Algorithm for finding optimal paths in a public transit network with real-time data

November 9, 2010

Jerald Jariyasunant, Corresponding Author, jjariyas@berkeley.edu
Department of Civil Engineering, University of California, Berkeley
621 Sutardja Dai Hall Berkeley, CA 94720
Tel: (510) 642-2468

Eric Mai, ericmai@berkeley.edu
Department of Civil Engineering, University of California, Berkeley
621 Sutardja Dai Hall Berkeley, CA 94720
Tel: (510) 642-2468

Raja Sengupta, sengupta@ce.berkeley.edu
Department of Civil Engineering, University of California, Berkeley
640 Sutardja Dai Hall Berkeley, CA 94720
Tel: (510) 642-9540 Fax: (510) 643-8919

Word Count:

Number of words:	5339
Number of figures:	8
Number of tables:	0
Total:	7339

Abstract

Recently, transit agencies have been opening up their route configuration and schedule data to the public, as well as providing online APIs (Application Programming Interfaces) to real-time bus positions and arrival estimates. Based on this infrastructure of providing transit data over the internet, we developed an algorithm to calculate the travel times of K-shortest paths in a public transportation network where all wait and travel times are only known at real-time. Although there is a large body of work in routing algorithms in transit networks, we take cues from an algorithm to find shortest paths in road networks, called Transit Node routing. Our approach is based on a simple observation intuitively used by humans: when you take transit, you will look for a particular set of transfer points that connect transit routes that lead from the origin and destination. We precompute a lookup table of feasible paths between the origin stop of every bus route to the terminus of every other bus route using the transfer points. This precomputation of paths significantly reduces the computation time and number of real-time arrival requests to transit agency servers, the bottleneck in computing this problem. The computational complexity of the algorithm is linear in real-time and implementation results show that queries from a webserver are returned in 3 seconds in the worst-case.

1 INTRODUCTION

The focus of the paper is the introduction of a new algorithm which overcomes the problem of computing shortest paths in a transit network which pulls real-time data from a third-party Application Programming Interface (API). Over a web-based or smartphone interface, a user enters a geo-coded origin and destination (O-D), and the algorithm must respond by returning K-shortest paths based on the real-time state of the transit network. Thus, our principal performance measure is the time elapsing between input of the O-D and output of the path, which we call the path computation time. Web usage studies show this time should be less than 7 seconds [1, 2].

Paths cannot be wholly pre-computed since the aim is to return results based on real-time state, which needs to be acquired at run-time. We call the time required to get this information the real-time state acquisition time. Industry Service Level Agreements (SLAs) create constraints on how this real-time state can be acquired. The industry appears to be evolving towards an open model where the public agencies are making real-time bus data available on the web, allowing third party developers to use this data to provide transit information via web-based services or native applications on smartphones or over the web. The real-time data needs to be consumed through web-based APIs that limit the amount of data per request to the transit agency web server and the number of requests per second. For example, NextBus, a provider of real-time data in 61 cities, limits real-time data to 300 stops per request.

Data analyzed for this paper showed that in most transit networks, running a common K-shortest path algorithm after obtaining predictions for all transit stops in a network is not feasible. For example, the Washington DC transit network has 28,627 stops, which takes an average of 75 seconds to acquire the real-time data for the entire network. Moreover, the 75 seconds does not account for the request rate constraint, 1 request every 10 seconds, imposed by the data provider[3]. It takes 96 requests to get all the Washington DC data, which adds at least another 9600 seconds to the time required to acquire the data for the entire network. This time is long enough for at least some portion of the data to become obsolete by the time the data for the entire network has been acquired. The transit routing algorithm in this paper needs to acquire its real-time data and compute with the K-shortest paths in less than 7 seconds. Thus, we need an algorithm able to route each trip by acquiring the real-time data for only some small part of the total network.

Fortunately part of the literature is helpful. The obvious approach might be to model the transit network as a graph, albeit time expanded [4, 5, 6, 7] or time dependent [8, 9, 10, 11], and run one of the widely used shortest path algorithms on it [12, 13, 14]. There has been research on speed up techniques [15, 16, 17] to Dijkstra's algorithm as well as heuristic algorithms developed [18, 19, 20, 21]. The literature also covers the K-shortest path [22, 23, 24, 25] and reasonable (multi-objective) path computation [26] problem. This literature helps with schedule based transit routing, i.e, when the edge costs of the graph are single numbers known a priori. The purpose of the discussion in this paragraph is to show the real-time transit routing problem is not easily reduced to this case. These algorithms require that all link costs (travel time) in the network are known at all times. However, this time required to acquire the edge costs for the network graph from the transit agency web servers is the major bottleneck due to the software architecture of implemented transit information systems today. Thus, the run-time of traditional K-shortest path algorithms in this problem is extremely slow.

There is also literature on routing in transit networks formulated as shortest path problems in stochastic and dynamic networks [27, 28, 29, 30, 31]. This literature helps consider the reliability of travel time as well or other measures of robustness. These methods could be blended to improve the algorithm in this paper to optimize higher moments of travel time. In this paper we do not leverage this literature.

Feder [32] addresses the routing problem when edge costs are approximately known but can be made more precise at run-time at a cost, a parallel to the problem of determining the wait and travel times at bus stops by accessing a real-time API. Likewise, Pallottino developed an algorithm for transit graphs where the edge costs are known, but subject to small changes (i.e. updates in real-time information) [33]. Though this paper is about a problem similar to the ones in these papers, we use a different solution technique, one leveraging off-line pre-computation in a manner similar to Bast, Sanders, and Schultes[34, 35], for routing in road networks.

Through 770,000 simulations covering transit networks in 77 US cities, we find this approach is able to keep the real-time data acquisition time under 1.5 seconds and the total path computation time less than 3 seconds for 99% of the O-D requests.

The structure of the paper is as follows. Sections 1.1 and 1.2 provide further information on open transit data and the routing methods of Bast et al, respectively. Section 2 describes our algorithm. Section 3 is the evaluation. We evaluate the algorithm by doing 10,000 simulations of trip requests for each transit network in 77 cities. Worst-case bounds on path computation time, real-time data acquisition time, off-line pre-computation time, and run-time memory requirements, are based on all 770,000 simulations. We measure the time required to service it by actually executing the request that communicates over the Internet in real-time to the transit agency data server for the city relevant to the O-D. Since execution times are machine dependent measures, we also provide histograms showing machine independent measures determining the complexity of our algorithm. The route computation time depends on the size of a database of paths produced by pre-computation plus the real-time data acquisition time. The real-time data acquisition time depends on the number of bus stops per transit trip request. We present histograms on these measures for Washington DC. The DC transit network turns out to be the most complex of the 77 cities, including Los Angeles, Chicago, and San Francisco. Section 4 concludes the paper.

1.1 Recent Trends of Open Data

The development of a real-time transit routing algorithm has been spurred by two trends in public transportation technology: open data, and real-time tracking of buses.

Starting in 2007, a growing number of transit agencies have packaged their route configuration and schedule data into a format called the Google Transit Feed Specification (GTFS)[36] in order to have trips planned by Google Transit. The benefit of this was the integration of public transit into online trip-planners, but more importantly, the release of the GTFS data by transit agencies has allowed for a plethora of innovative apps being created by third party and independent developers. For example, an open-source multi-modal trip planner, Graphserver, was developed and has been embraced by the online transit developer community. Last year, our research group introduced a system for real-time transit trip planning on mobile phones called BayTripper. The potential travel time benefits for incorporating real-time data into travel time predictions were evaluated[37, 38]. As of July 2010, BayTripper is the only real-time transit trip planning application released and available to the public.

The second trend is the use of GPS to track buses, not only for transit agencies to monitor their fleet, but to open the data up for users to code with. Pioneered by TriMet in Portland Oregon, transit agencies have begun to not only release route configuration and schedule data, but also real-time feeds of the location and expected arrival time of their transit vehicles. Transit agencies throughout the United States have slowly been opening data to the public. As of July 2010, there exist 787 transit agencies in the United States, 107 of which provide open data and a small fraction those agencies provide open real-time data. This number has been growing, as has research involving the value of real time data[39].

1.2 Relation to Transit Node Routing in Road Networks

Although there has been an enormous body of work on routing in public transit networks, the algorithm we developed takes cues from the work of Bast, Sanders, and Schultes, who developed an innovative algorithm for routing in road networks. Their algorithm, called Transit Node routing is currently the fastest static routing technique available [34, 35]. The main observation of theirs was something intuitively used by humans: When you drive to somewhere far away, you will leave your current location via one of only a few access routes to a relatively small set of transit nodes interconnected by a sparse network relevant for long-distance travel. Likewise, in public transit networks, when you travel to a destination that is not served by a bus/train route near your origin, you will look at a set of potential transfer stations to change buses. Transit Node Routing precomputes distance tables for important (transit) nodes and all relevant connections between the remaining nodes and the transit nodes. As a result, the difficult problem of finding shortest paths on extremely large road networks is "almost" reduced to about 100 table lookups. This lookup table only stores data for a certain subset of points to speed up the performance of the algorithm, which is a major innovation. The same strategy is used in the real-time transit routing algorithm presented in Section 2. We calculate a set of feasible paths for only a subset of nodes in the transit network and store the results in a lookup table.

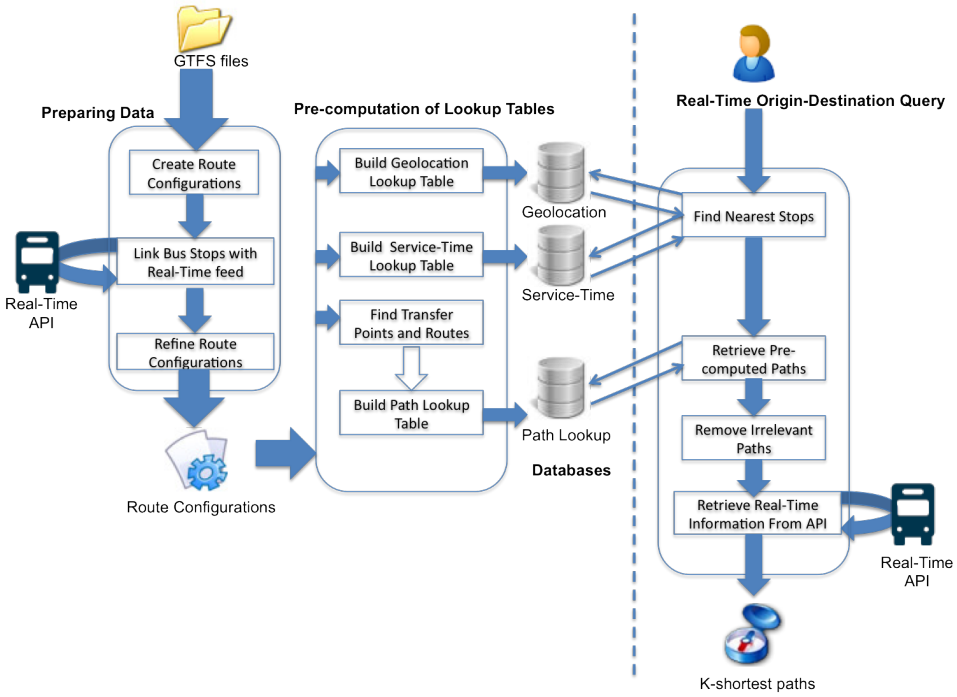

Figure 1: Flowchart of the algorithm described in Sections 2.1, 2.2, and 2.3.

2 ALGORITHM IMPLEMENTATION

As described in Section 1.2, the algorithm takes cues from Transit Node Routing. The Transit Node Routing algorithm has a precomputation step to store distances between important nodes. The nontrivial concept is defining 'important' nodes such that optimal routes are found. In our routing algorithm, the 'important' nodes to perform the precomputation are defined by the following statements about routing in transit network:

1. If the origin and destination are not served by the same bus route, humans intuitively plan trips by finding transfer points to connect between bus routes.

2. The set of feasible paths from any bus stop along Route X to any bus stop along Route Y is a subset of all feasible paths from the origin station of Route X to the terminus station of Route Y.

Statement 2 defines the important nodes that are precomputed; it is only necessary from the origin node of each bus line to the terminus of every other bus line to create a lookup table the size of the number of total bus routes squared. Statement 1 defines the process by which the precomputation is done: storing transfer points to build a sequence of locations of where to transfer to reach the terminus of the bus line. In this paper, a feasible path is defined as any path that can be traversed by taking a series of buses between an O-D with a maximum of four transfers. The constraint of four transfers puts a cap on the number of paths needed to be precomputed, and prior research has shown that the accuracy of real-time data deteriorates significantly when predicting arrival times over 20 minutes into the future, well over the time a fourth transfer would be made [38]. Any O-D query that requires more than four transfers cannot be computed by the algorithm.

The routing algorithm is broken up into three steps: preparing the schedule data received from third-party providers, building a lookup table of feasible paths from a subset of O-D pairs, and dynamically performing the real-time calculation of the K-shortest paths using the lookup table. These steps are shown in a flowchart in Figure 1.

Figure 2: Example transit network structure built from open transit data with no link travel times defined.

2.1 Preparing Data

Software was built to convert schedule data to build a graph of the network with no link travel time data as shown in Figure 2. Link travel time data is only known at real-time by accessing an API over the internet. Although the amount of data that is needed to be transferred to deliver all real-time predictions for all bus stops in an entire network is not large, today's API's are not designed to do this. Until the groups that perform the real-time arrival predictions create a standard for this, this paper presents a solution that is implementable immediately to perform routing with real-time data.

2.1.1 *Convert Open Transit Data To Route Configurations*

Transit data is read into the system from two different sources, Google, and NextBus.com. Google has established a standard for encoding transit data, GTFS, which can be converted into route configuration files that store the following relevant information in the structure specified in Figure 3, the route name, direction name, stop / intersection name, latitude, longitude, stop sequence, and agency name.

A route configuration is categorized by a route, direction, and agency name. Each configuration specifies a set of stop names with corresponding latitude and longitude points in the sequence which the bus traverses the route. Configurations with the same route, direction, and agency name are grouped together.

2.1.2 *Link Bus Stop To Real-Time Feed*

GTFS does not have a specification for a real-time data format, thus, each stop must be linked to a real-time URL by an unique stopcode. This assignment is done by matching the route configurations with the real-time feeds provided by Nextbus.com and Portland TriMet. Real time predictions are accessed through HTTP by specifying the agency name, bus route, direction, and stopcode.

Not all transit agencies that have released open data in the GTFS format are served by NextBus.com, however any future real-time feeds provided by those agencies can be integrated into the algorithm.

2.1.3 *Refine Route Configurations*

Most train routes only run in two opposing directions, however there are numerous cases in which bus routes have different route configurations but run under the same route name. The transit routes are broken up into segments as described in Figure 4 such that each unique segment defines a route configuration that all buses follow. If the set of stations in a configuration that a bus follows is a subset of another configuration

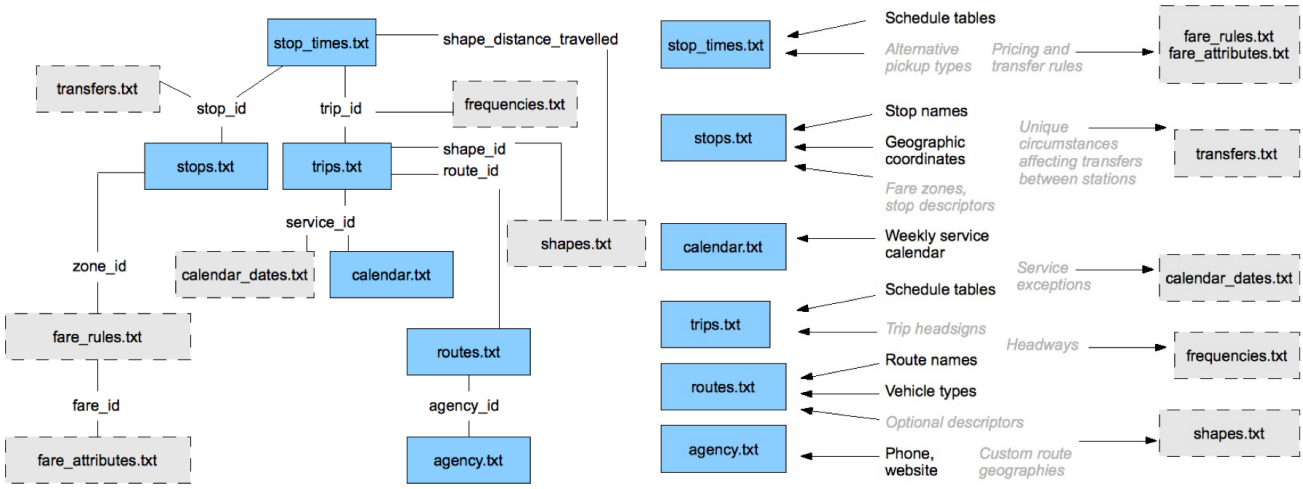

Figure 3: GTFS file structure. Optional Files in dashed boxes, Required files in solid boxes (a) Files and linking columns (b) Properties of files needed to be extracted

of stops, the two are considered the same. Whether or not a bus stops at every station is determined at run-time.

2.2 Precomputation of Lookup Tables

The precomputation step generates three tables, a path lookup table, a bus service-time table, and a geolocation table, which are defined in the following sections.

2.2.1 *Find Transfer Points and Routes*

The entire set of stops is sorted by either latitude or longitude, and stops within a reasonable distance are stored as a potential transfer points between the two routes. In the implementation, the reasonable transfer distance is arbitrarily chosen at half a mile. Previous research has shown that transit riders' preference for walking between transfers varies quite a bit, and is dependent on many variables, including land use, open space, and topographical features [40]. In a future implementation, this transfer distance can be varied to account for these variables.

Bus routes are often designed to share common stations, such as in a trunk-line configuration. In these route configurations, there exist a large number of potential transfer points between lines, which increases the size of the lookup table linearly. To address this scenario, all transfer points between the two lines are removed except for the optimal point. This optimal point is defined by the station with the lowest probability of missing a transfer as specified by the GTFS schedule.

2.2.2 *Build Path Lookup Table*

A set of feasible paths is built from the origin stop of every bus route to the terminus of every bus route in each direction and stored in a path lookup table. The set of bus routes is iterated through to create a list of reachable routes in one to four transfers. For each O-D pair, the transfer points corresponding to each of the list of reachable routes are stored in the path lookup table. Paths that require more than four transfers will not be found, and thus not stored in the database.

The size of this database grows polynomially with the number of bus routes in the transit network, as shown in Figure 7. The potentially infinite size of this database is limited in two ways: by setting the number of possible transfers at four transfers, and excluding paths that take two transfers more than any existing feasible paths between an O-D. The second limitation measure is based on an analysis of GTFS schedule

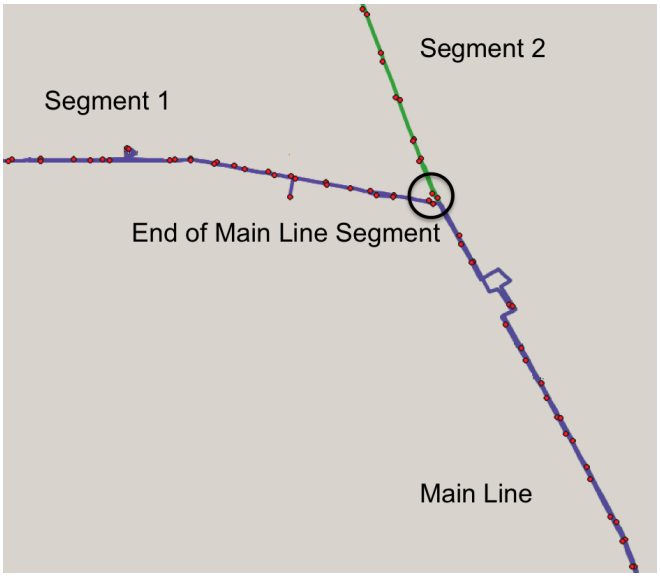

Figure 4: Example of a bus route with two termini. This particular route is broken up into three segments.

data, which shows that a direct route between two points is always faster than a route with two additional connections with no waiting time at transfer points.

2.2.3 *Build Service-Time Lookup Table*

The lookup table includes paths from all bus routes in the transit agency's network. However, not all bus routes are in service at the same time: there are often nighttime or rush hour only routes being run. The final step in the precomputation of routes builds a "service-time" table of hours in which each bus route is in service. In real-time, the bus service table used to remove any paths which include a bus route not in service.

2.2.4 *Build Geolocation Lookup Table*

All bus stops from the route configuration files are aggregated and a geolocation table is created to store the set of all transit stops in the entire network, indexed by their latitude and longitude.

2.3 Real-Time Queries

The following section describes the algorithm when it responds to real-time queries from users. A query is a request for the shortest path between an O-D, specified by a latitude and longitude point. The process is very simple: a set of paths is retrieved from the lookup table, the travel times are accessed from the real-time API, and the total travel times for each path are calculated and sorted. Thus, in real time, the computational complexity of this algorithm is linear in the number of paths retrieved from the lookup table.

2.3.1 *Find Nearest Stops*

Transit routes in a half-mile walking radius of both the user's origin and desired destination are looked up from the geolocation table. For each route, the stop nearest to the user is saved, and all other stops are not considered. If no stops are within this distance, the search radius is increased until a bus route is found.

2.3.2 *Retrieve precomputed paths*

Precomputed paths are retrieved for each O-D pair of bus routes retrieved from the geolocation table. When the query is made in real-time the set of transfer points is sent to the real time API to retrieve bus arrival predictions at each node. Every time a query is made by a user, new real-time arrival predictions are sent

to the real-time API, retrieving the latest data for each user request. Thus, the algorithm instantly handles changes in bus arrival predictions: even in major situations such as a bus breaking down, or multiple buses bunching together, such that the wait time at any bus stop is greater than the specified headway.

2.3.3 *Remove irrelevant paths*

As described in Section 2.2, the precomputed lookup table contains the superset of all feasible paths from the origin station of every bus route to the terminus station of every bus route. Therefore, not all paths retrieved from the path lookup table are physically possible to be made and are thus ignored. In addition, paths which include a bus route not in service are removed.

2.3.4 *Retrieve Real-Time Information From API*

The set of feasible paths contains information for all bus stops to board, get off, and transfer. These stops are all bundled into a HTTP request to the real-time API. At this point, the algorithm is dependent on the speed at which the third-party server can process the bus arrival predictions and internet traffic between the third party server and the localhost.

The total time of each path is calculated, all paths are sorted and returned to the user.

3 PERFORMANCE AND RESULTS

The real-time routing algorithm was tested with transit data from 77 different cities. The data was obtained from GTFS Data Exchange [41], an online aggregator of GTFS files, NextBus [42], and Portland TriMet [43]. For each transit agency's network 10,000 simulated queries for real-time paths were made. The 10,000 different O-D points were randomly drawn from the set of bus stops served by each agency. As described in Section 2.3.1, the K-shortest-paths algorithm computes paths to all bus stops within a half-mile walking distance of the random O-D. In the implementation of the iPhone application, only the five shortest paths are displayed for each request in order to provide users with a variety of options to reach their destination.

The route computation time depends on the size of a database of paths produced by pre-computation plus the real-time data acquisition time. The real-time data acquisition time depends on the number of bus stops per transit trip request. The tradeoff of using the real-time routing algorithm is the time to precompute the lookup table and size of the table, versus the reduced time to respond to queries in real-time. To evaluate this trade off, we collected data for four metrics:

1. **Number of paths.** We show that the algorithm searches through a set of paths that is tractable at runtime, and even in the worse case scenario it responds to queries in a timely manner.
2. **Number of requests sent to real-time API.** We show that our algorithm makes a small number of queries to the real-time API for all networks analyzed.
3. **Number of total paths stored in precomputed lookup table.** We show that the size of the lookup table is small enough to be loaded into memory to run on a server on even a home computer.
4. **Time to generate precompute lookup table.** We show that the time to compute the lookup table is small: under 1 minute for most networks, with a worst-case of 100 minutes.

3.1 Implementation Details

The simulated queries were run on an Amazon EC2 server located in Northern California. Amazon EC2 is a cloud computing service that allows users to rent virtual machines called "instances". The algorithm is running on the smallest instance available, which provides 1.7 GB memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB instance storage, on a 32-bit platform at a cost of 4 cents per hour. In addition to the tests run for the purpose of this paper, the server is used daily 350 times daily by 600 active users as of July 2010 via the BayTripper iPhone application. The EC2 server serves requests from the

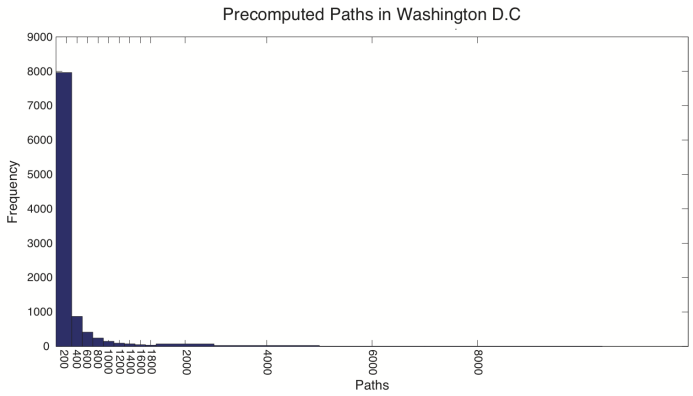

Figure 5: Histogram of number of paths searched Washington D.C.. 99% Percentile: 1900 paths. 80% Percentile: 304 paths. 50% Percentile: 224 paths. X-Axis: Number of precomputed paths which are computed in real-time. Y-Axis: Frequency of occurrences

iPhone application and the web for real-time routing on the BART and MUNI systems in the San Francisco Bay Area.

3.2 Real-Time Algorithm Performance

In the following section we present histograms of metrics 1 and 2: the number of paths and requests to the real-time arrival API. We also focus on discussing two worst-case scenarios found in all 770,000 simulated queries, the O-D pair that requires computation of travel times amongst the largest number of paths, and another scenario that requires the largest number of hits to the real-time API. In both the worst-case scenarios the algorithm computes the K-shortest paths problem in less than 3 seconds, while taking 1 second to query the real-time API in 99% of cases. The speed of the algorithm is affected by the number of precomputed paths retrieved from the lookup table. The larger number of paths, the more estimates for wait and travel times are needed to be retrieved from the real-time API, the bottleneck in the system.

Both of the worst-case scenarios were found in Washington D.C., the agency with the largest number of bus stops and routes served that we tested. The worst-case scenario for number of feasible paths retrieved from the precomputed lookup table is 10,127. The worst-case scenario for number of total hits to the real-time API was 127. Figure 5 shows a histogram of the number paths computed in real-time, generated from 10,000 simulated queries. Washington D.C. is presented because to demonstrate that even in the worst-case the algorithm computes in under 3 seconds.

The performance of the system is a function of the real-time API and constraints placed on accessing the API. In current real-time APIs the number of stations able to be queried in a HTTP request is limited to 300 bus stops per request, and 10 stops per bus route. Additionally, consecutive requests cannot be made more often than every 10 seconds. Thus, as the number of hits required to the API increases, the response time grows approximately as a step function. The worst-case scenario for number of queries to the real-time API was 127, in Washington D.C. Figure 6 shows a histogram of the number of hits to the real-time API for the simulated queries in Washington D.C. The number of bus stops in the worst case scenario, 127, can be made in a single request and fits well within the limitation of the real-time API.

The real-time algorithm makes a new request for bus arrival estimates to the real-time API every time a user request is made, ensure the most up to date estimates are used in the algorithm. As a comparison, the total number of stops in the network is 28,627; the time to query this is 75 seconds. The alternative of querying every route in the system for this network and running Dijkstra's algorithm on the transit graph is infeasible, as bus positions update more frequently than every 75 seconds.

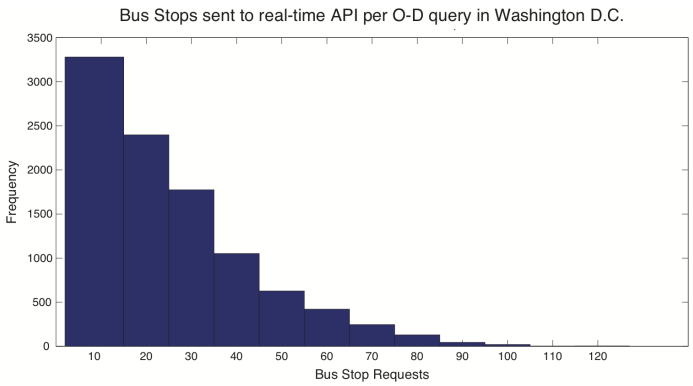

Figure 6: Histogram of number of bus stops sent to real-time API per O-D query in Washington D.C. 99% Percentile: 82 hits. 80% Percentile: 40 hits. 50% Percentile: 26 hits. X-Axis: Number of hits to real-time API. Y-Axis: Frequency of occurrences

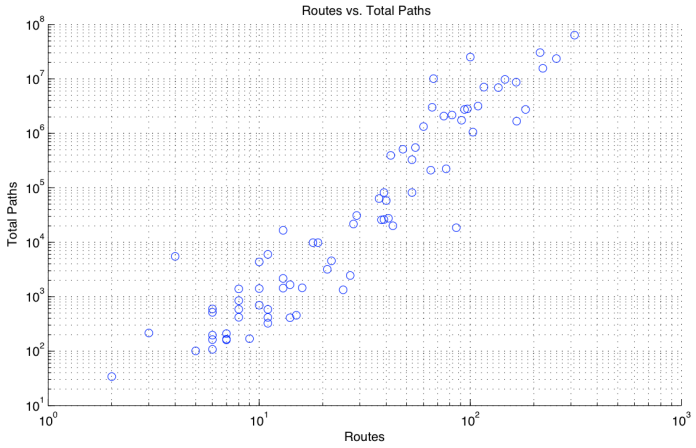

Figure 7: Scatterplot of lookup table size for all transit agencies. Y-Axis: Size of path lookup-table generated for transit agency X-Axis: Number of routes run by transit agency

3.3 Precomputation Performance

In the prior section we demonstrate that the real-time performance of the algorithm returns results in 3 seconds in the worst-case, with 1 second being spent with the real-time API query, versus 75 seconds to query the entire network. This performance comes at a cost of having to precompute a lookup table.

3.3.1 *Memory Usage*

Each path takes on average 20 bytes to store. The size of the path lookup table that has to be loaded to memory ranges from .664KB to 1.2GB for the largest transit network we analyzed. The geolocation and bus service tables range from 8KB to 2MB.

As shown in Figure 7, the size of the lookup table increases polynomially as the number of routes served by a transit agency grows.

3.3.2 *Precomputation Time*

The largest region we analyzed, Washington D.C. took 99 minutes to precompute 63,308,845 paths,

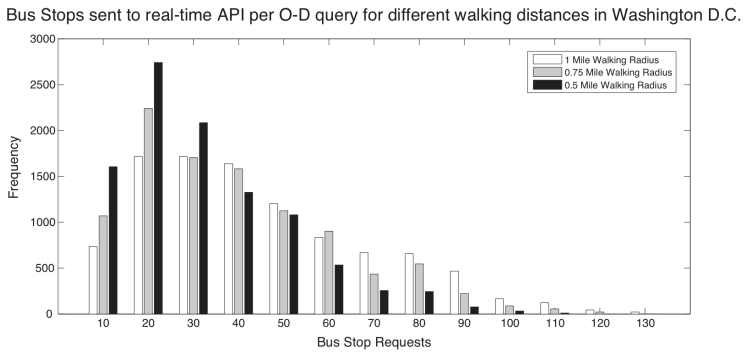

Figure 8: Histogram of number of bus stops sent to real-time API per O-D query in Washington D.C. for three different walking distances X-Axis: Number of hits to real-time API. Y-Axis: Frequency of occurrences

while 70% of all other cities took under 1 minute to complete. Every time a transit agency makes a route configuration change, the lookup table must be re-computed.

3.4 Sensitivity Analysis

The number of paths searched through at runtime is dependent on the distance a person is willing to walk to his origin, and from the last bus stop to his destination. Although the previous simulations were made with a fixed half-mile distance, the distance can be varied per the user's request. Figure 8 shows a histogram of the number of bus stops sent to the real-time API for three different walking distances, .5 miles, .75 miles, and 1 mile. With a 1 mile walking radius, the number of hits to the real-time API increases, but never exceeds 140 requests. The response time from the server for 140 requests still is 1 second on average, thus there is a negligible increase in the time to compute the K-shortest-paths with this walking distance.

Additionally, the number of paths searched at runtime is dependent on an even greater variety of specific features of transit agencies and metropolitan regions, such as number of bus stops served, number of transfer stations, as well as geographic locations and spatial distribution of these stops. These factors affect the following performance metrics of the algorithm in ways which have not been fully explored, but are important to consider for future implementations of the algorithm in larger-scale metropolitan transit networks than Washington D.C.

4 CONCLUSION

This paper presented a new algorithm for calculating K-shortest paths in a transit network with real-time data. We describe that the modeling of this particular shortest path problem is a result of the state of technology used by transit agencies to disseminate information today. It was shown that precomputation of a lookup table for paths between certain transit stops significantly reduces the run-time speed of responding to user requests. Transit data in 77 cities was precomputed into lookup tables, and 10,000 simulated user requests for each transit agency were made, showing that in the worse case scenario, the algorithm computes K-shortest paths in 3 seconds. Based on these results, additional testing is proposed to investigate the performance of the algorithm in larger regional transportation areas which include multiple transit agencies and intercity networks.

References

[1] R. Jain, T. Raleigh, C. Graff, and M. Bereschinsky, "Mobile internet access and qos guarantees using mobile ip and rsvp with location registers," *IEEE Int. Conf. Commun.*, vol. 3, pp. 1690–1695, 1998.

[2] T. Erl, Ed., *Service-oriented architecture (SOA): concepts, technology, and design.* Prentice Hall, 2005.

[3] N. T. Conditions, *http://www.sfmta.com/cms/asite/nextmunidata.htm.*

[4] M. Muller-Hannemann and K. Weihe, "Pareto shortest paths is often feasible in practice." *In Proc. 5th Workshop on Algorithm Engineering (WAE 2001)*, vol. 2141, pp. 185–198, 2001.

[5] S. Pallottino and M. G. Scutella, *Equilibrium and advanced transportation modelling, chapter 11.* Kluwer Academic Publishers, 1998.

[6] F.Schulz, D.Wagner, and K.Weihe, "Dijkstra's algorithm on-line: An empirical case study from public railroad transport," *ACM Journal of Experimental Algorithmics*, vol. 5(12), 2000.

[7] F. Schulz, D. Wagner, and C. Zaroliagis, "Using multi- level graphs for timetable information in railway systems," *In Proceedings 4th Workshop on Algorithm Engineering and Experiments*, pp. 43–59, 2001.

[8] G. Brodal and R. Jacob, "Time-dependent networks as models to achieve fast exact timetable queries," *In Proc. 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003)Electronic Notes in Theoretical Computer Science*, vol. 92, issue 1, Elsevier, 2003.

[9] K. Nachtigal, "Time depending shortest-path problems with applications to railway networks," *European Journal of Operations Research*, vol. 83, pp. 154–166, 1995.

[10] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *Journal of the ACM*, vol. 37(3), 1990.

[11] ——, "Minimum weight paths in time-dependent networks," *Networks*, vol. 21, 1991.

[12] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[13] R. Bellman, "On a routing problems," *Quarterly of Applied Mathematics*, vol. 16(1), pp. 87–90, 1958.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, pp. 100–107, 1968.

[15] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Efficient models for timetable information in public transportation systems," *ACM Journal of Experimental Algorithmics*, vol. 2.4, 2007.

[16] R. Bauer, D. Delling, and D. Wagner, "Experimental study on speed-up techniques for timetable information systems," *Networks*, no. 0.1002/net.20382, 2009.

[17] D. Delling, T. Pajor, and D. Wagner, "Engineering time- expanded graphs for faster timetable information." *Lecture Notes in Computer Science*, vol. 5868, pp. 182–206, 2009.

[18] G. Desaulniers and D. Villeaneuve, "The shortest path problem with time windows and linear waiting costs," *Transportation Science*, vol. 34, pp. 312–319, 2000.

[19] R. Dial, "Transit pathfinder algorithms," *Highway Research Record*, vol. 205, pp. 67–85, 1967.

[20] J. Granat and F. Guerriero, "The interactive analysis of the multicriteria shortest path problem by the reference point method," *European Journal of Operational Research*, vol. 151 (1), pp. 103–118, 2003.

[21] T.-N. Chuang and J.-Y. Kung, "The fuzzy shortest path length and the corresponding shortest path in a network." *Computers and Operations Research*, vol. 32(6), pp. 1409–1428, 2005.

[22] N. Van der Zijpp and C. Fiorenzo, "Path enumeration by finding the constrained k-shortest paths," *Transportation Research Part B*, vol. 39 (6), pp. 545–563, 2005.

[23] W. Xu, S. He, R. Song, and S. S. Chaudhry, "Finding the k shortest paths in a schedule-based transit network," *Computers & Operations Research*, 2010.

[24] Y.-L. Chen and H.-H. Yang, "Finding the first k shortest paths in a time-window network." *Computers and Operations Research*, vol. 31 (4), pp. 499–513, 2004.

[25] L. Zhao, Y. Xiong, and H. Sun, *The K Shortest Transit Paths Choosing Algorithm in Stochastic Transit Network*, R. Sets and K. Technology, Eds. Springer, 2008.

[26] M. Tan, C. Tong, S. Wong, and J. min Xu, "An algorithm for finding reasonable paths in transit networks," *Journal of Advanced Transportation*, vol. 41, No. 3, pp. pp. 285–305, 2010.

[27] R. W. Hall, "The fastest path through a network with random time-dependent travel times," *Transportation Science*, vol. 20(3), pp. 182–188, 1986.

[28] D. E. Kaufman and R. L. Smith, "Fastest paths in timedependent networks for intelligent vehicle-highway systems applications," *IVHS Journal*, vol. 1(1), pp. 1–11, 1993.

[29] M. P. Wellman, K. Larson, M. Ford, and P. R. Wurman, "Path planning under time-dependent uncertainty," in *Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 532–539.

[30] O. E. Karasan, M. C. Pinar, and H. Yaman, "The robust shortest path problem with interval data," manuscript, August 2001.

[31] P. Zielinski, "The computational complexity of the relative robust shortest path problem with interval data," *European Journal of Operational Research*, vol. 158, pp. 570–576, 2004.

[32] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy, "Computing shortest paths with uncertainty," *Journal of Algorithms*, vol. 62, pp. 1–18, 2007.

[33] S. Pallottino and M. Scutell, "A new algorithm for reoptimizing shortest paths when the arc costs change," *Operations Research Letters*, vol. 31 (2), pp. 149–160, 2003.

[34] H. Bast, S. Funke, P. Sanders, and D. Schultes, "Fast routing in road networks with transit nodes," *Science*, vol. 316(5824):566, 2007.

[35] D. Schultes, "Route planning in road networks," Ph.D. dissertation, Universitat Karlsruhe (TH), 2008.

[36] Google, "Google transit feed secification," *http://code.google.com/*.

[37] J. Jariyasunant and E. Mai, "Baytripper," *www.baytripper.org*. [Online]. Available: www.baytripper.org

[38] J. Jariyasunant, D. B. Work, B. Kerkez, R. Sengupta, S. Glaser, and A. Bayen, "Mobile transit trip planning with real-time data," in *Proceedings of the 2010 Transportation Research Board Annual Meeting*, 2010.

[39] B. Ferris, K. Watkins, and A. Borning, "Onebusaway: Results from providing real-time arrival information for public transit." in *Proceedings of CHI*, 2010.

[40] Z. Guo and N. H. M. Wilson, "Assessment of the transfer penalty for transit trips geographic information system-based disaggregate modeling approach," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1872, pp. 10–18, 2004.

[41] GTFS-DataExchange, *www.gtfs-data-exchange.com*.

[42] NextBus, *www.nextbus.com*.

[43] TriMet, *developer.trimet.org*.