Paper No. 01-2207

Comparative Assessment of AVI Tag Matching Algorithms for Estimating Vehicle Travel Times

Bruce Hellinga, PhD, P.Eng.

Assistant Professor, Department of Civil Engineering
University of Waterloo, Waterloo Ontario Canada N2L 3G1
Phone: 519-888-4567 Ext. 2630
Fax: 519-888-6197
Email: bhellinga@uwaterloo.ca

Transportation Research Board
80th Annual Meeting
January 7 - 11, 2001
Washington, D.C.

Number of words =	5,425
Tables (1 x 250 words) =	250
Figures (8 x 250 words =	2,000
Total equivalent words =	7,675

ABSTRACT

This paper examines the computational complexity associated with three candidate AVI tag matching algorithms that could be used to obtain individual vehicle travel time data in real-time. These algorithms are suitable for application to a linear roadway facility using transponder tags that do not have programmable memory. Analytical expressions are derived to estimate the worst-case and average computational load associated with each algorithm. A simulation is performed to test the validity of the assumptions made in these derivations, and also to perform a sensitivity analysis on several key system parameters, including the rate of flow of AVI equipped vehicles, the mean travel time between tag reader stations, the coefficient of variation of travel time, and the proportion of vehicles that pass the upstream tag readers.

INTRODUCTION

Recent advances in short-range communication technologies and computing infrastructure have enabled the development and deployment of a variety of automated vehicle identification (AVI) systems. These systems are able to uniquely identify individual vehicles within a traffic stream, usually through the use of a transponder or AVI tag that is affixed to the vehicle. AVI systems have been deployed in North America for the purposes of electronic toll collection (e.g. Highway 407 in Toronto, Canada; LeeWay ETC System in Lee County Florida (Burris and Byers, 1999), the TRANSCOM system in New York and New Jersey (Mouskos et al, 1999)) and for the purposes of obtaining real-time vehicle travel time data (e.g. the TransGuide system in San Antonio; the TranStar system in Houston, Texas; the COMPASS system in Toronto) to support traffic management and control.

Three general types (generations) of tag technologies currently exist. Type I tags are limited to provide *read*-only communications, meaning that the tag is not capable of receiving and storing data. Type I tags act only as a transmitter, broadcasting a unique tag identification code whenever activated by a signal from a roadside tag reader unit. Type II and Type III tags have *read* and *write* communication capabilities and are able to receive and store data in programmable memory. For example, Type II and III tags are able to receive and store the identification code broadcast by a tag reader and the time of the broadcast, and then re-transmit this information to the next tag reader station that is encountered.

The estimation of individual vehicle travel times requires that vehicles equipped with tags be uniquely identified at two locations and that the time (time stamp) associated with each identification be known. For the travel times to be useful for traffic management purposes, it is also required that the route taken by vehicles between each pair of tag reader locations be known.

The process of finding time stamp entries from two tag reader stations for the same vehicle is called *matching*. When Type II or Type III tags are used, the matching of unique vehicles can be considered a distributed process, as it takes place at each tag reader. When a vehicle equipped with a Type II or Type III tag passes a tag reader, it transmits the time stamp and tag reader ID associated with the tag's previous tag reading transaction (i.e. roadside tag reader previously passed by the vehicle). The tag reader then is able to determine immediately the vehicle's travel time and the associated roadway segment over which the vehicle has traveled.

However, when Type I tags are used, the matching process must be done centrally. Each tag reader transaction consists of only a time stamp and vehicle (tag) ID. All tag readers transmit transaction data to a central processing computer. This central computer is responsible for carrying out the matching process. If the matching process is being conducted to compute vehicle travel times to support traffic management and information provision, then the process must be conducted in real time. If the travel time data (or identification of road segments traveled) are to be used solely for toll billing or compilation of historical data, then the process need not be conducted in real time.

In this research, we consider the case in which Type I tags are used for the purpose of obtaining travel time data to support real-time traffic management and/or information provision. We are interested in estimating the computational complexity of the matching process associated with a single pair of tag readers. Specifically, we examine the computational complexity of three different matching algorithms. We also examine the sensitivity of the computational complexity to a number of key systems parameters, including the traffic volume passing each tag reader, the mean and variance of the travel time between tag reader stations, and the proportion of vehicles that pass one but not both stations.

In the next section, we describe the three matching algorithms considered in this research. In the following section we develop analytical expressions to estimate the average and worst-case computation load associated with these algorithms for a general case. Using simulation, we quantify the computation load of these three matching algorithms for a specific roadway and tag reader geometry, and compare these results to those obtained from the analytical expressions. Finally, the sensitivity of the computation load to several key traffic network characteristics is quantified.

MATCHING ALGORITHMS

Matching algorithms are normally considered to be a specific application of searching algorithms, which attempt to find an item within a list of items. The field of searching and sorting algorithms has become well developed over the past 40 years, with much of the current research motivated by the need for greater computational efficiency for Intranet and Internet applications (e.g. internet search engines, data routers, etc.), and data management.

While a wide range of searching and sorting algorithms have been developed, the choice of an appropriate algorithm depends on the system application. Therefore, before describing the algorithms that will be examined, we describe the AVI vehicle ID matching system.

System Description

Consider two tag reader stations, *A* and *B*, located on a single direction of roadway as shown in Figure 1. Station *A* is located at the upstream end of the roadway segment. Station *B* is located at the downstream end. As each AVI tag equipped vehicle passes a tag reader, a transaction record is generated consisting of the vehicle tag ID, the time at which the transaction took place, and the tag reader ID (*A* or *B*) at which the transaction occurred. As each record is generated, it is transmitted to the central processing computer. We assume that the delay that occurs between the time that the transaction takes place, and the time that the record is available for processing at the central computer, is negligible or constant and therefore need not be considered. The central computer maintains an active list (likely in memory, though this is not required) of transaction records associated with tag reader *A* (*List A*). Each time a record associated with *B* is received, the central computer initiated the matching process in order to determine the travel time associated with the vehicle passing *B*.

We note, that because of the entry and exit ramps, vehicles passing *A* may not necessarily pass *B* and those that pass *B* have not necessarily passed *A*. The probability that a vehicle passing *B* has also passed *A*, and the probability that a vehicle passing *A* will also pass *B*, depend on the traffic demands between origins 1 and 2 and destinations 3 and 4 (Equations 1 and 2). We ignore traffic demands from origin 2 to destination 4 as these vehicles do not pass either *A* or *B* and therefore never generate a transaction record at *A* or *B*.

$$P_{BA} = \frac{D_{1,3}}{D_{1,3} + D_{2,3}} \tag{1}$$

$$P_{AB} = \frac{D_{1,3}}{D_{1,3} + D_{1,4}} \tag{2}$$

where:

P_{BA}	=	the probability that a vehicle generating a transaction record at *B* will have also generated a transaction record at *A*
P_{AB}	=	the probability that a vehicle that has generated a transaction record at *A* will have also generated a transaction record at *B*
$D_{i,j}$	=	traffic flow between origin *i* and destination *j* (vehicles per hour)

Method 1: Sequential Search Algorithm

In Method 1 we assume that the list of transaction records from tag reader *A* (*List A*) is maintained in the order in which the records are received (i.e. chronological order). The problem of finding a match for each record generated at *B* falls in the category of searching within an unsorted list, for which the Sequential Search Algorithm (SSA) is the most common, and therefore is the one we select in this research.

Thus, for Method 1, the computational load is solely associated with the size of *List A* as the sequential search algorithm must be applied for each record generated at tag reader *B*.

Method 2: Modified Sequential Search Algorithm

Method 2 is a variation of the SSA described in Method 1 in which we attempt to reduce the number of records in *List A* that must be examined. We make use of two characteristics of the tag matching problem. First, vehicles require some minimum time to travel from reader *A* to reader *B* (t_{min}). Second, *List A* is sorted in non-descending order with respect to the time at which each transaction record was generated. Vehicles that generate an AVI tag record at *B* at time *t*, can not have passed tag reader *A* any later than time $t - t_{min}$. Therefore, we maintain a dynamic pointer (*p*) that indicates the last vehicle from *List A* that could have generated a record at *B* by the current time *t*.

To illustrate this method, consider Figure 2. *List A* contains the records generated by AVI vehicles passing tag reader *A*. List *B* contains the records generated by AVI vehicles passing tag reader *B*. The current time, *t*, is 14:34:23. The minimum time required for a vehicle to travel from tag reader *A* to reader *B* is 3 minutes. *List A* contains *n* records with record generation times ranging from 14:26:13 to 14:34:03. Pointer *p* indicates the last record in *List A* for which the record generation time is less than (t - t_{min}) = 14:31:23. All records in *List A* that follow pointer p (i.e. $p < i \leq n$) have been generated by vehicles that could not have reached tag reader B by the current time, t, and therefore do not need to be examined.

Thus, for Method 2, the computational load is associated with the size of *List A*, the minimum travel time between tag reader A and B, and the temporal distribution of vehicle arrivals at tag reader A.

Method 3: Binary Search Algorithm

Method 3 differs from Methods 1 and 2 in that we maintain *List A* as a sorted list with respect to the vehicle tag ID. That is, each time a record is received from tag reader A, we insert this record into *List A* such that records are maintained in non-descending order according to the tag ID. When *List A* is a sorted list, we can use a search algorithm that is more efficient than the Sequential Search Algorithm to find a match for vehicles that pass B. In this research we consider the popular Binary Search Algorithm (BSA) (see for example Banachowski *et al*, 1991; or Baase, 1998).

Thus, for Method 3, the computational load is associated with two separate processes, namely maintaining the order of *List A* and the use of the Binary Search Algorithm.

In the next section we develop analytical expressions to estimate the average computation load and the worst-case computation load for both methods.

ANALYTICAL ASSESSMENT OF COMPUTATIONAL LOAD

Searching and sorting algorithms are typically compared on the basis of their computational efficiency, that is the number of basic operations required to complete the desired task (i.e. sort and/or search). Since the lists, to which these algorithms are applied, are generated by random processes and the algorithms are applied repetitively, it is not particularly useful to compare algorithm performance for a list containing a specific series of entries. Rather, the comparison is typically conducted to determine the average and the maximum (worst case) number of computations required to sort a list (*List A*) having n entries. We assume that X is the vehicle tag ID that was most recently identified at tag reader B, and for which we seek to find an entry (a match) in *List A*.

When searching for X in a list, the basic operation can be considered to be a comparison of X with an entry in the list. When sorting a list of numbers, the basic operation can be considered to a comparison of two entries in the list. In this paper, we refer to a basic operation as a *computation*, and consider a computation to be a single comparison between an entry of the list (*List A*) and the value being searched for (*X*). Normally this comparison is implemented using an *if …then…else* construct in a computer programming language.

Worst Case Analysis

Methods 1 and 2: SSA and Modified SSA

For Methods 1 and 2, the only computational load is associated with the Sequential Search Algorithm. In the worst case analysis, the value being sought is located in the last position in *List A* (i.e. at position *n*). Therefore, the maximum number of comparisons that can be required for a single application of the SSA with a list having *n* entries, is *n*.

$$W_1(n) = W_2(n) = n \tag{3}$$

where:

$W_1(n)$	=	maximum number of comparisons associated with Method 1 for a list with *n* entries
$W_2(n)$	=	maximum number of comparisons associated with Method 2 for a list with *n* entries

Method 3: Binary Search Algorithm

Method 3 consists of two stages, namely the sorting of *List A*, and then the searching of *List A* for entry *X*. Sorting of *List A* is required to ensure that each time a new record is generated at tag reader *A*, it is inserted into *List A* in a position such that all *n* records in *List A* are in non-descending order by vehicle ID. This can be accomplished using a number of sorting algorithms. In this research, we have chosen to use the Binary Insertion Sort Algorithm (BISA), a variation of the Binary Search Algorithm, since this sorting method is computationally efficient and is particularly well suited to the repetitive insertion of a single record into an already ordered list.

The second stage consists of conducting a search of *List A* for a vehicle tag ID *X*. Since *List A* is sorted by vehicle tag ID, it is possible to use the Binary Search Algorithm (BSA) to locate *X*.

The BSA is based on the concept of "divide and conquer". During each search iteration, a value from the middle of the list is considered. If *X* is greater than this value, then the lower half of the list (i.e. L_1, L_2, …, $L_{n/2}$) is eliminated from further consideration and the midpoint of the upper half of the list if chosen for consideration in the next iteration. Thus, each time a comparison is made, one half of the remaining list is eliminated from further consideration.

The maximum number of comparisons associated with a single application of the BSA can be shown to be (Baase, 1988)

$$W_2(n) = 1 + W_2\left(\tfrac{n}{2}\right) = 1 + \log_2(n) \qquad (4)$$

where:

$W_2(n)$	=	maximum number of comparisons associated with Method 3 for a list with n entries

Analysis of Average Number of Computations

Method 1: Sequential Search Algorithm

The average number of comparisons depends on the number of entries in the list being searched (n) and the probability that $X = L(i)$. We assume that X is equally likely to be in any position in the list. Furthermore, we assume that there is a probability P_{BA} that X is in the list.

$$P(X = L(i)) = \begin{cases} \dfrac{P_{BA}}{n} & 1 \le i \le n \\ 1 - P_{BA} & i = n+1 \end{cases} \qquad (5)$$

where:

X	=	value being search for
$L(i)$	=	entry at position i of list L.
n	=	number of entries in list L
P_{BA}	=	probability that X is contained within list L

The average number of comparisons associated with any input I (i.e. set of list entries) can be computed as the product of the probability that X is equal to $L(i)$ and the number of comparisons required when X is equal to $L(i)$.

$$A_1(n) = \sum_{i=1}^{n} \left[P(X = L(i)) \times i\right] + P(X = L(n+1)) \times n \qquad (6)$$

where:

$A_1(n)$	=	expected average number of comparisons required to find X in a list of length n

Then, the average number of comparisons can be determined by substituting Equation 5 into 6 to give Equation 7.

$$A_1(n) = \sum_{i=1}^{n} \left(\frac{P_{BA} \times i}{n} \right) + (1 - P_{BA})n \tag{7}$$

Since

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{8}$$

then

$$A_1(n) = \tfrac{1}{2} P_{BA}(n+1) + (1 - P_{BA})n \tag{9}$$

Equation 9 provides an estimate of the average number of comparisons that must be performed for each application of the SSA. Since this algorithm must be applied for each record generated at tag reader *B*, the average number of comparisons that will need to be made when *k* records are generated at *B* is

$$A_1(n,k) = \tfrac{1}{2} k P_{BA}(n+1) + (1 - P_{BA})nk \tag{10}$$

where:

$A_1(n,k)$ =	expected average number of comparisons required to find $X_1, X_2, \ldots, X_k$ in *List A* having *n* records

Method 2: Modified Sequential Search Algorithm

Method 2 differs from Method 1 in that knowledge of the minimum travel time between tag reader *A* and *B* is used to define a subset of the *n* entries in *List A*. This subset consists of the first *p* records in *List A*. As with Method 1, we assume that *X* is equally likely to be in any position in this truncated list, and that there is a probability (1-P_{BA}) that *X* is not in *List A* at all.

$$P(X = L(i)) = \begin{cases} \dfrac{P_{BA}}{p} & 1 \le i \le p \\ 0 & p < i \le n \\ 1 - P_{BA} & i = n+1 \end{cases} \tag{11}$$

where:

X	=	value being search for
L(i)	=	entry at position *i* of list *L*.
n	=	number of entries in list *L*

p = number of entries in list L for which $P(X=L(i)) > 0$

P_{BA} = probability that X is contained within list L

The calculation of a value for p depends on the average rate at which AVI equipped vehicles pass tag reader A (V_A) and the minimum travel time (t_{min}).

$$p = n - \left(\frac{t_{\min}}{3600} V_A \right) \tag{12}$$

where:

p = number of entries in list L for which $P(X=L(i)) > 0$

t_{min} = minimum travel time from tag reader A to tag reader B (seconds)

V_A = average rate at which AVI equipped vehicles pass tag reader A (vph)

Then, similar to the development shown for Method 1, the average number of comparisons can be determined by Equation 13.

$$A_2(n,k) = \tfrac{1}{2} k P_{BA} (p+1) + (1 - P_{BA}) pk + k \tag{13}$$

where:

$A_2(n,k)$ = expected average number of comparisons required to find $X_1, X_2, \ldots, X_k$ in *List A* having n records

The last term in Equation 13 reflects the computational load required to maintain the position of pointer p. Each time a new record is generated at reader B, the pointer position is advanced within *List A*. Thus, the total number of comparisons associated with this positioning is equal to k, the number of records generated at B.

Method 3: Binary Search Algorithm

If we assume that X is equally likely to be found in any position in the list, then Equation 14 expresses the average number of comparisons required by the Binary Search Algorithm (BSA) to find X in a list of length n (Baase, 1988). In Method 3 the BSA algorithm is applied each time a new record is generated at tag reader A (to insert the record into the correct location in *List A*). The BSA is also applied for each record generated at tag reader B to find the corresponding vehicle record at A. If m is the number of records added to *List A* (i.e. the number of AVI tag reads at reader *A)*, and k is the number of records at B, then Equation 15 expresses the expected average number of comparisons for Method 3.

$$A(n) = \log_2(n) + \tfrac{1}{2} \tag{14}$$

$$A_3(n,k,m) = (m+k)(\log_2(n) + \tfrac{1}{2}) \tag{15}$$

Equation 15 assumes that there are *n* entries in *List A* at all times. The validity of this assumption is likely to depend on the specific scenario being examined. Each time a new record is generated at *A* another entry is added to *List A* and consequently *n* is increased by 1. However, each time a match is found in *List A*, that record is deleted from *List A*, and *n* is decreased by 1.

SAMPLE APPLICATION

Consider a sample application in which 800 records are generated at tag reader *A* (m=800); 1000 records are generated at tag reader *B* (k=1000); on average 65 records are in *List A* (n=65); the proportion of AVI equipped vehicles passing *B* that have also passed *A* is 0.8 (P_{BA}=0.8), the minimum travel time from tag reader *A* to *B* (t_{min}) is 126 seconds, and the average rate at which AVI equipped vehicles pass tag reader A (V_A) is 800 vph.

Equations 10, 13, and 15 can be used to estimate the associated average computational load for Methods 1, 2, and 3 respectively (A_1=39,400; A_2 =23,600, and A_3 = 11,740). Clearly these estimates indicate that Method 3 provides significant benefits in reduced computational load.

A simulation study was performed to confirm the validity of the analytical expressions developed in the previous section. AVI equipped vehicles were generated with exponentially distributed headways. Vehicle travel times from tag reader *A* to tag reader *B* were assumed to follow a log-normal distribution with a mean of 300 seconds and a standard deviation of 50 seconds. The same traffic demands were used as were assumed for the application of the analytical expressions (i.e. m=800; k=1000; and P_{BA}=0.8). Note that for this analysis, all vehicles that pass reader *A* also pass reader *B* (i.e. $D_{1,4}$ = 0 and therefore P_{AB} = 1.0). The consequences of $P_{AB} > 0$ are examined later in this paper.

Methods 1, 2 and 3 were applied to the time series of simulated AVI data. Ten repetitions of the simulation were completed to provide an estimate of the average behavior. From these results, it was found that the average number of comparisons required to process these data was 18,002 for Method 1, 13,720 for Method 2, and 10,448 for Method 3.

The number of comparisons predicted by the analytical expressions and the simulation experiment differ by 54%, 42% and 11% of the analytical estimate for Methods 1, 2, and 3 respectively. These large differences can be explained by two factors. First, and most significantly, the derivation of the analytical expressions assumed that *X* (the vehicle being searched for) would be equally likely to be found in any position within *List A*. Figure 3 illustrates the relative frequency distribution of the position within the stack (*List A*) at which *X* is found. Since the number of entries in *List A* changes, position is presented as relative position and is computed as i'/n, where i' is the record number at which *X* is found, and *n* is the number of records in *List A*. These results show that *X* is not uniformly distributed across all positions within *List A*, but is more likely to be found near the top of the stack. For example, there is more than a 70% probability that *X* is found within the first 25% of the records in *List A*. The large relative frequency ($\approx$ 21%) associated with a relative position of 1.0 (i.e. the end of the list) corresponds with *X* being found in the last record in the list and with *X* not being found in the list at all (20% likely). Since *X* is more likely to be found near the beginning of the list, fewer comparisons are required to find *X*, especially for the Sequential Search Algorithm (Methods 1 and 2).

The second factor is related to the number of records assumed to be in *List A* (i.e. *n*). For the application of the analytical expressions, a value of $n = 65$ was chosen. In the simulation results, the value of *n* is not constant, but changes with time as new records are generated at tag reader *A* and added to *List A*, and as matches are found for tag reads at *B* and deleted from *List A*. Figure 4 illustrates the distribution of the number of records in *List A* for one of the individual simulation runs. While the average number of records (*n*) in *List A* is 63, *n* ranges from a maximum of 78 to a minimum of 1.

On the basis of these results it appears that the analytical expressions provided in Equations 10, 13 and 15 may over-estimate the computational load associated with AVI tag matching, particularly for the Sequential Search Algorithm (Method 1) and the Modified Sequential Search Algorithm (Method 2).

SENSITIVITY ANALYSIS OF COMPUTATIONAL LOAD

In this section we examine the sensitivity of the AVI matching computation load as a function of 5 relevant system parameters, namely mean travel time between tag reader *A* and *B* ($\bar{t}$), coefficient of variation of travel time (COV), total traffic demand passing tag reader *A* ($D = D_{1,3} + D_{1,4}$), the probability that a vehicle generating a transaction record at *B* will have also generated a transaction record at *A* (P_{BA}), and the probability that a vehicle that has generated a transaction record at *A* will have also generated a transaction record at tag reader *B* (P_{AB}).

Table 1 provides the values of the parameters examined. The impact of varying the parameter values was determined by simulating the application of Methods 1, 2, and 3 independently for each parameter. For example, all simulations conducted to examine the impact of mean travel time used default values for the remaining 4 parameters (i.e. COV = 0.10; D = 4000; P_{BA} = 0.6; P_{AB} = 1.0).

Mean Travel Time

The mean travel time between tag reader *A* and *B* directly influences the number of records that must be maintained in *List A*, and consequently the computational load associated with the AVI matching process. As expected, the results indicated that the number of records in *List A* increases linearly with increases in the mean travel time. Correspondingly, the computation load associated with Methods 1, 2, and 3 increases nearly linearly as a function of mean travel time, albeit at a different rate for each method. The rate of increase in computation load (measured as thousands of computations per 1-minute increase in mean travel time) was 162, 71, and 1.9 for Methods 1, 2, and 3, respectively.

These results have two implications. First, the mean vehicle travel time between tag reader *A* and *B* is determined by two factors, namely the distance between the readers and the level of traffic congestion experienced on this section of roadway. The design of the AVI matching system should account for both factors when assessing the expected computation load. Second, the computational load of Methods 1 and 2 are much more sensitive to average stack size, and consequently the mean travel time, than is Method 3.

Variation in Travel Time

Experimental results indicate that the variation of vehicle travel times about the mean travel time has a much more limited impact on the computational load that does the mean travel time. Figure 5 illustrates the computational load associated with Methods 1, 2, and 3 as a function of the coefficient of variation of vehicle travel times. COV values from 0.01 to 0.25 were examined, however, field observations indicate that a value of 0.10 is typical. The results illustrated in Figure 5 indicate that Method 2 is the most sensitive to the COV. When very little variance in travel times exist, Method 2 provides a computational load that is even smaller than that provided by Method 3. Physically, when the variation in travel time becomes smaller, the minimum time required for vehicles to travel from tag reader *A* to tag reader *B* approaches the mean travel time, and p (from Equation 12) becomes smaller. Conversely, as the variation in travel time becomes larger, the minimum travel time becomes very small and p approaches n, such that the computational load of Method 2 approaches that of Method 1.

Method 1 is moderately sensitive to variation in travel time. When the COV is 0.25, the computational load of Method 1 increases by 11% over the load at COV = .010. Method 3 is insensitive to COV, with an increase of only 0.03% for COV=0.25 as compared to COV=0.10.

Traffic Demand Passing Tag Reader A

It is expected that the computational load is directly influenced by the total AVI equipped traffic demand on the roadway. Figure 6 illustrates the computational load of Methods 1, 2 and 3 as a function of the total AVI equipped traffic demand, *D*. The computational load of Method 1 increases exponentially with increasing demand, while the computational load of Method 3 increases almost linearly. At a demand of 1,000 vph, Method 1 has a computational load that is approximately 4 times the computational load of Method 3. At a demand of 10,000 vph, the computational load of Method 1 is approximately 28 times that of Method 3. Method 2 provides a computational load that is approximately midway between Methods 1 and 3.

Proportion of Vehicles Passing Reader B that have also Passed Reader A

If the roadway segment between tag reader *A* and tag reader *B* contains one or more on-ramps, then not all of the vehicles passing tag reader *B* will have also passed tag reader *A*. Figure 7 illustrates the computational load associated with Methods 1, 2, and 3 as a function of the proportion of vehicles passing reader *B* that have also passed reader *A*. As P_{BA} increases, then the computational load decreases. These results are consistent with expectation, since for each vehicle that generates a record at *B*, but does not generate a corresponding record at *A*, the AVI matching process must search *List A* until no records remain before it can be concluded that a match cannot be found.

Proportion of Vehicles Passing Reader A that also Pass Reader B

If the roadway segment between tag reader *A* and tag reader *B* contains one or more off-ramps, then not all of the vehicles passing tag reader *A* will also pass tag reader *B*. Figure 8 illustrates the computational load associated with Methods 1, 2, and 3, as well as the average number of records in List *A* and the number of vehicles passing tag reader *B*, as a function of the proportion of vehicles passing reader *A* that also pass reader *B*. It is assumed that no mechanism is implemented to delete records from *List A*, even if no matching record is ever generated at tag reader *B*. As P_{AB} increases, the computational load of Methods 1 and 2 initially increases, and then for P_{AB} greater than approximately 0.5, the computational load decreases. These results can be best understood by examining the average number of records in *List A*, and the number of records generated at tag reader

B, as functions of P_{AB}. The concave relationship of computational load versus P_{AB} for Methods 1 and 2 is a result of the interaction between the number of matches that must be conducted (i.e. *k*), and the number of records in the *List A* (*n*). As P_{AB} increases, the number of records in *List A* decreases at a non-linear rate, and the number of vehicles passing *B* increases at a linear rate. The computational load is a function of both *n* and *k*, and consequently, there is a value for P_{AB} (approximately 0.5) at which the computation load is a maximum.

The assumption that records in *List A* for vehicles that do not pass tag reader *B*, are never deleted is not realistic. In practice, some decision rule (usually a time-based heuristic) is used to determine if a record in *List A* is likely to be an "unmatchable" record (i.e. a record for which a corresponding record at the downstream tag reader station will not be generated). For example, a recent AVI matching system deployed on Highways 427 and 409 in Toronto, Ontario, uses a travel time threshold, *T*. If a match has not been found within *T* minutes, then the record is automatically deleted from *List A*.

CONCLUSIONS AND RECOMMENDATIONS

The use of AVI tag technologies to obtain individual travel times requires the implementation of a real-time AVI matching process. The choice of architecture and algorithms for this process has significant bearing on the computation load experienced by the AVI matching computer. This paper has identified three candidate AVI matching systems and has developed analytical expressions that can be used to determine the maximum and average computational load associated with each. These expressions are based on the assumption that the probability that the tag ID being searched for is equally likely to be found in any one of the records in the list.

A simulation analysis of a typical AVI matching scenario has shown that this assumption is violated, with the result that the analytical expressions over-estimate the expected computation load by as much as 54% for the Sequential Search Algorithm, and 11% for the Binary Search Algorithm.

Both the analytical expressions and the simulation results indicate that Method 3 (sorting *List A* in non-descending order by vehicle tag ID and applying the Binary Search Algorithm) is much more computational efficient than either Method 1 (maintaining *List A* in chronological order and applying the Sequential Search Algorithm) or Method 2 (maintaining *List A* in chronological order and applying the Modified Sequential Search Algorithm).

A sensitivity analysis of five system parameters indicates that the most critical factors for influencing computation load are number of AVI equipped vehicles, mean travel time between tag readers, and the proportion of vehicles passing tag reader *B* that have also passed tag reader *A* (P_{BA}).

REFERENCES

Baase, Sara (1988) "Computer Algorithms – Introduction to Design and Analysis, 2nd Edition". Published by Addison-Wesley Publishing Company.

Banachowski, L., Kreczmar, A., and Rytter, W. (1991) "Analysis of Algorithms and Data Structures". Published by Addison-Wesley Publishing Company.

Burris, M., and Byers, M. (1999) *Customer Response to Lee County's Electronic Toll Collection and Variable Pricing Program*, Presented at the 6th ITS Congress held in Toronto, Canada.

Mouskos, K., Niver, E., Batz, T., and Sadegh, A. (1999) *Costs, Benefits and Institutional Issues of the TRANSMIT System*, Presented at the 6th ITS Congress held in Toronto, Canada.

List of Tables:

List of Figures:

Table 1: Experimental Design Parameter Values

Parameter	Values tested	Default Value
Mean travel time, $\bar{t}$ (minutes)	{2, 4, 6, 8, 10, 15}	{6}
Coefficient of Variation (COV = std/mean)	{0.01, 0.05, 0.10, 0.15, 0.2, 0.25}	{0.10}
Traffic Demand, D (vph)	{1000, 2000, 4000, 6000, 8000, 10000}	{4000}
Probability that vehicle passing B will also have passed A, (P_{BA})	{0.2, 0.4, 0.6, 0.8, 1.0}	{0.6}
Probability that a vehicle passing A will also pass B, (P_{AB})	{0.2, 0.4, 0.6, 0.8, 1.0}	{1.0}

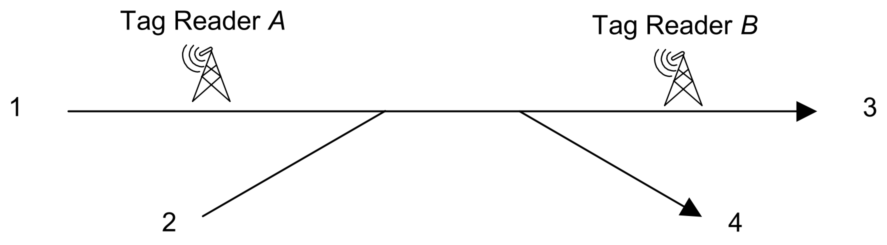
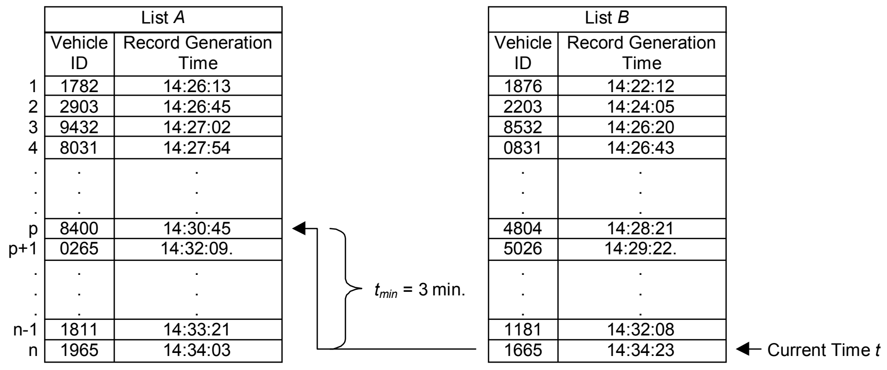


Figure 1: Roadway and tag reader configuration



	List *A*	
	Vehicle ID	Record Generation Time
1	1782	14:26:13
2	2903	14:26:45
3	9432	14:27:02
4	8031	14:27:54
. . .	. . .	. . .
p	8400	14:30:45
p+1	0265	14:32:09.
. . .	. . .	. . .
n-1	1811	14:33:21
n	1965	14:34:03

List *B*	
Vehicle ID	Record Generation Time
1876	14:22:12
2203	14:24:05
8532	14:26:20
0831	14:26:43
. . .	. . .
4804	14:28:21
5026	14:29:22.
. . .	. . .
1181	14:32:08
1665	14:34:23

Figure 2: Modified Sequential Search Algorithm

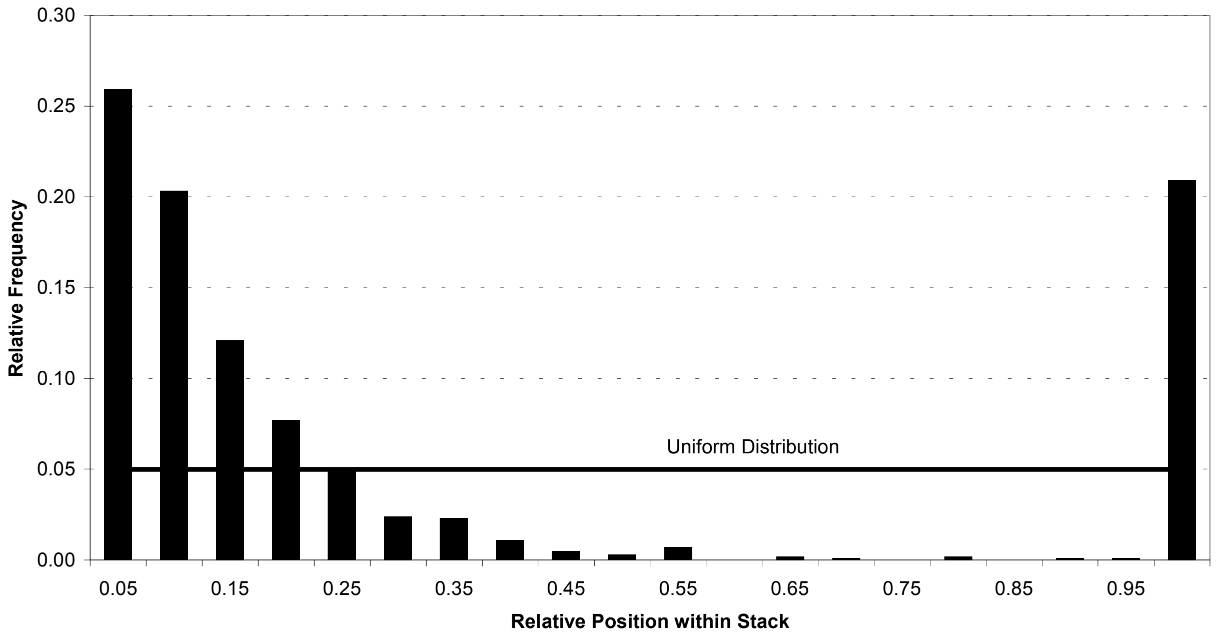


Figure 3: Distribution of Position of *X* within *List A*

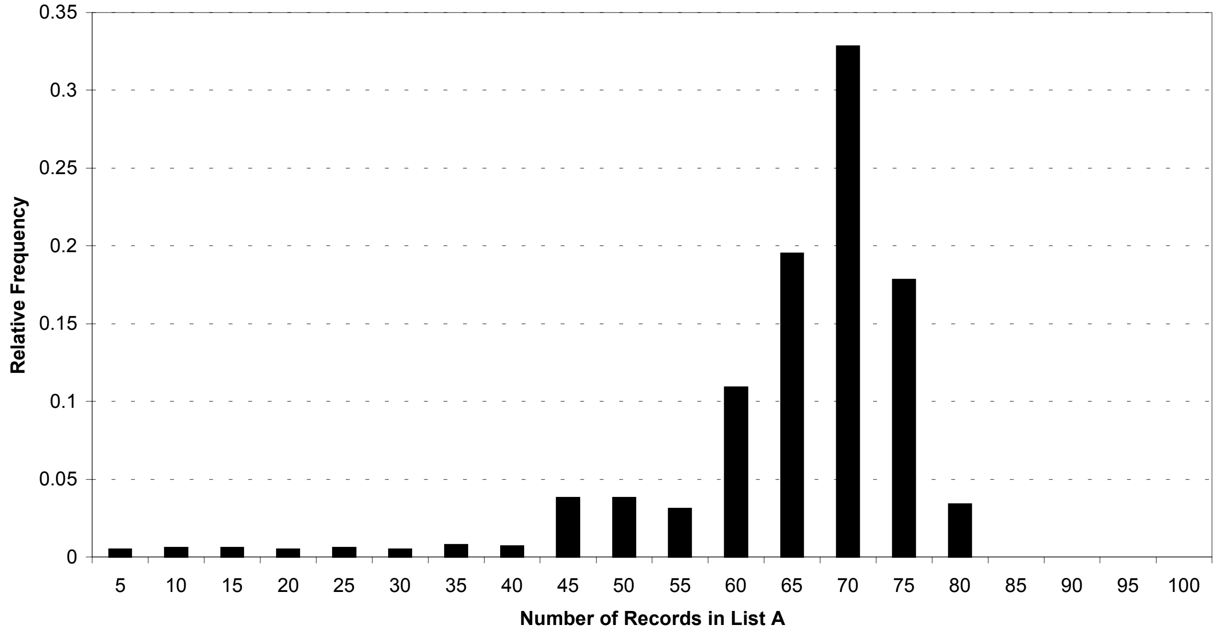


Figure 4: Distribution of the Number of records in *List A*

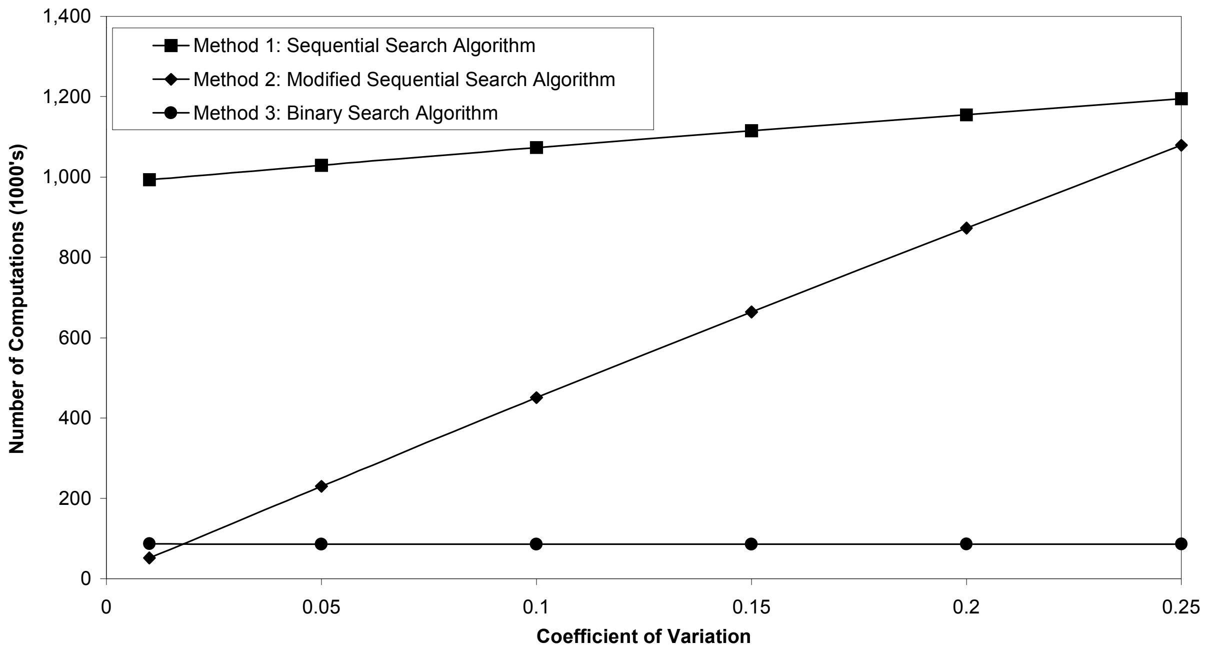


Figure 5: Computational load as a function of variation in vehicle travel time

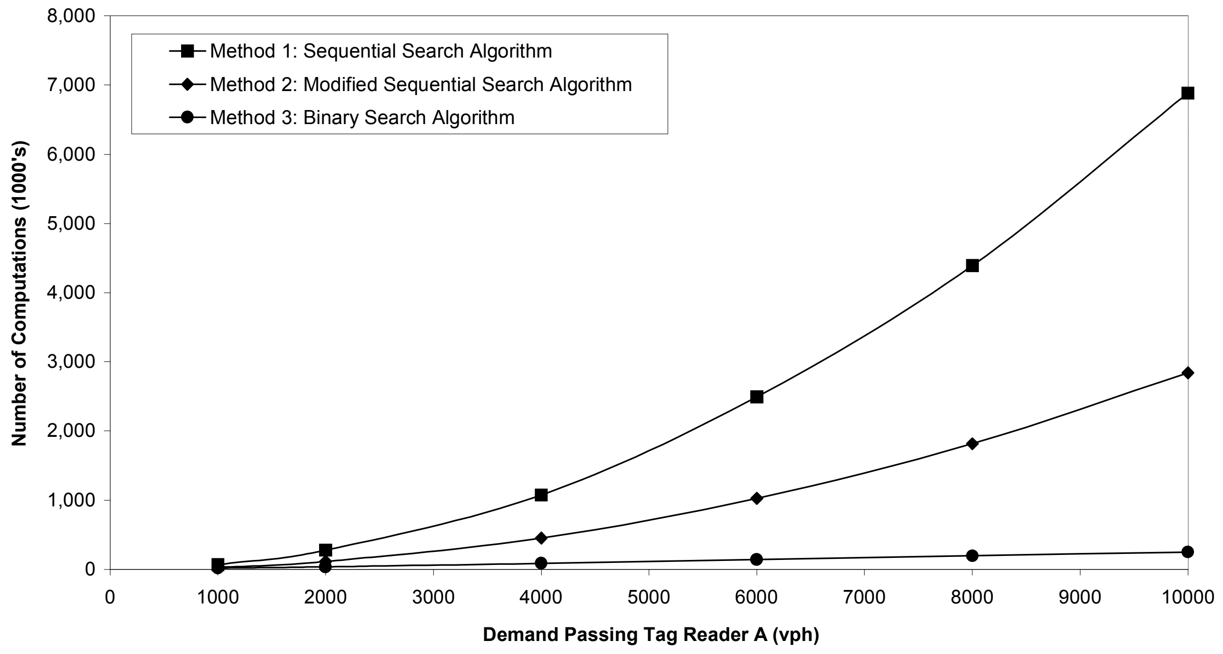


Figure 6: Computation load as a function of AVI equipped traffic demand *D*

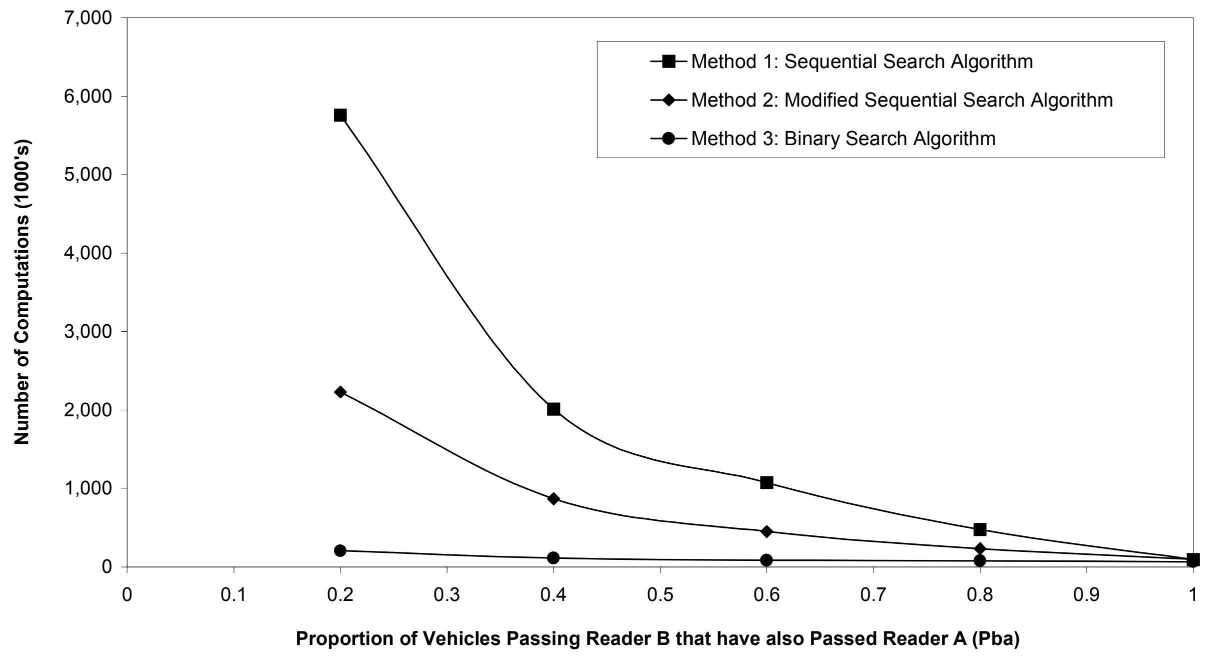


Figure 7: Computational load as a function of P_{BA}

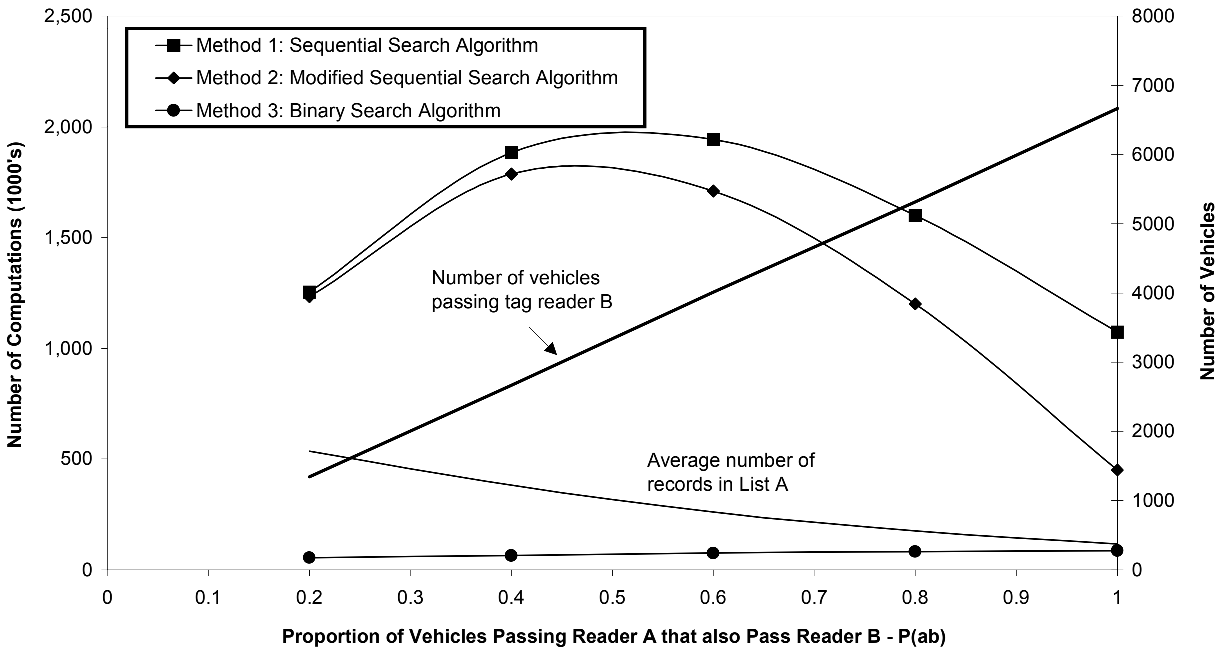


Figure 8: Computation load, stack size, and number of vehicles passing reader B as a function of P_{AB}