between two GPS coordinates, one might use the Vincenty formula [20] which uses an ellipsoidal model of the earth, or the Haversine formula, which uses a spherical model. We use the later since it was already available in the framework.

$$a = sin(\frac{\theta_2 - \theta_1}{2})^2 + cos(\theta_1) * cos(\theta_2) * sin(\frac{\lambda_2 - \lambda_1}{2})^2$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a})$$

$$d = R * c$$

where $\lambda_i, i = 1, 2$ are the latitudes, $\theta_i, i = 1, 2$ are the longitudes, $R \approx 6371$ [km] is the earth's radius.

3.2 Taxi candidates

Such an estimation is a lower bound on a road shortest path from a taxi location B and the request location A. Therefore, the selection of taxis to be considered as candidates is constraint to be at a distance no greater than $L \cdot (1 + \lambda)$, where we set λ to 2. In other words, we only consider taxis that are at an estimated distance no greater than two times the chosen neighbourhood radius. Found taxi candidates are then linked with the road map under consideration, as well as the request location.

3.3 Taxi list

This last step consists in applying the RBD Algorithm 6 on the road map G, the source s as the request location, and a chosen neighbourhood limit L. Note that this parameter L might be given for example as a maximal distance from the customer, or, more appropriately, as the maximal amount of waiting time for the customer.

4 Practical application

Several geographical maps are available for geolocalised applications like the one we built. The most famous ones are google maps[3], bing maps[4], nokia maps[5] or openstreetmap[6] (OSM). The latter is a collaborative project to develop a free editable map of the world. OSM is our choice for geographical data needed to compute road distances. OSM data for Western Europe is quite complete, readily available for download from web site like Cloudmade[7]. Complete maps containing roads segments, buildings, boundaries, ... can be downloaded, or only highways maps which refers to roads available for traffic. We use the highways map as such data file is considerably lighter in size than complete maps, and therefore quicker to process.

OSM is used in a lot of projects which facilitate map integration or exploitation, like OsmSharp[8], which is presented as "an open-source mapping tool designed to work with OpenStreetMap-data. ". We use its routing and OSM data processing library to test our shortest paths computations on real data sets. We randomly generated taxis and customers geographical coordinates within our map boundaries as (latitude,longitude) points. OsmSharp resolves the (latitude,longitude) points by finding the nearest usable road for each of them. OsmSharp, written in C#, is based on Microsoft's .NET framework. Although primarily used with Microsoft Windows operating system, it is also possible to use it in a Linux operating system. To do so, the Mono framework emulates the .NET framework and OsmSharp code can be compiled and executed successfully.

Sample testing is done with random taxi and customers location and time. They are stored in a MySQL database. To use C# and MySQL, the Oracle?s SQL Connector for .NET, available freely from Oracle, has to be downloaded and installed.

5 Conclusion

The concept developed in this note focuses on one of the most well known shortest path algorithm: the Dijkstra algorithm. Although the latter is sufficiently efficient for small network like a city-sized one, its running time for country size or continental size geographical maps is prohibitive for real-time application. As our application concerns micro-mobility, e.g. for a city-sized map, the computation time, which is less than a second in our case, the use of a Dijkstra-like algorithm is appropriate. Some refinement of the method can also be done, as the use of a two sided reverse bounded Dijkstara algorithm. Moreover, our method is also applicable to other types of shortest path algorithms on graph network, but this has not yet been tested yet.

One of the benefit from running shortest paths algorithms on complete graphs, like the Dijkstra algorithm, and not on contracted graph, as it is the cas for contraction hierarchies algorithms, is the ability to add additional information on traffic jam, perturbation, restriction access,

[3]https://maps.google.com/
[4]http://www.bing.com/maps/
[5]http://m.here.com
[6]http://www.openstreetmap.org
[7]http://downloads.cloudmade.com/
[8]http://www.osmsharp.com/

speed limitation, etc. directly associated with the edges or arcs of the graph. Doing so enable accurate information on the routing time from a point A to a point B. It might therefore be suitable to work directly on non contracted maps with real time road information for efficient solutions to real city-based applications.

6 Acknowledgment

This research was funding thanks to the CTI grant 15229-1 PFES-ES, for which the author is very thankful. This Reverse Bounded Dijkstra algorithm has been implemented by Vytenis Janilionis as part of his Master Thesis [21].

References

[1] E. W. Dijkstra, A note on two problems in connexion with graphs., Numerische Mathematik 1 (1959) 269–271. doi:10.1.1.165.7577.

[2] P. Sanders, D. Schultes, Engineering fast route planning algorithms, in: Proceedings of the 6th international conference on Experimental algorithms, WEA'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 23–36.
URL http://dl.acm.org/citation.cfm?id=1768570.1768573

[3] P. Sanders, D. Schultes, Highway hierarchies hasten exact shortest path queries, in: 13th European Symposium on Algorithms, Vol. 3669 of LNCS, Springer, 2005, pp. 568–579.

[4] P. Sanders, D. Schultes, Engineering highway hierarchies, in: 14th European Symposium on Algorithms, Vol. 4168 of LNCS, Springer, 2006, pp. 804–816.

[5] R. Geisberger, Contraction hierarchies: Faster and simpler hierarchical routing in road networks, Master's thesis, Institut für Theoretische Informatik, Universität Karlsruhe (2008).

[6] C. Vetter, Fast and exact mobile navigation with OpenStreetMap data, Master's thesis, Institute for Theoretical Computer Science, Karlsruhe University (2010).

[7] A. Goldberg, The hub labeling algorithm, in: V. Bonifaci, C. Demetrescu, A. Marchetti-Spaccamela (Eds.), Experimental Algorithms, Vol. 7933 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 4–4. doi:10.1007/978-3-642-38527-8_2.

[8] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, R. F. Werneck, Highway dimension and provably efficient shortest path algorithms, Tech. Rep. MSR-TR-2013-91, Microsoft Research (2013).

[9] E. Frentzos, K. Gratsias, N. Pelekis, Y. Theodoridis, Algorithms for nearest neighbor search on moving object trajectories, GeoInformatica 11 (2) (2007) 159–193. doi:10.1007/s10707-006-0007-7.

[10] K. Raptopoulou, A. Papadopoulos, Y. Manolopoulos, Fast nearest-neighbor query processing in moving-object databases., GeoInformatica 7 (2) (2003) 113–137, nearest Neighbors. doi:10.1023/A:1023403908170.

[11] Y.-K. Huang, Z.-W. Chen, C. Lee, Continuous k-nearest neighbor query over moving objects in road networks, in: Proceedings of the Joint International Conferences on Advances in Data and Web Management, APWeb/WAIM '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 27–38. doi:10.1007/978-3-642-00672-2_5.

[12] G. S. Iwerks, H. Samet, K. Smith, Continuous k-nearest neighbor queries for continuously moving points with updates, in: Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, VLDB '03, VLDB Endowment, 2003, pp. 512–523, nearest Neighbors.
URL http://dl.acm.org/citation.cfm?id=1315451.1315496

[13] R. H. Güting, T. Behr, J. Xu, Efficient k-nearest neighbor search on moving object trajectories, The VLDB Journal 19 (5) (2010) 687–714, nearest Neighbors. doi:10.1007/s00778-010-0185-7.

[14] J. Niedermayer, A. Züfle, T. Emrich, M. Renz, N. Mamoulis, L. Chen, H.-P. Kriegel, Probabilistic nearest neighbor queries on uncertain moving object trajectories, PVLDB 7 (3) (2013) 205–216, nearest Neighbors.
URL http://www.vldb.org/pvldb/vol7/p205-niedermayer.pdf

[15] X. Yu, K. Q. Pu, N. Koudas, Monitoring k-nearest neighbor queries over moving objects, in: K. Aberer, M. J. Franklin, S. Nishio (Eds.), ICDE, IEEE Computer Society, 2005, pp. 631–642, nearest Neighbors. doi:10.1109/ICDE.2005.92.

[16] R. Benetis, S. Jensen, G. Karĉiauskas, S. Ŝaltenis, Nearest and reverse nearest neighbor queries for moving objects, The VLDB Journal 15 (3) (2006) 229–249, nearest Neighbors. doi:10.1007/s00778-005-0166-4.
URL http://dx.doi.org/10.1007/s00778-005-0166-4

[17] S. Ma, Y. Zheng, O. Wolfson, T-share: A large-scale dynamic taxi ridesharing service, 2013 IEEE 29th International Conference on Data Engineering (ICDE) 0 (2013) 410–421. doi:10.1109/ICDE.2013.6544843.

[18] I. F. Ilyas, G. Beskales, M. A. Soliman, A survey of top-k query processing techniques in relational database systems, ACM Comput. Surv. 40 (4) (2008) 11:1–11:58, nearest Neighbors. doi:10.1145/1391729.1391730.

[19] T. H. Cormen, C. Stein, R. L. Rivest, C. E. Leiserson, Introduction to Algorithms, 2nd Edition, McGraw-Hill Higher Education, 2001.

[20] T. Vincenty, Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations, Survey Review 22 (176) (1975) 88–93.
URL http://www.movable-type.co.uk/scripts/latlong.html

[21] V. Janilionis, Shortest paths with moving objects: a dynamic dial-a-ride problem application for taxi fleets, Master's thesis, University of Edinburgh (2013).

Cahiers de recherche du Centre de Recherche Appliquée en Gestion (CRAG) de la Haute Ecole de Gestion - Genève

CRAG - Centre de Recherche Appliquée en Gestion
Haute école de gestion - Genève
Campus de Battelle, Btiment F
7, route de Drize - 1227 Carouge - Suisse

✉ crag@hesge.ch
www.hesge.ch/heg/crag
☎ +41 22 388 18 18
🖷 +41 22 388 17 40

Tous les cahiers de recherche de la HEG sur RERO DOC :
http://doc.rero.ch