

Algorithms for Real-Time Gathering and Analysis of Continuous-Flow Traffic Data

Jeffrey Miller
Department of Computer Science
University of Southern California
Jeffrey.Miller@usc.edu

Ellis Horowitz
Department of Computer Science
University of Southern California
horowitz@usc.edu

***Abstract* – In this paper we present a new approach to gathering traffic data over a continuous-flow of traffic rather than at discrete locations, as is the case with existing technologies. Loop detectors and video cameras, among other devices, currently provide the primary means for gathering data, but the data is gathered at discrete locations. Using cellular technology, the speed and location of each vehicle can be gathered in real-time over a continuous flow, which will allow more novel applications, such as incident identification and probabilities of an incident occurring, 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 destination is maintained. We present and analyze algorithms that can achieve these goals in an efficient manner.**

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 recent study performed by Texas A&M in 2004 [1] 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 [1].

Assume that a person would like to travel by car from downtown Los Angeles to his home in the San Fernando Valley during rush-hour on a weekday. There are many different routes that he could take, but how would he decide which highways and streets to use? He could use a street map to determine his route or use electronic maps, such as Mapquest [2] or Google Maps [3]. 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 freeway 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 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) has embedded hardware devices called loop detectors in freeways no closer than one-quarter mile apart. 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 sensor 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.

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 highway. Loop detectors, video cameras, and reports by commuters will give information about a current location, but does not give any data about the overall flow of traffic on the highway. In addition, based on loop detector data, commuters know nothing 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.

The expense is another limitation in gathering data at discrete locations. Loop detectors must be embedded in the highway, which means they must be considered when building new highways or the highway must be closed

temporarily to allow construction to embed the detector. In either case, there is a significant cost associated with installing 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 an unattractive one.

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. In addition, 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, we 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. Piggybacking on the multi-billion dollar cellular network will allow communicating real-time traffic data to/from vehicles via wireless, mobile devices. This data can then be used to dynamically determine and adjust optimal paths, determine precise locations of incidents, identify conditions that lead to higher probabilities of incidents, and allow researchers and transportation organizations to create applications that may not even have been considered yet that use this enormous amount of data to help improve the problems with traffic congestion in a freeway system.

In section 2, we will discuss the work that has already been done in this area and how our research differs. In section 3, we will present our approach for representing a freeway system as a graph and how vehicles will send their speed and location data to our system. In section 4, we will provide an explanation of the algorithms used to gather the speed and location data from each vehicle and then generate the fastest path to each vehicle's destination from its current location based on this data. We will discuss future work in section 5, and the paper will be concluded in section 6.

II. RELATED WORK

There has been significant work done on monitoring traffic, both academically and professionally. Many transportation organizations use loop detectors and video cameras to monitor traffic flow and report the gathered information via some form of media (i.e. TV, radio, internet, etc.) [6]. More recent work academically has focused on using sensor networks to gather traffic data, since sensors are much less costly than the hardware-intensive loop detectors and can be installed virtually anywhere with very little overhead [10, 11]. However, the sensors are autonomous entities that run on a limited energy source, so as the batteries die, there will be a cost associated with repairing certain nodes.

There is a private company called Airsage [12] that is attempting to gather traffic data via cellular phones, though they are not leveraging any Global Positioning System technology because of overhead and privacy issues. Airsage determines the speed and location of a user by how he

moves from one cellular tower's zone to another. Unfortunately, this approach will not work for densely populated areas where there are a significant number of vehicles on surface streets that are immediately adjacent to highways, for the Airsage application does not know whether the cellular phone is on the highway or the surface street.

Assuming that data at discrete locations can be obtained (which is currently done by many transportation organizations), there has been some work done in estimating the amount of time that it takes to travel along a specific path [13]. The Berkeley PeMS project has even gone so far as to attempt to predict travel times based on historical data [14]. However, most of this analysis is based on only obtaining traffic data at specific points, as is the case with loop detectors [15, 16].

Recently, there has been some research into traffic flow on highways [17], and traffic flow utilizing a mixed sequence of automated and manually driven vehicles [18]. It has been predicted that vehicles that drive themselves on the highways will increase the throughput and help aid in the traffic problems, though this will not be an overnight change, so considerations have to be taken as to mixed-mode operation of automated and manually driven vehicles.

Some work has also been done in determining how much data collected from a freeway system is actually necessary to accurately route vehicles to reduce travel time [19]. If the amount of data can be reduced, then the actual running times of the algorithms will be reduced.

With the research that has been and is being conducted, there has been little analysis of traffic assuming that a complete simulation of the freeway system can be obtained at any given point in time. This is due to the fact that this data has never been available, though we believe that technology is fast reaching the point of being able to obtain this data and use it to aid in the traffic problems facing commuters.

III. APPROACH

Assume that a freeway system can be characterized as a directed graph $G=(V, E)$, where a vertex $v \in V$ is defined as an on-ramp or off-ramp of a highway and an edge $e \in E$ 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:

$$P(i, j) = \{v_1, \dots, v_j\} \mid i \leq k < j, (v_k, v_{k+1}) \in E$$

Note that one limitation in the above definition with respect to our application is that v_j cannot be an on-ramp, as the destination of a path cannot be a vertex at which the vehicle cannot exit the highway. However, v_i may be an on-ramp or an off-ramp, for the source vertex of a path will change as a vehicle is moving along the highway. This means that the source vertex may be an off-ramp if the vehicle is closest to an off-ramp when the path is being determined. Let us also define the weights of the edges as the amount of time it will take to drive from one ramp to the next ramp. In other words, the amount of time to travel along that segment of

the freeway in a vehicle will be 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 we will consider in this paper. The first problem is how to gather the speed and location of all of the vehicles in the highway 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, we 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 cars, 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. We are assuming that all cars will support this capability. These two pieces of data will then be transmitted to our system for storage and analysis. The transmission will occur over a cellular link, so any mobile device that has access to the Internet will be able to transmit this data to our system. We will not have to be concerned with users who are not on highways transmitting data because our 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 the desired destination location 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.

IV. 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 our system, speeds will be updated continuously in real-time, which imply that the amount of time to traverse an edge could change often. This leads to re-computation of the shortest path needing to be done frequently, and efficient algorithms to achieve this are crucial.

To compute the shortest path between two points in a directed graph, we discuss three classes of algorithms: the Naïve class, the Dynamic class, and a new class which we call the Pre-Computed class. In section IV.A, we discuss the Naïve class of algorithms, in section IV.B, we discuss the Dynamic class, and in section IV.C, we discuss the new Pre-Computed class of algorithms.

A. Naïve Class of Algorithms

The Naïve class of algorithms includes Floyd-Warshall’s Algorithm [20] and Johnson’s Algorithm [21], both of which compute the shortest path for all pairs of vertices in a graph. Floyd-Warshall’s Algorithm takes advantage of “intermediate vertices”, and the paths to which intermediate vertices are a part. An intermediate vertex of a simple path is any vertex that is not an end-point of the path. Floyd-Warshall’s Algorithm uses a recursive approach that determines a minimum-weight path along each path from one node to every other node in the graph. The running time of Floyd-Warshall’s Algorithm is $O(V^3)$.

Johnson’s Algorithm, on the other hand, re-weights the graph in a pre-processing step (if negative-weight edges exist), and then uses Dijkstra’s Algorithm [22] from every vertex for determining the shortest paths. Since our specific application does not contain any negative-weight edges, the pre-processing step of Johnson’s Algorithm is irrelevant. Dijkstra’s Algorithm iterates through all of the vertices that are neighbors of the initial vertex and determines the shortest path. It then iterates through all of the vertices that are neighbors of the initial node’s neighbors and determines the shortest path to them. It continues in this fashion until the shortest path to all of the nodes in the graph has been found. 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 have good pseudo-code and explanations in [23].

Analyzing these algorithms with respect to our specific problem requires the algorithm to be executed every time an edge update occurs, which is when an updated speed is received. Although other algorithms we 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. These algorithms will be our basis for creating other algorithms that run faster or have other advantages over these.

B. Dynamic Class of Algorithms

The Dynamic All-Pairs Shortest Path class of algorithms was developed by Demetrescu and Italiano [24], who created an algorithm that can have a constant query cost of determining the path with minimum cost between

TABLE I. RUNNING TIME COMPARISON FOR FIVE ALGORITHMS DISCUSSED

| | 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 | $O(V^2 \log^3 V)$ | $O(1)$ |
| All-Pairs All-Paths Pre-Computed – Constant Update | $O(V^2 E!)$ | $O(1)$ | $O(Vm)$ |
| All-Pairs All-Paths Pre-Computed – Constant Query | $O(V^2 E!)$ | $O(V^2 m \log m)$ | $O(1)$ |
| All-Pairs All-Paths Pre-Computed – Hybrid | $O(V^2 E!)$ | $O(V^2 m)$ | $O(m)$ |

two vertices in a graph with an edge update cost of $O(V^2 \log^3 V)$. This class of algorithms supports edge updates, deletions, and insertions by noticing that changing the weight of an edge to have infinite cost will make that edge never become part of a minimum cost path, which essentially removes that edge from the graph. Edge inserts behave just as edge updates do, though the weight of the edge will be changed from infinite (i.e. the edge did not previously exist) to its new value. With respect to our specific problem, the Dynamic All-Pairs Shortest Path algorithms execute more efficiently than the Naïve algorithms, though the edge update cost is still executing in polynomial time. The fastest path running time is constant, just as it was in the Naïve case. For more information on the Dynamic All-Pairs Shortest Path algorithms, refer to Demetrescu and Italiano [24], Klein and Subramanian [25], and King [26].

C. Pre-Computed Class of Algorithms

The All-Pairs All-Paths 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 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 $O(V^2 E!)$. 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.

The following three algorithms all use the pre-computation algorithm described above, though how they update edge weights and determine shortest paths differ. The first algorithm allows for constant edge updates whenever a new speed is received from a vehicle. The second algorithm

allows for constant fastest path queries whenever a vehicle requests a fastest path. The third algorithm is a hybrid

approach between the previous two, attempting to optimize the algorithms.

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 way the algorithm works is by not maintaining the fastest paths at all times, but rather computing the fastest path when requested by a vehicle. When a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first¹. If the average speed on that edge has changed by s miles per hour, the time to traverse that edge will be updated, which can be 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, 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 running time becomes $O(mV)$.

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 a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first. If the average speed on that edge has changed by s miles per hour, 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 $V^2 m$. To insert an updated edge (implemented as a binary search tree) will take $O(\log m)$, giving an overall running time of $O(V^2 m \log m)$. However, to retrieve a path merely requires extracting the minimum path from the binary search tree, 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 a sorted sequence of paths, however, as the Constant Query algorithm

¹ The average number of vehicles passing a specific point in the LA freeway system in an hour is 2000 [17]. With 4 lanes on the freeway, this gives 667 cars every 5 minutes.

did. When a new speed is transmitted to the system, the application will buffer the data for that specific edge for 667 vehicles or 5 minutes, whichever comes first. If the average speed on that edge has changed by s miles per hour, 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. There are only m paths for each pair of vertices, giving a running time of $O(m)$.

Table 1 contains a recapitulation of the running times discussed in this section. The 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. We can hypothesize that we only need to consider a constant number of paths between any two nodes, though running these algorithms in a simulation will prove the validity of that statement. 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 set of algorithms will run most efficiently for this problem. With that being the case, we would conclude that the Constant Update algorithm will execute the fastest in a live setting.

V. FUTURE WORK

We plan to build a simulation of the Los Angeles freeway system and provide it with data from CalTrans. This will allow us to analyze the algorithms described above and verify that the computing time analyses do actually predict their performance. Moreover, we expect that the simulation will lead us to several variations of the above algorithms that will identify alternate strategies for drivers other than the fastest route to their destinations.

Assuming that we can gather data from all vehicles as described in this paper, it becomes possible to provide drivers with lane-travel information. For example, if an accident has occurred in the first lane of a highway, the system can notify drivers to move to the fourth lane if they are still going to traverse a route that passes the incident. This is dependent on extremely accurate GPS data, which is not currently available in the consumer market.

With the large amount of data that we will be gathering, it becomes possible to identify the location of incidents on the highway quite easily. A future improvement to this work would be to determine what conditions surround an incident. We believe that based on road conditions, weather, traffic conditions, speed of vehicles, time of day, location on highway, etc., that we will be able to identify situations that have a high probability of resulting in an incident.

VI. CONCLUSION

In this paper, we have provided a new approach to gathering traffic data from a continuous flow of traffic 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 cellular phones, leveraging the existing multi-billion dollar cellular infrastructure, allows the data to be gathered from any vehicle that contains a cell phone or a device capable of transmitting data to a cellular tower. With this large amount of data, additional applications that are not possible based on the current way 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.

We 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 destination. Based on the fact that the number of edge updates will greatly exceed the number of fastest path queries, the All-Pairs All-Paths Pre-Computed – Constant Update algorithm appears to execute 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 that is not going to go away overnight. Adding lanes to highways or adding additional hardware is not always a viable solution based on cost and space limitations. Using existing infrastructure and gathering data from a continuous flow of traffic can improve the overall flow of traffic on all highways with a very limited cost.

VII. REFERENCES

- [1] Shrank, David, Tim Lomax. "2004 Urban Mobility Report." *Texas Transportation Institute's Annual Urban Mobility Report*, September 2004.
- [2] Mapquest. <http://www.mapquest.com>.
- [3] Google Maps. <http://maps.google.com>.
- [4] CalTrans 2003 AADT – Average Annual Daily Traffic, 2003. <http://www.dot.ca.gov/hq/traffops/saferesr/trafdata/>.
- [5] Performance Measurement System (PeMS). <http://pems.eecs.berkeley.edu>.
- [6] Jia, Zhanfeng, Chao Chen, Ben Coifman, Pravin Varaiya. "The PeMS algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors." *IEEE 4th International Intelligent Transportation Systems Conference*, February 12, 2001.
- [7] Sigalert.com. <http://www.sigalert.com>.
- [8] Yahoo Maps. <http://maps.yahoo.com>.
- [9] Howe-Steiger, Linda. "Staying in the Loop." *Fall 2001 Tech Transfer Newsletter*, UC Berkeley, 2001.
- [10] Coleri, Sinem, Sing Yiu Cheung, Pravin Varaiya. "Sensor Networks for Monitoring Traffic." *42nd Annual*

Allerton Conference on Communication, Control, and Computing, September 2004.

[11] Ding, Jiagen, Sing-Yiu Cheung, Chin-Woo Tan, Pravin Varaiya. "Signal Processing of Sensor Node Data for Vehicle Detection." *IEEE 7th International Intelligent Transportation Systems Conference*, October 2004.

[12] Airsage. <http://www.airsage.com>.

[13] Rice, John, Erik van Zwet. "A Simple and Effective Method for Predicting Travel Times on Freeways." *IEEE 5th International Intelligent Transportation Systems Conference*, August 2002.

[14] Choe, Tom, Alexander Skabardonis, Pravin Varaiya. "Freeway Performance Measurement System (PeMS): An Operational Analysis Tool." *81st Annual Meeting Transportation Research Board*, January 2002.

[15] Kwon, Jaimyoung, Benjamin Coifman, Peter Bickel. "Day-to-Day Travel Time Trends and Travel Time Prediction from Loop Detector Data." *Transportation Research Board #1717*, 2000.

[16] Petty, Karl F., Peter Bickel, Jiming Jiang, Michael Ostland, John Rice, Ya'acov Ritov, Frederic Schoenberg. "Accurate estimation of travel times from single-loop detectors." *Transportation Research, Part 1 (Policy and Practice)*, Volume 32A, #1, January 1998.

[17] Bickel, Peter, Chao Chen, Jaimyoung Kwon, John Rice, Pravin Varaiya, Erik van Zwet. "Traffic Flow on a Freeway Network." *Proceeding of MSRI Workshop on Nonlinear Estimation and Classification*, March 19-29, 2001.

[18] Li, Kim, Petros Ioannou. "Modeling of Traffic Flow of Automated Vehicles." *IEEE Transactions on Intelligent Transportation Systems*, Volume 5, Number 2, June 2004.

[19] Kim, Seongmoon, Mark E. Lewis, and Chelsea C. White, III. "State Space Reduction for Nonstationary Stochastic Shortest Path Problems With Real-Time Traffic Information." *IEEE Transactions on Intelligent Transportation Systems*, Volume 6, Number 3, September 2005.

[20] Floyd, Robert. "Algorithm 97 (SHORTEST PATH)." *Communications of the ACM*, 1977.

[21] Johnson, Donald. "Efficient Algorithms for Shortest Paths in Sparse Networks." *Journal of the ACM*, 1977.

[22] Dijkstra, E.W. "A note on two problems in connexion with graphs." *Numerische Mathematik*, 1959.

[23] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms – 2nd Edition*. The MIT Press, 2001.

[24] Demetrescu, Camil and Giuseppe Italiano. "A New Approach to Dynamic All Pairs Shortest Paths." *ACM Symposium on Theory of Computing*, June 2003.

[25] Klein, P.N. and S. Subramanian. "A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs." *Algorithmica*, Volume 22, 1998.

[26] King, Valerie. "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs." *IEEE Symposium on Foundations of Computer Science*, 1999.