Association for Computing Machinery

Advancing Computing as a Science and Profession

IWCTS 2009

Proceedings of
The Second International Workshop on Computational Transportation Science
in conjunction with ACM SIGSPATIAL GIS 2009

November 3, 2009, Seattle, WA, USA.

The Association for Computing Machinery, Inc.
1515 Broadway
New York, New York 10036

ACM ISBN: 978-1-60558-324-2/08/11

Additional copies may be ordered prepaid from:

	Phone: 1-800-342-6626
ACM Order Department	(U.S.A. and Canada)
P.O. BOX 11405	+1-212-626-0500
Church Street Station	(All other countries)
New York, NY 10286-1405	Fax: +1-212-944-1318
	E-mail: acmhelp@acm.org

Printed in the U.S.A.

FOREWORD

IWCTS 2009, held in conjunction with the seventeenth international conference on the advances in geographic information systems (ACM SIGSPATIAL GIS 2009), was the second international workshop on Computational Transportation Science.

The aim of the workshop was to bring together the researchers from the areas of computer science and transportation science to explore areas of synergy and lay a foundation for research and development in the emerging discipline of Computational Trasnportation Science (CTS). CTS is a discipline that combines computer science with modeling, planning, and economic aspects of transportation. It aims to go beyond navigation methods and plans to address data management issues and data mining in the area of transportation science thereby improving efficiency, equity, mobility, accessibility, and safety of current transportation systems.

In the recent years, we have seen enormous progress in the areas of vehicular technology, hand-held devices, and communication infrastructure that often involves millions sensors that can communicate with each other. These advances definitely offer enormous potential to improve the performance of current transportation applications. Miniature computing devices and advances in wireless communication and sensor technology have definitely moved computing from stationary desktops to mobile devices making computation in transportation more ubiquitous, opening up enormous opportunities for information technology. However, the impact of information technology on these applications does not match the dramatic effect it has had, on other domains in business and science. In addition to implementing new computing strategies specially suitable for popular wireless devices such as cell phones and PDAs, computer science can also bring sophisticated geospatial and spatio-temporal information management capabilities to the area of transportation science.

This workshop is the result of a tremendous team effort. We wish to express our immense gratitude towards the authors for their valuable contribution. We thank the program committee members for their timely and thorough reviews. We are extremely grateful to Prof Ouri Wolfson for his guidance and generous help. Finally, we would like to express our appreciation to the ACM SIGSPATIAL for sponsoring the workshop.

We hope that you found the program lively and thought-provoking, and that the workshop provided you with a venue to share and discuss your ideas with other researchers and practitioners.

November, 2009 IWCTS 2009 Organizing Chairs

IWCTS 2009 Organization

General Co-Chairs

Shashi Shekhar, University of Minnesota, USA.
Glenn Geers, NICTA, Australia.

Program Committee Co-Chairs

Betsy George, Oracle Corporation, USA.
Sangho Kim, ESRI, USA.

Steering Committee

Ouri Wolfson, University of Illinois at Chicago, USA.
Budhendra Bhaduri ,Oak Ridge National Laboratory, USA.

Program Committee

Gennady Andrienko , Fraunhofer Institute IAIS, Germany.
Walid Aref , Purdue University, USA.
Claus Brenner , Leibniz Universitat Hannover, Germany.
Chen Cai, Neville Roach Laboratory, National ICT Australia.
Oscar Franzese, Oak Ridge National Laboratory, USA.
Kostas Goulias, University of California, Santa Barbara, USA
Le Gruenwald, University of Oklahoma, USA.
Wee-Liang Heng, ESRI, USA.
Der-Horng Lee, National University of Singapore, Singapore.
Cheng Liu, Oak Ridge National Laboratory, USA
Harvey Miller, University of Utah, USA.
Jim Nutaro, Oak Ridge National Laboratory, USA.
Stacy Patterson, University of California, Santa Barbara, USA.
Jay Sandhu, ESRI, USA.
Monika Sester, University of Hannover, Germany.
Pravin Varaiya, University of California, Berkeley, USA.
Stephan Winter, The University of Melbourne, Australia.
Jack Wang, Oracle Corporation, USA.
Bo Xu, University of Illinois at Chicago, USA.
Bruce Ralston, University of Tennessee, USA.

Table of Contents

Shortest Paths in Time-Dependent FIFO Networks Using Edge Load Forecasts [*]

Frank Dehne
School of Computer Science
Carleton University
Ottawa - Canada
frank@dehne.net

Masoud T. Omran
School of Computer Science
Carleton University
Ottawa - Canada
mtomran@scs.carleton.ca

Jörg-Rüdiger Sack
School of Computer Science
Carleton University
Ottawa - Canada
sack@scs.carleton.ca

ABSTRACT

We study the problem of finding shortest paths in time-dependent networks with edge load forecasts where the behavior of each edge is modeled as a time-dependent arrival function with FIFO property. Here, we present a new algorithm that computes for a given start node s and destination node d, the shortest paths and earliest arrival times for all possible starting times. Our algorithm runs in time $O((F_d + \lambda)(|E| + |V| \log |V|))$ where F_d is the output size (number of linear pieces needed to represent the earliest arrival time function) and λ is the input size (number of linear pieces needed to represent the local earliest arrival time functions for all edges in the network). Our method improves significantly on the best previously known algorithm which requires time $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between the start node s to any node in the network. It has been conjectured that there are cases where F_{max} is of super-polynomial size; however, even in such cases, F_d might still be of linear size. In such cases, our algorithm would take polynomial time to find the solution, while other methods require super-polynomial time. Both of the above methods are not useful in practice for graphs where F_d is of super-polynomial size. For such graphs, we present the first approximation method to compute for all possible starting times at s, the earliest arrival times at d within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s.

1. INTRODUCTION

Finding shortest paths in networks is one of the basic operations in Transportation Science, Computer Networks, Robotics, VLSI Design and many other applications. Although well-studied conventional static shortest path algorithms play a fundamental role in applications with non-changing nature, many real-world applications are changing over time [1, 2, 6, 4]. For example, in a road network, the shortest path from a given start node to a destination node during rush hour is not the same as during low traffic periods. Here, we study dynamically changing applications in which network properties are changing over time in a predictable manner and are given as edge load forecasts. For example, in many road networks the traffic load on each link changes predictably during the day. We are interested in finding the shortest path between two nodes of the network for any given time during the day. More precisely, a time-dependent network with edge load forecasts is modeled as a directed graph $G = (V, E)$ where each edge (v, w) is assigned an arrival time function $a_{vw}(t)$ which represents the arrival time at node w for departure time t at node v. Typically, piece-wise linear arrival time functions are used to approximate more complex functions. The notion of arrival time function is extensible to any path $p = \langle v_1, v_2, \ldots, v_k \rangle$ of the network. Starting from v_1 at time t implies an arrival at v_k at time $a_p(t) = a_{v_{k-1}v_k}(\ldots(a_{v_2v_3}(a_{v_1v_2}(t))))$. We note that if the $a_{v_{i-1}v_i}(t)$ are piece-wise linear then $a_p(t)$ is piece-wise linear as well. Given a start node s and a destination node d, our goal is to find the earliest arrival time function $A_{sd}(t)$ from s to d for all $t \in [0, T]$. $A_{sd}(t)$ is the minimum over all $a_p(t)$ for all possible paths p from s to d. As discussed by Orda and Rom in [15], this problem is NP-hard in its general form but there are variations of the problem which are not. For example, in earlier work [10], we considered a version of the problem, where the slope of each linear piece is either 0 or 1. This can be viewed as a network in which edges are available during given intervals and the travel time in each interval is of fixed value. In such a network, if an edge is not available for some arrival time, then one can wait until the next interval becomes available. For such networks, we proposed an $O(\kappa(|E| + |V|Log|V|))$ time algorithm. where κ denotes the total number of availability intervals in the entire graph.

Here, we consider a general class of time-dependent shortest path problems in which the FIFO property holds. The FIFO property is very common in many networks, including road networks, and is defined as having non-decreasing arrival time functions on all edges of the network. This means that for every edge (v, w), a later start at v implies a later arrival at w which is typically the case for predictable dynamic networks. A naive algorithm for this case which computes the earliest arrival time function for every possible path from s to d and then calculates the lower envelope would need ex-

[*]Research supported by NSERC, SUN Microsystems of Canada and HPCVL.

ponential time in many cases because many networks would have an exponential number of possible paths between s and d. As shown by Orda and Rom in [15], the time-dependent shortest paths problem for a time-dependent network with FIFO property can be solved in time $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between s and any node in the network.

In this paper, we present a new algorithm for solving the time-dependent shortest paths problem for a time-dependent network with FIFO property. The algorithm runs in time $O((F_d + \lambda)(|E| + |V| \log |V|))$, where F_d is the output size (number of linear pieces needed to represent the earliest arrival time function) and λ is the input size (number of linear pieces needed to represent the local earliest arrival time functions for all edges in the network). Our method improves significantly upon the best previously known method by Orda and Rom [15]. It has been conjectured [9] that there are cases, where F_{max} is of super-polynomial size. Since $F_{max} \geq F_d$, even in such cases, F_d might still be of linear size. In such cases, our algorithm would take polynomial time to find the solution, while other methods require super-polynomial time.

We also study the case where the output size F_d might be [9] super-polynomial. All previously known methods (including our method outlined above) are not useful in practice for such graphs. In this paper, we present the first approximation method for such instances of the time-dependent shortest path problem. Our method computes for all possible starting times at s the earliest arrival times at d within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E|+|V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s.

The remainder of this paper is organized as follows. In the following Section 2 we discuss previous work and related relevant results from similar problem settings. In Section 3 we discuss some structural properties of the time-dependent shortest path problem. Our new algorithm which solves the time-dependent shortest paths problem for a time-dependent network with FIFO property in time $O((F_d + \lambda)(|E| + |V| \log |V|))$ is presented in Section 4. Our approximation algorithm for time-dependent shortest path instances with possibly super-polynomial size output is presented in Section 5. Section 6 concludes the paper.

2. PREVIOUS ALGORITHMS AND RESULTS

The problem of finding a time-dependents shortest path was first proposed in 1966 by Cooke and Halsey [7]. They considered time to have discrete values. In real-world applications, arrival time functions are usually continuous time functions and approximated by piece-wise linear functions. For the remainder, we assume the FIFO property to hold since this is the case for most practical networks and makes the problem polynomial time solvable. In the following paragraphs, we review previous results for this problem setting.

2.1 Lower Envelope Algorithms

By definition, the earliest arrival time function from s to d is the minimum over all arrival time functions for every path from s to d. This leads to a simple algorithm: compute the arrival time functions of all paths from s to d and compute the lower envelope. For more information on lower envelope algorithms see e.g. [16]. Although this gives the correct solution, such an algorithm is not efficient in that there could be an exponential number of paths from s to d leading to exponential time complexity.

2.2 Label Correcting Algorithms

A slightly modified version of standard label-correcting algorithms (e.g., Bellman-Ford [3]) solves the time-dependent shortest path problem. Here, instead of computing labels for a specific time, one can do this simultaneously for all values of t. In this case, instead of working with scalar arrival times at each node, we consider earliest arrival time functions over all values of time. Orda and Rom [15] proposed such an algorithm which on a FIFO network with piece-wise linear functions has time complexity $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between s and any node in the network. This has been the best known approach since 1990 when it was presented. Our algorithm is a significant improvement of this method as well as of the methods outlined in the following paragraph.

2.3 Label-Setting Algorithms

In a label-setting algorithm the goal is to compute, in small pieces, actual correct values of output functions rather than iteratively revising these functions. This approach is similar to Dijkstra's algorithm [11] for the static shortest path problem. In contrast to label-correcting algorithms, it is not possible to simply replace scalar label values by functions to solve the problem because a minimum element (i.e., one function which is minimum over the entire domain) may not exist. The main idea of the algorithm is to determine the latest time ϕ, for each node, so that the current earliest arrival time function for any time less than ϕ gives the actual earliest arrival time to the node. For FIFO networks with piece-wise linear arrival time functions, Dean [9] suggested a label-setting algorithm that performs a single chronological scan through time to establish output functions. The algorithm employs the same approach used for solving parametric shortest path problems [5]. In the worst-case, this algorithm has a running time of $O(F^*|E| \log |V|)$ where F^* is the total number of pieces over all output functions in the network.

Recently, Ding et al. [12] presented a simpler label-setting algorithm for the time-dependents shortest path problem for FIFO networks with piece-wise linear functions. The algorithm scans a sequence of time steps the size of which depends on the values of the arrival time functions. Careful analysis of this algorithm shows that this approach yields a solution with time complexity $O(\gamma(|E| + |V| \log |V|))$ which contains a factor γ that is possibly unbounded because it depends on the relative *values* of arrival time functions. An example instance showing that the number of scanned time steps can be unbounded and independent on $|E|, |V|$, and λ is depicted in Figure 1.

3. STRUCTURAL PROPERTIES

Our new algorithm makes extensive use of some structural properties of the problem discussed in this section.

3.1 Solution Function Structure

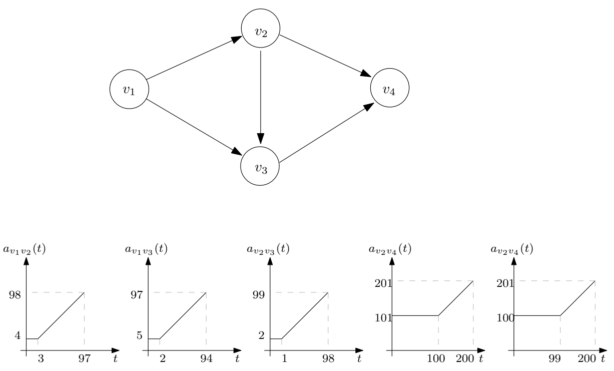


Figure 1: An example instance showing that the number of scanned time steps for the Ding et al. [12] algorithm, can be unbounded.

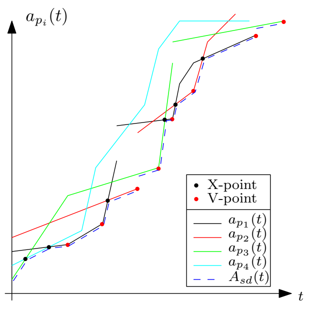


Figure 2: X and V-points

The earliest arrival time function from s to d, $A_{sd}(t)$, is a piece-wise linear function since all input arrival time functions are assumed to be piece-wise linear functions and the function operators used (*function inverse, linear combination, function compound, min, max*) do not change the linearity of the result. We are interested in the points on the curve $A_{sd}(t)$ that connect its different linear pieces, and will refer to them as *change points*. We differentiate between two types of change points. First, a change point may represent the intersection between two pieces of arrival time functions on different paths. Second, a change point may represent a change point on one of the input arrival time functions for a path from s to d. We refer to the first type as X-point and to the second type as V-point. Figure 2 depicts an arrangement of arrival time functions and identifies X and V-points. Every V-point corresponds to a change point on the arrival time function, $a_p(t)$, on some path p from s to d. Each change point on the $a_p(t)$ function is the result of a boundary point between two linear pieces of arrival time functions on an edge of p introduced because of a compound operation for computing $a_p(t)$. In the following lemma, we will show that every boundary point of an edge arrival time function can create at most one V-point on $A_{sd}(t)$.

LEMMA 1. *Suppose P_e is the set of all paths that include edge $e = (v, w) \in E$ and $a_e(t)$ is the arrival time function on e which has λ_e linear pieces. Then, all arrival time functions $a_{p_e}(t), p_e \in P_e$, create in total at most λ_e V-points on $A_{sd}(t)$.*

Proof: Let

$$a_e(t) = \begin{cases} \alpha_e^1 t + \beta_e^1 & 0 \le t \le T_e^1 \\ \alpha_e^2 t + \beta_e^2 & T_e^1 < t \le T_e^2 \\ \vdots & \vdots \\ \alpha_e^{\lambda_e} t + \beta_e^{\lambda_e} & T_e^{\lambda_e - 1} < t \le T_e^{\lambda_e} \\ \infty & T_e^{\lambda_e} < t \end{cases}$$

be the arrival time function on e. For every boundary point $T_e^i, i = 1 \ldots \lambda_e$, consider path p_e^i to be the concatenation of a path with the latest starting time (LST) from s which arrives at v at time T_e^i and a path with earliest arrival time (EAT) to d which starts from v at time T_e^i. Because of the definition of p_e^i, for any path $p_e \in P_e$ other than p_e^i, T_e^i will create a change point either at (LST, EAT) or to the left and above this point. Since (LST, EAT) is on $a_{p_e^i}(t)$ and FIFO property holds, any points that fall to the left and above (LST, EAT) are not on $A_{sd}(t)$, thereby other paths can not create new change points on $A_{sd}(t)$. This proves that all arrival time functions $a_{p_e}(t), p_e \in P_e$, create in total at most λ_e V-points on $A_{sd}(t)$. $\square$

Let $\lambda = \sum_{e \in E} \lambda_e$ be the total number of linear pieces on edge arrival time functions in the entire network. Since every V-point comes from a boundary point on some edge arrival time function, Lemma 1 implies that there can not be more than $O(\lambda)$ V-points on $A_{sd}(t)$.

3.2 Possibly Super-polynomial Output Size

In [9], the author conjectured that in a FIFO network with piece-wise linear arrival time functions on edges, the earliest arrival time function $A_{sd}(t)$ from a source node s to a destination node d may have more than a polynomial number of linear pieces. This means that there possibly exist network structures that result in super-polynomial complexity for earliest arrival time functions to some nodes of the network.

The *super-polynomial structure* could appear as a subgraph of the actual input network, resulting in earliest arrival time functions with super-polynomial number of pieces for destination nodes whose shortest path from s passes through the super-polynomial structures. However, the earliest arrival time function from s to d could easily be of linear size since the earliest arrival time path may not intersect the super-polynomial structure at all. In this case, F_{max} would be of super-polynomial size and F_d would be of linear size.

4. A NEW ALGORITHM FOR INSTANCES WITH POLYNOMIAL SIZE OUTPUT

Our new algorithm is based on the idea that instead of building all earliest arrival time functions for all nodes in the network, we find the earliest arrival time function to destination node d directly. The main problem here is to find all starting times for which the earliest arrival time function from s to d, $A_{sd}(t)$, changes from one linear piece to another as well as all linear functions between these change points. In Section 3, we introduced two types of change points in $A_{sd}(t)$: V-points and X-points. In Section 3, we also showed that at most $O(\lambda)$ V-points exist on $A_{sd}(t)$, where λ is the total number of pieces in all input arrival time functions. Moreover, using Dijkstra's static shortest path algorithm for every change point of all arrival time functions of the input, both forward to d and backward to

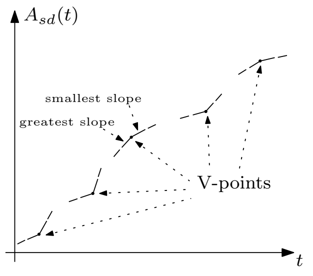


Figure 3: A sample $A_{sd}(t)$ function with all V-points and their adjacent linear functions.

s, we capture all V-points that can potentially be on $A_{sd}(t)$. (For reversibility of the time-dependent shortest path problem see [8].) To construct the entire function $A_{sd}(t)$, we also compute the linear pieces to the left and to the right of each V-point. These are pieces with the earliest arrival time and the smallest (greatest) slope on the right (left) vicinity of each V-point. Figure 3 shows a sample $A_{sd}(t)$ function once all V-points have been detected. Note that, given a time t_0, the smallest (greatest) slope piece can be computed while computing the earliest arrival time to d for starting time t_0. This is accomplished by keeping the product of slopes for each node in the shortest path tree as a secondary key when Dijkstra's algorithm finds two or more entries of the heap that have the same arrival time value. In this case, selecting the entry with smallest (greatest) slope leads to the smallest (greatest) slope linear piece. To show the correctness of this approach consider any two paths from s to d starting at time t_0 with equal arrival times but different slopes on their arrival time functions. The first time where they have equal arrival time values is either at d or at some earlier node d'. In the latter case, they will share the same "postfix" path from d' to d. In either case, selecting the smallest (greatest) slope product from the heap, among equal arrival time values, maintains the smallest (greatest) slope.

Thus far, we have determined all V-points and the slope of $A_{sd}(t)$ in their vicinity. We build the remaining part of $A_{sd}(t)$ by adding the missing piece between every pair of consecutive V-points on $A_{sd}(t)$. Due to the linearity of the input arrival functions, the X-points between two consecutive V-points are in concave position (seen from below). Consider two consecutive V-points, V_l and V_r, found in the previous step together with the linear pieces in their vicinity. Two cases arise for the linear pieces to the right of V_l and to the left of V_r. They either overlap, or they intersect in some point $I = (x_I, y_I)$. If they overlap, then the piece connecting the two V-points is the solution (Figure 4-a). In case of an intersection, two cases are possible. First, if calculating a static shortest path at time x_I returns the same arrival time y_I as for the intersection point then the pieces, v_l to I and I to v_r are the solution (Figure 4-b). Second, if calculating a static shortest path at time x_I returns a value less than y_I, then we found a new linear piece that is hiding the intersection point $I = (x_I, y_I)$ (Figure 4-c). The linear pieces to the right of V_l and to the left of V_r intersects the new piece, and we recurse. See Theorem 1 for further details.

Algorithm 1 below shows the entire TDSP algorithm. It determines both V-points and X-points in separate sections. After initializing $A_{sd}(t)$ at the beginning of the algorithm (Line 2), in Lines 3 through 11 we capture all V-points along

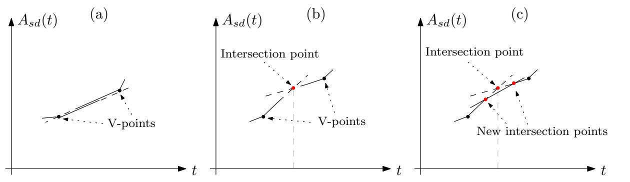


Figure 4: (a) Overlapping pieces (b) Intersection point on $A_{sd}(t)$ (c) Intersection point hidden by another linear piece

with their adjacent linear pieces to their left and right. In Lines 12 through 36 we detect all X-points on $A_{sd}(t)$. In the first phase, for every edge $e = (v, w)$ in the network and for every boundary point T_e^i corresponding to an edge e, the algorithm finds the latest starting time (LST) from s to arrive at v at time T_e^i by calculating a static shortest path algorithm (Dijkstra) backwards from v to s at time T_e^i (line 5). We also execute a forward static shortest path to obtain the earliest arrival time (EAT) at d starting from v at time T_e^i (Line 6). This provides the rightmost and lowest V-point that could be found on $A_{sd}(t)$ as a result of boundary point T_e^i. As shown earlier, for all other paths that include e, V-points for T_e^i will be hidden by some other linear pieces. In order to make sure that (LST, EAT) is on the final solution we calculate a static shortest path from s to d starting at time LST (Line 7). If the arrival time is the same as EAT, then (LST, EAT) is indeed a V-point on $A_{sd}(t)$. In this case, to find the linear pieces near V-points on $A_{sd}(t)$, we find the linear pieces with the smallest slope and the largest slope adjacent to the left and right of each V-point, respectively. Finally, we add each V-point found along with its adjacent linear pieces to a list for use in the next step (Lines 8 through 11).

In the second part of the algorithm, we first sort all V-points by ascending order of LST value (Line 12). Then, starting from the first two V-points, we read pair of consecutive points from the list and build the $A_{sd}(t)$ function between them as we scan these points (Lines 13, 15 and 35). Two cases are possible: either the linear function to the right of the first V-point, RF_1, and the linear function to the left of the second V-point, LF_2, overlap or they intersect. In case of overlap, the linear piece between the two points must be a piece of $A_{sd}(t)$ (Lines 16 and 17). If RF_1 and LF_2 intersect, we add the intersection point to a stack (Lines 19-21). Here, either there is a linear piece below the intersection point that prevents it from being on the final solution or the intersection point is on $A_{sd}(t)$. In the first case, we find the linear piece with the greatest slope and add two new intersection points to the stack (Lines 29-33). If the intersection point is on the final solution (Lines 24-27) we add the linear piece on RF_1 between the first V-point and the intersection point to $A_{sd}(t)$. Line 34 adds the linear piece to the left of the right V-point. Finally, in Line 36, we add an unbounded linear piece if the last linear function is unbounded.

Theorem 1. *Given a source node s and destination node d, the TDSP algorithm outlined above correctly determines $A_{sd}(t)$ for all $t \in [0, \infty)$.*

Proof: The algorithm first finds all V-points on $A_{sd}(t)$ along with linear pieces to the left and right of each V-point.

Algorithm 1: $TDSP(G, V, E, s, d)$

```
1  begin
2      $A_{sd}(t) \leftarrow NULL$
3      for every edge $e = (v, w) \in E$ do
4          for $i = 0$ to $\lambda_e$ do
5              $LST \leftarrow SPbackward(v, s, T_e^i)$
6              $EAT \leftarrow SPforward(v, d, T_e^i)$
7              $TMP \leftarrow SPforward(s, d, LST)$
8              if $EAT = TMP$ then
9                  $f_l \leftarrow GSEATfunction(s, d, LST)$
10                 $f_r \leftarrow SSEATfunction(s, d, LST)$
11                 $InsertToList(L, \{LST, EAT, f_l, f_r\})$
12     $Sort(L)$
13     $\{LST_1, EAT_1, LF_1, RF_1\} \leftarrow RemoveItem(L)$
14     while $NotEmpty(L)$ do
15         $\{LST_2, EAT_2, LF_2, RF_2\} \leftarrow RemoveItem(L)$
16         if $Overlap(RF_1, LF_2)$ then
17             $AddLinearPiece(A_{sd}(t), RF_1, LST_1, LST_2)$
18         else
19             $(PX_1, PY_1) \leftarrow (LST_1, EAT_1)$
20             $(PX_2, PY_2) \leftarrow IntersectionPoint(RF_1, LF_2)$
21             $Push(S, (PX_2, PY_2, RF_1, LF_2))$
22             while $NotEmpty(S)$ do
23                 $(PX_2, PY_2, f_l, f_r) \leftarrow Pop(S)$
24                 $TMP \leftarrow SPforward(s, d, PX_2)$
25                 if $TMP = PY_2$ then
26                     $AddLinearPiece(A_{sd}(t), f_l, PX_1, PX_2)$
27                     $PX_1 \leftarrow PX_2$
28                 else
29                     $f_m \leftarrow GSEATfunction(s, d, PX_2)$
30                     $(IX_1, IY_1) \leftarrow IntersectionPoint(f_l, f_m)$
31                     $(IX_2, IY_2) \leftarrow IntersectionPoint(f_m, f_r)$
32                     $Push(S, (IX_2, IY_2, f_m, f_r))$
33                     $Push(S, (IX_1, IY_1, f_l, f_m))$
34             $AddLinearPiece(A_{sd}(t), f_r, PX_2, LST_2)$
35         $\{LST_1, EAT_1, LF_1, RF_1\} \leftarrow$
           $\{LST_2, EAT_2, LF_2, RF_2\}$
36     if $EAT_1 \neq \infty$ then
       $AddLinearPiece(A_{sd}(t), RF_1, LST_1, \infty)$
37     return $(A_{sd}(t))$
38 end
```

By Lemma 1, no V-points other than those considered can be on $A_{sd}(t)$. Then, the algorithm picks every two consecutive V-points to compute all X-points between them. Let $v_l = (x_l, y_l)$ and $v_r = (x_l, x_r)$ be two consecutive V-points on $A_{sd}(t)$. Also, suppose that RF and LF are the linear pieces to the right of v_l and to the left of v_r, respectively. Either RF and LF overlap or they intersect. If overlap, the linear piece on RF (or LF) from x_l to x_r is part of the solution function since no other V-points are possible between v_l and v_r. On the other hand, if the two functions intersect in some point $I = (x_I, y_I)$ and the intersection is on $A_{sd}(t)$, then the linear piece on RF from x_l to x_I is on $A_{sd}(t)$ since no other V-points are possible between v_l and v_r. If I is not on $A_{sd}(t)$, then there must be another linear piece preventing it from being on the solution. The algorithm determines such a piece with maximum slope. The extension of the linear piece must intersect both RF and LF since otherwise there must be another V-point between v_l and v_r. Let I_l and I_r be the two intersection points. We now recursively perform what we did for I, first for I_l and then I_r. Starting from v_l, we add linear pieces to the solution function once I_l is found to be on $A_{sd}(t)$. Then, we move to the next intersection. As a last step, the algorithm adds to the solution function the last piece on LF between the last intersection and x_r. Since we verify every X-point for being on $A_{sd}(t)$ and no more V-points are possible between two consecutive V-points, the algorithm finds all X-points. Since V-points and X-points are the only change points on $A_{sd}(t)$, Algorithm TDSP correctly finds all linear pieces of $A_{sd}(t)$. □

THEOREM 2. *The time complexity of the TDSP algorithm outlined above is* $O((F_d + \lambda)(|E| + |V| \log |V|))$.

Proof: First, the algorithm executes a slightly modified version of Dijkstra's shortest path algorithm both forward and backward for each edge of the graph to find all possible V-points. It then executes another modified version of Dijkstra's algorithm to find greatest and smallest slope pieces close to each V-point. Supposing that for every edge (v, w) and a given starting time at v we can compute the arrival time at w in $O(1)$ time, for a given starting time at s, the earliest arrival time at d is computed in the same time as Dijkstra's algorithm, namely $O(|E| + |V| \log |V|)$ using Fredman and Tarjan's implementation [13]. Consequently, the time-complexity of the first part is $O(\lambda(|E|+|V| \log |V|))$ where λ is the total number of linear pieces in all edge arrival time functions. Then, in the second part, the algorithm executes a modified Dijkstra's algorithm as many time as we find intersection points. At each intersection point found, we determine the linear piece with greatest slope that hides the intersection point. This guarantees that, every time we run a modified Dijkstra's algorithm at the intersection point we obtain a new linear piece which is part of the solution $A_{sd}(t)$. As a result, we will execute the modified Dijkstra's algorithm at most as many times as there are X-points on $A_{sd}(t)$. With F_d defined as the number of linear pieces on $A_{sd}(t)$, the second part runs in time $O(F_d(|E|+|V| \log |V|))$. Hence, the total time complexity of the TDSP algorithm is $O(F_d + \lambda)(|E| + |V| \log |V|))$ □

5. AN APPROXIMATION ALGORITHM FOR INSTANCES WITH SUPER-POLYNOMIAL SIZE OUTPUT

We now present an approximation algorithm for instances where the output size F_d might be super-polynomial. Our method computes for all possible starting times $t \in [0, T]$ at s the earliest arrival times at d within error at most ϵ. The algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the earliest arrival time at d for the latest possible starting time at s. In Section 3, we showed that the number of V-points on $A_{sd}(t)$ is bounded by λ, the total number of linear pieces in all edge arrival time functions of the network (input size). Hence, for instances where the output size F_d is super-polynomial, it follows that the number of X-points on $A_{sd}(t)$ must be super-polynomial. Our approximation algorithm first computes all V-points on $A_{sd}(t)$ as outlined in the previous section. Then, for every two consecutive V-points $v_l = (x_l, y_l)$ and $v_r = (x_r, y_r)$ we compute an approximation of $A_{sd}(t)$. If $y_r - y_l \leq \epsilon$ we simply connect v_l and v_r through a linear piece. If $y_r - y_l > \epsilon$ we calculate the static shortest path backward from d to s at time $y_m = \frac{y_l + y_r}{2}$ and obtain a point $v_m = (x_m, y_m)$ on $A_{sd}(t)$. We recursively perform this splitting operation until the difference between arrival times is less than ϵ. As a final step, we connect all points obtained (V-points plus new points) through linear pieces. With $\Delta = A_{max} - A_{min}$ defined as the difference between the earliest arrival times $A_{min} = A_{sd}(t = 0)$ and $A_{max} = A_{sd}(t = T)$, the time complexity of this algorithm is $O((\frac{\Delta}{\epsilon} + \lambda)(|E| + |V| \log |V|))$. We can improve this time complexity to $O((\frac{\Delta}{\epsilon})(|E| + |V| \log |V|))$ by avoiding the calculation of the V-points altogether. We calculate the static shortest paths backwards from d to s at all times $A_{min} + i\epsilon$, $i = 1, \ldots, \lfloor \frac{\Delta}{\epsilon} \rfloor$, and then connect the points obtained by linear pieces. Here, the main complication is the possibility of discontinuities in $A_{sd}(t)$. Handling this case within the same time complexity is possible.

6. CONCLUSION

In this paper, we presented a new algorithm which solves the time-dependent shortest paths problem for a time-dependent network with FIFO property. The running time of our algorithm is $O((F_d + \lambda)(|E| + |V| \log |V|))$, where F_d is the output size and λ is the input size. Our method improves significantly on the best previously known bound by Orda and Rom [15]. We also study instances where the output size F_d is super-polynomial, for which all previously known methods (including our first method presented here) require super-polynomial time. We present the first approximation method for such instances of the time-dependent shortest path problem. Our method computes for all possible starting times the earliest arrival times within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s. The methods presented in this paper are independent of the underlying static shortest path algorithm, so that more efficient shortest path algorithms than the generic Dijkstra algorithm can be used when applicable. E.g., in planar networks, applying linear time shortest path algorithms [14] will further improve our results. In many practical networks heuristics such as A^* can be applied to improve the practical performance of our methods. We are currently implementing our algorithm.

7. ACKNOWLEDGEMENT

We thank Rolf Klein and Florian Berger for insightful discussions on this topic.

8. REFERENCES

[1] Ravindra K. Ahuja, James B.Orlin, Stefano Pallottino, and Maria G.Scutella. Dynamic shortest paths minimizing travel times and costs. *Networks*, 41:205, 2001.

[2] Ravindra K. Ahuja, James B. Orlin, Stefano Pallottino, and Maria Grazia Scutellà. Minimum time and minimum cost-path problems in street networks with periodic traffic lights. *Transportation Science*, 36(3):326–336, 2002.

[3] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[4] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electr. Notes Theor. Comput. Sci.*, 92:3–15, 2004.

[5] Patricia June Carstensen. *The complexity of some problems in parametric linear and combinatorial programming.* PhD thesis, University of Michigan, 1983.

[6] Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Fates: Finding a time dependent shortest path. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 165–180, London, UK, 2003. Springer-Verlag.

[7] K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

[8] Carlos F. Daganzo. Reversibility of the time-dependent shortest path problem. *Transportation Research Part B: Methodological*, 36(7):665–668, August 2002.

[9] Brian C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical report, MIT Department of Computer Science, 2004.

[10] Frank. Dehne, Masoud T. Omran, and Jorg-R. Sack. Minimum travel time on networks with time-dependent edge availabilities. Technical report, Carleton University, Ottawa, 2009.

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

[12] Bolin Ding, Jeffrey Xu Yu, and Lu Qin. Finding time-dependent shortest paths over large graphs. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 205–216, New York, NY, USA, 2008. ACM.

[13] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[14] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[15] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.

[16] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, New York, NY, USA, 1996.

Fuel-cache site-selection for polar research: A Summary of Results

Mark Dietz
Department of Computer Science
University of Minnesota
mdietz@cs.umn.edu

Shashi Shekhar
Department of Computer Science
University of Minnesota
shekhar@cs.umn.edu

ABSTRACT

Scientists conducting polar research in Antarctica must contend with harsh environmental conditions that constrain their movements and raise their costs. One on going challenge is choosing cache sites for aircraft refueling. Given a data-gathering mission (e.g. set of flight destinations), aircraft fuel-consumption model, and infrastructure (e.g. base and cache-sites), the Fuel-Cache Site-Selection (FCSS) problem identifies the optimal use of cache sites to fulfill the mission. The FCSS problem is important for planning expeditions in infrastructure-poor areas for scientific or military purposes. However, the FCSS problem is computationally challenging due to interaction across different flight-routes. Related approaches from literature concerning routing are inadequate due to assumptions about the cost of providing infrastructure. This paper proposes heuristics and a filter-and-refine based exact algorithm, evaluation using analytical and experimental methods, and a case study with end-users, e.g. polar scientists.

Categories and Subject Descriptors

I.2.1 [**Applications and Expert Systems**]: Route Finding

General Terms

Algorithms

Keywords

Routing, Assignment, Optimization.

1. INTRODUCTION

Scientists conducting polar research in Antarctica must contend with harsh environmental conditions that constrain their movements and raise their costs. One on going challenge is choosing cache sites for aircraft refueling. Given a data-gathering mission (e.g. set of flight destinations), aircraft fuel-consumption model, and infrastructure (e.g. bases and cache-sites), the Fuel-Cache Site-Selection (FCSS) problem identifies the optimal use of infrastructure to fulfill the mission. Each destination must be visited by a separate flight due to the amount of time that must be spent at each destination and constraints on the total time spent in the field. Thus a solution to the FCSS problem specifies a path from the home point to the flight destination and back to the home point. Some points can be reached directly. Others require a refueling stop, which can only be made at predefined cache-sites. All fuel taken from the cache-sites must first be placed there by a separate refueling flight. Therefore the FCSS problem consists of minimizing the fuel consumption of two types of flights: research flights to the specified destinations of interest and refueling flights to place fuel at the cache sites in support of the research flights.

The FCSS problem is important for planning expeditions in infrastructure-poor areas for scientific or military purposes. The logistics costs for supporting these missions are very high due to the cost incurred establishing the necessary infrastructure to support the mission. There is much value in reducing these costs.

However, the FCSS problem is computationally challenging due to interaction across different flight-routes. The choice of refueling site for a particular research point may vary based on whether there is left over fuel from a previous refueling flight at any of the cache sites.

Related Work: Prior research on problems similar to FCSS falls into two categories: vehicle routing with fuel constraints and multi-depot vehicle routing.

Research into vehicle routing with fuel constraints ([1] and [2]) attempts to find the least cost path between two points in a graph for a vehicle with a limited fuel tank capacity. The vehicle must stop to refuel at designated refueling stations. The graph edges are weighted by the fuel required and each refueling station is a node on the graph with a fixed fuel price.

Multi-depot vehicle routing ([3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] and [15]) is the problem of transporting materials to customers using a number of depots. Customers are assigned a depot and vehicles are routed to satisfy the customers' demands while attempting to minimize the costs of fuel, vehicles and drivers.

In the FCSS problem, the fuel cache sites take the role of the refueling stations or depots and the research points take the role of the destination points or customers. However, the approaches in the literature assume the cost of choosing a cache site for a research point is independent of the other research points using that site. In FCSS, the cost of fuel at the fuel cache site is the cost to transport the fuel there. This cost varies based on which research points are assigned to that fuel cache.

Our Contributions: This paper defines the fuel cache site selection problem (FCSS). We believe the problem is NP-hard due to similarities with other routing problems and propose two heuristics. The paper also provides an algorithm to find the optimal solution using properties of the problem to filter the possible cache site choices for each research point and performing an exhaustive search of the remaining choices. Finally, it presents a case study showing how these solution methods combined with a visualization tool (Google Earth) can be used by human planners to assess the fuel cost impacts of adding a new cache site or removing an existing site.

Outline: The rest of the paper is organized as follows. Section 2 formally defines FCSS and provides an example. Section 3 presents our methods for solving the selection problem. Section 4 shows an analytical and experimental comparison between the solution methods on real and synthetic data. Section 5 discusses our conclusions.

2. PROBLEM DEFINITION

2.1 Definitions

Definition 2.1.1: Aircraft model – We assume the aircraft used for all flights can be specified by the following model:

- Maximum Fuel Capacity
- Fuel Burned per Mile
- Minimum Fuel Reserve
- Base Operational Weight: weight of the aircraft and crew without cargo, passengers, or fuel
- Maximum Operational Weight: weight of the aircraft plus crew, cargo, passengers, and fuel

Definition 2.1.2: $fuelCost(h, p)$: fuel cost for a round trip from home point h to point p – either a cache site or a research point.

Definition 2.1.3: $fuelCost(h, r_i, c_j)$: fuel cost for a round trip from home point h to research point r_i using fuel cache c_j to refuel.

Definition 2.1.4: $refuelCost(h, r_i, c_j)$: fuel cost to place enough fuel at cache c_j to support a round trip research flight to r_i that uses c_j to refuel. For example, if the research flight to r_i needs to take 1000 lbs of fuel from the site c_j and the maximum payload for a refueling flight to c_j is 800 lbs, then $refuelCost(h, r_i, c_j)$ is twice $fuelCost(h, c_j)$.

2.2 Formal Problem Definition

Given:

- A single home point h represented by latitude / longitude coordinates.
- Set $C = \{c_1, c_2, \dots, c_m\}$ of fuel cache sites represented as latitude / longitude coordinates.
- Set $R = \{r_1, r_2, \dots, r_n\}$ research points of interest represented as latitude / longitude coordinates.
- Aircraft model.

Find:

- For every $r_i \in R$ find a feasible path from h to r_i and back to h. The feasible path contains no other research points and at most one stop to refuel at a site $c_{i1} \in C$ en route from h to r_i and at most one stop to refuel at $c_{i2} \in C$ en route from r_i to h.
- The amount of fuel to place at each cache site.
- The number of refueling flights needed to place fuel at each cache site.

Objective:

- Minimize the total fuel consumed by the research flights (flights to the points in *R*) and the refueling flights (flights to the cache sites in *C* to deposit the fuel used to refuel during the research flights).

Constraints:

- All refueling is performed at h or a point in *C*.
- h is assumed to be an infinite fuel source.
- All fuel taken from a site $c_i \in C$ must be flown from h to c_i using a refueling flight.
- A flight is either a refueling flight for placing fuel or research flight for visiting a research point, not both.
- Refueling flights carry fuel in drums of a specified size and weight. For example, each drum weights 400 lbs and contains 350 lbs of fuel.
- Sites in *C* have the capacity to store an infinite number of fuel drums.

2.3 Example

Figure 1 shows an example problem instance with $R = \{r_1, r_2, r_3, r_4\}$ and $C = \{c_1, c_2, c_3\}$. In this example, r_1 is close enough to h to be reached directly without stopping to refuel. r_2 can only be reached by refueling at c_1 on the outbound flight and refueling at c_1 again on the home bound flight. Therefore these assignments must be included in any optimal solution. The only choices to be made for this problem instance are selecting cache sites for r_3 and r_4. Point r_3 can be visited with just one stop to refuel at either c_2 (closest) or c_3 (a little farther), but c_1 is too far away. Point r_4 can be visited with just one refuel stop at c_3 (closest) or c_2 (a little farther). It can also be reached by refueling at c_1, but at a much larger cost. Table 1 shows the costs for r_3 and r_4 for assignments to c_2 and c_3.

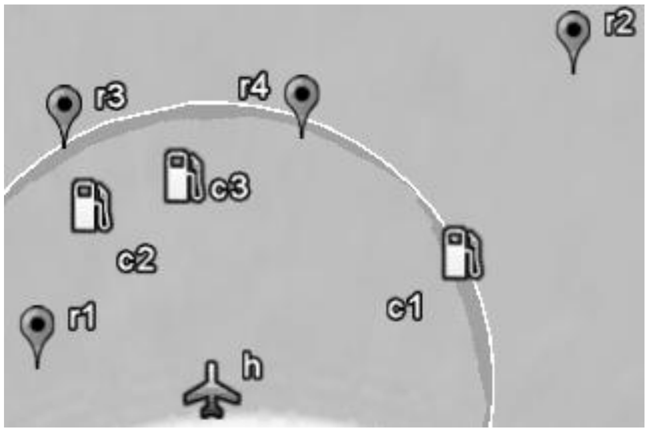


Figure 1: Example of cache selection problem with four research points and three fuel caches. Circle indicates maximum round trip flight range without refueling.

Table 1: Feasible solutions for r_3 and r_4 in example problem

Solution #	Research Point	Cache site to refuel	Fuel consumed by research flights	Total Plan for r_3, r_4 Cost With Fuel Placement
1	r_3	c_2	2807	9090
1	r_4	c_3	2774	
2	r_3	c_2	2807	9590
2	r_4	c_2	3245	
3	r_3	c_3	2850	7364
3	r_4	c_3	2774	
4	r_3	c_3	2850	9534
4	r_4	c_2	3245	

Table 1 shows that Solution #1 optimizes the fuel consumed by the flights to the research points and would be the best choice if there were no cost for fuel placement. However, Solution #3 is the optimal solution when fuel placement costs are considered. In Solution #3, fuel placement costs are lower because fuel needs to be placed at only one location to support r_3 and r_4 thus allowing consolidation of refueling flights. Note Solution #2 also chooses the same site for r_3 and r_4, but these flights require more fuel to be placed at c_2 than can be carried in one refueling flight so two refueling flights are necessary.

2.4 *Contrast FCSS Routing with other Routing*

In many routing problems, such as the Traveling Salesman Problem, a route must be found that can visit many points of interest. By contrast in FCSS we must find a separate route from the home point to each point of interest and back to the home point. The route cannot include any other research points. This constraint arises from real world conditions in polar research. Scientists visiting a research point must spend several hours taking measurements and installing sensors. There is no time to visit a second research point in a day and Antarctica's inhospitable climate makes an overnight stay in the field impossible. Other problems that would include this constraint are:

- an installation task where the vehicle can carry the materials for only one installation at a time
- a research task where the tools needed are very specific to the particular task and so not all tools can be carried at once
- a transportation task where vehicle capacity is limited to the items for one customer

3. CACHE SELECTION METHODS

Due to the belief that this problem is NP-hard, there is not likely a scalable algorithm to find an exact solution. We present two heuristics and one exact algorithm for solving the FCSS problem as presented in section 2.2. The first step of each is to identify any research points that are close enough to the home point h to not need a refueling stop at a cache site. These points are assigned a direct flight and then ignored. Even though we have shown that finding the globally optimal solution requires the cache selections to be interdependent, the heuristics each make different assumptions that allow the selections to be treated independently. The exact algorithm identifies bounds on the solution and filters out choices that do not fit within these bounds. It then performs an exhaustive search of the remaining choices. Once assignments are made, all three methods tally the total fuel required at each cache site and assign the minimum number of refueling flights needed to place the fuel at each cache site. This allows some refueling flights to be consolidated. However, only the exhaustive search considers the gains from consolidating refueling flights when making the cache site selections.

3.1 *Heuristic: Research Flights Only*

Figure 3 shows pseudo-code for the Research Flights Only heuristic. This heuristic is the most similar to the current manual process for solving the FCSS problem. For each research point $r_i \in R$, the heuristic chooses the cache $c_j \in C$ that minimizes $fuelCost(r_i, c_j)$. Ignoring the cost of fuel placement will provide an under estimate of the total fuel required to visit a research point. When executed with the input from example 2.3 this heuristic gives Solution #1 from Table 1. An execution trace follows:

- Only r_1 can be reached directly from h. Assign a direct flight for r_1.
- r_2 can only be reached by refueling at c_1. Assign c_1 to r_2.
- r_3 can be reached by refueling at c_2 with $fuelCost(h, r_3, c_2) = 2807$ or c_3 with $fuelCost(h, r_3, c_3) = 2850$. Assign c_2 to r_3.
- r_4 can be reached by refueling at c_1 with $fuelCost(h, r_4, c_1) = 3462$ or c_2 with $fuelCost(h, r_4, c_2) = 3132$ or c_3 with $fuelCost(h, r_4, c_3) = 2774$. Assign c_3 to r_4.
- The amount of fuel needed at each site is: $c_1 = 2213$ lbs., $c_2 = 357$ lbs., and $c_3 = 394$ lbs.
- Refueling flights to c_1 can deposit a maximum of 700 lbs. of fuel per flight so c_1 requires 4 refueling flights.
- Refueling flights to c_2 can deposit a maximum of 1050 lbs. of fuel per flight so c_2 requires 1 refueling flight.
- Refueling flights to c_1 can deposit a maximum of 1400 lbs. of fuel per flight so c_1 requires 1 refueling flight.
- The total cost is $fuelCost(h, r_1) + fuelCost(h, r_2, c_1) + 4 \times fuelCost(h, c_1) + fuelCost(h, r_3, c_2) + fuelCost(h, c_2) + fuelCost(h, r_4, c_3) + fuelCost(h, c_3) = 24681$

```
ResearchFlightsOnly {
   foreach rᵢ in {r₁,…,rₙ}
      minFuelCost = ∞;
      foreach cⱼ in {c₁, …, cₘ}
         if(fuelCost(h,rᵢ,cⱼ) < minFuelCost)
            minFuelCost = fuelCost(h,rᵢ,cⱼ);
            cacheSiteAssignment[rᵢ] = cⱼ;
   return cacheSiteAssignment[];
}
```

Figure 3: Pseudo-code for Research Flights Only

3.2 *Heuristic: Independent Refueling Flights*

Figure 4 shows pseudo-code for the Independent Refueling Flights heuristic. For each research point $r_i \in R$, this heuristic chooses the cache $c_j \in C$ that minimizes $fuelCost(h, r_i, c_j) + refuelCost(h, r_i, c_j)$. The cache selection does not take into account any opportunity for combining flights to place fuel at cache sites. This is not optimal because if the fuel needed for research flights to two research points that use the same cache can be supported with only one refueling flight, the cost will be counted against both research flights when evaluating alternatives. When executed with the input from example 2.3 this heuristic gives Solution #1 from Table 1. Note that while in this case Independent Refueling Flights chooses the optimal solution, this is not the case in general. An execution trace follows:

```
IndependentRefuelingFlights {
   foreach rᵢ in {r₁,…,rₙ} {
      minFuelCostIncludingRefueling = ∞;
      foreach cⱼ in {c₁, …, cₘ}{
         if(fuelCost(h,rᵢ,cⱼ)
               + refuelingCost(h,rᵢ,cⱼ)
               < minFuelCostIncludingRefueling)
            minFuelCostIncludingRefueling =
                  fuelCost(h,rᵢ,cⱼ)
                  + refuelingCost(h,rᵢ,cⱼ);
            cacheSiteAssignment[rᵢ] = cⱼ;
   return cacheSiteAssignment[];
}
```

Figure 4: Pseudo-code for Independent Refueling Flights

- Only r_1 can be reached directly from h. Assign a direct flight for r_1.
- r_2 can only be reached by refueling at c_1. Assign c_1 to r_2.
- r_3 can be reached by refueling at c_2 with $fuelCost(h, r_3, c_2) + refuelCost(h, r_3, c_2) = 4575$ or c_3 with $fuelCost(h, r_3, c_3) + refuelCost(h, r_3, c_3) = 4520$. Assign c_3 to r_3.
- r_4 can be reached by refueling at c_1 with $fuelCost(r_4, c_1) + refuelCost(h, r_4, c_1) = 8090$ or c_2 with $fuelCost(r_4, c_2) +$

$refuelCost(h, r_4, c_2) = 5014$ or c_3 with $fuelCost(r_4, c_3) + refuelCost(h, r_4, c_3) = 4514$. Assign c_3 to r_4.

- The amount of fuel needed at each site is: $c_1 = 2213$ lbs., $c_2 = 0$ lbs., and $c_3 = 794$ lbs.
- Refueling flights to c_1 can deposit a maximum of 700 lbs. of fuel per flight so c_1 requires 4 refueling flights.
- c_2 supplies no fuel so needs no refueling flights.
- Refueling flights to c_1 can deposit a maximum of 1400 lbs. of fuel per flight so c_1 requires 1 refueling flight.
- The total cost is $fuelCost(h, r_1) + fuelCost(h, r_2, c_1) + 4 \times fuelCost(h, c_1) + fuelCost(h, r_3, c_3) + fuelCost(h, r_4, c_3) + fuelCost(h, c_3) = 22955$

3.3 *Algorithm: Exhaustive Search With Filter*

Figure 5 shows pseudo-code for the Exhaustive Search With Filter algorithm. This algorithm finds the upper bound for the total fuel consumption contribution for each research point and a lower bound for the fuel consumption contribution for the assignment of each cache site to each research point. These are shown in Lemma 3.3.1 and 3.3.2

Lemma 3.3.1: The upper bound on the fuel cost contribution for r_i is the amount of fuel required to visit r_i without considering other research points:
$MINIMUM\{\forall c_j \in caches \mid fuelCost(h, r_i, c_j) + refuelCost(h, r_i, c_j)\}$
Proof: Suppose there exists an optimal solution S where the contribution from r_i is greater than the proposed upper bound. Construct *S'* by replacing the flight to r_i and supporting refueling flights in S with a flight to r_i using c_j to refuel and supporting refueling flights. Then the total cost of *S'* is less than the total cost of *S*, a contradiction. Thus the contribution from r_i in the optimal solution *S* is less than the proposed upper bound.

Lemma 3.3.2: Lower bound on fuel cost contribution for r_i using cache site c_j is $fuelCost(h, r_i, c_j) + refuelCost(h, r_i, c_j) - fuelCost(h, c_j)$. This is the lower bound because there is at most one refueling trip's payload of fuel left over at c_j from refueling trips to support other research points. Thus refueling at the same site as other research flights saves at most one refueling flight.

Using the upper bound of $fuelCost(h, r_i, c_j) + refuelCost(h, r_i, c_j)$ we form the list of possible assignments for each $r_i \in R$. A cache $c_j{}'$ is removed from the list of possible assignments if its lower bound $fuelCost(h, r_i, c_j{}') + refuelCost(h, r_i, c_j{}') - fuelCost(h, c_j{}')$ is higher than the upper bound $fuelCost(h, r_i, c_j) + refuelCost(h, r_i, c_j)$. Then all combinations of the choices are examined to find the optimal solution. When executed with the input from example 2.3, this algorithm gives Solution #1 from Table 1. An execution trace follows:

- Only r_1 can be reached directly from h. Assign a direct flight for r_1.
- r_2 can only be reached by refueling at c_1. Assign c_1 to r_2.
- The maximum contribution from r_3 (Lemma 3.3.1) is $fuelCost(h, r_3, c_3) + refuelCost(h, r_3, c_3) = 4520$.
- The minimum contribution from r_3 with c_2 (Lemma 3.3.2) is $fuelCost(h, r_3, c_2) + refuelCost(h, r_3, c_2) - fuelCost(h, c_2) = 2807 < 4520$. c_2 cannot be filtered, so optimal selection for r_3 is in $\{c_2, c_3\}$.
- The maximum contribution from r_4 is $fuelCost(h, r_4, c_3) + refuelCost(h, r_4, c_3) = 4514$.
- The minimum contribution from r_4 with c_1 is $fuelCost(h, r_4, c_1) + refuelCost(h, r_4, c_1) - fuelCost(h, c_1) = 5776 > 4514$. c_1 can be filtered.
- The minimum contribution from r_4 with c_2 is $fuelCost(h, r_4, c_2) + refuelCost(h, r_4, c_2) - fuelCost(h, c_2) = 3245 < 4514$. c_2 cannot be filtered so optimal selection for r_4 is in $\{c_2, c_3\}$.
- All solutions are equal except for the choice of cache for r_3 and r_4. Table 1 enumerates costs contributed by r_3 and r_4 for all combinations, so exhaustive search chooses c_3 for both r_3 and r_4.
- The total cost is $fuelCost(h, r_1) + fuelCost(h, r_2, c_1) + 4 \times fuelCost(h, c_1) + fuelCost(h, r_3, c_3) + fuelCost(h, r_4, c_3) + fuelCost(h, c_3) = 22955$

```
ExhaustiveSearchWithFilter {
   foreach rᵢ in {r₁,…,rₙ}
      computeUpperBound[rᵢ]
      foreach cⱼ in {c₁,…,cₘ}
         if(fuelCost(h,rᵢ,cⱼ) > upperBound(rᵢ))
            removePossibleAssignment(rᵢ,cⱼ)
   examineCombinationsOfPossibleAssignments
   return optimalAssignment
}
```

Figure 5: Pseudo-code for Exhaustive Search With Filter

4. Analysis

We evaluated the methods analytically, experimentally against synthetic datasets of different sizes, and experimentally against a real world dataset provided by the Antarctica Geospatial Information Center (AGIC).

4.1 *Analytical*

4.1.1 *Correctness*

The solutions provided by the Research Flights Only and Independent Refueling Flights heuristics are guaranteed to be feasible because each starts with no selections for any research point and only makes selections where the cost is not infinite (line 5 in Figures 3 and 4). This means all selections made by the heuristic are feasible.

The solutions given by the Exhaustive Search With Filter algorithm are guaranteed to be feasible because it starts with no selections for any research point and all infeasible selections are filtered by the bounding process (line 5 in Figure 5).

4.1.2 *Computational Complexity*

It is easy to see that the heuristics – Research Flights Only and Independent Refueling Flights – each execute in $O(|R| \times |C|)$ time for all cases. Each possible cache site is examined once for each possible research point.

The worst case execution time for Exhaustive Search With Filter is $O(|C|^{|R|})$. Line 8 in Figure 5 is the non-polynomial part of the algorithm because it must examine all combinations of assignments. In the worst case, the filter is unable to rule out any cache site for any research point. However, if the filter is able to rule out many of the cache site choices, then the execution time can be cut down substantially. In the best case for execution time, the filter eliminates all but one choice for each research point, allowing the Exhaustive Search With Filter to execute in $O(|R| \times |C|)$ time. The average case however remains $O(|C|^{|R|})$.

4.2 Experimental Analysis

Synthetic problem instances were generated in order to compare the heuristics and the exact algorithm in terms of overall solution quality for problems with varying numbers of research points and fuel cache sites. Figure 6 shows the overall evaluation process. First random research points were selected in an area around a home point. Points within range of a direct flight from the home point were excluded. Cache sites were selected such that at least half of the research points could be reached from each cache site. Finally any research points that were not reachable from any cache site were removed and replaced with random points that were reachable from at least one cache site and not reachable from the home point directly. These restrictions on the synthetic data allowed the size of the data set to correspond to the number of alternatives the algorithm must evaluate.

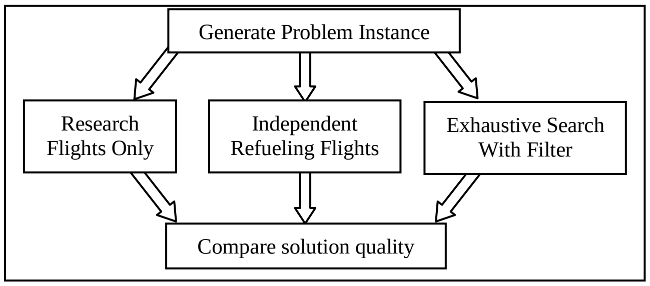

Generate Problem Instance
Research Flights Only
Independent Refueling Flights
Exhaustive Search With Filter
Compare solution quality

Figure 6: Experiment Design

The first experiment ran all three methods against problems with 2 cache sites and 10 – 20 research points. We ran 30 trials for each problem size. Figure 7 shows the percentage of the total cost contributed by refueling for the solutions. This is the cost divided by the total fuel required to visit all research points using the nearest fuel cache. Table 2 shows the average execution times over the trials. The run time of the exact algorithm was very volatile. The execution time for trials with 20 research points varied from 0.02 sec to 20 hours. Figure 7 and Table 2 show the Exhaustive Search With Filter provides the highest quality solutions, but it also has considerably longer execution times.

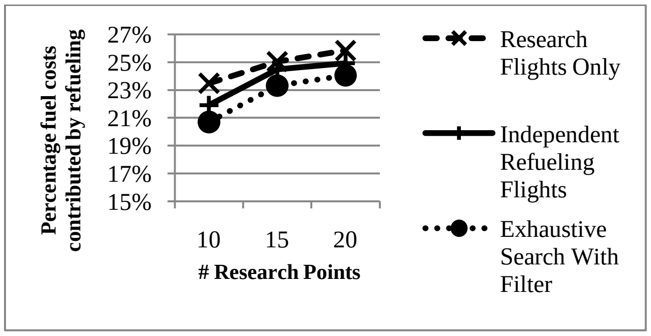


Figure 7: Percentage fuel costs contributed by refueling for problems with 2 fuel cache sites and 10 - 20 research points.

Table 2: Execution time for datasets in Figure 7

Method	Average Execution Time		
	10 points	15 points	20 points
Research Flights Only	0.001 s	0.003 s	0.005 s
Independent Refueling Flights	0.003 s	0.050 s	0.063 s
Exhaustive With Filter	0.1728 s	22.7 s	1 hour

We further evaluated the two heuristics against larger synthetic datasets in order to characterize the quality of their solutions. The first heuristic evaluation used 4 cache sites with the number of research points varying from 10 to 100. We ran 1000 trials for each problem size. Figure 8 shows the percentage of the total cost contributed by refueling. The exhaustive search algorithm was not able to consistently finish in a reasonable time on problems of this size.

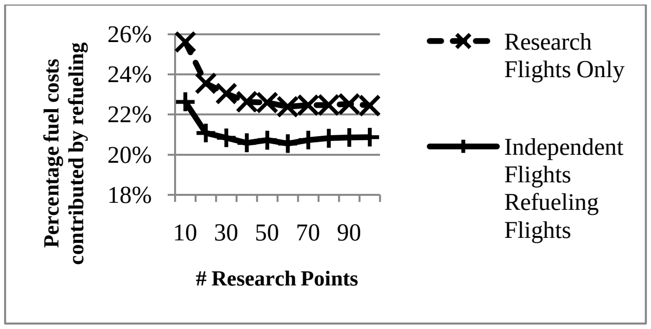


Figure 8: Percentage fuel costs contributed by refueling for problems with 4 fuel cache sites and 10 - 100 research points.

The second heuristic evaluation used 50 research points with the number of fuel cache sites varying from 2 to 10. We ran 1000 trials for each problem size. Figure 9 shows the percentage of the total cost contributed by refueling. Figure 9 shows that the Independent Refueling Flights is better able to take advantage of more fuel cache site choices than Research Flights Only. Figures 8 and 9 show that generally the Independent Refueling Flights heuristic provides a better solution. However, there were cases when the Research Flights Only heuristic provided a better solution. Both heuristics execute efficiently so it is worthwhile to try both to find solutions in an actual planning scenario.

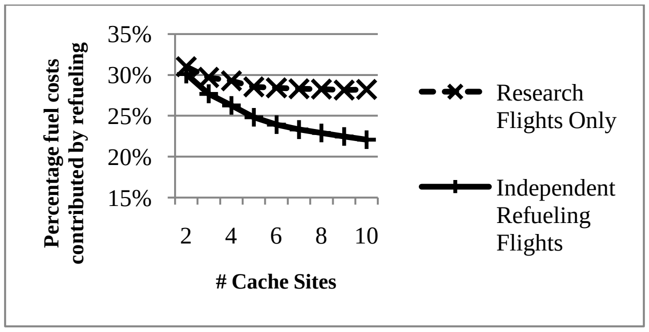


Figure 9: Percentage fuel costs contributed by refueling for problems with 60 research points and 2 - 10 fuel cache sites.

4.3 Case Study: AGIC

The Antarctica Geospatial Information Center provided a dataset for an instance of the FCSS problem. The dataset contained 26 research points, 3 existing fuel cache sites, and 2 proposed fuel cache sites. Sixteen research points were reachable by a direct flight and two were not reachable even when using the existing or proposed cache sites for refueling.

Currently AGIC evaluates proposed cache sites using two Excel spreadsheets. One spreadsheet is used to calculate the payload capacity and fuel consumption for a flight given its start, destination, and refueling stops. This method provides no clues to help the user determine which fuel cache site is a good choice for accessing a research point. The second spread sheet is used to tally the results and keep track of the amount of fuel required at each of the cache sites.

This process suffers from several deficiencies. First, even with a given set of fuel cache sites, there is no help for choosing which cache to assign to a research point. Second, adding a new proposed cache site to the set of possible choices takes hours so only a few possible new cache sites can be evaluated. Finally, opportunities for consolidating refueling flights are missed due to the difficulty of keeping track of the solution.

An automated solution to the FCSS problem, combined with a visual representation of the problem, solves the deficiencies in the current system. For the dataset provided by AGIC, the Exhaustive Search With Filter was able to find a solution that reduced the total fuel consumption by 10% compared to the solution arrived at manually using the previous system. In addition, many more alternative cache sites can be evaluated due to the efficiency gained by solving the FCSS problem quickly. Finally, by displaying the home point, research points, and fuel cache sites in Google Earth, and presenting visual feedback on the range of research and refueling flights, the user was able to quickly identify good locations for candidate fuel caches sites and then quickly evaluate the impact of a new site on the total fuel consumption. Figure 10 shows a screenshot of the visual representation of a flight plan. The circles show the range of a round trip flight from the home point and from the fuel caches. These let a user know where it is feasible to place a fuel cache site. The lines connecting the points represent the flight path chosen by the FCSS algorithm. This is either a direct flight from the home point or a flight with a stop to refuel at a fuel cache.

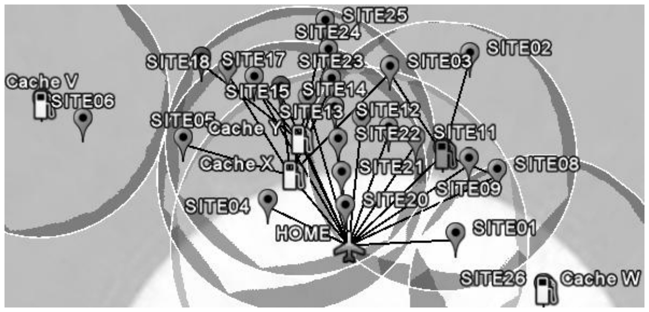


Figure 10: The problem set provided by AGIC. The circles around home and cache sites indicate the range of a flight from those sites with the given payload.

5. Conclusions and Future Work

We have characterized the fuel cache site selection (FCSS) problem and demonstrated the importance of the problem with respect to polar research and other operations in infrastructure poor regions. We proposed two heuristics and an exact algorithm. We showed through synthetic and case study data that the heuristics provide useful solutions when compared to the existing methods of cache selection, and that both the heuristics and the exact algorithm can find solutions to datasets that arise in practice.

Future work includes proving the FCSS problem is NP-hard, characterizing the problem instances where each heuristic is more effective, and finding better heuristics and alternate formulations such as Mixed Integer Programming to facilitate more efficient exact solutions. AGIC also wants to pursue algorithms to recommend new cache sites given an instance of the FCSS.

6. ACKNOWLEDGMENTS

Special thanks to Paul Morin and Michelle LaRue from the Antarctica Geospatial Information Center and to the research scientists in Antarctica for their help identifying and defining the FCSS problem. Thanks also to Kim Koffolt for improving the readability of the paper. This work was supported by the National Science Foundation and US Department of Defense.

7. REFERENCES

[1] Lin, S., Gertsch, N., Russell, J., 2007, A linear-time algorithm for finding optimal refueling policies. *Operations Research Letters*, 35, (3), 290-296

[2] Lin, S., 2008. Finding optimal refueling policies: a dynamic programming approach, *Journal of Computing Sciences in Colleges*, v.23 n.6, 272-279

[3] Wu, T.H., Low, C. and Bai, J.W. 2002. Heuristic solutions to multi-depot location-routing problems. *Computers & Operations Research*, v. 29, 1393-1415

[4] Avella, P., Boccia, M. and Sforza, A., Solving a fuel delivery problem by heuristic and exact approaches. European Journal of Operational Research. v152. 170-179.

[5] Cornillier F, Boctor FF, Laporte G and Renaud J. 2007. An exact algorithm for the petrol station replenishment problem. *Journal of the Operational Research Society*, doi:10.1057/palgrave.jors.2602374.

[6] Giosa, I., Tansini, I. and Viera, I. 2002. New assignment algorithms for the multi-depot vehicle routing problem. *Journal of the Operational Research Society*. v53. 977-984.

[7] Nagy, G. and Salhi, S. 2005. Heuristic algorithms for the single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*. v162. 126-141.

[8] Renaud, J., Laporte, G. and Boctor, F. 1996. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*. V23. 229-235, doi:10.1016/0305-0548(95)O0026-P

[9] Laporte, G., Nobert, Y. and Taillefer, S. 1988. Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems. *Transportation Science*. v22. 161

[10] Tuzun, D. and Burke, L. 1999. *European Journal of Operational Research*. v116. 87-99. doi:10.1016/S0377-2217(98)00107-6

[11] Parthanadee, P. and Logendran, R. 2006. Periodic product distribution from multi-depots under limited supplies. *IIE Transactions*. v38. 1009 – 1026

[12] Liu, S. and Lee, S. 2003. A two-phase heuristic method for the multi-depot location routing problem taking inventory control decisions into consideration. *The International Journal of Advanced Manufacturing Technology*. v22. 941-950. doi: 10.1007/s00170-003-1639-5

[13] Pisinger, D. and Ropke S. 2007. A general heuristic for vehicle routing problems, *Computers and Operations Research* v34. 2403–2435.

[14] Bard, J. F., Huang, L., Jaillet, P. and Dror, M. 1998. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation Science* v32. 189-203.

[15] Wu T.-H., Low C., Bai J.-W. 2002. Heuristic solutions to multi-depot location-routing problems. *Computers and Operations Research*. v29. 1393-1415. doi:10.1016/S0305-0548(01)00038-7

Towards Modeling the Traffic Data on Road Networks

Ugur Demiryurek, Bei Pan, Farnoush Banaei-Kashani and Cyrus Shahabi
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
{demiryur,beipan,banaeika,shahabi}@usc.edu

ABSTRACT

A spatiotemporal network is a spatial network (e.g., road network) along with the corresponding time-dependent weight (e.g., travel time) for each edge of the network. The design and analysis of policies and plans on spatiotemporal networks (e.g., path planning for location-based services) require realistic models that accurately represent the temporal behavior of such networks. In this paper, for the first time we propose a traffic modeling framework for road networks that enables 1) generating an accurate temporal model from archived temporal data collected from a spatiotemporal network (so as to be able to publish the temporal model of the spatiotemporal network without having to release the real data), and 2) augmenting any given spatial network model with a corresponding realistic temporal model custom-built for that specific spatial network (in order to be able to generate a spatiotemporal network model from a solely spatial network model). We validate the accuracy of our proposed modeling framework via experiments. We also used the proposed framework to generate the temporal model of the Los Angeles County freeway network and publish it for public use.

1. INTRODUCTION

The latest developments in online map services (e.g., Google Maps) and their widespread usage in hand-held devices and car-navigation systems have led to the recent prevalence of the location-based services. Many of the location-based services rely on efficient computation of the shortest path between a source and a destination in road networks. While the majority of the previous studies (e.g., [16, 10, 12, 3]) simplistically assume the travel-time of each segment of the network is constant, in reality the actual travel-time of a segment heavily depends on the traffic flow on the segment; hence, varies as a function of time. Recently, an increasing number of studies [8, 4, 5] consider time-dependent shortest path computation in road networks. However, most of these studies resort to using simplistic models and/or synthetic datasets to represent the temporal aspect of the road networks, mainly because collecting and working with real temporal data from road networks is costly and difficult, and the available temporal datasets are often proprietary and cannot be released for public use. Obviously, inaccurate temporal representation of road networks can seriously affect the validity of the design and evaluation of any proposed path planning technique for such networks; hence, the need for realistic models for traffic flows in spatiotemporal networks.

In this paper, we propose a framework for realistic and accurate modeling of traffic flows on road networks. The benefit of the proposed framework is twofold. First, anyone (e.g., governmental agencies) in possession of a real traffic data collected from a road network can use the proposed framework to derive and generate a realistic temporal model for the corresponding network, to be shared for public use (e.g., for researchers and policy planners) without infringing the copyright laws and jeopardizing the privacy of the dataset. As an example, we have used the proposed framework to generate and publish a realistic model for traffic flows in all freeways of the Los Angeles County based on the real (and proprietary) data provided to us by the county (see Section 4.1 for more details about this dataset)[1]. Second, as we describe in Section 4 (since the traffic in Los Angeles County is arguably typical and generic) one can use the proposed framework to generate realistic traffic flows specific to and customized for any given road network; hence, transferring the road network model to its corresponding spatiotemporal network model. Towards this end, we use a semi-supervised hierarchial clustering approach (based on the spatial characteristics of the network) to generate the spatiotemporal model of the network. To the best of our knowledge, our work is the first attempt in generating realistic temporal models for road networks.

The remainder of this paper is organized as follows. In Section 2 we review the related work. In Section 3 we provide the preliminary definitions, and subsequently in Section 4 we establish the theoretical foundation of our proposed traffic flow modeling framework and discuss the three-phase modeling process of this framework. In Section 5, we present the results of our experiments to verify and validate the accuracy of this framework. Finally, in Section 6 we conclude and discuss our future work.

2. RELATED WORK

In [2], Brinkhoff introduces a system called Network-based Generator of Moving Objects that models and simulates the behavior of moving objects (e.g., vehicles) on road networks. This system has been extensively used to benchmark k-nearest neighbor and location based search algorithms in road networks. While the focus of this system is the moving objects and their mobility in road networks, we primarily study to model the traffic flow on the network segments. In addition, this work relies on some simplistic assumptions about the network parameters such as minimum and maxi-

[1]http://infolab.usc.edu/projects/transdec/model.html

mum speed assignment for the network segments.

The freeway Performance Evaluation Monitoring System (PeMS) [13] developed by UC Berkeley collects and stores data from loop detectors operated by Caltrans. The main goal of PeMS is to convert the freeway sensor data into graphs and tables that show performance measures and traffic patterns on freeways in the State of California. The scope of PeMS is limited to collection and analysis of the historical freeway sensor data. However, our goal is to model the traffic flow for any given road network (even without sensor data) as described in Section 4.

Most of the traffic simulators developed in the recent decade use microscopic simulation models (aka, agent-based models) [14, 6] to simulate the traffic flow in road networks. The microscopic simulation models focus on the behavior of the system entities (e.g., vehicles and drivers) as well as their interactions with the system parameters (e.g., traffic lights). For instance, for each vehicle in the stream, a lane-change is described as a detailed chain of drivers' decisions. These simulation models, however, ignore the global descriptions of the traffic flows such as flow-rate, density and velocity and often are restricted to synthetic or simplified data.

There also exist several machine learning techniques developed for the purpose of traffic modeling. In [7], Kamarianakis et al. proposed a space-time autoregressive integrated moving average model to estimate the traffic flows on road networks. In [9], Lint et al. introduced a neural network based technique to model the traffic flow on freeways. However, all of these approaches are univariate and ignore most important factors such as road network geometry and spatiotemporal characteristics of the traffic flow.

3. DEFINITIONS

In this section, we formally define a road network with traffic flow as a spatiotemporal network. We assume a spatial network (e.g. the Los Angeles road network) containing a set of nodes and segments. We model the spatial network as a time-dependent weighted graph (i.e., spatiotemporal network) where the weights are time-varying travel-times (i.e., traffic flow) between the nodes. Below, we formally define our terminology

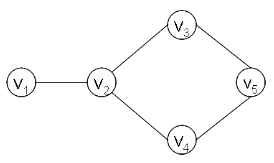
DEFINITION 1. *Spatiotemporal Network*
A Spatiotemporal Network is defined as a graph $G_T(V, E, W)$ where $V = \{v_i\}$ is a set of nodes representing the intersections and terminal points, and E ($E \subseteq V \times V$) is a set of edges representing the network segments each connecting two nodes. Each edge e is represented by $e(v_i, v_j)$ where v_i and v_j are starting and ending nodes, respectively, and $v_i \neq v_j$. For every edge $e(v_i, v_j) \in E$, there is an edge travel-time function $w_{i,j}(t) \in W$, where t is the time variable in time domain T. An edge travel-time function $w_{i,j}(t)$ specifies how much time it takes to travel from v_i to v_j starting at time t.

Figure 1 illustrates a spatiotemporal network modeled as $G_T(V, E, W)$. While Figure 1(a) shows the network structure with five nodes and five edges, Figures 1(b), 1(c), 1(d), 1(e), 1(f) illustrate the time-dependent edge costs (i.e., travel-times) for the edges of the network.

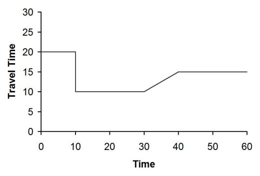
4. METHODOLOGY

Our modeling framework is based on the real-world traffic data collected from the freeways in Los Angeles County (LA). The proposed framework offers solutions to the following two cases. In the first case, given the historical temporal data (time-series of traffic flow possibly collected from various sensor locations) of a road network, our framework creates the spatiotemporal model of that network using the temporal data only. We refer to this case as Modeling with Temporal Data (MTD). However, the temporal data may not be available for most of the road networks as acquiring such data is a complex and sometimes prohibitively expensive task. In this (second) case, our framework generates a spatiotemporal network model from the *spatial characteristics* and the topology of the road network. We refer to the second case as Modeling with Spatial Characteristics (MSC).

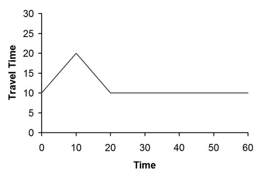
Our approach involves the following three steps. In the first step, we compute the time-dependent travel-times on each network segment using historical time-series sensor data (*traffic flow generation*). In the second step, we attach semantic information to the network by labeling the regions of the network based on its spatial characteristics (*spatial characterization*). Finally, in the third step, we employ a semi-supervised clustering algorithm to group the traffic flows of similar kind into respective spatial characteristics by using the data obtained in the first and second steps (*hierarchical semantic traffic flow clustering*). The main idea here is to find the most representative traffic flows in and between the network regions based on their spatial characteristics. As we describe in Section 4.3.2, the traffic flows found in the final step can be used to model the traffic of any given road network without temporal data. Specifically, one can transfer any road network model to its corresponding spatiotemporal network model by using the similar spatial characteristics introduced in our model. While the techniques developed in the first step can result in MTD, we employ the second and third step to achieve MSC. Below, we explain each of these steps in turn.



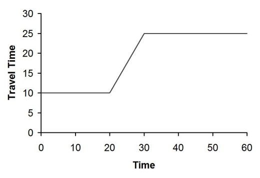
(a) Graph G_T



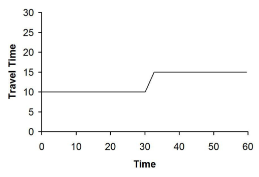
(b) $w_{1,2}(t)$



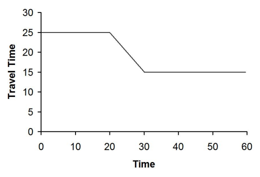
(c) $w_{2,3}(t)$



(d) $w_{2,4}(t)$



(e) $w_{4,5}(t)$



(f) $w_{3,5}(t)$

Figure 1: A Spatiotemporal network $G_T(V, E, W)$

4.1 Traffic Flow Generation

In the past one year, through a system called RIITS [15], we have been continuously collecting and archiving the sensor (i.e., loop detector) data from a collection of approximately 1500 sensors located on the freeways of LA County. The urban area of Los Angeles County has an area of 4752 square miles (12,308 km^2) and population of approximately nine million people. Figure 2 shows

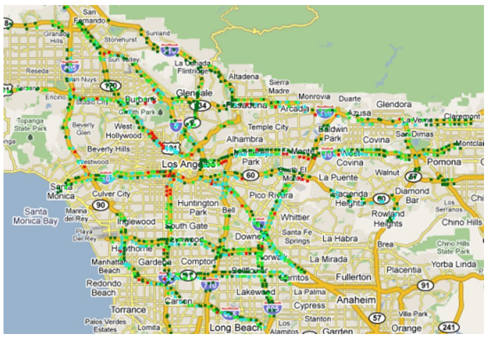


Figure 2: Traffic sensor layout in LA County

the spatial span (covering 1183 miles) of the traffic sensors on a map. The sampling rate of the sensor data is 1 reading/sensor/min. We average the readings over three consecutive time intervals in order to ease the implementation and smooth out the noise. Therefore, each sensor provides 480 distinct measurements per day. We only consider the readings during the weekdays. The storage space required for this streamed dataset is approximately 350 MB/day without indexing overheads. Currently, our data warehouse consists of data from the period of October 2008 to June 2009.

The main traffic parameters collected from the loop detectors are *occupancy* and *volume*. The loop detectors turn on and off as cars pass over them. The number of *'on'* readings within a time interval (e.g., 60 seconds) determines the occupancy measure. Occupancy is defined as the percentage of time a point on network segment is occupied by vehicles. The other parameter, volume, is defined as the number of vehicles flowing past a point during a time interval. We derive a third parameter, speed, from the occupancy and volume readings using the formula introduced in [1] $Speed = \frac{C*V}{O}$ where C is a constant proportional to the average length of a car, V is volume, and O is occupancy.

In order to determine the time-dependent travel-time on each network segment, we employ a two step process. First, using the spatial query operators, we map the coordinates of the individual sensors to network segments. Then, for each segment, we aggregate the desired traffic measure in both time and space dimensions by considering the distances between the sensors. For instance, for a given time instance, we compute the travel-time of a segment by the following formula $Travel\_Time = \sum_{i=1}^{n} \frac{D(s_i, s_{i+1})}{S_i}$ where S_i, $D(s_i, s_{i+1})$ and n represents the speed measured on sensor i at time t, the distance between two consecutive sensors, and the number of sensors on the segment, respectively. Figure 3 shows the graph of travel-time on a segment of I-405 freeway in LA between 6:00 AM and 8:00 PM on a weekday.

4.2 Spatial Characterization

In this section, we describe how we characterize the road network using geographical and topological characteristics of the network. Studying the real-world traffic data, we observe the following three main spatial and temporal characteristics of the traffic flow which motivated us to pursue the approach discussed in Section 4.3. First, the traffic flow on network segments demonstrates a strong *periodicity* at various spatial and temporal scales (daily, weekly, monthly, and quarterly). For example, the traffic flow on particular segment may exhibit a huge peak on each day at around 8:00 AM, a smaller one at around 4:00 PM, and an absolute minimum at around 2:00 AM during the weekdays in fall season. Second, the traffic flow is highly affected by the *spatial characteristics* of the network. That is, the traffic flow follows different patterns near major residential areas, city centers (aka, downtown), attraction areas (e.g., shopping centers, sports stadiums), and the regions in between. For instance, while a segment connecting a residential area to downtown is congested during morning hours, the opposite segment connecting downtown to a residential area is usually congested in the afternoon. Third, the traffic flows are also affected by the topology (i.e., another spatial characteristic) of the network. For example, a dense network topology which contains numerous nodes (hence many alternative routes) is usually congested in the hubs (i.e., intersection of the nodes) depending on the time of the day but has steady traffic flow in the rest of the region.

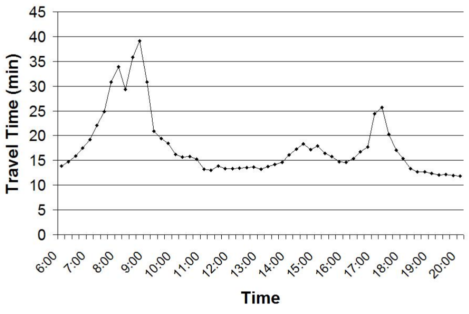


Figure 3: Real travel-time during a weekday on a segment of I-405 in LA County

As we discussed, the main idea behind incorporating the spatial characteristics of the network to our model comes from the observation that the traffic flow in certain parts of the network can be affected by the geographical and topological characteristics. Although, there are various other characteristics (e.g., population and demographics) that are also good candidates to characterize a road network, we select two major characteristics for the purpose of this study namely, geographical region and density. We plan to include more spatial characteristics into our model in the future. For our study, we developed a graphical user interface (i.e., a map mashup) that enables users to label the geographical regions (i.e., residential, downtown, and attraction) of the road network. To capture the density information, the map interface allows users to partition the road network into regular grid cells (e.g., 5x5 km) and label the sub-networks (overlapping the grid cells) as dense or sparse based on the distribution of the number of nodes in each grid cell. Note that the map interface allows users to control the grid cell size. Clearly, these characteristics do not consider all possible aspects of the traffic flow and their specific definitions may vary. Our main focus is to establish a framework that considers the spatial characteristics of the road network for generating a spatiotemporal network model. We emphasize that our framework allows users to select their preferred spatial characteristics among the pre-defined ones. For example, one can only select regional information (ignoring density) to generate the spatiotemporal model of a particular network. In the following section, we explain how we incorporate the spatial characteristics of a network in to our proposed semi-supervised clustering algorithm.

4.3 Hierarchical Semantic Traffic Flow Clustering

The goal of this step is to cluster the time-series data constructed in Section 4.1 by enforcing the spatial characteristics mentioned in Section 4.2. Such clustering enables us to find the most representative traffic flows for the corresponding network regions. Towards this end, we propose Hierarchical Semantic Traffic Flow Clustering ($HSTFC$) method that is based on the semi-supervised clustering algorithm introduced in [17]. Although the unsupervised clusters can identify the natural groups, it is extremely difficult to construct the mapping between the representation of the groups and their semantic meanings. Semi-supervised clustering addresses this issue by relating domain knowledge (in the form of labels and constraints) in to clusters. In other words, semi-supervised clustering not only creates natural groups with similar features but also provides semantic meanings to the cluster results. Therefore, in the context of our problem, semi-supervised clustering technique enables us to associate spatial characteristics of the network with their traffic flows. In the following sections, we first explain pairwise constraint clustering (a semi-supervised clustering method) and discuss how it fits in to our problem. Second, we present our proposed hierarchical clustering structure.

4.3.1 Pairwise Constraint Clustering Method

Pairwise constraint clustering (PCC) [17] is a classic technique to employ semi-supervised clustering. PCC, during the cluster computation, incorporates the domain knowledge (of the data instances) in the form of pairwise *cannot-link* and *must-link* constraints, and make the cluster results maximally satisfy the constraints. While *must-link* constraint specifies that two instances should be assigned into the same cluster, *cannot-link* constraint specifies that two instances should be assigned into different clusters. Let us now explain how this technique is adopted to our problem. As we discussed, in typical transportation networks, segments demonstrate different traffic patterns based on their geographical areas. For example, the traffic pattern of freeways near downtown may be entirely different than that of a suburban area. On the other hand, the segments which are spatially close to each other (e.g., two freeway segments near Hollywood) may generate similar traffic patterns. Hence, we can capture the knowledge in the latter case in the form of *must-link* constraint and the former case in the form of *cannot-link*. The formulation of pairwise constraint clustering is given below.

Let M be the set of must-link pairs such that $(x_i, x_j) \in M$ implies x_i and x_j should be assigned to the same cluster, and C be the set of cannot-link pairs such that $(x_i, x_j) \in C$ implies x_i and x_j should be assigned to different clusters. Let $W_m = w_{ij}$ and $W_c = \overline{w}_{ij}$ be the two sets that give the weight to the constraints in M and C, respectively. Let l_i be the assigned cluster number of instance x_i, and μ_{l_i} be the centroid of the cluster l_i. The cost of violating these pairwise constraints is typically the sum of violating pair(s) times their penalty weight. Specifically, the cost of violating a must-link constraint is given by $w_{ij} * f(l_i \neq l_j)$, where f is the indicator function, with $f(true)$ = 1 and $f(false)$ = 0. Similarly, we could get the cost of violating the cannot-link constraint as $\overline{w}_{ij} * f(l_i = l_j)$. Using this model, the problem of PCC is formulated as the minimization problem on the following objective function: $\frac{1}{2} \sum\limits_{x_i \in D} \|x_i - \mu_{l_i}\|^2 + \sum\limits_{(x_i,x_j) \in M} w_{ij} * f(l_i \neq l_j)$
$+ \sum\limits_{(x_i,x_j) \in C} \overline{w_{ij}} * f(l_i = l_j)$

Algorithm 1 presents our pairwise constraint (k-means) clustering algorithm. The algorithm takes the dataset of the traffic flow (D), a set of must-link constraints (M), and a set of cannot-link constraints (C). Note that M and C are derived from the spatial characterization step. First, we call the function *POPULATE-CONSTRAINTS* to generate transitive closure over pair-wise constraints denoted as M', C'. Next, we initialize the cluster center by choosing k points from the cannot-link constraints pairs in C' as long as they do not have must-link constraints in M'. If we cannot find such k points, we terminate the algorithm to enrich the input constraint set from the dataset, and restart. Finally, the algorithm returns the centroids of clusters that satisfy all the specified constraints. It is important to note that with Algorithm 1, we utilize the pairwise constraints for initializing the cluster centroid. For example, if two instances have cannot-link constraint, they should have distinct spatial category information. This enables us to guide the clustering process that generates two clusters maintaining distinct spatial characterizations. We assume that the cluster number (i.e., k) is equal to the number of pre-defined spatial characteristics.

Algorithm 1 Pairwise Constraint K-means Clustering Algorithm

```
Input: Traffic flow D, must-link constraints $M \subseteq D \times D$, cannot-link constraints $C \subseteq D \times D$, Cluster Number k
Output: The cluster index of each variable $l_1, ...l_n$
1: Call POPULATE-CONSTRAINTS($M, C$);
2: Initialize the cluster center $\mu_1, ...\mu_k$
3: For each point $x_i$ in $D$, assign it to the closest cluster $l_j$ such
   that VIOLATE-CONSTRAINTS($di, l_j, M, C$) is false.
4: For each cluster $C_i$, update its center by averaging all of the
   points $d_j$ that have been assigned to it.
5: Iterate between (3) and (4) until convergence.
6: Return $l_1, ...l_n$.
POPULATE-CONSTRAINTS(must-link constraints set M, cannot-link constraints set C)
1: For each a: if both $(a, b), (a, c) \in M$, $M = (b, c) \cup M$
2: For each a: if $(a, b) \in M$, and $(a, c) \in C$, $C = (b, c) \cup C$
3: Return $M$, $C$ and denoted as $M'$, $C'$
VIOLATE-CONSTRAINTS(data point x, cluster L, must-link constraints M, cannot-link constraints C)
1: For each $(x, y) \in M$: If $y \notin L$, return true.
2: For each $(x, y) \in C$: If $y \in L$, return true.
3: Otherwise, return false.
```

4.3.2 Hierarchical Pairwise Constraint Clustering

So far we have explained the pairwise constraint clustering, but PCC itself is not sufficient to solve our problem. This is because, some network segments may lead to multiple (and possibly contradictory) pairwise constraints depending on their spatial characterization. For example, let us consider both region and density information as two types of spatial characteristics that guide the pairwise constraint clustering. During the must-link and cannot-link constraint construction, two instances which have the same density value may lead to a must-link constraint. Meanwhile, a cannot-link constraint may also be assigned to these two instances due to their difference in the region values. In this case, since the two instances have both must-link and cannot-link constraints simultaneously, PCC technique will suffer. To avoid this problem, we propose a hierarchical pairwise constraint clustering method that guides the clusters in multiple levels by considering a single type of characteristics at each level. It is important to note that our hierarchical structure makes it very easy to add new characteristics (e.g., segment length) to the system. Currently, we only have two hierarchies namely, region and density.

Fig.4 depicts an example of our hierarchical clustering method for two spatial characteristics namely, region and density. As illustrated, at the first level, the region information is used to compute the initial clusters. At the second level, based on the results from the first level, the density information is used to guide the semi-supervised clustering. Finally, the output are the traffic flows (i.e., centroid of clusters) corresponding to each spatial characteristics.

Let us now explain how this step is useful to achieve MSC case

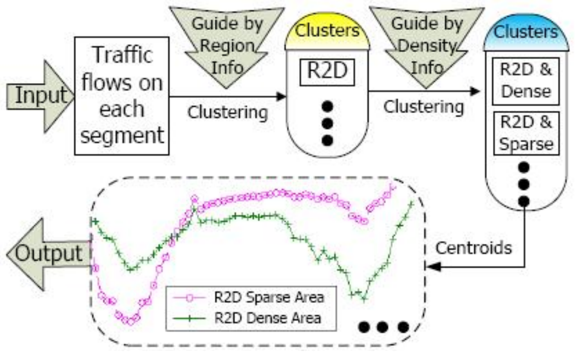


Figure 4: Hierarchical semantic clustering flowchart.

discussed in Section 4. As we explained, after clustering the traffic flows (for LA county dataset) based on the pre-defined spatial characteristics, we obtain a representative traffic flow (i.e., cluster centroid) corresponding to each spatial characteristics. Assuming that the traffic pattern in LA county is typical and generic, we use the proposed framework to generate the traffic flow for any given road network that has no temporal data but has similar spatial characteristics. Specifically, given a road network and its spatial characterization, we first group the network segments based on their spatial characteristics and then assign each group the corresponding cluster centroid obtained from LA county dataset.

5. PERFORMANCE EVALUATION

5.1 Experimental Setup

We conducted several experiments with different road networks and parameters to evaluate the performance of our algorithm. As we mentioned in Section 4.1, we used the real-world Los Angeles freeway traffic sensor data to construct our model. Since the traffic flow on freeways is much simpler than that of the local road network (i.e., no traffic light, no pedestrian), it requires less characterization. Therefore, to simplify our experiments, we only evaluated our model on freeway data. The sensor dataset is collected from 1592 sensors on the freeways during the period from October 2008 to June 2009. In order to represent the traffic flow on each segment, we computed the average travel time (from the historical sensor data) from 6:00 AM to 9:00 PM with 15 minute time intervals. As our road network dataset, we used Los Angeles (LA) and San Joaquin County (SJ) freeway network data. We obtained these datasets from NAVTEQ [11]. Using NAVTEQ dataset, we constructed the graph G(V,E) representation of LA and SJ freeway networks. Each network segment is represented in the vector data format and described by more than 20 attributes such as direction, speed limit, zip code, location, density, geographical location (e.g., residential), etc. Based on the location and direction information, we labeled the freeway segments into eight spatial categories namely, *RR*, *R*, *D*, *A*, *R2D*, *D2R*, *R2A*, *A2R*. The descriptions of these labels are presented in Table 1. Moreover, in addition to region labels, we defined another label capturing the density information of the network segments. In order to assign density label to the network segments, we partitioned both LA and SJ freeway networks into 5 $\times$ 5 km regular grid cells. Based on the average number of nodes(α) in each grid cell (assuming uniform distribution of the nodes), we labeled the segments as *Dense Area* (i.e., area that has more nodes than α) or *Sparse Area*. We conducted our experiments on a workstation with 2.7 GHz Pentium Core Duo processor and 12GB RAM memory. Due to the space constraints, we only present the experimental evaluations from LA dataset.

Table 1: Spatial Label Description

Label	Spatial Information for Freeway Segments
R	Residential Area
RR	Remote Area, area far from downtown and res.
D	Downtown Area
A	Attraction Area
R2D	From Residential Area to Downtown Area
D2R	From Downtown Area to Residential Area
R2A	From Residential Area to Attraction Area
A2R	From Attraction Area to Residential Area

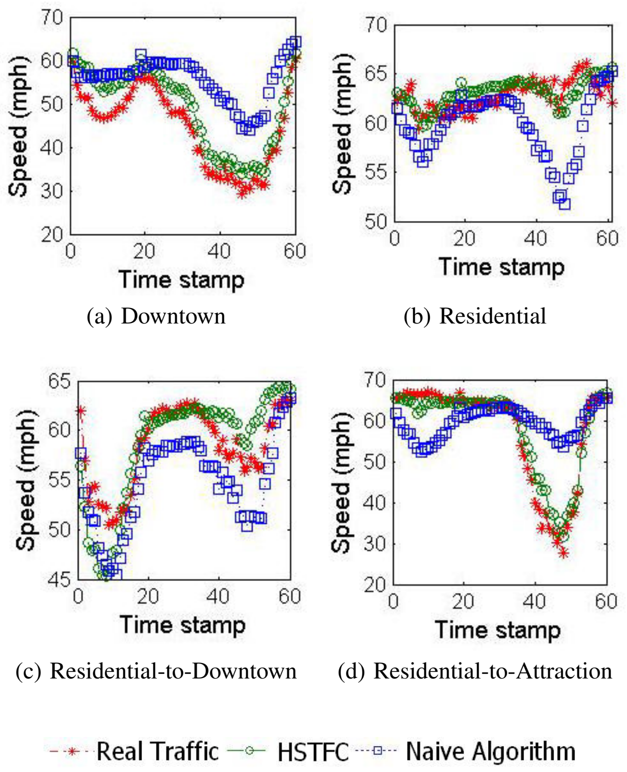


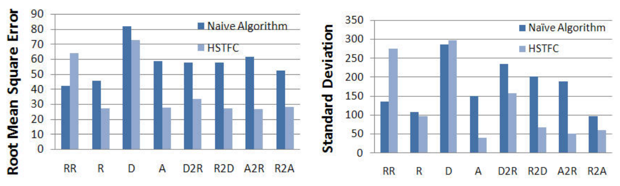
Figure 5: Traffic flow comparison

5.2 Performance Study

For performance evaluation, we compared our algorithm with a naive approach that is based on decision tree. To implement decision tree, we used eight spatial categories (represented in Table 1) and density information (i.e., dense or sparse) as the nodes of the decision tree. The leaves of the decision tree contained the traffic flow information of the segments in the same category. Since each leaf can contain more than one traffic flow, we took the average value of the traffic flows to represent the corresponding leaf with one traffic flow. In our experiments, we measured the traffic flow similarity, general error rate and confidence interval.

5.2.1 Traffic Flow Similarity Comparison

In this set of experiments, we compare the traffic flow obtained from the two algorithms with actual (observed) traffic flow on the segments. We randomly choose one instance in four categories: D, R, $R2D$, $R2A$. Figure 5 shows the traffic flow with respect to these four categories. The graphs cover the period from 6:00 AM



(a) Mean (b) Variation

Figure 6: General error rate comparison

(represented as 0 in the figures) to 9:00 PM with 15 minutes time intervals. As illustrated, the traffic flow generated by our HSTFC algorithm is more consistent with actual traffic flows. This is because, in real-world, some traffic patterns do not follow the major traffic flow trend in the same category due to some special events (e.g., accidents, lane closure). However, the naive decision tree approach considers that each instance contributes equally towards the construction of the category presentation. This assumption causes the results deviate from the major pattern trend hence leading to imprecise traffic flow representation. On the other hand, HSTFC considers both the spatial correlations and the traffic flow features; therefore, the centroid is calculated only based on the major trend of each category without possible noisy instances.

5.2.2 *General Error Rate Comparison*

In the second set of experiments, we compare the overall performance of two algorithms based on average root mean square error (MSE) and standard deviation(STD). These two measures enable us to quantify the amount by which the estimated centroids differ from the real instances. MSE and STD are calculated based on the distances between an individual instance and its corresponding centroid. The lower the value of these measures, the more precise the corresponding algorithm. Figure 6 depicts the performance of the two algorithms with respect to eight spatial categories. In general, the results show that the naive approach is less accurate than our algorithm with respect to both MSE and STD measures except for the RR category. The reason is that for RR, we require more types of characterizations to capture its traffic flow.

5.2.3 *Confidence Interval Evaluation*

In the final set of experiments, we use confidence intervals (CI) to indicate the reliability of our estimates. In particular, we evaluate the intensity of the featured clusters generated by the algorithms using CI. We consider the level of confidence interval is 90%, and use the mean of all distances between the instances and their cluster centroids as the observed mean value. Therefore, the lower the mean value, the denser the cluster. Figure 7 depicts the Euclidean distance between the instances and the cluster centroids (Y-axis) for eight spatial categories (X-axis). As illustrated, the naive algorithm has more sparse population of instances in each category.

6. CONCLUSION AND FUTURE WORK

In this paper, we introduced a framework for realistic and accurate modeling of traffic flows in road networks. We explained the design and implementation of our framework based on a real-word traffic sensor dataset. We intend to extend this work in two directions. First, we plan to extend the set of spatial characteristics supported by our framework to a complete minimum set that allows for modeling all typical road networks. Second, we plan to incorporate temporal characteristics (e.g., congestion intervals) of the road networks into our framework.

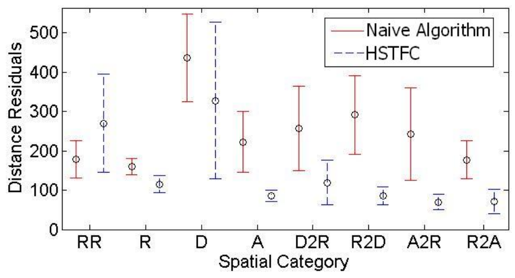


Figure 7: Confidence interval evaluation

7. ACKNOWLEDGMENTS

This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0534761, and CNS-0831505 (CyberTrust),the NSF Center for Embedded Networked Sensing (CCR-0120778) and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] P. Athol. Interdependence of certain operational characteristics within a moving traffic stream. In *TRB*, 1967.
[2] T. Brinkhoff. A framework for generating network-based moving objects. In *Geoinformatica*, 2002.
[3] U. Demiryurek, F. B. Kashani, and C. Shahabi. Efficient continuous nearest neighbor query in spatial networks using euclidean restriction. In *SSTD*, 2009.
[4] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, 2008.
[5] B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, 2007.
[6] S. P. Hoogendoorn and P. Bovy. State-of-the-art of vehicular traffic flow modelling. In *Journal of Systems and Control Engineering*, 2001.
[7] Y. Kamarianakis and P. Prastacos. Space-time modeling of traffic flow. 2007.
[8] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE*, 2006.
[9] H. v. Lint, S. P. Hoogendoorn, and H. J. v. Zuylen. State space neural networks for freeway travel time prediction. In *ICANN*, London, UK, 2002.
[10] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, 2006.
[11] Navteq. http://www.navteq.com. Last visited June 17, 2009.
[12] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.
[13] PeMS. https://pems.eecs.berkeley.edu/. Last visited May 15, 2009.
[14] M. Pursula. Simulation of traffic systems-an overview. In *Journal of GIS and Decision Analysis*, 1999.
[15] RIITS. http://www.riits.net/. Last visited December 25, 2008.
[16] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
[17] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. K-means clustering with background knowledge. In *ICML*, 2001.

A Scalable Heuristic for Evacuation Planning in Large Road Network

Dafei Yin
Institute of GIS & Remote Sensing
Peking University
Beijing, China, 100871
dafeiyin@pku.edu.cn

Abstract
Evacuation planning is of critical importance for civil authorities to prepare for natural disasters, but efficient evacuation planning in large city is computationally challenging due to the large number of evacuees and the huge size of transportation networks. One recently proposed algorithm Capacity Constrained Route Planner (CCRP) can give sub-optimal solution with good accuracy in less time and use less memory compared to previous approaches. However, it still can not scale to large networks. In this paper, we analyze the overhead of CCRP and come to a new heuristic CCRP++ that scalable to large network. Our algorithm can reuse search results in previous iterations and avoid the repetitive global shortest path expansion in CCRP. We conducted extensive experiments with real world road networks and different evacuation parameter settings. The result shows it can gives great speed-up without loosing the optimality.

Categories and Subject Descriptors
F.2.0 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity -- *General*

General Terms
Algorithms, Performance, Experimentation

Keywords
Evacuation Planning, Shortest Path, CCRP

1. Introduction
Evacuation, generally defined, is to transfer people at risk from dangerous sources to safe destinations. The evacuation planning gives optimized route and schedule for people to evacuate efficiently. The challenge to give out an optimized evacuation plan lies in dealing with the conjunction. Basically, if we did not consider the capacity constrains along the route and not separate people well spatially/temporally, there will be terrible conflict as everyone would like to take shortcut. Most existing evacuation planning works in relatively small scale, e.g. building or campus. In this work, we consider this problem in large scale, where people must get out of town via the transportation network. It is an important function for crisis management authority in big city to deal with the disaster like earthquakes, hurricanes, terrorist attacks, etc. Although some agencies have setup some guides on how to reach the safe area, these rough plans seldom consider the actual road network capacity during the disaster and did not consider the position of people when the disaster happens. Here, we assume we can get people's position by tracking their cell phone signal, and we can monitor the road network situation by CCTV camera etc. Based on this information, we can come out a set of plans optimized for everyone, rather than for the interest of individual. And we can give each citizen detailed instruction (via e.g. SMS to their cell phone) on which route to take, as well as when to start/stop along the route. If they all follow our planning, that will be the best for all of them to evacuate soon.

An efficient evacuation planning should consider the trade-off between minimizing the egress time [1] and minimizing the algorithm run-time. The optimal solution generates plan that minimize the egress time. Unfortunately, most of previous optimal solutions run too slow and require too much memory. They often convert evacuation problem to Quickest Flow problem over Time-Expended network [4] [6]. It is hard for them to scale because the time-expended network model needs to replicates the entire network in each time slice and add links in between these networks. Practically, we would like to minimize the algorithm run-time to give quick response, even that sacrifices the optimality a little. The recent work CCRP (Capacity Constrained Route Planning) [1] achieved great improvement on both run-time and memory size. However, it still needs more than a day to give the result for mid-sized city [2]. In this paper, we propose a new heuristic algorithm CCRP++ based on the similar semantic as CCRP, but more scalable to large road network. There are many definitions on "scalable". Here we give one definition in the context of evacuation planning. A scalable evacuation algorithm means: the run-time to generate plan for each group will not increase as the size of network increase. Like CCRP, our algorithm will also generate one plan for one group in each iteration, but the run-time per plan is almost constant regardless the network size.

The rest of the paper organizes as following: in Section 2, we formally define the evacuation problem, and analyze the overhead of CCRP algorithm. In Section 3, we present the idea of our algorithm. In order to make it easier to understand our design, we discuss several naive implementations before introducing the

[1] Egress time is the amount of time it takes for all evacuees to reach the safe destinations. It is the duration from the first person start moving to the last evacuee arrives at the safe place.

CCRP++. In Section 4, we present our experiment to verify its optimality (egress time) and the efficiency (algorithm run-time). In Section 5, we discuss why we not loose the optimality, compared to CCRP. We conclude our work and point the future work in Section 6.

2. Problem Definition and Previous work

2.1 The Evacuation Problem

Given: A transportation network represented as graph G (V,E) with some source nodes S, some destination (exit) nodes D and some middle node N. There are p_s evacuees in each source $s \in S$ which need to be sent to any of the safe destination $d \in D$. Every node $n \in N$ has maximum capacity Cap(n). Each edge(link) $e \in E$ has capacity constrains Cap(e) and travel time Cost (e).

Output: An evacuation planning consists of a set of plans for each individual. Since people from the same source share the same route/schedule, they are generated together as a group. So each plan consists of the following information: the number of people, the origin-destination routes, the arrival and departure time in each middle node. It can be interpreted as a integer flow, f(u,v) where u, v $\in$N, moving along the path. The maximum number per plan should observe the capacity constraints.

Goal:

1) Minimize egress time

2) Minimize the algorithm run-time.

Constrains:

Capacity Constrains: $f(u,v) <= Cap(e)$; $f(u,v) <= Cap(v)$;

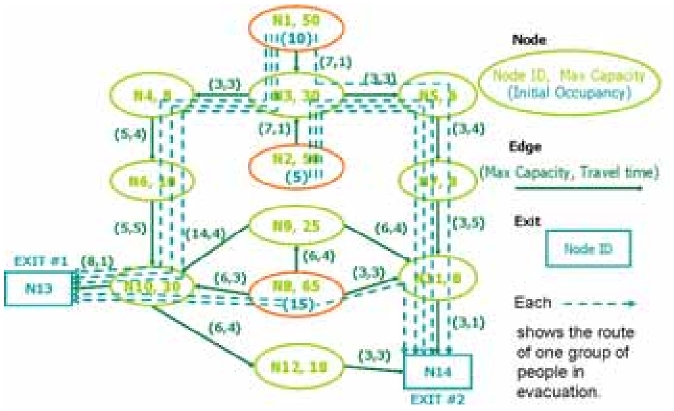
Flow Conservation: $\Sigma f_s(u,v) = p_s (u \in S)$;

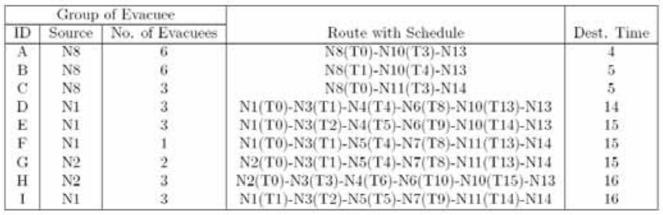
$\Sigma f_v(u,v) = \Sigma p_s \ (v \in T)$;

Integer Constrains: $f(u,v) > 0$; $f(u,v) \in N$;

Model:

Following our previous work, we model the road network as graph consisting of a set of sources nodes with certain number of evacuees, a set of destination nodes as safe place, a set of middle nodes with maximum holding capacity and the edges with travel cost and capacity linking the nodes. Figure1 shows a small network model and corresponding evacuation plan example.





Group of Evacuee			Route with Schedule	Dest. Time
ID	Source	No. of Evacuees		
A	N8	6	N8(T0)-N10(T3)-N13	4
B	N8	6	N8(T1)-N10(T4)-N13	5
C	N8	3	N8(T0)-N11(T3)-N14	5
D	N1	3	N1(T0)-N3(T1)-N4(T4)-N6(T8)-N10(T13)-N13	14
E	N1	3	N1(T0)-N3(T2)-N4(T5)-N6(T9)-N10(T14)-N13	15
F	N1	1	N1(T0)-N3(T1)-N5(T4)-N7(T8)-N11(T13)-N14	15
G	N2	2	N2(T0)-N3(T1)-N5(T4)-N7(T8)-N11(T13)-N14	15
H	N2	3	N2(T0)-N3(T3)-N4(T6)-N6(T10)-N10(T15)-N13	16
I	N1	3	N1(T1)-N3(T2)-N5(T5)-N7(T9)-N11(T14)-N14	16

Fig1. The network model and example evacuation plan [1]

Notation[2]:

We list some of notations we will in the rest of the paper

S (si)	Source node set ; Source node
D (di)	Destination node set; Destination node;
N (ni)	Middle node set; middle node;
P (pi)	Total number of evacuee; Num of people in node Ni
S0	Virtual super source node
R	Route; One path /schedule plan per iteration
EA	Earliest Arrival Time to any destination from one source
Q	The priority queue used in CCRP-- and CCRP-
RQ	The priority queue used in CCRP+ to store Reserved Route
PreRQ	The priority queue used in CCRP++ to store Non-Reserved Route

2.2 The CCRP Algorithm

CCRP is a greedy algorithm to give sub-optimal evacuation planning. The core heuristic is to choose the source with the "Earliest Arrival time (EA)" and send maximum possible people within capacity constraints along the path in each iteration. The intuition behind this heuristic could be: evacuate the people "nearest" to any destination first will give others more chance to take shorter path. Here, "nearest" is not based on distance but on temporal. It means "the arrival time to any destination from one source is the earliest among all sources". Besides accelerating the run-time, the CCRP requires less memory since it uses only the original network instead of the time-expanded network.[3] The complexity of CCRP is **O**(P*N*logN), where P is the total number of evacuees in network, and N is the number of nodes[4].

CCRP iteratively runs Dijkstra shortest path algorithms from all source nodes to find the source-destination pair with EA. It starts from all sources' earliest possible start time, revises the EA for each middle node, and label setting the node with minimum EA until reach any destination. Then it retrieves the route by back tracking from this destination to the source Si. Si becomes the EA

[2] In this work, we do not differentiate the similar concepts like: "start" and "source", "exit" and "destination", "path" and "route", and "edge" and "link".

[3] CCRP uses time aggregated model [3] to represent the capacity changes for each edge during the entire evacuation process. It still need to record the available capacity for every edges in each time slot, but it at least avoid the additional link in-between the network of different time slots in time expanded network model.

[4] **O**(NlogN) is Dijkstra shortest path complexity in sparse graph. In the worst case, there will be one person per plan/iteration.

source in this iteration. After that, it reserves[5] maximum of available capacity of the edges along the path. Such process of single iteration will go on until no evacuees in any source.

CCRP is not scalable because as the network size increases, the expansion becomes more and more expansive as the source number increase. Considering the spatial distribution of the source and destination nodes will give us a more clear perspective to understand this process. In each iteration, like water wave circle expanding from different sources, CCRP expand from different source in parallel until one expansion touches any destination. One observation is there are many "unfinished" expansions remains when the first source reaches its destination. For example, in Fig2, there are three source nodes, S1, S2, and S3. The earliest arrival path is S1-D1 in the first iteration and S2-D2 in the second iteration. In both iterations, the expansion of S3 is “unfinished” and “repetitive”. If there are S sources, there will be S-1 such overhead expansions in single iteration, and it will be (S-1)*P such expansions in the worst case. Even worse, most of these expansions will repeat again and again in the future iterations until they become the first one to touch any destination. However, without these parallel expansions, no one can tell which source's expansion will touch the destination first. These unfinished expansions are the "overhead" of CCRP[6].

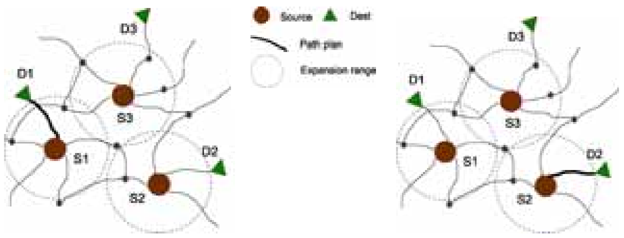


Fig2. The expansion of S3 is “unfinished” and “repetitive”

3. Algorithm

3.1 Overview of Algorithm Design

Based on the observation of the overhead of CCRP, we can see there are many unfinished shortest path expansions neither contribute to the result in current iteration nor benefit in future. We would like to ask: if only one source's shortest path is directly contributing in single iteration, can we delay the expansions from other sources? Or can we complete the unfinished expansions to re-use in the future iteration?

We come out a series of algorithms to answer the above questions. Some are naïve but worth to present for better understanding the final one CCRP++. All the algorithms share the same idea to "delay update the other sources' shortest path until it get chance to reserve path". They share the following procedure:

a. Record the shortest path length from every source nodes.
b. Pick the source node with EA to send people (reserve path).
c. Update the new shortest path from this source.

Following the heuristic of CCRP, our algorithms give the person near the exit higher priority to evacuate. However, we expand from one source in each iteration, and reuse this shortest path in the future iterations. They all use the priority queue with <EA, Source> as <key, value> to record and pick the source node with the earliest arrival time. All the algorithms terminate when there are no people in any source. They different from each other in initialize and update policy, which will be introduced below.

3.2 The Naive Approaches

3.2.1 CCRP--

There is a naive algorithm CCRP--, which empty sources one by one. It first calculates the EA for each source without considering the capacity constrains. Then it puts the source nodes into a priority queue Q with the order of their EA. After that, it pops the source node in the queue, and evacuates all the people in this source. The major drawback of this approach is that it does not keep the semantic of CCRP. For example, the sources ordered in queue are S1, S2, and S3. S1's second path may be longer than the path from S2 and S3, but it still reserves the second path from S1.

3.2.2 CCRP-

Another algorithm called CCRP- initialize Q the same as CCRP--. Then, in each iteration, it pops the first element <EA_i, S_i> in the queue, and "checks" the availability of route R_i in case some edges are taken by other source's reservation. If it passes the check, it reserves the max available capacity along this route. Then, it finds a new available route R’ and insert <EA_i’, S_i>. If it fails the check, it will just find another new route R’’ and re-insert queue the <EA_i’’, S_i>. Unlike CCRP--, CCRP- gives interleaved path among different sources. It avoids the "unfinished" and "repetitive" expansion in CCRP, because we only need to update the shortest path from single source when it pops the queue.

The major problem of the CCRP- is the route in the queue will invalidate frequently. Because we insert available but not yet reserved path into the queue, it is possible that the reservation on top of the queue will take the edge in the path from other sources. If not updated in time, the invalid route will ruin the order of the queue. For example, the sources ordered in queue are S1, S2, and S3. It is possible that S2's route R2 blocked by S1's reservation, so the EA2 is invalid. The EA of S2’s available route R2’ will be EA2’, which maybe longer than EA3. However, the algorithm can not detect this wrong order before S3 was popped. It pops S2 before S3 using the invalidated key EA2.

If we want to keep the correct order all the time, one way is to re-calculate all the available paths for all sources and re-order the queue, which will lead to similar overhead as CCRP[7]. Even if we sacrifice the correct order of the queue, we may still suffer from too many failed checking. The updated route maybe invalid again

[5] Reserve not means exclude others to take this path. Only the “full reservation” which takes the last available capacity would exclude the other source’s chance to use edge e at the same time.

[6] In the later work of CCRP, this process is interpreted as single source shortest path tree expansion from a virtual super source S_0 to any destination by adding edges from S_0 to S_i with length as the earliest departure time of S_i. It maintains the same semantic because finding the shortest path from one super source S_0 is the same to find EA from any one source S_i. However, it does NOT avoid the overhead to expand from all source nodes from S_0 due to tie breaking.

[7] There should be some intelligent way to identify the influenced sources instead of re-calculating all the shortest paths. But this is beyond the discussion of this paper.

and again before it valid (when it is popped the queue). The frequent updating is an unpredictable overhead.

3.2.3 CCRP+

In order to deals with the "invalidation" problem of CCRP-, we come to the CCRP+ algorithm. Different from CCRP-, it "reserves" the updated available path before reinsert into the queue RQ, which only consist of reserved path. So it guarantees the routes validate before popped and avoid the checking (The first time one source node being popped still needs to be check, because the route we insert at the beginning is not reserved).

The drawback of this approach is it not guarantees the correct order of non-reserved source node before it first popped. E.g. At the very beginning, the order in RQ is S1, S2, S3. It is possible that S1's reservation will take the edge in S2's route R2. And the S2's next available route R2' may be longer than S3's route R3. So we should update S3 before S2. One way to keep the correct order of non-reserved route is to recalculate path for all the remaining non-reserved sources. However, it may take long time before the last source to pop first time.

3.3 The CCRP++

In order to guarantee the correct order of non-reserved source node, we introduce an auxiliary priority queue PreRQ to order the non-reserved sources before insert into the priority queue RQ. The <key, value>in both queues are <EA_i, S_i>. PreRQ keeps the sources whose path NOT yet reserved. The RQ keep the source whose path been reserved.

Algorithm 1. CCRP++

```
Initialization: Pre-compute the path for all sources to its nearest destinations. Put <$EA_i$, $S_i$> into PreRQ.

Pop<$EA_1$, $S_1$> from PreRQ and insert it into RQ;

while (exit_num < evac_num)
  Q1 = RQ.top();
  P1= PreRQ.top();
  if ( P1.EA< Q1.EA )
      PreRQ.pop();
      EA = Check (P1); // get the available EA
      if  (P1. EA == EA)
          RQ.push(P1); // push <$EA_i$, $S_i$> into RQ
      else
         P1.EA =  EA;
         PreRQ.push (P1);   // push back to PreRQ
  else
      RQ.pop(); //Send people and release the capacity
      Q2 = RQ.top();
      while ( Q1.EA <= Q2.EA )
            Q1.EA =  Update(Q1.S); //find next route
```

Fig 3. The CCRP++ algorithm

Fig3 is the pseudo code of CCRP++. We first pre-compute the EA time from all sources and order them in PreRQ (without considering the capacity constrains). Then we pick the top one from PreRQ to insert into RQ if its path is shorter and valid. In order to verify this, we compare the EA of the top elements: P1 from PreRQ, and Q1 from RQ. a) If P1.EA< Q1.EA, it means the P1 maybe have the chance to reserve. We first check its availability. If it is available, we reserve this path and put it into the RQ. If it fails the checking, we find an available path and insert the updated <EA_i', S_i> back to PreRQ. b).Otherwise, if P1.EA >= Q1.EA, it means the Q1 have the "privilege" to continue reserve. So we reserve another available path, until the Q1's updated route is longer than Q2's.

We omit the Check(), Update() functions. They are simply a shortest path using Dijkstra algorithm from single source and return the Earliest Arrival time. Once all the people from one source node are evacuated, it will not be re-insert into the RQ anymore. The algorithm continues until there is no source in any queue.

4. Experiment

We use binary heap to realize priority queue PreRQ and RQ. We store the entire path, including the arrival and departure time at each middle node. Since we have no prediction on how long will it takes, we use dynamic array Cap[Edge][Time] to keep track the available capacity of each edge during evacuation. Obviously, linked list is another option. The test result we presented here are based on the experiment in Windows 32bit platform (OS: Windows XP; CPU: IntelCore2, 1.66GHz / Memory: 1.5GB). The code is written in C++ and compiled with Microsoft .NET Visual Studio 2005.

4.1 Experiment on Different Network Size

We use both real and synthetic data to conduct our experiment. We use three datasets provide by [6] as basic road network. And we can control different evacuation parameters: the number of source, destination, and generate different number of evacuee on sources. Before conducting these tests on large network, we have tested both CCRP and CCRP++ in a small building network to verify its correctness. Below is some of our test results.

Table 4.2 a. Test on the small building data

(47 nodes, 148 edges, 41950 evacuees, 12 sources, 1 destination).

Algorithm	Egress time	Groups	Run-time (sec)	Run-time per Plan (sec)
CCRP	138	820	2.656	0.0032
CCRP++	105	1058	1.179	0.0011

Table 4.2 b. Test on the OL (City of Oldenburg) data (6105 nodes, 7029 edges, 511636 evacuees, 999 sources, 1038 destinations)

Algorithm	Egress Time	Groups	Run-time (sec)	Run-time per Plan (sec)
CCRP	2535	3090	132.176	0.0427
CCRP++	2533	3483	15.454	0.0044

Table 4.2 c. Test on the TG (City of San Joaquin County) data (18263 nodes, 23797edge, 1429655 evacuees, 2844source, 3983 destinations)

Algorithm	Egress Time	Groups	Run-time (sec)	Run-time per Plan (sec)
CCRP	2283	8171	2700.018	0.3304
CCRP++	2282	9067	53.578	0.0059

Table 4.2 d. on the SF (San Francisco) data

(174056 node, 221802 edge, 13037136 evacuees, 26019 sources, 44245 destinations)

Algorithm	Egress Time	Groups	Run-time (sec)	Run-time per Plan (sec)
CCRP	--	--	--	--
CCRP++	714	81887	2668.562	0.0326

We get 2, 10 and 50 times speed up in building network, small and mid-sized city respectively[8]. In all the test sets, we achieved better run-time compared to the CCRP. Obviously, the larger the network size is, the more speed up we can gain.

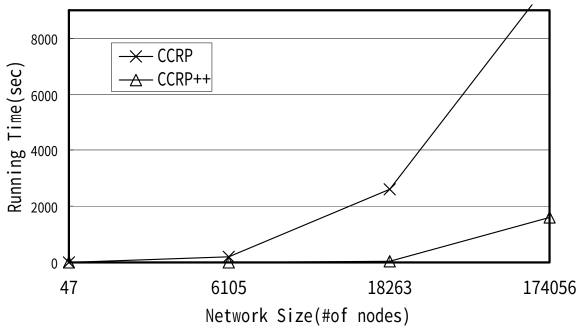


Fig.4. The run-time of CCRP and CCRP++

Another observation is the run-time per plan not increasing as much as the network size increases. The slight increment comes from tie breaking[9] rather than the shortest path calculation. Our algorithm is scalable to large network.

We noticed the egress time in most of our tests is less than CCRP. Besides, the number of groups generated by CCRP++ usually is more than CCRP, i.e., CCRP++ tends to partition the people into finer groups. Nevertheless, there is no semantic to control the granularity of groups in both CCRP and CCRP++. Any way, the good news is CCRP++ did not loose the optimality compared to CCRP. We will further discuss this in Section 5.

4.2 Experiment on Different Parameter

We also conducted experiments on different parameters to verify how evacuee number, source number, destination number influence the results. The experiment shows our algorithm not only scalable to the networks size, but also scalable to different evacuation parameters.

Table 4.3 a Different Evacuee Number Test on OL data set (nodes: 6105; sources: 998 regular: 4069 destinations:1038)

Test set	Evacuees	Egress time	Groups	Run Time (sec)
OL-1-1-1.net	511636	2533	3483	15.454
OL-2-1-1.net	1023787	2609	6581	24.625
OL-3-1-1.net	1533616	2646	9628	33.469

As the number of evacuee increase, the number of group and the run-time increase accordingly. Interestingly, the egress time just increases slightly. Maybe it is because we have so many destinations that the conflict is not obvious.

Table 4.3 b. Different Source Number Test on OL data set (Nodes= 6105 Evacuee ~= 511636 Destination = 1038)

Test set	Sources	Egress time	Groups	Run Time (sec)
OL-1-1-1.net	998	2533	3483	15.454
OL-1-2-1.net	2091	3049	3931	28.360
OL-1-3-1.net	3079	3047	4534	38.953

As the source number increases, the run-time increases obviously. However, we did not anticipate the egress time also increases. Maybe that is due to more conflict from different sources.

Table 4.3 c. Different Dest Number Test on OL data set (Number of nodes: 6105 (Evacuee ~= 511636 Source ~= 998)

Test set	# of Dest	Egress time	Groups	Run Time (sec)
OL-1-1-1.net	1038	2533	3483	15.454
OL-1-1-2.net	1537	1892	3201	15.781
OL-1-1-3.net	2080	1892	2958	15.031
OL-1-1-4.net	3109	1067	2512	11.422

As the destination number increase, the evacuation time, group generated, and run-time got decrease in general.

5. Discussions

The core idea of our heuristic is to let the source keep updating its own shortest path tree, rather than updating the entire shortest path tree from S_0. But the dilemma is without expanding from all the sources, it does not guarantee the same result as CCRP. The CCRP++ actually changes the semantic a little. It does not find the path with the earliest arrival time in every iteration. Instead, it picks the path with the minimum EA in the last expansion to update. This enables us to delay updating other sources not likely with EA in current iteration.

There is yet another way to interpret how we avoid the overhead expansion in CCRP. Similar to the idea of Dijkstra shortest path algorithm, assume we have an OPEN heap to store candidate routes and CLOSE list to store the determined plans. We put the path reserved before i_{th} iteration in CLOSE, then find the new path candidates and put them in the OPEN. It chooses from the OPEN list the route with minimum length as $i+1_{th}$ route and inserts it into the CLOSE. The difference between the CCRP and CCRP++ is the way they updating the OPEN list. The CCRP will generate the OPEN from scratch in every iteration by expanding from all the sources. An more detailed complexity analysis will be **O**(P*S* (N/S)* log (N/S)). Instead, the CCRP++ chooses only the source that just reserved path to update its new path. Most of the paths in OPEN are reused by previous expanding. The complexity of CCRP++ is **O** (P* (N/S)* log (N/S)). That is why CCRP++ always runs faster than CCRP. [10]

Another problem is how will the semantic change of CCRP++ influence the result, especially on the optimality, compared to CCRP?

[8] We did not obtain the result of CCRP in large city because the original CCRP algorithm use static capacity table. With large time slot, it exceed memory limit in Windows 32bit platform.

[9] If there are more source nodes, there are more nodes with the same EA, which lead to more nodes re-insert to PreRQ.

[10] We would like to point out the idea to re-use the OPEN list is widely used in repetitive/dynamic shortest path planning, e.g. [8]

One natural question would be: the CCRP++ may impair the "fairness" of the CCRP due to "over-reservation", which means it is possible that the updated route on top of RQ will take the edge on "intended path" of the source beneath. This will leave less chance for them to reserve shorter path.

First, it is true but we argue fairness is not help to improve efficiency. We admit by using CCRP++, some source who could evacuate earlier in CCRP lose the chance to evacuate that early in CCRP++. For example, in our test on building data, one node N6 did not get chance to evacuate any people until 69th iteration with EA= 44, which is later than 48th iteration with EA = 40 in CCRP. However, it worth to point out that the entire egress time of CCRP++ is 105, which much less than 136 by CCRP. The "fairness" is neither the goal of CCRP nor CCRP++. Actually we concern more on efficiency. To be concrete, we concern whether one group could arrive at any destination earlier by taking certain edge.

Second, the optimality of CCRP++ is at least not worse than CCRP. As the CCRP will find the earliest available path in each iteration, we prove the CCRP++ is able to reserve that path too.

Lemma1. (FIFO property) If an edge e is reserved earlier by S_1 in t_1, and reserved later by S_2 in t_2, then $EA_1<=EA_2$. (The earlier to take the critical edge, the better to reach destination earlier.)

Proof: (By contradiction) if S_1 arrived e at t_1 find its EA_1 will not earlier than EA_2, the S_1 would rather wait at e until t_2, and then take the same route as S2.

Corollary1. (Early reserve deserve) If S reserve route R by taking edge e, then any source which would like to reserve the edge e later could not reach destination earlier than S.

Theorom1: CCRP++ reserves the earliest available path as CCRP.

Here we argue that once there exists an earliest available path which passes edge e, e must be taken by the popped source node in RQ in CCRP++ at the first time.

Proof: Assume in the current iteration, S_1 is the source popped from RQ. S_2 is beneath S_1 in RQ with R_2 as reserve path. S_3 is the source in PreRQ, with the R_3 as intended path. By updating its old route R_1, S_1 wish to take e to be part of the new route R_1'. 1). If there is available capacity on edge e, the S_1 would reserve it and re-insert <P_1', S_1> into the RQ. It will be popped later. 2). If S_3 in PreRQ could take e and have available path shorter than R_1', it should have been entered the RQ, and will pop next time. S3's route R3 will be the plan for next iteration. 3).If S_2 in RQ have reserved to take the edge e earlier, the route R_2 should be shorter than R_1'. In that case, <EA_1', S_1> will enter the RQ beneath the <$EA_{2,}S_2$ > and pop later than R_2.

No matter which source in PreRQ or on top of RQ takes the edge e, the CCRP++ have ability to reserve earliest arrival path. ■

Third, we would like to know why the egress time is less in CCRP++ in most of our test case. We have seen from above proof that CCRP++ is able to reserve the earliest available route immediately. Further more, the CCRP++ gives source on top of the RQ more chance to keep reserving until its route longer than the earliest arrival path. We wonder if that makes CCRP++ more aggressive than CCRP by giving higher priority to whom take certain edge earlier. We would like to investigate if the following guess could give some hint.

Conjecture: CCRP++ is more aggressive than CCRP.

6. Conclusion

In this paper, we propose a new scalable algorithm to accelerate the evacuation planning based on the semantic of the CCRP. Because we do the local shortest path expansion from single source rather than the global expansion from all sources, and we reuse the result for future iteration, we achieve less run-time compared to CCRP. We noticed that CCRP++ is greedier to reserve the critical edges. We will further investigate if that is the reason we get less egress time in most of our test cases. Anyway, both the experiment and the theoretical analysis show the CCRP++ not only outperforms the CCRP in efficiency(run-time), but also keeps the optimality(egress time).

In the future, we would like to improve the CCRP++ in two aspects. First, although we have avoided most of the redundant shortest path expansions in each iteration, we believe the updates in different iterations can be further accelerated by adapting, e.g. [8], which can accelerate the repeated shortest path expansions from single source. Second, we would like to investigate the spatial-temporal distributing nature of the evacuation planning problem, and utilize for example, the Map-reduce distributing computing schema, to further accelerate the run-time.

Acknowledgement

The author would like to thank Dr. Sangho Kim, Dr. Farnoush Banaei-Kashani, and Prof. Cyrus Shahabi for the great discussion and encouragement during this research. The author is funded by Ph.D. Student Exchange Program from Chinese Scholarship Council [2007-3020] during the study in Infolab, Department of Computer Science, Viterbi School of Engineering, USC, and 863 High Tech Program [2007AA120502] from Chinese Technology Administration during the study in Gis4g Lab, Earth and Space Science Dept., Peking University.

References

[1] Q. Lu, B. George, and S. Shekhar. Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. *Proceedings of 9th International Symposium on Spatial and Temporal Databases (SSTD'05)*, 291-307, 2005.

[2] K., Sangho, B. George, and S. Shekhar. Evacuation Route Planning: Scalable Heuristics, *(ACMGIS '07)*, Seattle, WA, 2007

[3] B. George, K. Sangho and S. Shekhar. Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results, *(SSTD'07)*, Boston, July, 2007.

[4] H. Hamacher and S. Tjandra, Mathematical Modeling of Evacuation Problems: A State of the art. *Pedestrian and Evacuation Dynamics*, 227-266, 2002.

[5] E. M. Hooks, and S. S. Patterson, On Solving Quickest Time Problems in Time-Dependent and Dynamic Networks, *Journal of Mathematical Modeling and Algorithms*, Vol. 3 No.1. 39-71, 2005.

[6] Spatial Dataset contribute by Dr. Li Feifei (http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm)

[7] E. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* (1959) 1, pp. 269-271.

[8] S. Koenig, M. Likhachev and D. Furcy. Lifelong Planning A*. *Artificial Intelligence Journal*, 155, (1-2), 93-146, 2004.

Video Analytics for Multi-camera Traffic Surveillance [*] [†]

Dongyu Ang
University of North Texas
da0097@unt.edu

Yao Shen
University of North Texas
shenyao1016@hotmail.com

Prakash Duraisamy
University of North Texas
pd0075@unt.edu

ABSTRACT

A low-cost Video Image Detection Systems (VIDS) is introduced. Using video analytics, the system can count the number of vehicles making a left (or right) turn at an unseen intersection plus collect statistics on other traffic conditions. Other functionality can be added. Two cameras with non-overlapping views were used as the information source. Between them was situated a "T" intersection. Comparing the automated data collected from the two cameras' video with manually generated truth data, no errors were found over five minutes of video.

1. INTRODUCTION

In an increasingly automated and controlled traffic surveillance environment, situational awareness is more difficult for human operators without system assistance. A new generation of internet is approaching. All sorts of electronic devices could be connected to global high speed internet including traffic surveillance cameras creating what is coming to be called cyber-physical systems. A huge numbers of real-time surveillance real time will make it impossible for human traffic monitors to manage. Thus, a system which can monitor the traffic automatically at intersections is crucial for offloading labor from a traffic management center (TMC).

Video analytics is a solution to traffic surveillance. The notion is to provide camera-side processing to collect statistics such as volume, speed, and occupancy while being obser-vant for aberant events. Rather than depend on an operator at a TMC to continuously monitor the video, the TMC is alerted only when there is an unusual occurrence. For example, market products such as Xcam-1 from Citilog, Inc., Vantage Edge 2 from Iteris, Inc., and Autoscope Solo Tera from Econolite, Inc. are capable of detecting wrong-way vehicles, stalled vehicles, and other events. The advantage of such products is operating costs – it is not necessary to include the labor-intensive activity of continuous monitoring by an operator. Our goal is to demonstrate the feasibility of multi-camera video analytics for which the inference of events not within the field of view of any camera is possible.

We assume, in this experiment, two cameras. They must be within line of sight of each other in order to achieve low-cost wireless communication. Both cameras have on-board processing. Both are cognizant of the road network lying between them, yet, parts of that network are not visible to either camera.

In typical visual surveillance or ambient intelligence systems, event analysis is based on the tracking and identification of visual objects given multiple camera views. Objects are often captured in more than one view and it is desirable that these multiple instances of the same visual object can be automatically identified. The tracking of multiple visual objects in one view is a classic vision problem that has received much attention and finds application not only in surveillance systems but also in other machine vision scenarios, such as robotics. Multiple-view tracking has only recently received much research attention. The main advantages of using many cameras for tracking in surveillance scenarios [9, 21] are an arbitrarily large coverage of any given area. For most environments, a single camera is not able to provide adequate coverage or tracking performance. This is especially important in critical areas and where more robustness against occlusion is desirable [18]. While a single-camera tracker searches for correspondences only between frames, the additional task of a multi-camera tracker is to establish correspondences between observations of objects across cameras. The goal is to correctly tag all instances of the same visual object at any given location and at any given time instant.

Specific models can add constraints that simplify this task. Kalman filters (KF) [13] are an efficient algorithm to estimate the state of dynamical system. However, it is limited to the Gaussian and linear assumptions [4]. The Extended

[*]This study was supported in part by the Texas Department of Transportation under grant No. 0-6432

Kalman filter (EKF) was introduced to address this limitation by using non-linear functions and non-Gaussian noise models. Since the non-linear functions are applied only to the sample means iteratively, the EKF may diverge quickly if either the initial state estimation or the process model is incorrect [20]. Isard and Blake [10] proposed a more powerful particle filter (PF) tracker, which is considered to be both efficient and flexible in solving non-linear and non-Gaussian problems [1, 19].

Although, the tracking algorithms in [4, 20, 10, 1, 19] are proven both accurate and efficient, they suffer from the occlusion problem due to the utilization of just a single camera. Multiple cameras can help solve occlusion problems, although the establishment of correspondences between moving objects in different views is a difficult challenge.

The rest of this paper is organized as follows. In Section 2, related work is introduced. In Section 3 we present the methodology we used for background modeling, foreground vehicle detection , vehicle identification, trajectory estimation and verification. In section 4 we describe our experiments and give out our results. Lastly, we present the conclusions and future work in section 5.

2. RELATED WORK

An intelligent transportation system (ITS) is an application that incorporates electronic, computer, and communication technologies into vehicles and roadways for monitoring traffic conditions, reducing congestion, enhancing mobility, and so on. Many researchers have devoted themselves to investigating different solutions for traffic monitoring and the tracking of vehicles [7, 3]. Nearly all of these studies are limited to a single traffic video camera at one time. With data from multi-cameras, we can fuse the information to achieve a higher level of service for transportation networks.

In the multiple camera problem, geometric information is used to obtain robust results. Cai and Aggarwal [5], used multiple calibrated cameras for surveillance. Kettnaker and Zabih [14] used a Bayesian formulation of the problem to reconstruct the paths of objects across multiple cameras. Collins et al. [8] developed a system consisting of multiple calibrated cameras and a site model. [17] proposed an approach for tracking in cameras with overlapping FOV that does not require calibration. Khan et al. [15] used field of view (FOV) line constraints for tracking in cameras with overlapping views. Javed et al. [11] extended this approach for tracking in non-overlapping cameras. Multiple cameras are used to recover the homographic relations between camera views of the same scene from different perspectives. For example, Black et al. [2] use a combined 2D/3D Kalman filter for object tracking. Alignment-based approaches rely on recovering the geometric transformation between cameras automatically. This can be done using spatial image alignment methods and incorporating time information [6] or by matching motion trajectories in different cameras. In [16] this begins by finding the limits of the field of view of each camera that are visible in the other cameras. Also, [22] proposes a ground-based fusion method for camera handover using space-time constraints and stereo segmentation. In our case, cameras were located in locations not permitting field of view overlaps, yet nonetheless allow establishing path dependencies between them using probabilistic models [14, 12].

3. METHODOLOGY

Two cameras were used to monitor road traffic and adjacent observation areas. With these two cameras the same vehicle can be observed from different positions and angles. The objects of interest were identified from image data by foreground processing methods. Figure 1 is the workflow diagram for the surveillance system that was implemented.

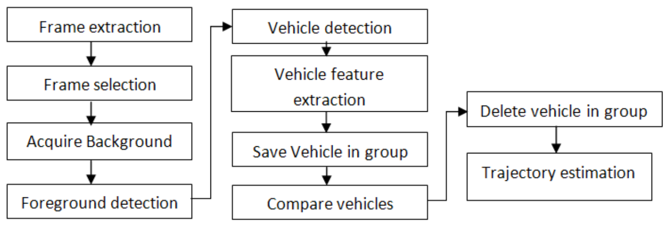


Figure 1: Flowchart for our surveillance system

3.1 Background modeling and foreground vehicle detection

Background modeling is an important aspect of video surveillance systems. It is necessary that this module detect the relevant details of the scene while excluding irrelevant clutter. It also must be computationally efficent to allow for real-time processing of video sequences. An adaptive background model was used for the entire region of awareness and for segmenting the moving objects that appear in the foreground.

Stationary cameras captured traffic video. The overall accuracy of the system depended on robust foreground object detection. However, inherent changes in the background itself is not completely stationary due to such phenomena as wavering trees and flags, water surfaces, and so on. Therefore, background modeling for traffic video surveillance needs must meet certain requirements. The foreground modeling component detects moving objects (blobs) regardless of whether they represent a vehicle or non-vehicle.

To achieve the capability of detecting moving objects in the scene, the scene must be ”learned.” The first four blocks of figure 1 are preprocessing including frame extraction, frame selection, background estimation and foreground detection. The background estimation algorithm creates a reference image which contains the background components of the scene. The background image, as noted, is fundamental in detecting moving objects and tracking them. It is used to distinguish foreground components. To avoid the worst effects of noise, a median filter was applied across five consecutive frames to obtain a good estimate of the background. The specific operation of the median filter was as follows: For each pixel, the intensities were collected for the five frames; the intensities were sorted; the median value was selected as the background pixel value. This process was repeated for each pixel to construct the final background image.

Given the background image, the foreground image was obtained by the subtracting it from the current frame. To re-

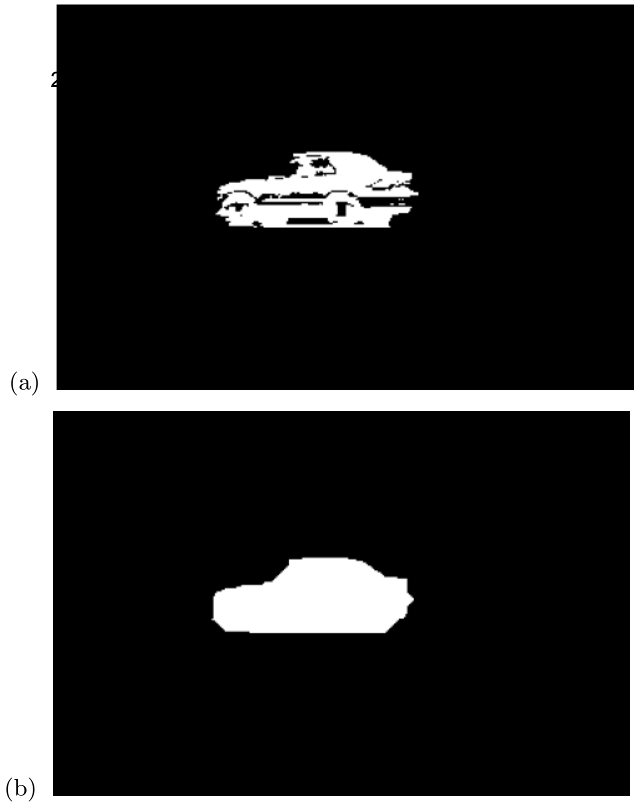


Figure 2: Foregound image(a) and foregound image(b) after morphological opening

move small objects created by noise, the morphology opening was applied to the foreground image (Figure 2). The morphology opening operator is composed of two basis operators – erosion and dilation. An erosion is capable of deleting completely small objects, so the dilatation operation which follows does not reinstate them. Thus the result of a morphology opening operator on a binary image is the original image with very small objects removed.

Once subtraction and morphological filtering is complete, the remaining blobs are the objects in the foreground. Blob features such as the centroids, the areas, and the shapes were calculated. These blob features were used to differentiate between objects and noise. Moreover, the extraction of color features of vehicles were also dependent on these features.

3.2 Feature Extraction

In order to identify the same vehicle in two scenes, feature extraction of the objects is essential. Color histograms were calculated from RGB representations. The images were coded in RGB. Six bins were populated from each channel, R,G, and B, respectively.

In order to obtain the histogram of each feature, the kernel based density $q_t(x) = \{q_t(n;x)\}_{n=1\ldots N}$ of the feature distribution was estimated by

$$q_t(n;x) = C_h \sum_{i=1}^{N} (K \left\| \frac{y-x_i}{h} \right\|^2) \delta[b(x_i) - n]$$

where x_i are the pixel locations centered at y with radius h. $b(x_i)$ denotes the bin index of x_i. δ is the Kronecker delta function, C_h is a normalization constant ensuring $\sum_{n=1}^{N} q_t(n;x) = 1$.

At time t, the feature model $q_t(x)$ obtained in one video frame was compared to the feature model in another video frame $q_o = \{q_o(n)\}_{n=1\ldots N}$ with $\sum_{n=1}^{N} q_o(n) = 1$. In our experiments, the histogram values were chosen beginning at location x, the centroid of the blob.

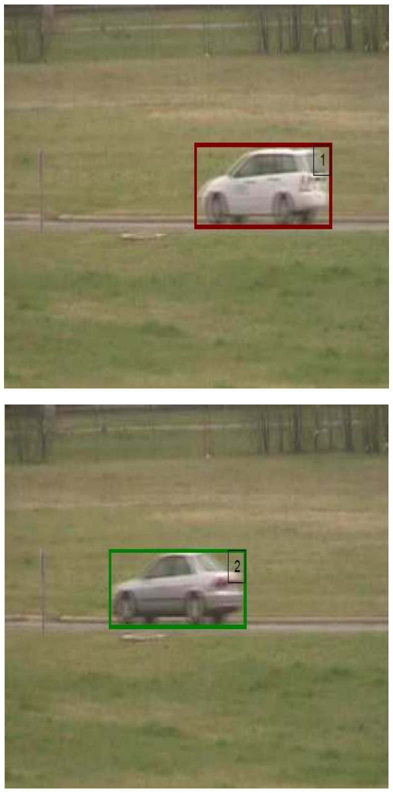


Figure 3: vehicles in camera 2

3.3 Vehicle Identification by Feature Matching

Given the real-time nature of VIDS, it was necessary to have matching criteria embedded in algorithms that were computationally fast as well as accurate. The distance between two feature distributions was defined as

$$D = \sqrt{1 - \rho[q(n;x), q_o(n)]}$$

which is called Bhattacharyya distance. where

$$\rho[q(n;x), q_o(n)] = \sum_{n=1}^{N} \sqrt{q(n;x) q_o(n)}$$

The larger ρ, the greater the similarity that exists between two distributions. For example, given two identical histograms of two images, ρ=1 would be obtained, which indicates a perfect match. Two classification criteria were employed in the study. One was a color match criterion and a second a vehicle direction with timestamp match criterion.

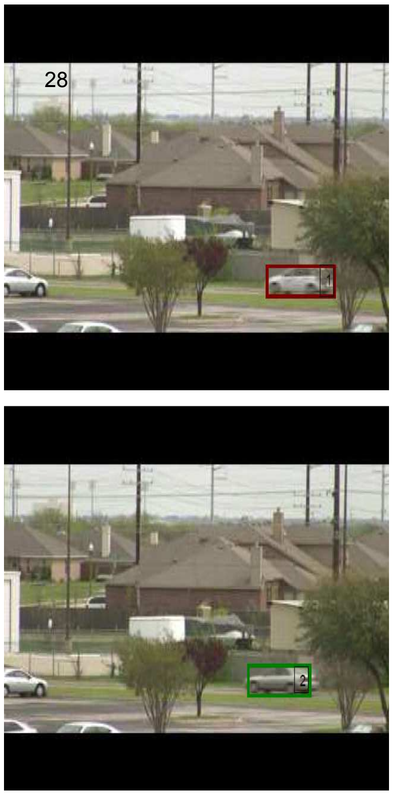


Figure 4: vehicles in camera 1

Both criteria utilize the features extracted from the vehicle blobs (Figures 3 and 4). To recapitulate, the vehicle was detected by foreground segmentation algorithm described earlier. The states of the vehicle, color, direction and timestamp, were saved in vehicle description tables, one for each of the two cameras. Vehicle features in one table were then matched with vehicle features from the other table. However, before doing so, the vehicles for which the timestamp had expired were deleted. An expired timestamp indicated that the vehicle had chosen a path not within the camera field of view. Thus, before deleting, its trajectory had to be classified as explained in the next subsection. With respect to vehicles in the two tables which pass the color match criterion before the timestamps expire, they were deleted once the trajectory path was classified.

3.4 Trajectory Estimation and Verification

The above section described how vehicles were matched between camera views in real time. The remaining task is the classification of vehicle trajectories based on matches or, in some cases, lack of a match. As mentioned in our problem definition we had two cameras at a "T" intersection, referring to Fig 5. Two tables, one corresponding to each camera, were maintained. Each table was used to store the features of vehicles that entered the field of view of the respective camera. Each time a new vehicle was stored into a table, its features had to be compared with the vehicles in the other table. A unique decision logic was developed to fuse the vehicle information from two tables. See figure 5. To reduce the search time for future events, once the vehicle's trajetory was classified, the corresponding vehicle features were deleted from the tables. Referring to figure 5, trajectories were assigned to one of six classes named "1to2", "1to3", "2to1", "2to3", "3to1" and "3to2". The overall algorithm follows:

```
loop
  every new frame
  if a vehicle's timestamp has expired then
    delete the expired vehicle out the table and classify
    the vehicle's trajectory as 1to3 or 2to3
  end if
  if new vehicle detected in new frame then
    save vehicle information in table
    if the vehicle's direction is away from the intersection
    then
      compare the vehicle with those in the other table
      if find one match then
        classify the vehicle's trajectory as 1to2 or 2to1
        and delete vehicle from both tables
      else
        classify the vehicle's trajectory as 3to1 or 3to2
        (i.e., entering via the unseen intersection)
      end if
    else
      (the vehicle's direction is toward the intersection)
      do nothing other than save the vehicle and its in-
      formation in appropriate table
    end if
  end if
end loop
```

For example, in camera 1, if a vehicle with the direction from camera 1 to camera 2 is detected, it is sufficient to search for the corrsponding feature in the event table of camera 2 within a certain time interval. If the features (both direction and color) match, the vehicle is identified and its trajectory across both camera field of views can be classified. Otherwise, a vehicle remains in the event table until its timestamp expires. When the timestamp expires, it is deleted from the event table and is assumed to have made a turn at the "T" intersection. Upon deletion, its trajectory classification is assigned.

4. EXPERIMENTS AND RESULTS

Preliminary testing was performed in order to validate the efficiency and effectiveness of the Bhattacharyya distance. Only vehicles coming from road 1 to road 2 were used in this aspect of the experiment. Color histograms of all the vehicles detected by camera 2 were stored beforehand and Bhattacharyya coefficient was computed between each vehicle detected in camera 1 and all the vehicles in camera 2, ignoring the timestamp. The experimental results (Figures 6-9) indicate there exists a threshold that all correct matches exceed and no incorrect match exceeds.

Recall that the videos were taken from cameras that faced different segments of one straight, connected road. Between the two road segments, there was a "T" intersection not in the field of view of either camera. Thus, a vehicles identified in one camera but not both must have emerged from the unseen intersection or turned into the intersection. The distinction is made by examining the direction of movement within the single view available.

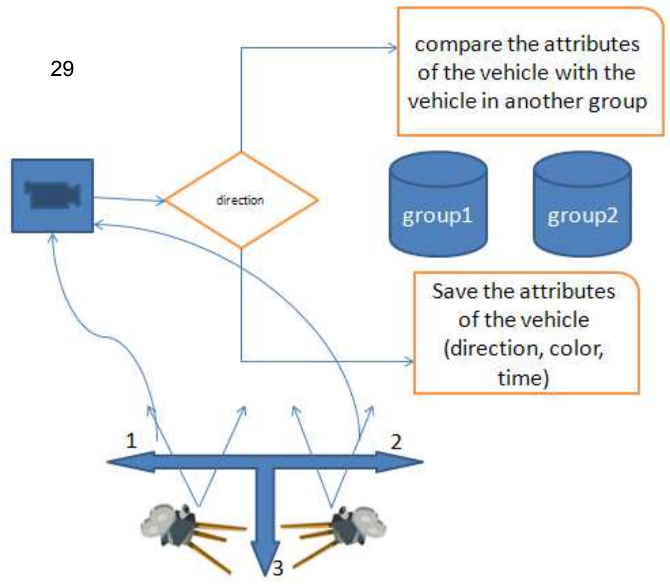

compare the attributes of the vehicle with the vehicle in another group
direction
group1
group2
Save the attributes of the vehicle (direction, color, time)
1
2
3

Figure 5: Method of identify vehicles

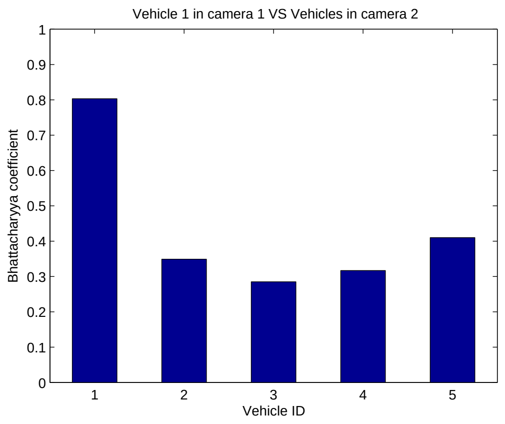


Figure 6: Vehicle 1 in camera 1 VS Vehicles in camera 2

We used over five minutes of traffic video taken at the "T" intersection for program testing. For reasons of safety, we confined our experimental observations to a low density traffic area. The total number of vehicles was 14. At the time of video capture, we manually recorded the trajectory of all vehicles to use as truth data. The 14 vehicles encountered included two buses, three pickups, one van, two SUVs and six sedans. By comparison with the manually tabulated data, the result from the program correctly classified the trajectories of all 14 vehicles.

5. CONCLUSIONS AND FUTURE WORK

We have presented the results of vehicle trajectory classification based on multi-camera views and processing that employed morphological filtering and pattern recognition methods. Using this method we have been able to effectively estimate trajectories for vehicles of various classes. More importantly, our method exhibits real-time operability, partly because frame sampling was used to discard all but five frames per second. The results are twofold: (1) identification of a set of features that correctly match vehicles as viewed from different cameras; (2) an algorithm that successfully classifies vehicle trajectories in real time. Our experiment showed that based on the features selected and on our method, the same vehicle in different views in different cameras can be identified correctly.

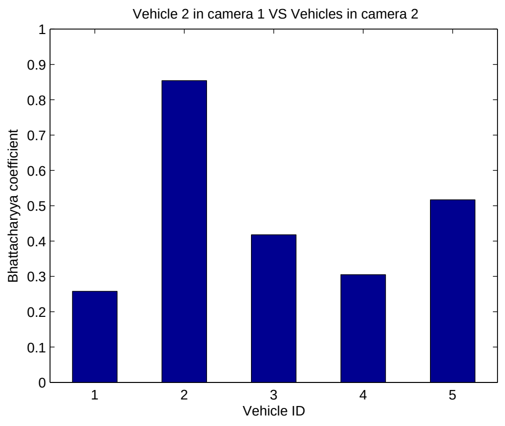


Figure 7: Vehicle 2 in camera 1 VS Vehicles in camera 2

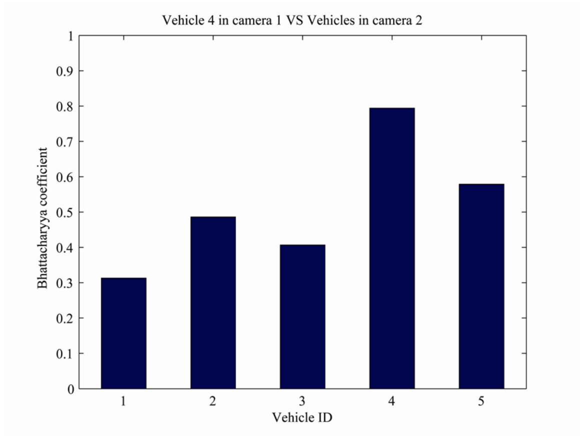


Figure 8: Vehicle 4 in camera 1 VS Vehicles in camera 2

We plan to continue development of video analytics for traffic monitoring using actual traffic surveillance video. We expect that maintaining the real-time operating mode will continue to be dependent on grabbing only five frames per second from the surveillance video. We plan to increase the functionality of our program by adding incident detection, emergency and wrong way vehicle detection, and other useful events.

6. ACKNOWLEDGEMENTS

We are heartily thankful to our supervisor, Bill Buckles, whose encouragement, guidance and support from the initial to the final version enabled us to develop an appreciation of traffic surveillance technology.

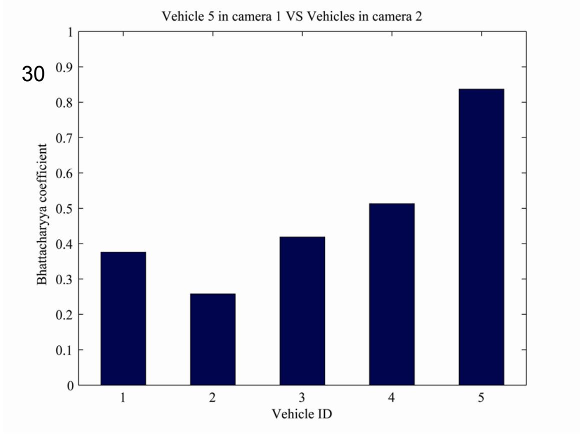


Figure 9: Vehicle 5 in camera 1 VS Vehicles in camera 2

Lastly, we offer regards and gratitude to Yan Huang, Yassine Belkhouche, and all those who supported us in any respect during the completion of this study.

7. REFERENCES

[1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian bayesian tracking. *IEEE Trans. Signal Process*, pages 174–188, 2002.

[2] J. Black, T. Ellis, and P. Rosin. Multi view image surveillance and tracking. *IEEE Workshop on Motion and Video Computing*, pages 169–174, 2002.

[3] J. E. Boyd, J. Meloche, and Y. Vardi. Statistical tracking in video traffic surveillance. In *IEEE Conf. Comput. Vis.*, pages 163–168, 1999.

[4] Y. Boykov and D. Huttenlocher. Adaptive Bayesian recognition in tracking rigid objects. In *Comp. Vis. and Pattern Rec*, pages 697–704, 2000.

[5] Q. Cai and J. Aggarwal. Tracking human motion in structured environments using a distributed camera system. *IEEE Trans. on PAMI*, 21(11):1241–1247, Nov 1999.

[6] Y. Caspi and M. Irani. A step towards sequence-to-sequence alignment. *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 682–689, 2000.

[7] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transp.Res. Part C*, 6(4):271–288, 1998.

[8] R. Collins, A. Lipton, H. Fujiyoshi, and T. Kanade. Algorithms for cooperative multisensor surveillance. In *IEEE*, 2001.

[9] G. Foresti, Micheloni, L. C. Snidaro, P. Remagnino, and T. Ellis. Active videobased surveillance system: The low-level image and video processing techniques needed for implementation. *IEEE Signal Process. Magazine 22*, 22(10):25–37, 2005.

[10] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *Int. J. of Comp. Vis*, pages 5–28, 1998.

[11] O. Javed, Z. Rasheed, O. Alatas, and M. Shah. Knightm: A real time surveillance system for multiple overlapping and non-overlapping cameras. In *ICME*, 2003.

[12] O. Javed, Z. Rasheed, K. Shafique, and M. Shah. Consistent labeling of tracked objects in multiple cameras with overlapping fields of view. *Tracking across multiple cameras with disjoint views*, 2:952–957, 2003.

[13] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME-J. Basic Eng.*, 82:35–45, 1960.

[14] V. Kettnaker and R. Zabih. Bayesian multi-camera surveillance. *IEEE Conference on Computer Vision and Pattern Recognition*, 2:252–259, 1999.

[15] S. Khan, O. Javed, Z. Rasheed, and M. Shah. Human tracking in multiple cameras. In *ICCV*, 2001.

[16] S. Khan and M. Shah. Consistent labeling of tracked objects in multiple cameras with overlapping fields of view. *IEEE Trans. on PAMI*, 25(10):1355–1360, Oct 2003.

[17] L. Lee, R. Romano, and G. Stein. Monitoring activities from multiple video streams: Establishing a common coordinate frame. *IEEE Trans. Pattern Anal. Machine Intell.*, 22(8):758–767, 2000.

[18] A. Mittal and L. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. internat. *Computer Vision*, 51(3):189–203, 2003.

[19] K. Nummiaro, E. Koller-Meier, and G. L. Van. A color-based particle filter. In *Workshop on Generative-Model-Based Vision*, June 2002.

[20] R. Rosales and S. Sclaroff. 3D trajectory recovery for tracking multiple objects and trajectory guided recognition of actions. In *Comp. Vis. and Pattern Rec*, pages 117–123, 1999.

[21] G. Wu, Y. Wu, L. Jiao, Y.-F. Wang, and E. Chang. Multi-camera spatio-temporal fusion and biased sequence-data learning for security surveillance. *ACM Internat. Conference on Multimedia,*, pages 528–538, 2006.

[22] T. Zhao, M. Aggarwal, R. Kumar, and H. Sawhney. Real-time wide area multicamera stereo tracking. *IEEE Conference on Computer Vision and Pattern Recognition*, 1:976–983, 2005.

Machine Learning Approach to Report Prioritization with an Application to Travel Time Dissemination

Piotr Szczurek Bo Xu Jie Lin Ouri Wolfson
University of Illinois at Chicago
{pszczu1,boxu, janelin ,wolfson}@uic.edu

ABSTRACT
This paper looks at the problem of data prioritization, commonly found in mobile ad-hoc networks. The proposed general solution uses a machine learning approach in order to learn the relevance value of reports, which represent sensed data. The general solution is then applied to a travel time dissemination application. Through the use of offline learning, the paper analyzes the feasibility of the proposed approach and compares the accuracy performance of several common machine learning algorithms. The results show that not all machine learning algorithms may be used for prioritization and that the use of the logistic regression algorithm is particularly suited for the problem. The learned logistic regression model is then used in a simulated VANET environment. The results of the simulations show that it is better at prioritizing reports in terms of their usefulness in aiding vehicles to choose the shortest travel time paths.

Categories and Subject Descriptors
H.4.3 [**Information Systems Applications**]: Communications Applications; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; I.5.2 [**Pattern Recognition**]: Design Methodology – *Classifier design and evaluation*;

General Terms
Algorithms

Keywords
data dissemination, VANET, data prioritization, machine learning, traffic information systems

1. INTRODUCTION
The current expansion of computing devices with wireless communication capabilities helped to motivate creation of various information dissemination applications. Many of these applications are related to communicating traffic information. Examples include systems for disseminating parking availability [3], travel speeds [11, 15], or traffic video clips [7, 8]. By disseminating the information, drivers can make better choices regarding their routes or destinations. However, due to the limitations of communication devices, it might not be possible to send all of the information. As a result, a prioritization scheme has to be developed which ranks the usefulness of the information and allows for only the most useful to be disseminated. The method for finding such a prioritization scheme is the subject of this paper.

To find a prioritization scheme for a variety of applications, this paper proposes the use of a machine learning approach. In this method, the information is assumed to be contained in reports which are disseminated over time. The receivers of such reports then use the contained information to possibly alter their behavior. By examining the characteristics of the reports including the sender information and analyzing its impact on the recipient, it can be determined which reports should be considered the most useful. A useful report is one that has an impact on the decision making process of the receiver. For example, in a travel time dissemination application, a report will be useful when it changes the path of a vehicle.

To find the usefulness of reports, this work uses machine learning algorithms. The results show that the machine learning technique achieves good accuracy and hence can be used reliably for prioritization of the reports. Additionally, simulations of a travel time dissemination application show that the learned model provides better information to vehicles than common heuristics in terms of providing better routes with lower travel times.

In the next section, some relevant work on the topic of prioritization will be discussed. The following section will describe the model used for the general machine learning approach. The method itself will then be described in the subsequent section. Later, a specific application (travel time dissemination) of the general approach will be described. This will be followed by a presentation and discussion of the results and a conclusion.

2. RELEVANT WORK
Prioritizing reports for memory (cache) management and bandwidth management in mobile wireless networks has been studied in a number of works. In [10] the rank of a report is a weighted sum of its popularity, reliability, and size. The paper does not discuss how the weights are determined. In [14] reports are ranked such that the number of replicas of each report is proportional to the square root of its access frequency. According to [5], such a distribution of replicas has the optimal replication performance in minimizing the query cost. However, for the dissemination of real-time traffic information, the access frequency is not a suitable solution because the access frequency to a newly produced report is always small but the newly

produced report is usually of most interest. Thus for traffic information we use machine learning to determine the report relevance. In [15], traffic reports are ranked using an ad-hoc formula in which the rank is in reverse proportion to the sum of the age and distance of a report. Finally, in [6][9][13] reports are ranked based on an abstract utility function which is to be defined by specific applications. Our ranking method can be viewed as an instantiation of the utility function.

3. MODEL AND PROBLEM DEFINITION

The system consists of a set of *mobile nodes*. A node is a physical entity capable of data computation, storage, and short range wireless communication. A node can also observe its environment through a *sensing device*. Examples of nodes include vehicles equipped with on-board computers and Wi-Fi. The sensing device may be a camera installed in the car, an odometer, or GPS.

At any point in time, a node may create a *report*, which contains the data derived from the sensing device. The data is formed as a fixed set of *attributes* and their values. An attribute identifies the type of the data value. An example of a report is a speed report, whose attributes are time and average speed. Other examples of reports include reports about traffic accidents or available parking spaces.

Every node carries a *reports database* of fixed size. The reports database contains reports the node has received or created over time. The reports stored in the database are communicated over time to a subset of other nodes in the network. The determination of when, how many, and which reports get communicated is controlled via a *communication protocol*. It is assumed that the communication capabilities are limited by the bandwidth and hence not all reports in the reports database may be transmitted. A reports prioritization mechanism is thus employed in order for the communication protocol to determine which reports to transmit.

Each node in the network has the ability to judge the *relevance of a report*. The relevance represents the utility a report holds when it would be sent to other nodes, given the sending node's current characteristics and the attribute values of the report. In other words, how useful would the report be to the recipient? This value is numeric and can be either Boolean or real valued. In cases where nodes can only judge whether the report was good or bad, the report's value is Boolean (0 for bad, 1 for good). For example, consider a report that represents the availability of a parking space. The report is judged by a node (vehicle) as "good" if the parking space remains available when the node reaches it. In cases where nodes assign numeric values, those will be assumed real valued in the range of 0 to 1.

The problem is thus to find a method for estimating the relevance of a report before it is sent, given the characteristics of the node holding the report and the attribute values of the report. By knowing the relevance, the communication protocol has greater information about which reports provide the most benefit to the nodes. Also, estimating the relevance can also be useful to determine which reports are kept and which are discarded by the node, given that its local database is of limited size.

In this paper we assume that the size is the same for all the reports. Thus the relevance of a report does not depend on its size. The extension to variable reports sizes is straightforward via a greedy solution to the Knapsack problem (see [1]).

4. METHOD DESCRIPTION

The general idea behind our method is to use the received reports as an input to a machine learning process. Given that nodes can make judgments regarding the relevance of a report, a supervised learning algorithm can be used with the judged relevance value as the given output. Over time, each node learns a model that can estimate the value of a report at any time.

To provide the necessary training data for the learning algorithm, each report is augmented with additional attributes related to the sender of the report. Although dependent on the actual application, the attributes in spatio-temporal environments would generally depend on time and space. By knowing these attributes, the receiving node can learn the mapping from the sender's and report's characteristics to the relevance value of a report. The receiving node, which would later resend the report, can then have a better estimate of the value to the next receiver.

There are generally two ways in which the machine learning method can be used: *online* or *offline*. In the online method, nodes continually learn the model by using the incoming reports as training examples. The communication protocol would then use the most current model to estimate the relevance value of reports. In the offline method, there are two stages. In the first stage, nodes only gather the training examples without modifying their models. Note that these training examples may be generated through simulations. After a period of time, the examples are gathered and fed to the machine learning process. After this, the learned model is used by all nodes in the network. The advantage of this method is that nodes would not have to incur the overhead of the machine learning algorithm. The disadvantage is that the learned model cannot adapt to changing situations. Nevertheless, since in the online case, nodes initially have no model, the offline method is useful for learning the model a priori, thereby providing a way of bootstrapping. Also, the offline method can be used to analyze which attributes should be used for learning and which machine learning algorithm is best for the given application. The focus of this paper is on the offline method and in particular, how it can be used in a specific, transportation related application.

The machine learning method for relevance value estimation can be applied to a variety of applications. The main constraint is that every node should be capable of evaluating the relevance of reports and that the value be based on a goal common to all nodes. This is frequently true for applications set in environments such as peer-to-peer networks, including mobile and vehicular ad-hoc networks (MANETs and VANETs). One application that has recently been studied by researchers is travel time information dissemination. The next section of the paper looks at how the machine learning approach can be used in this application.

5. APPLICATION – TRAVEL TIME DISSEMINATION

This section will discuss how the general machine learning method can be used in a particular application: dissemination of vehicle travel times on a road network. The dissemination is done by vehicles carrying computational devices with communication capability. The communication protocol, which those vehicles will use, is based on the TrafficInfo algorithm [15]. In this algorithm, a simple heuristic is used to evaluate which reports should be periodically broadcasted. In this section, the machine learning approach is applied in order to replace the heuristic with a learned model that is better at providing the relevance value of the reports. In the following sections, we define the model in which the TrafficInfo algorithm is applied and which will be used for the machine learning method.

5.1 Vehicles

The model consists of a set of vehicles. A subset of these vehicles is equipped with GPS and devices capable of computation and fixed-sized storage. Additionally, we assume the devices provide communication capabilities (i.e. 802-11b), and have transmission range of 250 meters. We will assume that each vehicle has a predetermined destination which it reaches via a shortest travel time path, according to the information it currently has in its database. When a vehicle reaches its destination, it chooses another one immediately.

5.2 Digital Map

Each vehicle holds a *digital map* used for storing information about travel times. The digital map is made up of a set of road segments. The properties of each road segment that are of interest in this paper are:

- Road segment identifier
- Coordinates of the segment endpoints
- Road type
- Travel time estimate
- List of reports used for the estimate
- Time period number

The road segment identifier uniquely determines the particular road segment in the digital map. The endpoint coordinates provide connectivity information used for shortest path calculations. Road type indicates the physical characteristics of the particular road segment. The road type could be identified as either a highway or a city street segment. The travel time estimate is the estimated time required to traverse the road segment in the given time period. This estimate is calculated as the average of travel times contained in the list of reports. The time period is a 5 minute interval between subsequent times that time in minutes is evenly divisible by 5. The time period number identifies the given period by a unique number. Time period numbers start at 0 during initial system startup and increase by 1 for each subsequent period. For example, if the first interval is 12:00pm-12:05pm; the time period number at time 12:11pm is 3 and represents the time period 12:10-12:15pm. The initial value of the time period number, when no reports have been received is -1. For the travel time estimation, the initial value is equal to the free-flow travel time.

5.3 Travel Time Measurements and Reports Database

As each equipped vehicle fully traverses a particular road segment, it uses its GPS to record the travel time. This information is then saved in a *travel time report*. This report stores the following fields:

- Report identifier
- Road segment identifier
- Travel time
- Time period number

The report identifier is unique to every created report and allows for duplicate detection. As in the digital map, the road segment identifier is used to uniquely match the report to a particular road segment. The travel time for reports is the time measured by the given vehicle's GPS. The time period number identifies the interval in which the time measurement was taken.

When a report is created, it is stored in a *reports database*. Each vehicle carries its own reports database, which can hold at most 250 reports. The reports are sorted in order, according to a value given by the *ranking function*. When it is the case that the reports database is full, upon insertion and re-ranking, the lowest ranked reports will be discarded until all reports can be stored within the given capacity.

5.4 Reports Dissemination

Vehicles exchange reports according to a peer-to-peer dissemination protocol. We will assume a particular dissemination algorithm called TrafficInfo is used by the vehicles. TrafficInfo uses a combination of flooding and periodic broadcasting to disseminate reports. Flooding is used for newly created reports, while periodic broadcasting is done for reports stored in the local database.. When a broadcast is initiated, all reports in the vehicle's reports database are ranked according to the ranking function. All top ranked reports that can fit inside the transmission message size are then broadcast. The goal is thus to find the relevance value of a report before the transmission in order to achieve the best ranking.

5.5 Travel Time Updates

Initially, all vehicles have the free-flow travel time as the estimate for every road segment. After every 5 minute interval ends, all reports in the database that have been received in that interval are used for updating the travel time. Before updating, the reports are first analyzed based on their period number. For each road segment, the period number of all reports for the segment is recorded and only the reports pertaining to the maximum period number are kept. All other reports are discarded.

The way the report is used for updating depends on the relation of its period number to the period number contained in the vehicle's digital map for the given segment. There are three cases:

1. Report's period is smaller than the digital map's. The report is thus discarded since it contains old information.
2. Report's period is greater than the digital map's. The report then replaces all previously received reports for the segment. The period in the digital map becomes the report's period.
3. The periods are equal. In this case, we will first make sure the received report is not a duplicate. If it is, it will be discarded. Otherwise, it will be added to the report list for the given road segment and the travel time estimate will be recalculated by averaging all reports' travel times in the list.

This updating is done for every report received in the last 5 minutes. At the end, the vehicle will recalculate the shortest travel time path to its destination.

5.6 Offline Learning

As described earlier, the first step in the method is to augment the reports with additional attributes. Since the data is of spatio-temporal type, the most obvious attributes to include relate to time and space. For time, the age of the report will be used, which is calculated as the difference between the current time period and the time period in which the travel time measurement was taken. To reflect the distance, the attribute used will be calculated as the free-flow travel time along the shortest path to the mid-point of the road segment specified by the report. Additionally the type of road will also be included as an attribute. The type can refer to either a highway or city street road segment.

Since the value of the road type attribute is the same for the sender and receiver, there will only be two additional attributes added to the travel time report. These attributes are:

- Age of the report at time of creation or sending
- Sender's (or creator's) free-flow travel time from vehicle's current position to the mid-point of the road segment

The age and free-flow travel time to the segment are initially filled in at the time of report's creation and updated when the report is broadcast. That way, the receiver of the report knows the condition of the sender when the report was broadcast. These report attributes will serve as the input to the machine learning process. The relevance value of a report, which will serve as the output, is determined by the receiving vehicle as either 1 or 0, depending on whether the report changed the vehicle's travel path. Additionally, any discarded report will also be labeled negatively (i.e., 0).

Although offline learning can be performed using real vehicles, the Scalable Wireless Ad hoc Network Simulator (SWANS) and STreet RAndom Waypoint (STRAW) [2,4] vehicle mobility model were used for the purposes of this paper. This simulator combines a mobile ad-hoc network communication simulator with a vehicular mobility simulator based on vehicles choosing random waypoints in the road network and using car-following theory to model individual vehicular movements.

In order to collect the necessary input/output pairs, the learning was done using *epochs*. Each epoch consisted of a single road network and a group of vehicles, each randomly placed and having a random destination. Every vehicle initially started with their digital map containing free-flow travel times for every road segment. In each epoch, a set of travel time reports is created about a single, randomly chosen, road segment. Each report in the set has its travel time set to a random number, chosen uniformly from 0 to the free-flow time for that segment. The time period of each report varies from 0 to 100. The number of reports is thus 101, with the first report having time period of 0, the next report 1, etc. The current time period is then set 100. Therefore, the ages of the created reports range from 100 to 0.

After the reports are created, each vehicle chooses a random number between 0 and 100 to serve as the period number for the given road segment in its digital map. While the travel time for that segment will still be the free-flow time, the use of the random period number will allow learning of the effects of age since the travel time updates are dependent on the relation of the report's period to the current period. Once the period is chosen for the vehicle, the 101 reports are then used to independently update the digital map of that vehicle. This means that after each update is performed, the digital map is returned to its original state.

During each update, it will be determined whether the shortest path of the vehicle would have changed as a result of the update. If so, an example labeled as 1 will be created, otherwise an example will be labeled as 0. The offline learning procedure is outlined formally below.

```
Algorithm: Offline learning for single region
Input:   R, road network within region w/ free-flow travel times
         n, number of vehicles
         e, number of epochs
Output:  L, List of labeled learning examples
For each epoch 1..e:
1.  Randomly place n vehicles on road network R
2.  Randomly choose a segment s in R
3.  Randomly choose a travel time t between 0 and free-flow
    travel time on segment s
4.  For each vehicle 1..n:
    a.  Choose a random period number p between 0 and 100
    b.  Create a travel time report about segment s with a free-
        flow travel time and period p and use it to update
        vehicle's digital map
    c.  For period number Pi: 0..100:
        i.    Create a travel time report about segment s with
              travel time t and period Pi and use it to update
              vehicle's digital map
        ii.   Recalculate the vehicle's path to destination given
              current state of digital map. If path has changed,
              set label to 1, otherwise 0.
        iii.  Create learning example consisting of age=100-p,
              distance=free-flow travel time from vehicle's
              location to segment s, roadType=(roadType of s),
              label
        iv.   Restore previous digital map state.
```

6. EVALUTION

The purpose of the evaluation is to establish the feasibility of the offline learning method. The next section describes the tests done to determine the model accuracy obtained by various machine learning algorithms. The following section uses one of the learned models to evaluate how it performs in terms of vehicle route choice.

6.1 Machine Learning Accuracy

The Weka learning toolkit [12] was used for evaluation of various machine learning algorithms. The training data consisted of a set of learning examples gathered using the offline learning algorithms for two regions from the city of Chicago.

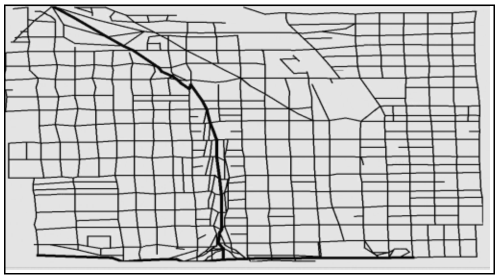


Figure 1. Region 1 road network.

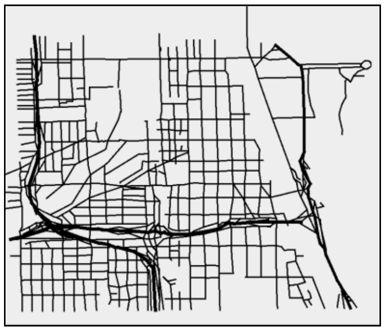


Figure 2. Region 2 road network.

The road networks of these two regions are shown in figures 1 and 2, respectively. Region 1 is approximately 2.76 sq. mi. of northwest Chicago, while region 2 is about 6.75 sq. mi. from Chicago's south side.

For each region, the offline learning algorithm was used with 25 epochs. 100 vehicles were used for region 1 and 250 for region 2. The examples output from both regions were combined into a single data set. Since the number of negative examples far outnumbered the positive ones, the SpreadSubsample Weka routine was used to downsample the negative examples. The result was a data set with 7677 positive and 7677 negative examples. This data set was then input to the following machine learning algorithms:

- Naïve Bayesian (NaiveBayes Weka implementation)
- Logistic Regression (using Logistic Weka implementation)
- Support Vector Machines (using SMO Weka implementation, w/ buildLogisticModels enabled)
- Artificial Neural Network (using Multilayer Perceptron Weka implementation)
- Decision Tree (using J48 Weka implementation)

Each of the algorithms was used with the default parameters used by Weka, with exceptions as stated above. The testing of learning algorithms was performed using a 10-fold cross validation method.

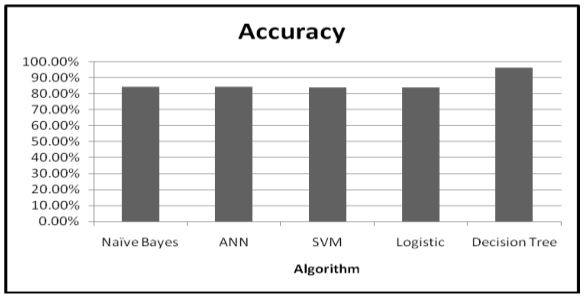


Figure 3. Accuracy of various machine learning algorithms.

Figure 3 shows the accuracy that resulted from using each of the 5 algorithms. All algorithms, with exception of decision trees, performed similarly with an accuracy of approximiatelly 83-84%. The decision trees had the best accuracy of 96.22%. It should be noted though, that while on one hand, the decision trees achieved extremely high accuracy, the resulting tree is very complex and not useful for report prioritization. The reason is that most of the leafs in the tree contained only one class of examples, which meant that almost every report would have a revelance value of 0 or 1. This is counterintuitive, given that two reports with different values of age, distance, or road type should be given a different probability of changing a vehicle's path. On the other hand, the other machine learning algorithms performed much worse than decision trees, but were able to capture the intuition behind the attribute relationships to the probability of changing a vehicle's path. The most understandable of these algorithms was the logistic regression model. It assumes that the relevance value of a report is a linear combination of the attribute values and finds the weights of the attributes that best fits the data. The model resulting from the used dataset was the following:

U = -0.0322*age - 0.02*distance + 0.3885*[road=highway] – 0.3885*[road=city street] + 4.9053

As can be seen, this model predicts that an increase in age will result in a decrease in the probability of changing path. This follows the intuition, since the travel time update policy discards many old reports. Similarly, the distance also varies inversely with relevance value. As could have been expected, the model also predicts more changes with highway segments than city streets. The approximately 80% accuracy achieved by the logistic regression method, coupled with the easy understanding of the resulting model, makes it the most promising in the use for prioritization.

Since the makeup of the road network might change the weights found in logistic regression, separate models for the two regions were also build to determine how the model would be affected. The results showed that both age and distance attributes did not vary much from the combined model. The age weight was determined to be 0.0313 for region 1 and 0.0336 for region 2. The distance weights were 0.0186 for region 1 and 0.0222 for region 2. The road type weight did significantly change between the two regions, with 0.7213 for region 1 and 0.0391 for region 2. This indicates a strong relationship between the road network and the road type of the segment on the probability of a report changing a vehicle's path. The constant factor between the two regions was relatively the same with -4.7806 for region 1 and -5.1552 for region 2.

6.2 Use of Learned Model in Prioritization

In order to evaluate the usefulness of the learned model, the SWANS/STRAW simulator was used to generate several scenarios in which vehicles disseminate travel time reports using the TrafficInfo algorithm. 100 vehicles were used for each scenario. The vehicles traveled along the road network constrained by region 1. In each scenario, vehicles were randomly placed and traveled to random destinations for 1 hour. After reaching a destination, the vehicle would choose another one immediately. The path to each destination was recomputed every 5 minute period according to the shortest travel time given the state of the vehicle's current digital map.

The highlight the differences in the various prioritization schemes, the TrafficInfo protocol was modified to disseminated only top 10 ranked reports. Additionally, the speed limits on most highway segments were lowered to 3 km/h in order to simulate a highly congested highway.

Two metrics were used for evaluation. The first is the average trip time, which measured the time it took to reach a destination averaged across all vehicles and destinations. This metric shows how it the prioritization scheme affects the travel time for an average vehicle. The second is the total travel time difference.

This metric was calculated by summing the absolute values of the differences between the travel time along the shortest path according to vehicle's current information and the path according to full information. Full information is defined as a digital map which was updated with every report ever created, as soon as it was created. The summation was performed every time a vehicle recalculated its path. This metric shows the level of information fidelity the prioritization scheme is able to achieve.

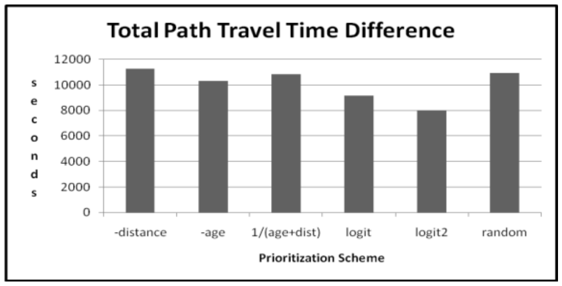


Figure 4. Total path travel time difference achieved by various prioritization schemes.

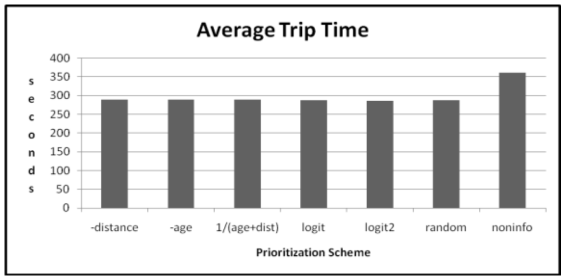


Figure 5. Average trip time using various prioritization schemes.

Figures 4 and 5 show the results of the simulations. The comparison was done using several heuristic methods and the learned models. The –distance and –age heuristics rank the reports inversely proportional to distance and age, respectively. 1/(age+dist) was the original heuristic used by TrafficInfo. The logit is the model learned using logistic regression using two regions. Logit2 is the model learned using only region 1. Lastly, the results were also compared to that of a randomized prioritization and a case, were no dissemination is performed (noninfo). The results show that the logit models outperformed all other methods in both metrics. In the total path travel time difference metric, the logit model outperformed original TrafficInfo heuristic by over 15%. The impact on the average trip time was small, with only 2 seconds difference. It should also be noted that the model (logit2) learned by using examples from the same region as was used in the simulations yielded much better performance in both metrics than the model learned from the combined data set (logit). This once again indicates a strong relationship of the road network to report's priority.

7. CONCLUSION

This paper proposed a machine learning approach to report prioritization for use in peer-to-peer environments. The method uses incoming reports in order to provide input to supervised machine learning algorithms. The learned model can then be used by all nodes in order to rank the reports to be disseminated. Through simulations, the paper was able to show the feasibility of using the machine learning method in a travel time dissemination application by achieving an accurate prediction model for the reports. Additional simulations showed that the learned model outperformed heuristics in terms of disseminating the information most likely to affect the vehicle's path.

8. ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation IGERT program under Grant DGE-0549489. It was also supported by National Science Foundation grants IIS-0847680, 0513736, 0326284, and 0957394.

9. REFERENCES

[1] C. Aggarwal, J. Wolf, P. Yu. Caching on the World Wide Web. TKDE, 11(1), pp. 94-107, 1999.

[2] R. Barr. An efficient, unifying approach to simulation using virtual machines. PhD thesis, Cornell University, 2004.

[3] M. Caliskan, D. Graupner, and M. Mauve. Decentralized discovery of free parking places. *VANET*, 2006.

[4] D. R. Choffnes and F. E. Bustamante. An Integrated Mobility and Traffic Model for Vehicular Wireless Networks. *VANET*, 2005.

[5] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. *SIGCOMM*, 2002.

[6] A. Datta, et. al. Autonomous Gossiping: A self-organizing epidemic alg. for selective information dissemination in wireless mobile ad-hoc networks. ICSNW'04.

[7] M. Guo, M. Ammar, E. Zegura. V3: A vehicle-to-vehicle live video streaming architecture. *PerCom* 2005.

[8] U. Lee, et al. Dissemination and Harvesting of Urban Data Using Vehicular Sensing Platforms. IEEE Transactions on Vehicular Technology, (58)2, 2009, pp. 882-901.

[9] F. Perich, et al. On Data Management in Pervasive Computing Environments. TKDE, 16(5), 2004.

[10] F. Sailhan and V. Issarny. Energy-aware web caching for mobile terminals. *ICDCSW'02*.

[11] L. Wischhof, A. Ebner, H. Rohling, M. Lott, and R. Halfmann. SOTIS – a self-organizing traffic information system. *IEEE Vehicular Technology Conference*, 2003.

[12] I. H. Witten and E. Frank. Data Mining: Practical machine learning tools and techniques, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[13] Y. Zhang, et al. ICEDB: Intermittently-Connected Continuous Query Processing. ICDE 2007.

[14] Y. Zhang, J. Zhao and G. Cao. Roadcast: A Popularity Aware Content Sharing Scheme in VANETs. *ICDCS*, 2009..

[15] T. Zhong, B. Xu, P. Szczurek, O. Wolfson. Trafficinfo: An Algorithm for VANET Dissemination of Real-Time Traffic Information. 5th World congress on ITS, 2008.

On the Performance of Adaptive Traffic Signal Control

Chen Cai[1] Bernhard Hengst Getian Ye Enyang Huang Yang Wang
Carlos Aydos Glenn Geers

Neville Roach Laboratory, National ICT Australia
Department of Computer Science and Engineering, University of New South Wales

ABSTRACT
In this paper, we present a study in understanding sensing error's impact on traffic signal control performance. Adaptive traffic signal control systems depend on information from traffic sensors to interpret the state of traffic. Signal timings are adjusted at real time according to the state of traffic. Queue length is an important element of the state of traffic, and errors in estimating queue length influences control decision and hence the performance. This paper presents the first attempt to quantify the effects of sensing error on control performance in the field of traffic control. A novel technique to estimate queue length using data from single loop detector is presented, and estimations are compared with parallel observations. The results show that moderate overestimation of queue length may significantly improve control performance. The benefit from overestimation suggests including arriving traffic in system state, and using look-ahead algorithms to calculate signal timings.

Categories and Subject Descriptors
I.2.1 [**Simulation and modeling**]: *model validation and analysis*

General Terms
Algorithms, Measurement, Performance.

Keywords
Traffic Signal, Sensor Error, Queue Estimation.

1. INTRODUCTION
In this paper we present a simple model based traffic signal controller that switches cycle time and splits based on the perceived queue lengths on the approaches. In reality queue lengths are hard to determine. We develop a novel Bayesian technique based on a single upstream loop detector in order to obtain a probabilistic estimate of the queue length. Our objective is to understand the impact of estimation error on control performance.

2. TRAFFIC SIGNAL CONTROL SYSTEMS
Conventional traffic signal control systems calculate fixed cycle time and green time splits for conflicting signal groups. While satisfying the overall demand structure at the signalized intersection, fixed-time systems inadequately accommodate systematic or random variations in traffic demand. Adaptive control systems, on the other hand, adjust to changes in traffic demand by calculating signal timings in real time.

The state-of-the-art for adaptive signal control usually involves state-space representation of the control system and sequential decision-making in real time. The control objectives are commonly set to optimize some measures of generalized control performance over a time period, whilst accommodating both systematic and random variations. The quantities to be calculated are the sequences of signal changes to be invoked and the associated timings.

Central to the state-space presentation are the techniques that use sensing information to construct the system state. The state of a traffic signal control system is defined in [1] as a composite of two elements: the state of traffic and the state of controller. The state of traffic at a junction can be specified by the number of vehicles queuing in each of the links and the arrivals of vehicles in the near future. The former of these is influenced by the signal controls applied. The state of the controller can be specified by the signals that are green, any changes that are currently underway, the times at which they will be completed and the times of expiry of any minimum or maximum permitted durations.

Since the state of the controller is easily accessible, the main challenge in constructing the system state is to achieve accurate estimation of queue length. The difficulty in this is that loop detectors are usually the only source of traffic information, and the norm is that only one loop detector may be available for a traffic lane. The loop detector can be installed upstream of the stopline or at the stopline. For the upstream detector, if the queue spills beyond the location of the detector, no more arriving vehicle can be detected until the queue length is less than the

[1] Corresponding author for the paper:
chen.cai@nicta.com.au, +61 (0)2 8306 0421

distance between the stopline and the detector position. This poses a difficult problem for updating the traffic state.

Literature addressing queue length estimation in real time with the possibility of spilling back is limited. A deterministic model using shockwave theories [2, 3] was proposed by Liu *et al.* [4]. An important assumption for this approach is that the progression of shockwaves in traffic flow can be detected by high-resolution detectors.

In this study, we present a novel technique called *the Q_tracker* that uses vehicle counts and vehicle speed from a single loop detector to estimate queue length.

3. THE Q TRACKER

The Q_tracker is a technique that estimates queue length in real time. It provides the probability distribution of queue length at a given time. The mean value of the distribution is used to construct the traffic state. A traffic controller implements the control policy according to the traffic state and the controller state.

3.1 Queue Length Estimation

For queue length estimation, the measurement obtained from inductive loop detector is denoted by o_t at each time instant t. In this work, o_t is the velocity measured by the loop detector. Given the measurements over time, the posterior distribution of queue length q_t can be iteratively updated at each time instant t. Using Bayes' rule, we have

$$\begin{aligned} p(q_{t+1} \mid o_{1:t+1}) &\propto p(q_{t+1} \mid o_{1:t}) p(o_{t+1} \mid q_{t+1}) \\ &= p(o_{t+1} \mid q_{t+1}) \sum_{q_t} p(q_{t+1} \mid q_t) p(q_t \mid o_{1:t}), \end{aligned} \tag{1}$$

where $o_{1:t} = \{o_i \mid i = 1, 2, ..., t\}$ is the history of measurements up to time instant t.

The transition probabilities of the queue length are assumed as the following:

$$p(q_{t+1} \mid q_t) = \begin{cases} \alpha, \text{ if } q_{t+1} = q_t \pm 1 \\ 1 - 2\alpha, \text{ if } q_{t+1} = q_t, 0 < q_t < q_{\max} \\ 1 - \alpha, \text{ if } q_{t+1} = q_t, q_t \in \{0, q_{\max}\} \\ 0, \text{ otherwise} \end{cases}, \tag{2}$$

where $q_{\max}$ is the maximum number of vehicles in the queue, and parameter α the transition probability factor. Considering vehicle inflow f_t^{in} and outflow f_t^{out}, the actual transition probability $p(q_{t+1} \mid q_t)$ is then shifted by $f_t^{out} - f_t^{in}$ toward the original point for q_{t+1}. Since f_t^{in} and f_t^{out} usually are not integers, linear interpolation is employed during the shift of the transition probability.

The queue length from the stop line to the loop detector is denoted by q_L. In the case where $q_t < q_L$, the observation likelihood $p(o_t \mid q_t)$ is approximated by a Gaussian distribution $N(o_t; \mu, \sigma^2)$. For a vehicle passing through the loop detector with constant deceleration, we have $\mu^2 \propto q_L - q_t$, where μ denotes the mean velocity. Hence we have

$$\mu = \alpha\sqrt{q_L - q_t}, \tag{3}$$

where parameter α adjusts the mean velocity. Similarly, it is assumed that $\sigma^2 \propto q_L - q_t$. Additionally, there exist two factors influencing the velocity distribution. The first factor β_1 concerns that even when the queue length is beyond the loop detector, vehicles in the queue may keep moving. The second factor β_2 concerns the bias in measuring vehicle velocity. This is because mean vehicle length rather than actual vehicle length is used for velocity estimation. Regarding these two factors, we have

$$\sigma^2 = \beta_1(q_L - q_t) + \beta_2 . \tag{4}$$

We further define the posterior distribution of the velocity as

$$p(o_t \mid q_t) \propto \begin{cases} N(o_t; \alpha\sqrt{q_L - q_t}, \beta_1(q_L - q_t) + \beta_2), \text{ if } q_t < q_L \\ N(o_t; 0, \beta_2), \text{ if } q_t \geq q_L \end{cases}, \tag{5}$$

where $0 \leq o_t \leq v_{\max}$, and $\int_0^{v_{\max}} p(o \mid q_t)\, do = 1$.

3.2 Vehicle Velocity Measurement

The mean vehicle velocity estimated from the loop detector signal is used as the observation or measurement at each time instant. A method for estimating mean velocity from the loop detector signal over time is presented in this work. To estimate the vehicle velocity, the relationship between vehicle count C, time occupancy O, vehicle velocity $\{v_i\}$, and vehicle length l can be described as:

$$O = \frac{1}{T} \sum_{i=1}^{C} \frac{l_i + l_{loop}}{v_i}, \tag{6}$$

where l_{loop} is the length of the loop detector, and T is the duration of the measurement. For the update process of the queue length tracking, it is assumed that velocity v_{mean} is constant within interval T and the mean vehicle length l_{mean} is computed from historical data Hence we have

$$O \approx \frac{C(l_{mean} + l_{loop})}{v_{mean} \cdot T}. \tag{7}$$

The measured mean velocity can be written as

$$o_t = v_{mean} = \frac{C(l_{mean} + l_{loop})}{O \cdot T}. \tag{8}$$

3.3 Inflow and Outflow

Inflow rate is defined as the average number of vehicles that have passed through a loop detector during a period of time. Inflow rate estimation is related to vehicle count as well as the detection of long queue. Vehicle count is the number of vehicles that have passed through a loop detector on a road and it can be obtained by counting the number of falling edges of the loop signal. If the length of a queue is longer than q_L, then the queue is beyond the loop detector and is considered to be a long queue. When a long queue occurs, the estimate of inflow rate is updated by using the historical information, e.g. the inflow rate calculated one or two days ago. Kalman filter has been employed to improve the robustness of the inflow rate estimation.

The outflow rate estimation is to estimate the flow rate at which vehicles enter an intersection from an approach. When the traffic light is red, the outflow rate becomes zero. Otherwise the outflow rate is simply approximated by a constant value in this work.

4. TRAFFIC SIGNAL CONTROL POLICY

A simple heuristic control policy is employed for this study. This policy is derived from exploiting the apparent optimality of *saturation flow algorithm* [5], in which the signal changes only when the favored queue is exhausted. The saturation flow algorithm was further studied in [6], where it was treated as the *basis policy* and was compared to two other *comparison policies*. One of the comparison policies considers switching traffic signal before the favored queue is empty. The other allows extending green after the favored queue is exhausted. D*ynamic programming* [7] was used to show that there were domains in the state space where the basis policy was optimal, and that there were other domains in which the comparison policies were optimal. Recent studies in adaptive traffic signal controlling using advanced dynamic programming techniques [8, 9] suggested that considering information of arriving traffic as an extra dimension of state space improves performance of the basis policy. In this study, we simply use detected vehicle headway as an indicator of the arriving traffic. The basis policy is often impractical in reality since it may extend green for too long. Regarding this, we add a few conventional constraints to traffic signal timings, including minimum green, maximum red, and maximum cycle length. An inter-green period is further added to ensure safety at the intersection. The control policy can be summarized as the following.

Decision (a). Switching signal if all the following conditions are satisfied: 1) favored queues are exhausted, 2) vehicle headways of the concerned approaches excess critical value, and 3) minimum green is exhausted. Decision (a) is overridden if decision (b) is applicable.

Decision (b). Switching signal if maximum red is exhausted or if maximum cycle time is exhausted.

5. NUMERICAL EXPERIMENT

In numerical experiments Q_tracker and actual observation of queue length are in turn employed to present the traffic state. Depending on the traffic state, the control policy supervises switching of the traffic signals. Performance results thus obtained are analyzed to show the impact of sensing error in control performance. We use the PARAMICS simulation software [10] for numerical experiments. A specific plug-in was supplied so that actual queue length can be observed parallel to the simulation. The configuration of the traffic intersection in PARAMICS is provided in Section 5.1. The definition of queue for the PARAMICS plug-in is introduced in Section 5.2. Analysis of numerical results is included in Section 5.3.

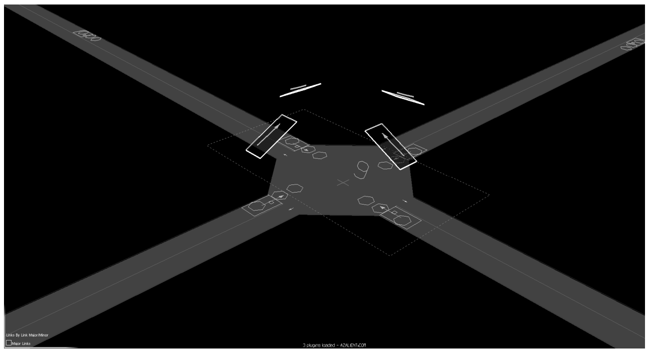


Fig. 1 A simulated traffic intersection in PARAMICS model

5.1 PARAMICS Model

We present a very simple PARAMICS network for this study. It contains one signalized intersection, consisting of two perpendicular roads. Each road is a two-way road with a single lane in each direction and a speed limit of 60 km/h. The intersection has four signal groups that can be independently controlled by the user. Each of the four signal groups controls one of the four approaches. The intersection does not allow right or left turns, therefore vehicles can only proceed straight ahead. This model is graphically shown in Figure 1.

Each of the four approaches to the intersection is 1100 metres long. The first 100 metres at the far ends of the network are zones or vehicle source areas. Vehicles loaded are only from a single vehicle type defined as a car of length 4.40 metres and mass of 1370 kg and the traffic volume loaded into the network can be configured by the user. The volumes of traffic demand are summarized in Table 1.

There are advance and stop line loop detectors in each approach. Only the advanced loop detector is used for estimating queue length. Each loop is 4.5 metres long. The advance loop detectors upstream edges are located 80 metres before the stop line. The stop line detectors' downstream edge coincides with the stop line position.

Table 1. Traffic demand volumes for the simulated intersection (vehicles per hour)

	Origins of traffic				
	West	East	South	North	Sum
West		600			600
East	600				600
South				400	400
North			400		400
Sum	600	600	400	400	2000

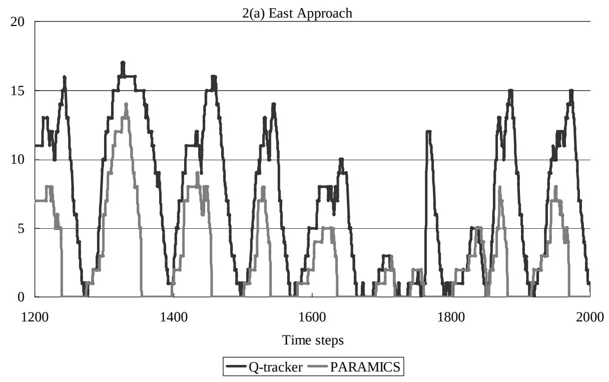
5.2 Definition of Queue

For the purposes of this paper, we require a definition of the queue length of vehicles for each individual lane of the intersection approaches. The queue length is the length between the lane stop line and the last stationary vehicle on that lane. A stationary vehicle is defined by two hysteresis thresholds of 5 km/h and 15 km/h. A vehicle joins a queue if its velocity falls below 5km/h, and leaves the queue if its velocity goes above 15 km/h. Flow conservation applies to the formulations of the queue. This definition of queue is written as a plug-in to the PARAMICS model, and runs parallel to the simulation process.

5.3 Results and Analysis

We first look at the accuracy of the Q_tracker in estimating queue length by comparing with the observation obtained from the PARAMICS plug-in, and then discuss the results in control performance.

5.3.1 Accuracy in estimation



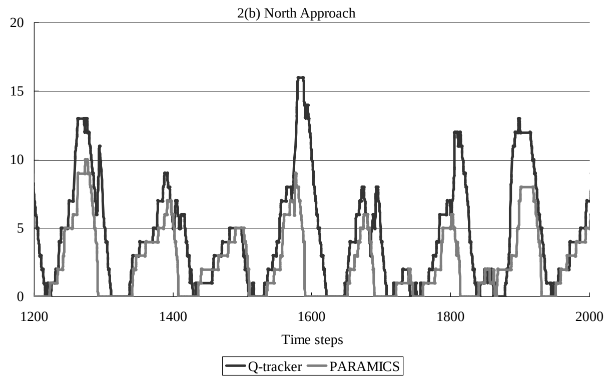


Fig. 2 Estimated and observed queue lengths between t = 1200: 2000; estimated queue from Q_tracker is plotted in dark (black) line, and observed queue in lighter (orange) line; (a) shows queue of the East approach and (b) of the

The results presented here were obtained from a single run of simulation, with one hour of simulated time and a time step of 1.0s. We illustrate the estimated queue and the observed queue in the East and North approaches in Figure.2(a) and 2(b) respectively. The Q_tracker estimation is marked by the dark (black) line and the observations by the lighter (orange) line. The dynamics of the queues are broadly similar, but the Q_tracker seems to be consistently overestimating the queue length. Overestimation results in longer time to exhaust the queue, and thus proivdes longer green time for the favored signal group, according to the control policy we stipulated. A visual comparison between Q_tracker and observation is shown in Figure 3.

Common methods for measuring estimation accuracy in time series are *mean absolute percentage error* (MAPE), which is calculated as

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Observation - Estimation}{Observation}\right| \times 100\%\ ,$$

and *mean absolute error* (MAE), which is calculated as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|Observation - Estimation\right|\ .$$

The MAPE method is more informative than the MAE, but has drawback in case with observations of zero value, which is true in our case (an empty queue is common). We therefore use the MAE, and the results are shown in Table 2. The error in estimation is greater for approaches with heavier demand. Given the consistent overestimation, the results suggest that, by using Q_tracker, approaches with heavier demand will usually be given more green time than in the case of using observation.

Table 2. Mean absolute error (in vehicles) as an indicator of accuracy in queue length estimation; Q_tracker as the estimation tool and PARAMICS plug-in as the source of observation

West	East	South	North
2.45	3.09	1.69	2.08

5.3.2 Performance in control

We obtained 10 paired results for the comparison in control performance. Each paired run of simulation is fed with independent seed with 1 hour simulated time and a time step of 1.0s. The measure for control performance is vehicle delay, which is indicated by the average vehicle seconds per seconds. Results are summarized in Table 3.

Despite the error in estimating queue length, the controller using Q_tracker seems better in performance. Using the hypothesis of equal mean, and by using the *paired t-test*, we found that t = -17.84 while the two tail *t-critical* = 2.26, thus rejecting the hypothesis. The controller using Q_tracker performs significantly better than using observation — reducing average vehicle delay by 23%.

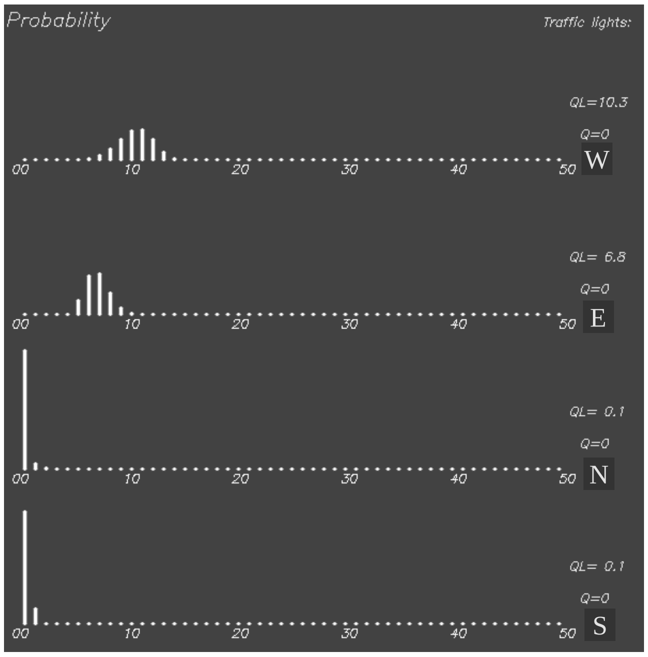
This combination of persisting bias in estimation and better performance presents an interesting scenario. An explanation for this is that delay is more sensitive to cycle time shortening than extension. Using Webster's [11] delay formula, we can find that the rate of delay increases rapidly for values of the cycle time smaller than the optimum, but less rapidly for values larger than the optimum. This means that it is better to err on the safe side by stretching the cycle length a little longer than the computed optimum, so does the controller equipped with Q_tracker and the basis policy.

The numerical results also suggest that considering arriving traffic as an extra dimension in state space is necessary and improves control performance as this may lead to an appropriate extension or shortening in green time. This requires the system to look ahead from the current state, and achieve optimality in the immediate term as well as in long-term. Dynamic programming techniques are ideal to address such problems. Reinforcement learning techniques may also be incorporated to progressively improve control policies.

Table 3. Control performance comparison between using Q_tracker and using observation

Run	Q_tracker	Observation
1	18.20	23.73
2	17.75	22.94
3	17.89	25.06
4	19.34	24.17
5	18.61	26.16
6	18.18	24.09
7	19.59	24.88
8	20.25	24.48
9	18.66	24.34
10	19.54	24.87

3(a) Estimated distribution of queue from Q_tracker at a particular time; there are 10.3 vehicles (mean value) in the queue in the West approach, and 8.8 vehicles (mean value) in the East approach



3(b) Observed queue length in PARAMICS model; 7 vehicles are queued in the West approach and 5 in the East approach

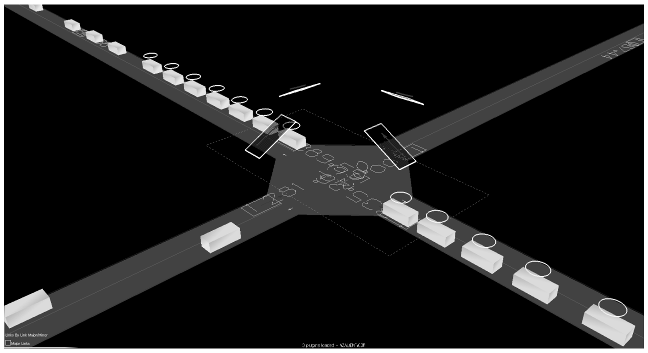


Fig. 3 A graphical illustration of queue estimation and observation

6. CONCLUSIONS

In this paper we have presented a simple model based traffic signal controller and compared its performance using two different traffic models: the notionally exact queue length returned by the Paramics plug-in and the approximate queue length returned by the Q tracker, with the latter giving a 23% improvement in vehicle delay when compared with the former.

By consistently overestimating the queue length the Q tracker based controller switches to green earlier than the real queue length based controller and gives longer green time which provides the bulk of the performance improvement. It will also ‘gap-off’ earlier providing additional performance improvement.

From this work it appears as if there is an optimal queue length for which phase switching should be triggered. It seems likely that this optimal queue length would be a function of traffic flow and could form part of the control parameters in a future study.

Accuracy of queue estimation can be further improved. The variance of estimation can be taken into account in the control policy. Results in this study suggests that the greater the variance, the longer the green extension to the favored queue.

7. ACKNOWLEDGMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

8. REFERENCES

[1] Bell, M.G.H, Cowell, M.P.H, Heydecker, B.G., 1990. Traffic-responsive signal control at isolated junctions. In: Yagar, S., Rowe, E. (Eds.), Traffic Control Methods. New York: Engineering Foundation, 273-294.

[2] Lighthill, M.J., and Whitham, J.B., 1955. On kinematic waves. I. Flow movement in long rivers. II. A theory of traffic flow on long crowded road. *Proceedings of the Royal Society A229*, 281-345.

[3] Richards, P.I., 1956. Shockwaves on the highway. *Operation Research*, **4**, 42-51.

[4] Liu, X., Wu, X., Ma, W., Hu, H. 2009. Real-time queue length estimation for congested signalized intersections, *Trans. Res. Part C,* **17**(4), 412-427.

[5] Dunne, M.C. and Potts, R.B. 1964. Algorithm for traffic control, *Operations Res.*, **12**, 870-881.

[6] Grafton, R.B. and Newell, G.F. 1967. Optimal policies for the control of an undersaturated intersection, *Proc. 3rd Intern. Symp. On Traffic Theory*, edited by Edie, L.C., Herman, R., Rothery, R.W., New York: Elsevier, 239-257.

[7] Bellman, R., 1957. Dynamic programming, Princeton: Princeton University Press.

[8] Heydecker, B.G., Cai, C., Wong, C.K. 2007. Adaptive dynamic control for road traffic signals. *Proceedings of 2007 IEEE International Conference on Networking, Sensing and Control*, London, United Kingdom, 193-198.

[9] Cai, C., Wong, C.K., Heydecker, B.G. 2009. Adaptive traffic signal control using approximate dynamic programming, *Transportation Research Part C*: *Special Issue on AI in Transport Analysis*,**17**(5), 456-474.

[10] Smith, M., Duncan, G., Druitt, S. 1995. PARAMICS: microscopic traffic simulation for congestion management, *IEE Colloquium on Dynamic Control of Strategic Inter-Urban Road Networks*, 8/1-8/3.

[11] Webster, F.V. 1957. Traffic signal settings, *Road Research Technical Paper*, No.39, Road Research Laboratory, London.

Trajectory Pattern Analysis for Urban Traffic

Fosca Giannotti[1] Mirco Nanni[1] Dino Pedreschi[2] Fabio Pinelli[1]
Pisa KDD Laboratory
[1]ISTI - CNR, Pisa, Italy
[2]Computer Science Dep., University of Pisa, Italy

ABSTRACT

The increasing pervasiveness of location-acquisition technologies (GPS, GSM networks, etc.) is leading to the collection of large spatio-temporal datasets and to the opportunity of discovering usable knowledge about movement behaviour, which fosters novel applications and services. In this paper, we apply a trajectory pattern extraction framework, called T-Pattern, to a real-world dataset, describing mobility of citizens within an urban area. The mining tool adopted is able to provide useful insights both in terms of common movements followed in the city, and, as by-product of the mining engine, in terms of spatial distribution and temporal evolution of the traffic density. Both kinds of results are provided in the paper in a visual form, aimed at helping the analyst to better interpret them and link them to his/her existing background knowledge of the domain.

1. INTRODUCTION

The large amounts of movement data collected by means of current tracking technologies such as GPS and RFID call for scalable software techniques helping to extract valuable information from these masses. This has stimulated active research in the fields of geographic information science, databases, and data mining. A general framework for analysis of movement data where database operations and computational methods are combined with interactive visual techniques is suggested in [1]. This paper works towards this direction, providing a case study of movement data analysis where simple statistical means and pattern mining methods are merged together with visualization to improve the understanding of the data.

This work is based on a set of tools that implement the T-pattern mining paradigm introduced in [4]. In such work, a new form of spatio-temporal pattern is studied, that succinctly show the cumulative behaviour of a population of moving objects, a kind of information that provides a useful abstraction to understand mobility-related phenomena, particularly indicated for domains such as sustainable mobility and traffic management in metropolitan areas, where the discovery of traffic flows among sequences of different places in a town (origin-destination flows) is a key issue [2].

The paper is organized as follows. First, Section 2 provides some indications of existing related work in the area of spatio-temporal patterns, i.e., approaches to pattern extraction from trajectories alternative to the one adopted in this paper. Then, Section 3 summarizes the basis of T-patterns, as introduced in [4]. Then, Section 4 presents the dataset considered in this paper, that is later analyzed with statistical means in Section 5 and T-patterns in Section 6. Finally, some conclusions are drawn in Section 7.

2. BACKGROUND AND RELATED WORK

In this section we summarize some relevant research related to spatio-temporal pattern mining, which is the main tool adopted in the analysis step of this work. To the best of our knowledge, the existing literature on this subject is composed of only a few recent works, that tackle the problem from different viewpoints.

The work in [3] considers patterns that are in the form of trajectory segments and searches approximate instances in the data; on the opposite, the work in [5] provides a clustering-based perspective, and considers patterns in the form of moving regions within time intervals, such as spatio-temporal cylinders or tubes and counts as occurrences all trajectory segments partially contained in the moving regions. A similar goal, but focused on cyclic patterns, is pursued in [6]: the authors propose an effective and fast mining algorithm for retrieving *maximal* periodic patterns, treating time as discrete, yet dealing with continuous spatial locations that are discretized dynamically through density-based clustering. Finally, in [7] patterns in the form of sequences of locations are mined, building candidate patterns over a predefined discretization of space (a grid) and time (fixed snapshots), and computing the support of a pattern as its *expected support* w.r.t. the location distributions of the input objects.

3. T-pattern

Trajectory patterns (T-patterns in short) are an extension of the traditional sequence mining paradigm, that integrates spatial and temporal information extracted from trajectory data. To our purpose, a trajectory of an object is a sequence of time-stamped locations representing the traces collected by some wireless/mobility infrastructure, such as the GSM mobile phone network, or GPS traces recorded by portable devices and transmitted to a central server. The location,

IWCTS '09, November 3, 2009, Seattle, WA, USA.
Copyright 2009 ACM ACM ISBN 978-1-60558-861-2 ...$10.00.

like a GSM cell or a lat-long pair, is abstracted using ordinary Cartesian coordinates.

3.1 Problem definition

The inclusion of spatial and temporal information in a sequential pattern can be obtained by making the patterns include spatial constraints on each element of the sequence and temporal constraints between consecutive elements:

Definition 1. A *Trajectory pattern*, called T-pattern, is a pair (S, A), where $S = \langle (x_0, y_0), \ldots, (x_k, y_k)\rangle$ is a sequence of points in $\mathbf{R}^2$, and $A = \langle \alpha_1, \ldots, \alpha_k\rangle \in \mathbf{R}^k_+$ is the *(temporal) annotation* of the sequence. T-patterns will also be represented as $(S, A) = (x_0, y_0) \xrightarrow{\alpha_1} (x_1, y_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} (x_k, y_k)$.

An occurrence of a T-pattern takes place when both spatial positions and transition times of the pattern approximatively correspond to those found in an input sequence:

Definition 2. (Spatio-temporal containment, $\preceq_{N,\tau}$) Given a trajectory T, a time tolerance τ, a neighborhood function $N : \mathbf{R}^2 \rightarrow \mathcal{P}(\mathbf{R}^2)$ and a T-pattern $(S, A) = (x_0, y_0) \xrightarrow{\alpha_1} (x_1, y_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} (x_k, y_k)$, we say that (S, A) *is contained in* T ($(S, A) \preceq_{N,\tau} T$, or simply $(S, A) \preceq T$, when clear from context) if and only if there exists a subsequence T' of T, $T' = \langle (x'_0, y'_0, t'_0), \ldots, (x'_k, y'_k, t'_k)\rangle$ such that:

1. $(x_i, y_i) \in N(x'_i, y'_i)$
2. $\forall_{1\leq j\leq k}.\ |\alpha_j - \alpha'_j| \leq \tau$

where $\alpha'_j = t'_j - t'_{j-1}$.

Intuitively, a T-pattern is contained in a trajectory if the latter contains an approximated instance of the former, the approximation being associated with both the spatial and the temporal dimensions. We notice that comparisons are not performed on absolute times, as spatio-temporal containment is based on the *transition times* between two consecutive elements in the sequence, expressed by the α_i and α'_i terms of condition 2 in Definition 2.

From containment, a natural definition of support and frequent pattern can be assigned, as well as a general definition of the trajectory pattern mining problem.

Definition 3. (Trajectory pattern mining) Given a database of input trajectories $\mathcal{D}$, a time tolerance τ, a neighborhood function $N()$ and a minimum support threshold s_{min}, the *trajectory pattern mining* problem consists of finding all *frequent* T-patterns, i.e., all T-patterns (S, A) such that

$$support_{\mathcal{D},\tau,N}(S, A) \geq s_{min}$$

where the support $support_{\mathcal{D},\tau,N}$ of a T-pattern (S, A) is the number of input trajectories $T \in \mathcal{D}$ such that $(S, A) \preceq_{N,\tau} T$.

3.2 Region-of-Interest approach

Notice that the neighborhood function is a parameter of the definition of containment of T-patterns in a input trajectory, and different neighborhood functions yield different variants of frequent T-patterns. In particular, the approach introduced in [4] makes use of a simple neighborhood function based on the idea of *Regions-of-Interest* (RoI):

$$N_R(x, y) = \begin{cases} A & \text{if } A \in R \ \wedge\ (x, y) \in A \\ \emptyset & \text{otherwise} \end{cases} \qquad (1)$$

where R is a set of disjoint spatial regions – each representing a *place* that is relevant for our analysis.

The neighborhood of a spatial point is the whole region it falls in, i.e., two points are considered similar iff they fall in the same region. All points that are not covered by the regions in R have an empty neighborhood, meaning that they are not similar to any point (including themselves). The result is that points disregarded by R will be virtually deleted from trajectories and spatio-temporal patterns.

Adopting the RoI approach, the resulting set of frequent T-patterns can be neatly represented as a set of temporally annotated sequences $R_0 \xrightarrow{\alpha_1} R_1 \cdots \xrightarrow{\alpha_n} R_n$, where each R_i is a region of R.

The algorithms developed in [4] provide three ways of defining the RoI:

1. take them from the user, based on his/her experience;
2. compute them automatically, by means of heuristics that evaluate the traffic density on the geographic space;
3. compute them as in point 2, but dynamically updating the traffic density along the extraction of each pattern, at each step focusing only on the trajectories involved in the partial pattern obtained so far.

We remark that the solutions 1 and 2 can be naturally combined, thus merging the user background knowledge with the suggestions provided by the density-based heuristics. Moreover, a by-product of the algorithms used in approaches 2 and 3 is the estimation of the density of traffic, that is visually represented highlighting the most dense geographical areas. Both these features will be exploited in this paper, in order to provide preliminary insights on the data and to obtain meaningful regions.

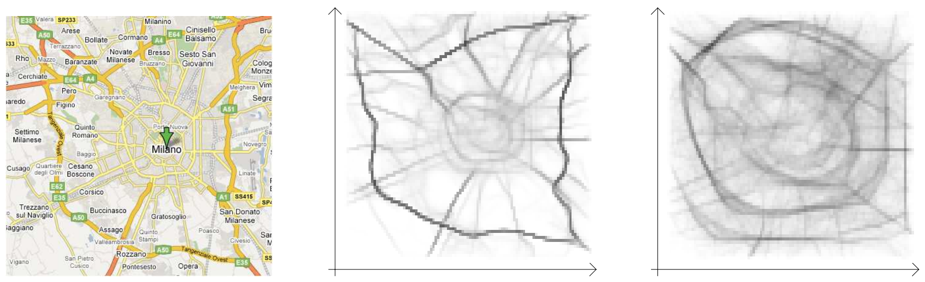
4. THE CASE STUDY

In this section, we explore a real case study by applying data mining spatio-temporal methods described in this paper. We develop two novel analytical services for mobility analysis and traffic management, designed and validated in collaboration with Milan Mobility Agency:

- a study describing how the traffic density is distributed on space and how it changes along the time. This is achieved by exploiting the density plots computed by the T-pattern miner algorithm and described in Section 5;
- a study of the frequent movement behaviours in terms of T-patterns, that highlights common routes and timings followed by the vehicles in some time windows. This study is presented in Section 6.

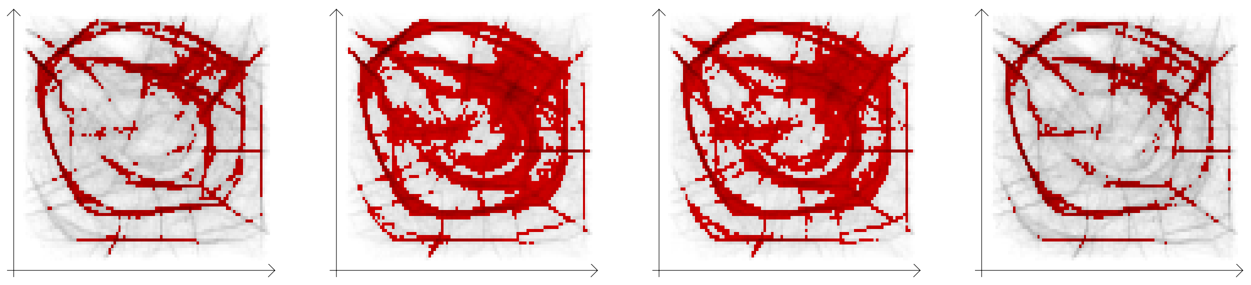
The dataset considered in this case study is a collection of GPS traces describing the movement of GPS-equipped cars in the urban area of Milan, Italy. The overall area covered by the data is approximately a rectangle of size 14 km by 22 km. The temporal span of the dataset is exactly one week, located in spring 2007. The number of vehicles tracked is ca. 17k, that generated ca. 2 million observed points (i.e., time-stamped locations).

Before starting the analysis, the raw dataset was preprocessed in order to:



(a) Map of Milan (b) Whole dataset (c) area analyzed

Figure 1: Map and dataset



(a) Sunday (b) Tuesday (c) Thursday (d) Saturday

Figure 2: Distribution of density along the week

- remove noisy input locations; and
- reconstruct the trajectories followed by the vehicles.

In the first step, a simple heuristics was applied, that computes the apparent speed required to move from each input point to the next one of the same vehicle, and then removes the second point if such speed is higher than a given threshold, fixed to ca. 200 km/h.

In the second step, the sequence of points corresponding to a single vehicle has been partitioned into maximal subsequences such that the time distance between two consecutive points less than a given threshold, fixed to 15 minutes. Higher delays between consecutive observations usually mean that the GPS device was simply switched off.

The preprocessing steps described above led to the removal of ca. 48K points, and the creation of ca. 200K trajectories, i.e., around 11 trajectories per vehicle.

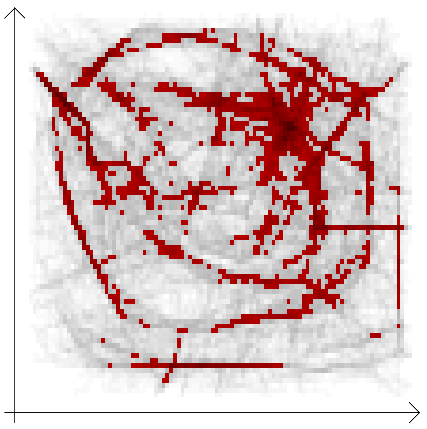
The two graphs on the left of Figure 1(c) report a general map of the city, and a plot of the traffic density (i.e., number of trajectories passing in the same area) computed over the whole dataset. The darker points, corresponding to higher density areas, cover the main roads of the city and, in particular, the peripheral ring road has a very high density. Preliminary experiments with both the types of analysis considered in this paper, i.e., density-based and pattern-based analysis, showed that this phenomenon tends to overshadow the density distributions and the movement patterns occurring in the rest of the area. For this reason the analysis has been focused to a smaller, central sub-area, apparently more interesting also in terms of better understanding the urban mobility of the city. The density distribution for the selected area is shown in Figure 1(c)(right).

5. A TRAFFIC DENSITY-BASED ANALYSIS

As visible in the map in Figure 1(c)(left), within the peripheral ring road the city has four smaller ones (though the external one does not form a complete circle) that carry most of the internal traffic, as visible in Figure 1(c)(center and right). In the following, we provide a more detailed analysis of how such traffic is distributed, by highlighting the areas that are visited by at least a given fraction of the all trajectories.

Figure 2(d) reports the daily density distribution of four days in the same week: Sunday, Tuesday, Thursday, and Saturday. The dark/red points correspond to areas visited by at least the 0.035% of trajectories. A first comparison between them shows that the overall traffic increases significantly during working days. In particular, while on Sunday and Saturday only the three external rings are dense, and only partially, during Tuesday and Thursday all four rings, including the very central one, are massively populated. On the rings, and especially on their south-eastern side, the traffic is significantly distributed also around the main road, probably corresponding to areas that contain residences and working sites, i.e., the main sources and destinations of movements in an urban area. Another significant phenomenon to notice is the presence of high traffic in a whole sector in the north-eastern part of the city, not limited to the main roads. In particular, during working days the traffic covers almost all the area, while during weekends it takes around a 50% of the area. A study of the site on the map reveals that it contains the central railway station of the city, as well as other connections to public transportation (mainly metro stops and bus stations).

In order to understand how the traffic changes during the


(a) Thursday morning (7-10am)

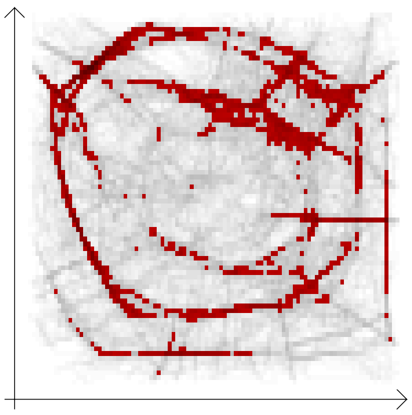

(b) Thursday afternoon (5-8pm)

Figure 3: Distribution of density

day, with particular emphasis to the daily movements between home and working place, a working day (Thursday) was selected, tracing the traffic in the early morning and late afternoon. Figure 3(b) reports the density distribution of Thursday limited to two time intervals of duration equal to 3 hours: one in the morning, from 7 a.m. to 10 a.m., and one in the afternoon, from 5 p.m. to 8 p.m. The density threshold, in this case, has been set to 0.01% of trajectories. The figure shows that in general the density is higher during the morning hours, in particular in the north-eastern sector mentioned in the previous analysis, on the eastern side of the second central road ring, and in the very center of the city. However, in the afternoon distribution the density slightly increases in the north-western and south-western sections of the third ring (which is the most external one in this sector of the city). Such differences can be an indication of an asymmetry either in the routes followed (due, for instance, to a large presence of one-way roads or to well-known traffic jams problems in some points of the city) or in the leaving/returning times of workers (for instance, due to large segments of population that leave home between 7 a.m. and 10 a.m., but return home earlier or later than 5-8 p.m., or vice versa); or, simply, to the presence of people that move also during working hours, which is especially reasonable for the central zone.

6. A T-pattern-BASED ANALYSIS

In this section we apply the T-Pattern miner algorithm to extract pattern over the same data analyzed in the previous section.

Selection of RoI

First the algorithm was run in order to automatically extract Regions-of-Interest solely based on density of traffic. The result was a set of 11 middle-sized regions that cover the main segments of the internal ring roads of the city, plus 79 very small regions that cover other dense spots. The very center of the city, although interesting for our analysis, was not taken into consideration due to its density that is slightly smaller than the used threshold (the same applied in the previous analysis, see Section 5). Therefore, as a user-driven refinement of the regions proposed, the small regions were removed, and 4 new square regions were added to cover the center of the city. The result can be seen in Figure 4(c) (yellow rectangles and the largest orange one).

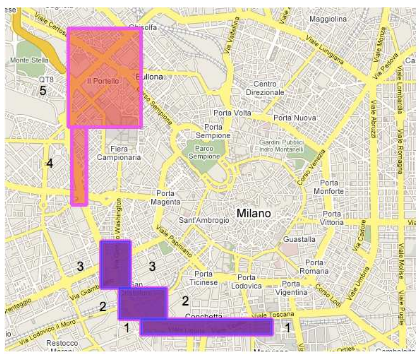
Comparing working and weekend days

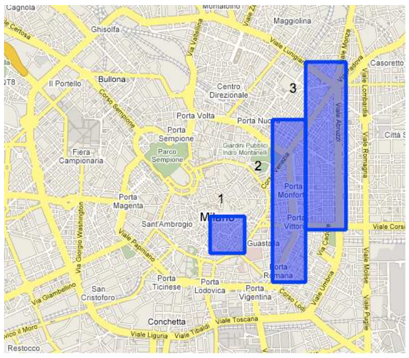
In order to better emphasize the behaviours followed during working days, we extracted frequent T-patterns on Sunday and on Thursday, comparing the results.

In Figure 4(a) we can see a pattern (depicted in blue and numbered from 1 to 3) composed of three steps that occurs during Sunday, starting from the southern segment of the third ring road (counting from the center) and following the road ring on the western side. The same pattern was found on Thursday (depicted in blue and red, numbered from 1 to 5), but continued with two more steps that touch the north-east side of the same road. That represents the fact that, while during working days it is common to travel along all the western side of the ring road (for instance to reach the highways that connect Milan to Turin, to the west), during the weekend the movements stop much earlier, probably meaning that the connections provided in the north-eastern part of the city are mostly used for working purposes.

Figure 4(b) reports another pattern, this time obtained only on Sunday, that starts from the very center of the city (a region manually provided by the user, added to those automatically extracted by the tool) and moves towards east and north-east, touching two consecutive ring roads. From the patterns found, this kind of apparently common movement seems to be actually frequent only during the weekend.

Finally, Figure 4(c) shows how, in some cases, the dynamic version of the T-pattern miner algorithm, that updates the RoI during the pattern extraction process, is able to *refine* the patterns found with the approach used so far. In particular, the Figure contains a refined pattern (depicted in orange and numbered from 1 to 4) that starts from the northern edge of the second ring road and continues the movement following precisely the same road for three successive steps. For comparison, the corresponding pattern discovered by means of the static RoI approach, is composed of region 1 and the large yellow region that contains all steps from 2 to 4, thus loosing the information of where exactly the movement develops – indeed, in the larger region


(a) Pattern on Sunday (blue) and Thursday (blue and red)


(b) Pattern on Sunday, using a user-provided region

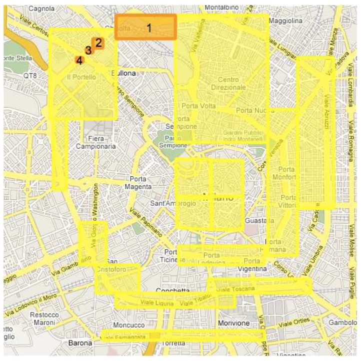

(c) pattern dynamic versus static regions

Figure 4: Typical itineraries in Milan

the road crosses several large streets making it difficult to make clear inferences from the static RoI-based pattern.

7. CONCLUSION

In this paper we presented a real-world application of the T-pattern mining paradigm, used both to extract preliminary information about the traffic density distribution over the geographic space and along the time, and to extract patterns describing frequent and precise behaviours followed by the monitored population. The results of this analysis provide useful insights for the understanding of the global behaviour and the *local* phenomena and, at the same time, suggests several improvements to better help the analyst, that we are going to pursue in the near future. Among them, we list the following:

- providing an effective graphical, interactive user interface to ease some steps of the analysis, and to enable new ones not possible for non-expert users;
- providing automatic means for merging background knowledge and extracted suggestions concerning the definition of Regions-of-Interest, for instance formulated as a information integration (and conciliation) problem;
- providing an interactive tool for navigating the output patterns, to allow the selection by means of spatial, temporal or qualitative criteria.

8. REFERENCES

[1] Gennady Andrienko, Natalia Andrienko, and Stefan Wrobel. Visual analytics tools for analysis of movement data. *SIGKDD Explor. Newsl.*, 9(2):38–46, 2007.

[2] K. Ashok. *Estimation and Prediction of Time-Dependent Origin-Destination Flows.* PhD thesis, Massachusetts Institute of Technology, 1996.

[3] H. Cao, N. Mamoulis, and D. W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, 2005.

[4] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli. Trajectory pattern mining. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (KDD'07).* ACM, 2007.

[5] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Proceedings of 9th International Symposium on Spatial and Temporal Databases*, pages 364–381. Springer, 2005.

[6] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD*, 2004.

[7] Jiong Yang and Meng Hu. Trajpattern: Mining sequential patterns from imprecise trajectories of mobile objects. In *EDBT*, pages 664–681, 2006.

Author Index