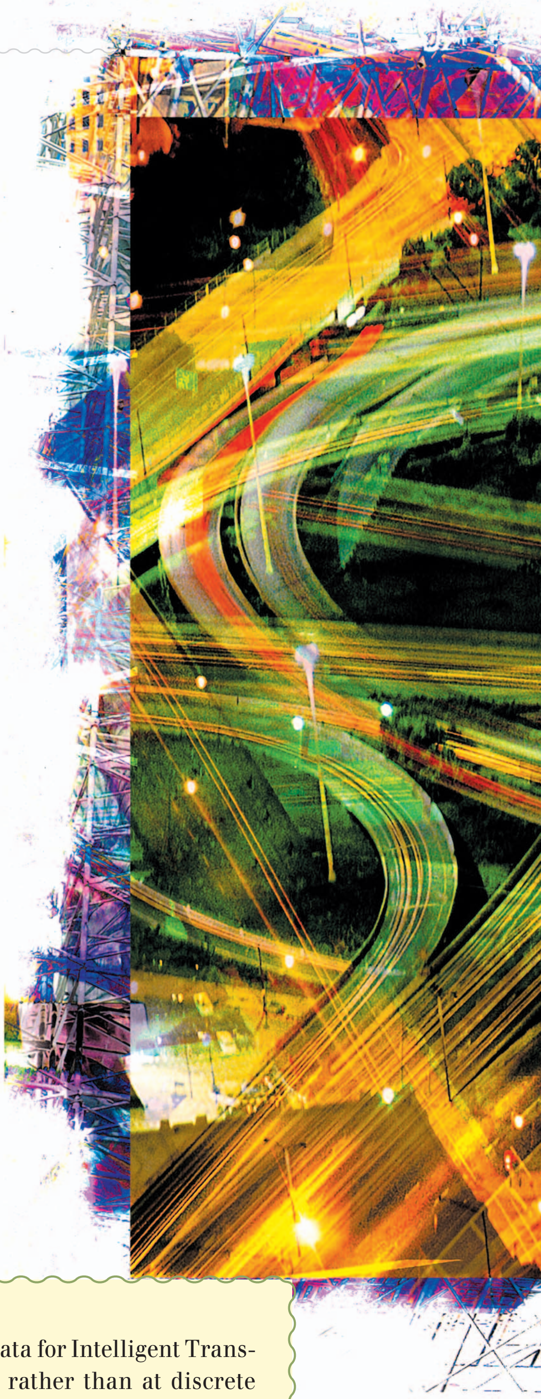
Dynamically Computing Fastest Paths for Intelligent Transportation Systems

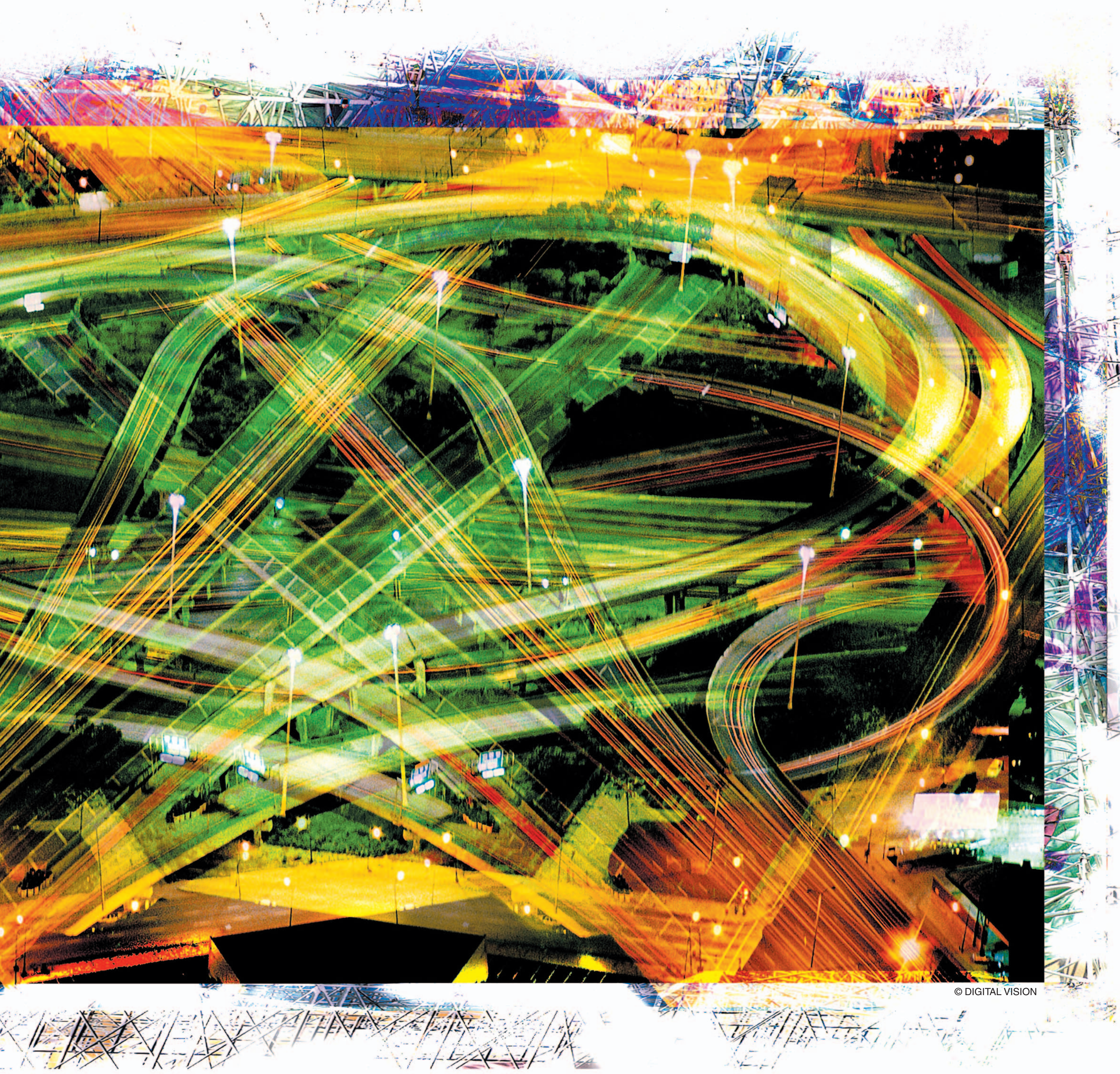


Jeffrey Miller

Department of Computer Systems Engineering
University of Alaska, Anchorage
jmiller@uaa.alaska.edu

Abstract **—**In this paper I present a new approach to gathering data for Intelligent Transportation System applications over a continuous-flow of traffic rather than at discrete locations, as is the case with many existing technologies. Loop detectors and video cameras, among other devices, currently provide the primary means for gathering data, though it has now become possible using mobile and GPS technology to gather the speed and location of each vehicle in real-time over a continuous flow, which will allow more novel applications, such as incident identification and hazard alerts, to be developed. In addition, as vehicles transmit updated speeds to the system, the fastest path of each commuter from his current location to his desired destination can be determined. The Pre-Computed class of algorithms determines fastest paths more efficiently than existing algorithms, with the assumption that the graph edges are rather static though the weights can change frequently. Different shortest and fastest path algorithms are presented and analyzed using FreeSim (http://www.freewaysimulator.com), which contains an implementation of all of the algorithms discussed.

Digital Object Identifier 10.1109/MITS.2009.932717


© DIGITAL VISION

I. Introduction

Traffic poses a major issue in many cities around the world. The amount of time to travel from one location to another can vary significantly based on the current traffic conditions. A study performed by Texas A&M in 2004 [3] concluded that traffic on freeways in the United States costs drivers an additional 3.5 billion hours each year. The study also noted that Los Angeles is the worst city for traffic in the United States, where, in 2002, drivers wasted an average of an additional 93 hours in traffic [3].

There are often many different routes a driver could take from his current location to his desired destination, but how can he decide which highways and streets to use to minimize the commute time? He could use a street map to determine his route or use electronic maps, such as Mapquest [2] or Google Maps [1]. A navigation system in his vehicle could aid in turn-by-turn directions based on his current location. Radio stations could help in providing traffic and incident updates based on data they have received from third-party organizations, telephone calls from drivers, and reports by helicopters flying over the transportation system. He could even look on the Internet before departing to determine the traffic conditions at that moment, though this may (and most likely will) change by the time he travels along the route. As traffic

Loop detectors, video cameras, and reports by commuters will give information about a specific location, but does not give any data about the overall flow of traffic on the roadway.

conditions change, however, the driver will not know if his fastest route has changed, and thus may be traveling along a route that will take significantly longer than other routes.

Many departments of transportation have attempted to combat this issue by gathering real-time traffic data. The California Department of Transportation (CalTrans), among many other DOTs, has embedded hardware devices called loop detectors in freeways at specific locations (generally corresponding to on-ramps and off-ramps). These devices track the number of vehicles passing that point and the amount of time that a vehicle is physically occupying the space over the detector. This data is made publicly available via the CalTrans website [4], though there is at least a 15-minute delay and no data is provided for areas between the hardware sensors. The Performance Measurement System (PeMS) project at Berkeley [5] has created an algorithm to approximate the speed of the traffic at each of the loop detectors based on estimating the length of a vehicle and using the occupancy data provided by the loop detector [6]. Some companies, such as Sigalert.com [7] and Yahoo Maps [8], have created user-friendly interfaces via the web that expose the data gathered by CalTrans at each of the loop detector locations. Sigalert.com [7] has even gone so far as to provide a driver with the ability to have this data sent to him via a text message to his cellular phone at a certain time (presumably when the driver is in the middle of his trip).

In addition, incidents are reported by these companies through communication with CalTrans and the emergency response agencies, such as the California Highway Patrol (CHP). However, the means by which CalTrans and the CHP gather information about incidents primarily comes from commuters reporting the incidents via telephone. Helicopters can also be used to report traffic data, though the information they provide is the same as that of drivers reporting an incident, and it may be slightly delayed based on how quickly the helicopter identifies the incident and reports it.

There are three main limitations to gathering data based on the current approaches described above. The first limitation is that the data is gathered only at discrete locations of the transportation network. Loop detectors, video cameras, and reports by commuters will give information about a specific location, but does not give any data about the overall flow of traffic on the roadway. In addition, based on loop detector data, commuters know little about the flow between the loop detectors. One may conclude that an incident has occurred between two loop detectors because the flow of traffic is faster at one than the other, but the exact location can not be determined.

Expense is another limitation in gathering data at discrete locations. Loop detectors must be embedded in the roadway, which means they must be considered when building new highways or the highway must be closed temporarily to allow embedding the detector. In either case, there is a significant cost associated with installing and maintaining the loop detector.

The third limitation is the reliability of loop detectors. In CalTrans' District 7 (which encompasses Los Angeles), anywhere from 15% to 50% of the loop detectors may not be working at any given time [9]. The unreliability coupled with the maintenance required on a loop detector makes this solution quite unattractive.

Consider a world in which a driver could obtain up-to-the-minute traffic information covering all possible routes he might follow to get to his destination. Further, imagine the additional information that could be gathered if, at any given point in time, an exact simulation of the flow of vehicles in a freeway system could be obtained. In this paper, I consider how one might apply the latest mobile communication technologies and efficient data processing to help alleviate the problem of traffic congestion in a freeway system. Using some roadway infrastructure, such as the multi-billion dollar cellular network, will allow communicating real-time traffic data to and from vehicles via wireless, mobile devices, through a vehicle-to-infrastructure (V2I) architecture [23]. There has been little analysis of traffic assuming that a complete simulation of continuous traffic flow of a freeway system can be obtained at any given point in time. This is due to the fact that this data has never been available. UC Berkeley's Mobile Millennium project [21] and MIT's CarTel project [22] are both attempting to gather vehicle data via cellular phones, though privacy issues and determining whether a cellular phone is in a vehicle are challenges they are facing. At the University of Alaska, Anchorage, vehicle-tracking devices have been installed in a number of vehicles, which eliminates these two challenges, though has an increase in cost. Data from these three projects (and others) are currently being gathered and will hopefully aid in the traffic problems facing commuters. This data can be used to dynamically determine and adjust optimal paths, determine precise locations of incidents, identify conditions that lead to the occurrence of incidents, and allow researchers and transportation organizations to create applications that may not have even been considered yet. Analyzing this enormous amount of data can help improve the problems with traffic congestion in a transportation system.

Imagine the additional information that could be gathered if, at any given point in time, an exact simulation of the flow of vehicles in a freeway system could be obtained.

II. Approach

Assume that a free-flowing transportation system can be characterized as a directed graph $G = (V, E)$, where a vertex $v \varepsilon V$ is defined as a specific location of a highway and an edge $e \varepsilon E$ is defined as the section of the highway connecting two vertices. A path P on a directed graph $G = (V, E)$ can then be defined in the usual way:

$$P(i, j) = \{v_i, \ldots, v_j\} \mid i \leq k < j, (v_k, v_{k+1}) \in E.$$

The number of vertices defined within a freeway system will determine the accuracy of the fastest paths computed. The more vertices in the network, the more bandwidth that will be required to transmit fastest paths, but the more accurate the fastest paths will be. The weight of an edge is defined as the amount of time it takes to travel from one vertex to the next vertex. In other words, the amount of time to travel along that segment of the freeway in a vehicle is the weight of the corresponding edge in the graph.

From this representation of the highway system and these definitions, there are two separate, though closely related, problems that need to be addressed for Intelligent Transportation System applications. The first problem is how to gather the speed and location of all of the vehicles in the transportation system. The second problem is how to use this data to optimally determine the fastest path between a vehicle's current location and its desired destination.

For the speed gathering problem, I assume the location data will be retrieved through a global positioning system, which allows a vehicle to determine its location as a (latitude, longitude) pair. Currently, navigation systems exist in many vehicles, and they routinely track the (latitude, longitude) position of the vehicle. The speed of the vehicle can be retrieved either through the vehicle's computer system or by using triangulation of two geographical locations divided by the time to travel between the points. I am assuming that all cars will support this capability. These two pieces of data will then be transmitted to the system for storage and analysis. The transmission will occur over a wireless link, so any mobile device that has access to the Internet will be able to transmit this data to the system. I will not need to be concerned with users who are not on highways transmitting data because the application will know the (latitude, longitude) pairs that are valid for the highways. If a speed is transmitted with a (latitude, longitude) pair that is not within a highway, the application will ignore it.

For the fastest path application, the vehicle will transmit its current location and desired destination to the system and receive the fastest path from the source to the destination in return. The fastest path will be a sequence of vertices, as defined earlier in this section. The data will be transmitted in both directions over a cellular link, so any mobile device with Internet access will be able to transmit and receive the data.

III. Algorithms

The algorithms developed for the speed gathering and fastest path applications must be very efficient. With the potential of up to 1 million vehicles in the Los Angeles freeway system at a given point in time [4], if the algorithms are not very efficient, the latency will be so great that the system may appear to be unusable. Acceptable latency amounts are not yet known since this is based on user perception; however, a latency of more than several minutes seems unacceptable, as drivers will lose focus and traffic conditions may change.

In a V2I architecture, speeds can be updated continuously in real-time, which implies that the amount of time to traverse an edge could change often. This leads to recomputation of the fastest path needing to be done frequently, and efficient algorithms to achieve this are crucial.

To compute the fastest path between two vertices in a directed graph, there are three general classes of algorithms: the Naïve class, the Dynamic class, and a new class which I call the Pre-Computed class. If the weights of the edges in a graph are provided as the amount of time to traverse that section of the transportation network, any shortest path algorithm can be applied as a fastest path algorithm. The Naïve class of algorithms includes Floyd-Warshall's Algorithm [10] and Johnson's Algorithm [11], both of which compute the shortest path for *all* pairs of vertices in a graph. Floyd-Warshall's Algorithm can compute the shortest paths between all pairs of vertices in a weighted directed graph in $O(V^3)$. Johnson's Algorithm, on the other hand, re-weighs the graph

Table 1. Algorithmic running time comparison for fastest path algorithms.

	Pre-computation	Update Edge	Retrieve Fastest Path
Naïve (Johnson)	N/A	$O(V^2 \log V + VE)$	O(1)
Dynamic all-pairs shortest path (Demetrescu, Italiano)	N/A	$O(V^2 \log^3 V)$	O(1)
Pre-computed—constant update	$O(V^2E!)$	O(1)	O(Vm)
Pre-computed—constant query	$O(V^2E!)$	$O(V^2 m\log m)$	O(1)
Pre-computed—hybrid	$O(V^2E!)$	$O(V^2 m)$	O(m)

in a pre-processing step (if negative-weight edges exist) using Bellman-Ford's Algorithms [19, 20], and then uses Dijkstra's Algorithm [12] from every vertex for determining the shortest paths. The running time of Dijkstra's Algorithm, if implemented with a min-priority queue is $O(VlgV + E)$, giving a total running time for Johnson's Algorithm of $O(V^2lgV + VE)$, which is slightly faster than Floyd-Warshall's. Both Floyd-Warshall's algorithm and Johnson's algorithm are explained and pseudo-code is provided in [13].

Analyzing these algorithms with respect to the specific problem discussed in section II requires the algorithm to be executed every time an edge update occurs, which is when an updated speed is received. Although other algorithms I will discuss may be faster for updates, re-running these algorithms every time a speed is received allows the fastest path to be determined in constant time, since the fastest paths will already have been computed for all pairs of vertices when the optimal path is requested.

The Dynamic All-Pairs Shortest Path algorithm, developed by Camil Demetrescu and Giuseppe Italiano [14, 17], attempts to improve over the Naïve algorithms by determining the path with minimum cost between two vertices in a graph in constant time, though there is an edge update cost of $O(V^2log^3V)$.

The Pre-Computed class of algorithms takes advantage of the fact that the graph is rather static. With no edge insertions, all of the paths between all pairs of nodes can be pre-computed, which then does not require the application to take the extra step of determining all of the paths between two vertices when a request for a fastest path is being answered. The worst-case time to pre-compute the number of paths from one vertex to every other vertex is factorial with respect to the number of edges in the graph, or $O(E!)$. Then, to compute all of the paths between all pairs of vertices would require multiplying the time to compute the number of paths from one vertex to every other vertex ($O(E!)$) by the number of pairs of vertices, which is V^2. The overall running time to compute all paths between all pairs of vertices is then $O(V^2E!)$. Although this value is factorial with respect to the number of edges, it is really irrelevant since the pre-computation only occurs one time for the entire freeway system. Only when a change occurs (a freeway segment is added or removed) will this algorithm have to be re-executed. Using this pre-processing step, there are three approaches to determining fastest paths – updating the speed in constant time (Constant Update Algorithm), determining the fastest paths in constant time (Constant Query Algorithm), and compromising between updating the speed and determining the fastest paths in constant time (Hybrid Algorithm).

The *Constant Update Algorithm* sacrifices the speed of retrieving fastest paths for the amount of time it takes to update the weight of an edge. The algorithm does not maintain the fastest paths at all times, but rather computes the fastest path when requested by a vehicle. When the speed on an edge has changed by a specified threshold, the time to traverse that edge will be updated, which can be

Table 2. Actual running time comparison for fastest path algorithms (in milliseconds).

	Precomputation	Update Edge		Retrieve Fastest Path	
	Los Angeles and LA Downtown	Los Angeles	LA Downtown	Los Angeles	LA Downtown
Naïve (Johnson)	0	3,354	13,229	0	0
Dynamic all-pairs shortest path (Demetrescu, Italiano)	0	6,218	3,358	25	18
Pre-computed – constant update	31,545	0	0	18	7
Pre-computed – constant query	31,545	4,647	2,091	0	0
Pre-computed – hybrid	31,545	1,532	9	8	0

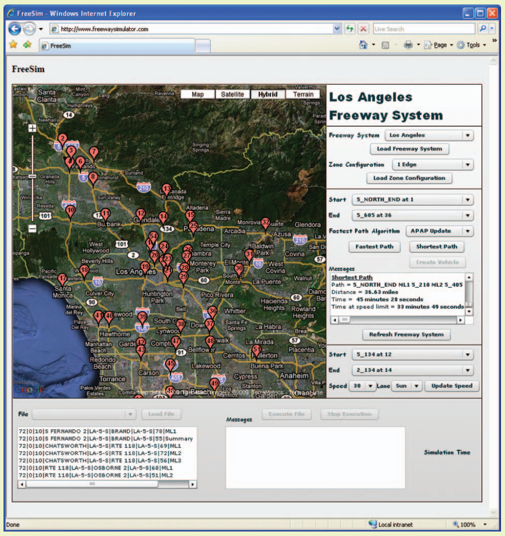
accomplished in constant time. To retrieve the fastest path, it is necessary to determine the time to traverse all m paths between the source vertex and the destination vertex (as determined by the pre-computation algorithm), and select the path with the minimum time. Assuming in the worst case that each of the m paths will contain a maximum of V vertices, the overall worst-case running time is $O(Vm)$.

The *Constant Query Algorithm* sacrifices the speed of updating an edge for the time to retrieve the fastest path. This algorithm always maintains the fastest path between all pairs of vertices by recalculating the fastest paths for all pairs of vertices that have a path containing the updated edge whenever an edge update occurs. When the speed on an edge has changed by a specified threshold, the time to traverse that edge will be updated, and the fastest paths for all pairs of vertices that have a path containing that edge will be recalculated. In the worst case, all paths for all pairs of vertices contain that edge, which are V^2 pairs, with m paths for each pair, giving V^2m. To insert an updated edge (implemented as a min heap) will take $O(logm)$, giving an overall running time of $O(V^2\ mlogm)$. However, to retrieve a path merely requires extracting the minimum path from the min heap (which is stored at the root), which can be done in constant time.

The *Hybrid Algorithm* attempts a compromise between the Constant Update and the Constant Query algorithms by maintaining the amount of time it takes to traverse each path whenever an edge update occurs. It does not maintain any type of ordering of paths, however, as the Constant Query algorithm did. When the speed on an edge has changed by a specified threshold, the time to traverse that edge will be updated, and all paths that contain that edge will have their overall time updated. If the updated edge is contained within every path of every pair of vertices, this algorithm will take $O(V^2\ m)$. To retrieve the fastest path, the application will only have to compare the times to traverse all m paths from the source to the destination and return the path with the minimum time, which can be accomplished in $O(m)$.

Table 1 contains a recapitulation of the algorithmic running times discussed in this section. One important observation to note is that, depending on the value of m, the Pre-Computed class of algorithms may execute faster than the other classes of algorithms. Running these algorithms through FreeSim [16, 18] proved that only a constant number of paths between any two nodes needs to be considered. Also important to note is that the number of edge updates will greatly exceed the number of fastest path queries, which should be taken into account when considering which algorithm will run most efficiently for this problem. With that being the case, the Constant Update algorithm will execute the fastest in a live setting.

Table 2 shows the actual running times in milliseconds of the fastest path algorithms when run on two different transportation networks. The Los Angeles network



Los Angeles Freeway System

Freeway System: Los Angeles
Zone Configuration: 1 Edge
Start: 5_NORTH_END at 1
End: 5_605 at 36
Fastest Path Algorithm: APAP Update

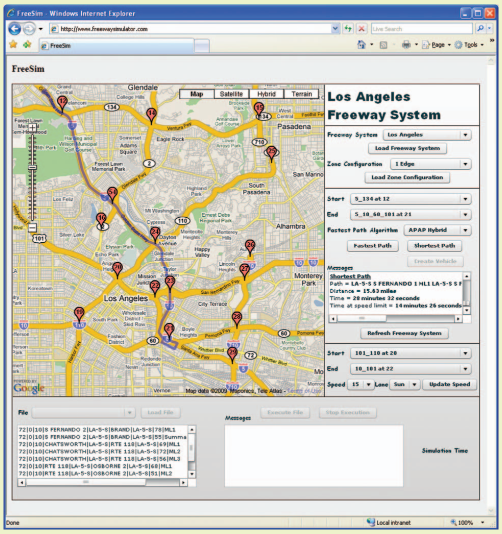
Messages
Shortest Path
Path = 5_NORTH_END ML1 5_210 ML2 5_405
Distance = 36.63 miles
Time = 45 minutes 28 seconds
Time at speed limit = 33 minutes 49 seconds

Start: 5_134 at 12
End: 2_134 at 14
Speed: 30 Lane: Sun

File
```
72|0|10|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|78|ML1
72|0|10|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|55|Summary
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|69|ML1
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|72|ML2
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|56|ML3
72|0|10|RTE 118|LA-5-S|OSBORNE 2|LA-5-S|68|ML1
72|0|10|RTE 118|LA-5-S|OSBORNE 2|LA-5-S|51|ML2
```

FIG 1 Freesim screenshot of Los Angeles freeway system graph with freeway intersections as vertices.

contains 53 nodes and 155 edges (where the nodes represent freeway intersections), while the LA Downtown network contains 97 nodes and 101 edges (where the nodes represent on-ramps and off-ramps). FreeSim screenshots of the graphs for these two transportation systems are provided in Figures 1 and 2. The values in Table 2 were obtained by running 10,000 random edge updates and 10,000 random fastest path queries, then averaging the amounts of time to compute each respectively. It is important to note that the actual number of milliseconds is not as important



Los Angeles Freeway System

Freeway System: Los Angeles
Zone Configuration: 1 Edge
Start: 5_134 at 12
End: 5_10_60_101 at 21
Fastest Path Algorithm: APAP Hybrid

Messages
Shortest Path
Path = LA-5-S S FERNANDO 1 ML1 LA-5-S S F
Distance = 15.63 miles
Time = 28 minutes 32 seconds
Time at speed limit = 14 minutes 26 seconds

Start: 101_110 at 20
End: 10_101 at 22
Speed: 15 Lane: Sun

File
```
72|0|10|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|78|ML1
72|0|10|S FERNANDO 2|LA-5-S|BRAND|LA-5-S|55|Summa
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|69|ML1
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|72|ML2
72|0|10|CHATSWORTH|LA-5-S|RTE 118|LA-5-S|56|ML3
72|0|10|RTE 118|LA-5-S|OSBORNE 2|LA-5-S|68|ML1
72|0|10|RTE 118|LA-5-S|OSBORNE 2|LA-5-S|51|ML2
```

FIG 2 Freesim screenshot of Los Angeles downtown freeway system graph with off-ramps and on-ramps as vertices.

Vehicle tracking data can be used to dynamically determine and adjust optimal paths, determine precise locations of incidents, identify conditions that lead to the occurrence of incidents, and allow researchers and transportation organizations to create applications that may not have even been considered yet.

as the relationship between the running times of the different algorithms. The Naïve class of algorithms (specifically Johnson's algorithm) proved to take the most amount of time to update an edge since that required re-computing the fastest paths between all pairs of nodes. In the LA Downtown graph, for which the vertices represented on-ramps and off-ramps of a freeway, Johnson's algorithm was an entire order of magnitude slower than any of the other algorithms. Ignoring the pre-computation step, it can be seen that updating the edges has the highest cost for all of the algorithms except the Pre-Computed – Constant Update algorithm, which updates an edge in a negligible amount of time. To compute fastest paths, Johnson's algorithm and the Pre-Computed – Constant Query algorithm perform the fastest since they had already computed the fastest paths when the edge update occurred and were just reading the fastest path from a data structure at query time, while the Pre-Computed – Hybrid algorithm is nearly as fast. Further, since the values are in milliseconds, all of the algorithms appear to run in nearly negligible time. As was stated earlier, in a V2I network, the number of speeds received from vehicles will outweigh the number of fastest paths requested, so the algorithm that will prove to be the fastest for this application practically will be the Pre-Computed – Constant Update algorithm, which substantially improves on the running times of other fastest path algorithms.

IV. Conclusion

In this paper, I have provided algorithms that can be used when gathering traffic data from a continuous flow of traffic via a V2I architecture rather than at discrete locations. Current means of gathering traffic data (loop detectors, video cameras, helicopters, etc.) are expensive to implement and maintain. Gathering data via mobile devices communicating with some roadway infrastructure allows the data to be transmitted from any vehicle that contains a device capable of sending data wirelessly. With the large amount of data gathered by individual vehicles, additional applications that are not possible based on the current means of gathering data can be developed. Specifically, updating a user as to his fastest path while in the middle of his commute, identifying precise locations of incidents, determining probabilities of incidents occurring, and improving the overall throughput of vehicles in a highway system are a few of the potential applications, though other applications of the data are sure to arise.

I have presented several algorithms used to update edges of the highway graph as updated speeds are received and to optimize the task of generating a fastest path for a commuter based on his current location and his desired destination. Realizing that the number of edge updates will greatly exceed the number of fastest path queries, the Pre-Computed – Constant Update algorithm executes the fastest. The value of *m*, which is the number of paths to consider between any pair of vertices, is also important to remain constant.

Traffic is a problem in many countries of the world. Adding lanes to highways or adding additional hardware is not always a viable solution based on cost and space limitations. Using existing roadway communication infrastructure and gathering data from a continuous flow of traffic has the potential to improve the overall flow of traffic in a transportation network with a very limited cost.

References

[1] Google Maps. [Online]. Available: http://maps.google.com
[2] Mapquest. [Online]. Available: http://www.mapquest.com
[3] D. Shrank and T. Lomax, "2004 Urban mobility report," College Station, TX, *Texas Transportation Institute's Annual Urban Mobility Rep.*, Sept. 2004.
[4] CalTrans 2003 AADT – Average Annual Daily Traffic, 2003. [Online]. Available: http://www.dot.ca.gov/hq/traffops/saferesr/trafdata/
[5] Performance measurement system (PeMS). [Online]. Available: http://pems.eecs.berkeley.edu
[6] Z. Jia, C. Chen, B. Coifman, and P. Varaiya, "The PeMS algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors," in *Proc. IEEE 4th Int. Intelligent Transportation Systems Conf.*, Feb. 2001.
[7] Sigalert.com. [Online]. Available: http://www.sigalert.com
[8] Yahoo Maps. [Online]. Available: http://maps.yahoo.com
[9] H.-S. Linda, "Staying in the loop," *Fall 2001 Tech. Transfer Newsletter*, UC Berkeley, 2001.
[10] F. Robert, "Algorithm 97 (SHORTEST PATH)," *Commun. ACM*, 1977.
[11] J. Donald, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, 1977.
[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959.
[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.
[14] C. Demetrescu and G. Italiano, "A new approach to dynamic all pairs shortest paths," in *Proc. ACM Symp. Theory of Computing*, June 2003.
[15] S. Yang. (2008 Feb.). "Joint Nokia research project captures traffic data using GPS-enabled cell phones," *UC Berkeley News* [Online]. Available: http://www.berkeley.edu/news/media/releases/2008/02/08_gps.shtml
[16] J. Miller and E. Horowitz, "FreeSim – A free real-time traffic simulator," in *Proc. IEEE 10th Int. Intelligent Transportation Systems Conf.*, Sept. 2007.
[17] C. Demetrescu and G. Italiano, "Experimental analysis of dynamic all pairs shortest path algorithms," *ACM Trans. Algorithms*, vol. 2, no. 4, Oct. 2006.
[18] FreeSim. [Online]. Available: http://www.freewaysimulator.com
[19] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, 1958.
[20] Ford Jr., R. Lestor, and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.
[21] UC Berkeley's Mobile Millennium Project. [Online]. Available: http://traffic.berkeley.edu
[22] MIT's CarTel Project. [Online]. Available: http://cartel.csail.mit.edu
[23] J. Miller, "Vehicle-to-vehicle-to-infrastructure (V2V2I) intelligent transportation system architecture," in *Proc. IEEE 4th Intelligent Vehicles Symp.*, June 2008.

ITS