Commuting with Delay Prone Buses

Mayur Datar*
Department of Computer Science, Stanford University, Stanford, CA 94305, USA.

Abhiram Ranade
Department of Computer Science and Engg., Indian Institute of Technology Bombay, Powai, Mumbai 400076, India.

Abstract
What is the fastest way to go from point X to point Y within a city using its public bus service, with bus changes as necessary? If buses ran according to a schedule this is an easy query - it can be solved by using classical shortest path algorithms. Unfortunately, as everyone knows, buses in most cities hardly confirm to a fixed schedule. Once you get into a bus it usually takes a predictable amount of time to reach its destination, but the amount of time you spend waiting for it is quite random and can usually be estimated only statistically.

We present an algorithm to generate travel plans for such conditions. Our plans allow actions of the form "wait for buses with route number R1,R2 ... if you get into R1 get off at ... and wait for ..., else if you get into R2 ..." and are modeled as a tree. We guarantee that the expected time taken by the plan we generate will be minimum over all possible plans for travel between the given starting point and destination. The algorithm has been implemented for the bus system of Mumbai, and it generates the (optimal) travel plan essentially instantaneously. A transcript of a session with the program is included.

1 Introduction
In this paper we consider the consumer side planning problem for a public transport system: what is the fastest way to get from point X to point Y within a city making (bus/train) changes as necessary. The bus service and even other modes of commuter transport in many cities hardly runs according to a reliable predetermined schedule (definitely not in Mumbai!). This is because of various factors such as traffic jams and breakdowns. Once you get into a bus it usually takes a predictable amount of time to reach its destination, but the amount of time you spend waiting for it is quite random and can usually be estimated only statistically.

In this paper we present an algorithm for planning travel under such conditions. Our algorithm has been implemented for the bus system of the city of Mumbai. This is a fairly large bus system with the following figures indicating its size:

- There are 417 bus routes, with buses plying in both directions on most routes.
- There are a total of 2018 bus stops in the city.
- On the average there are 35 bus-stops on each route.
- The number of buses stopping at each bus stop is widely variable, with as many as 130 bus routes passing through some nodal points (Khodadad circle in central Mumbai).

Our program doesn’t consider other modes of public transport (such as commuter trains) available in the city, but these may be included fairly easily. Of course, we expect our algorithm to be useful for planning travel in other cities also.

We note that generating minimum time travel plans would be an easy undertaking if buses/trains reliably adhered to their published schedules (assuming such schedules are published in the first place). In that case the problem would be solved by using a standard shortest path algorithm with minor modifications. Such programs are common on the Eurail network. However, with unpredictable waiting times the problem is substantially more complex, as will be seen from the following example.

Consider a fragment of a bus network as shown in Figure 1. Suppose we wish to travel from X to Y which fall on the route of bus B1. Suppose B1 runs approximately every 30 minutes, and takes about 60 minutes to travel between X and Y. Suppose that
there is a bus $B_2$ that runs from $X$ to $Z$ which is midway between $X$ and $Y$ which runs about once every 5 minutes, and there is a bus $B_3$ which runs from $Z$ to $Y$, also once every 5 minutes. What is the best travel plan to go from $X$ to $Y$? If we opted for bus $B_1$ alone, the expected time would be about 75 minutes (15 minutes of waiting time\(^1\) and 60 minutes of running time). If we opted for bus $B_2$ followed by $B_3$, the time would be 65 minutes (2.5 minutes waiting time and 30 minutes of running time for both $B_2$ and $B_3$).

However, the second strategy alone is also clearly suboptimal. The best strategy in this case is “Wait at $X$ for either $B_1$ or $B_2$. If you get $B_1$, fine; if you get $B_2$ go to $Z$ and wait there for $B_1$ or $B_3”$. Finding such plans is clearly more complex than finding standard shortest paths.

This paper makes two main contributions. First, a model is presented in which travel plans are represented as trees. The details of this are described later; however we note that for the above example, we would have three trees (associated with the three plans: (i) Take $B_1$, (ii) Take $B_2$ and then $B_3$, and (iii) Take whichever of $B_1$ or $B_2$ that comes first, and then take $B_3$ if necessary). With each such tree our model associates a cost which is simply the expected travel time (including the waiting time) were the plan associated with the tree to be followed. Finding the fastest plan to go from point $X$ to point $Y$ is equivalent in our model to finding the least cost tree from the set of trees corresponding to all the plans for traveling between $X$ and $Y$. Our second contribution is a polynomial time algorithm for finding the least cost tree. We have implemented the algorithm for the bus system of the city of Mumbai. On standard workstations it gives optimal plans seconds after the user types in the starting and ending points.

1.1 Overview

We begin by describing our model for bus routes and travel plans in Section 2. A detailed critique of the model and possible extensions appears in Section 6. In Section 3 we describe the algorithm for generating travel plans. Our algorithm requires a database describing the details of the bus system such as frequency of each bus route, the stops on each bus route and the running time between consecutive stops on the route. Unfortunately the bus company did not supply us data this detailed. In particular, the bus company only gave us estimates of the end-to-end running times for each bus route, from which we had to infer the running time between consecutive stops on each route. The strategies we used for generating the detailed data are described in Section 4.

Section 7 sketches possible extensions to our work.

2 Model and Notation

For the rest of this paper the term bus route will mean a unidirectional sequence of bus stops. Each bus route $b$ is associated with a frequency $F(b)$; arrivals of a bus on route $b$ are assumed Poisson distributed with expected time between arrivals being $1/F(b)$. We assume that the frequency of each route is known to us.

We assume that once you get into a bus its movement happens predictably. In particular, if $s$ and $t$ are consecutive bus stops on one or more routes, then all such buses take time $R(s, t)$ to travel between them. We assume we know $R(s, t)$ for all pairs of stops that happen to be consecutive on some route. We neglect the time buses wait at stops although it can be easily incorporated in the travel time $R(s, t)$ where $s$ is where the bus stops while $t$ is the next stop on its route. Let $s_1, s_2, \ldots, s_k$ be a sequence of consecutive bus stops on a route $b$. The time to go from $s_1$ to $s_k$ having already got into bus $b$ is denoted as $R_b(s_1, s_k)$ and is of course:

$$R_b(s_1, s_k) = \sum_{i=1}^{k-1} R(s_i, s_{i+1})$$

We note that $R_b(s_1, s_k)$ need not equal $R_{b'}(s_1, s_k)$ if $b$ and $b'$ do not halt at the same stops between $s_1$ and $s_k$.

We model travel plans as trees. Each node $u$ in the tree has a label $S(u)$ which must be a bus stop. Each edge $(u, v)$ has a label $B(u, v)$ which is a bus route that passes through stops $S(u)$ and $S(v)$ in that order (not necessarily consecutively). Further we require that the root must be labeled with the

\(^1\)This really depends upon the distribution of bus arrivals.
origin stop, and every leaf with the destination stop. The action indicated by a tree node u having children \( v_1, v_2, \ldots, v_k \) is:

Wait at stop \( S(u) \) for buses \( B(u, v_1), B(u, v_2), \ldots, B(u, v_k) \). If bus \( B(u, v_i) \) is the first to arrive, then take it to stop \( S(v_i) \). Then execute the action indicated by \( v_i \).

The plan execution begins at the root and follows a path to a leaf depending upon which buses arrive first.

2.1 Expected time of a plan

It should be clear that the expected time taken by a plan can be evaluated recursively, since subtrees of a given plan tree are themselves plan trees. Let \( T(u) \) denote the expected time to reach the destination starting at node \( u \) in the tree.

Let \( r \) be the root of the plan tree, and \( v_1, \ldots, v_k \) the children of the root. Let \( b_i = B(r, v_i) \). Then we note that

\[
T(r) = \frac{1}{\sum_i F(b_i)} + \sum_i \frac{F(b_i)}{\sum_i F(b_i)} \left( R_{b_i}(r, v_i) + T(v_i) \right)
\]

The first term in the time is simply the waiting time: because bus arrivals are Poisson distributed we see a combined frequency which is simply the sum of the frequencies of the buses that we are waiting for. The second term gives the time after we get into some bus. Suppose bus \( b_i \) is the first to arrive, then the time taken will be \( R_{b_i}(r, v_i) + T(v_i) \). But this happens with probability proportional to the frequency \( F(b_i) \), hence the weighting factor in the expression above.

Define the height of a plan as the height of the associated tree; the height is simply the maximum number of bus trips used in any execution of the plan.

3 Finding an optimal plan

Given the origin stop and the destination stop our algorithm finds an optimal plan of height \( H \), where \( H \) is a parameter set by the user. We do this by dynamic programming, successively constructing optimal height \( h \) plans (if any) in the order \( h = 0, 1, \ldots, H \). For \( h = 0, 1, \ldots, H - 1 \) we construct optimal plans to reach the destination from every bus stop. Of course, height 0 plans are trivially available: the destination stop can be reached (without taking any bus) only from the destination stop itself. We describe below the process of generating optimal height \( h \) plans given optimal height \( h - 1 \) plans.

We first need the notion of a bus restricted plan. A bus restricted plan is also a tree similar to an ordinary plan, except the root \( r \) has only one outgoing edge going to some node \( u \). There is also a difference as far as the precondition and the action associated with the root. In particular, the plan can be invoked only after the user is already aboard a bus of route \( B(r, u) \) at stop \( S(r) \). The expected time for a bus restricted plan is thus \( R_{B(r,u)}(r, u) + T(u) \), where \( T(u) \) can be evaluated as given in Section 2.1.

Let \( T(s, h) \) denote the expected time to reach the destination from stop \( s \) using an optimal height \( h \) plan. Let \( T_b(s, h) \) denote the expected time to reach the destination using a bus restricted plan of height \( h \) which starts in bus \( b \) at stop \( s \). Our algorithm works on the basis of the relationships between these quantities.

First observe that if you are already in bus \( b \) at stop \( s \), the optimal height \( h \) bus restricted plan will either (i) be the same as the optimal height \( h - 1 \) bus restricted plan, or (ii) require you to get down at some bus stop \( s_i \) following \( s \) on the route of \( b \) and follow the optimal height \( h - 1 \) plan from there. Thus we get:

\[
T_b(s, h) = \min \{ T_b(s, h - 1), \min_i \{ R_{b_i}(s, s_i) + T(s_i, h - 1) \} \}
\]

The second observation is that a height \( h \) plan starting at stop \( s \) can only consist of waiting for some set of buses \( B \), and having got into one of these buses \( b \in B \), following the corresponding bus restricted plan of height \( h \). Let \( B_s \) denote the set of the buses passing through stop \( s \). Then we have:

\[
T(s, h) = \min_{b \in B_s} \left( \frac{1}{\sum_{b \in B} F(b)} + \sum_{b \in B} \frac{F(b)}{\sum_{b \in B} F(b)} T_b(s, h) \right)
\]

The first term represents the time spent waiting for the buses, and the second the expected time spent executing the bus restricted plans.
The key question then is how to find the subset \( B \). Naively, it would seem that we need to explicitly consider all possible collections of buses passing through \( s \) -- which would require \( \Omega(2^{|B_s|}) \) work. As the following lemma shows, we can get by with just \( O(|B_s| \log |B_s|) \).

**Lemma 3.1.** Let \( B_s = \{b_1, b_2, \ldots, b_m\} \) denote the set of buses passing through stop \( s \). Let \( T_{b_s} = T_{b_s}(s, h) \). Without loss of generality let \( T_{b_1} \leq T_{b_2} \leq \ldots \leq T_{b_m} \). Let

\[
T_i^* = \frac{1}{F(b_j)} + \sum_{j=1}^{i} \frac{F(b_j)}{T_{b_j}}
\]

In other words, \( T_i^* \) is the expected time if we wait for buses \( b_1, \ldots, b_i \) at stop \( s \), and thereafter follow the corresponding optimal bus restricted height \( h \) plans. Let \( l \) be smallest such that \( T_i^* \leq T_{b_{i+1}} \). Then it suffices to wait for buses \( b_1, \ldots, b_l \), and ignore the other buses even if they arrive first. In other words:

\[
T(s, h) = T_l^*
\]

**Proof.** The proof is based on the following two ideas. First, if an optimal plan involves waiting at \( s \) for bus \( b_i \), then it must also involve waiting for bus \( b_{i-1} \). In other words, suppose that the optimal height \( h \) plan involves waiting for a set \( B \) of buses at stop \( s \). Then \( B \) can only be of the form \( B = \{b_1, \ldots, b_k\} \) for some \( k \) (proved in Part 1 below). After this we just need to argue that \( T_k^* \) takes its minimum value for \( k = l \), with \( l \) as defined in the Lemma statement. (Part 2 below).

**Part 1:** Suppose \( b_i \in B \) but \( b_{i-1} \notin B \). Since waiting for the buses in \( B \) at \( s \) is optimal we know that:

\[
1 + \sum_{b \in B} \frac{F(b)T_{b}}{\sum_{b \in B} F(b)} \leq 1 + \sum_{b \in B \setminus \{b_i\}} \frac{F(b)T_{b}}{\sum_{b \in B \setminus \{b_i\}} F(b)}
\]

\[
= \frac{(1 + \sum_{b \in B} F(b)T_{b}) - F(b_i)T_{b_i}}{(\sum_{b \in B} F(b)) - F(b_i)}
\]

Noting that \( \frac{x}{y} \leq \frac{x-aq}{y-q} \Rightarrow \frac{x}{y} \geq \frac{x+aq}{y+r} \) for positive \( x, y, a, q, r \), we see that:

\[
1 + \sum_{b \in B} \frac{F(b)T_{b}}{\sum_{b \in B} F(b)} \geq \frac{1 + \sum_{b \in B} F(b)T_{b} + F(b_{i-1})T_{b_i}}{(\sum_{b \in B} F(b)) - F(b_i)}
\]

\[
= \frac{1 + \sum_{b \in B} F(b)T_{b} + F(b_{i-1})T_{b_i}}{\sum_{b \in B} F(b)}
\]

since \( T_{b_i} \geq T_{b_{i-1}} \). But the right hand side is simply the time for the plan involving waiting for the buses in \( B \cup \{b_{i-1}\} \). Thus we may assume that the optimal plan involves waiting for buses \( b_1 \) through \( b_k \) for some \( k \).

**Part 2:** Suppose \( k > l \). We show that \( T_k^* \geq T_l^* \).

We first describe the intuition behind the argument for the case \( k = l + 1 \). We know that the time \( T_{b_{i+1}} \) needed to finish the journey even after \( b_{i+1} \) has already arrived is no less than the expected waiting and traveling time \( T_l^* \) if we wait for \( b_1, \ldots, b_l \). Thus it cannot help to take bus \( b_{i+1} \) even if it arrives earliest. In general:

\[
T_k^* = \frac{1 + \sum_{j=1}^{k} F(b_j)T_{b_j}}{\sum_{j=1}^{k} F(b_j)} \geq \frac{1 + \sum_{j=1}^{l} F(b_j)T_{b_j} + T_{b_{i+1}}}{{\sum_{j=1}^{k} F(b_j) + \sum_{j=l+1}^{k} F(b_j)}}
\]

Since \( T_{b_{i+1}} \leq T_{b_{i+2}} \leq \ldots \leq T_{b_k} \). But for positive numbers \( w, z, x, a, q, y \) we know that

\[
\frac{w}{z} \geq \frac{x + aq}{y + q} \quad \text{and} \quad a \geq \frac{x}{y} \Rightarrow \frac{w}{z} \geq \frac{x}{y}
\]

Thus we get:

\[
\frac{1 + \sum_{j=1}^{k} F(b_j)T_{b_j}}{\sum_{j=1}^{k} F(b_j)} \geq \frac{1 + \sum_{j=1}^{l} F(b_j)T_{b_j}}{\sum_{j=1}^{l} F(b_j)}
\]

by choosing:

\[
w = 1 + \sum_{j=1}^{l} F(b_j)T_{b_j}
\]

\[
z = \sum_{j=1}^{l} F(b_j)
\]

\[x = 1 + \sum_{j=1}^{l} F(b_j)T_{b_j}
\]

\[a = T_{b_{i+1}}\]
26

\[ q = \sum_{j=t+1}^{k} F(b_j) \]
\[ y = \sum_{j=1}^{k} F(b_j) + \sum_{j=t+1}^{k} F(b_j) \]

and observing that \( a = T_{k+1} \geq T_k^* = x/y \). But (3.2) simply asserts \( T_k^* \geq T_l^* \). In a similar manner we can show that this holds even for \( k < l \).

3.1 Algorithm Execution Time Analysis

Our algorithm for generating height \( h \) plans given height \( h-1 \) plans is as follows.

1. The times for the bus restricted plans are generated first, using equation 3.1. For each bus route \( b \), these are best computed in reverse order of the stops on route \( b \); if \( s_i \) and \( s_{i+1} \) are consecutive bus stops on the route, it follows from equation 3.1 that

\[ T_b(s_i, h) = \min(T_b(s_i, h-1), R_b(s_i, s_{i+1}) + T_b(s_{i+1}, h)) \]

Thus the bus restricted plans for each bus and any starting stop on its route are computed in time proportional to the number of stops on the route.

2. Next we determine \( B \) in the manner described in Lemma 3.1. For this, the \( m \) buses stopping at \( s \) must first be arranged in increasing order of \( T_b(s, h) \). This can be done in time \( O(m \log m) \) using sorting. Then all \( T_l^* \) can be calculated in time \( O(m) \). From this we can determine \( l \). The total time is thus \( O(m \log m) \) at each bus stop.

This description only considers the computation of \( T_b(s, h) \) and \( T(s, h) \); however it may be noted that the corresponding plans can be recovered if auxiliary data structures are maintained. Also, it will be noted that the auxiliary data structures can be maintained without changing the algorithm execution time estimates given above.

Let \( C \) denote the maximum number of buses stopping at any bus stop. Let \( L \) denote the sum of the lengths of all bus routes. Then the time for step 1 above is \( O(L) \), and that for step 2 is \( O(L \log C) \).

The time for the algorithm over all phases is thus \( O(HL \log C) \).

We note that even in theory the maximum value of \( H \) is certainly no more than the number of bus routes; hence the optimal travel plan (with unbounded \( H \)) can be computed in polynomial time. However, in practice we note that the maximum number of bus changes a traveler might tolerate would be small, say 2-3 at most.

4 Preprocessing

The bus company (Bombay Electric Supply and Transport (BEST)) gave us the following data:

1. List of stops on each route in order.
2. Frequency of each bus route. In the notation of the previous section, this is \( F(b) \) for route \( b \).
3. End-to-end travel time for each bus route. If the stops on a route \( b \) are \( s_1, s_2, \ldots, s_k \), then we are given the value of \( \sum_{j=1}^{k-1} R(s_j, s_{j+1}) \).
4. Stage length data. For the purpose of calculating fares, BEST marks some of the stops on each route as stage boundary stops. BEST has information about the length of each stage, i.e. the distance between consecutive boundary stops. In other words, if \( s_g \) and \( s_g' \) are successive stage boundary stops on the route of bus \( b \) and if \( d(s_j, s_{j+1}) \) denotes the distance between stops \( s_j \) and \( s_{j+1} \), then we are given the values of \( \sum_{j=1}^{k-1} d(s_j, s_{j+1}) \), for all stages on all buses.

For the algorithm described in the last section, we need the values of \( R(s, t) \) for every pair of consecutive bus stops \( s \) and \( t \) on every route \( b \). It should be clear that the data provided by BEST is grossly inadequate for us to determine these values precisely. The question then is: of all the possible values that may be assigned to \( R(s, t) \) consistent with the data supplied by BEST, which ones should we use?

We use a two phase approach for doing this. In the first phase, we generate estimates for distances \( d(s, t) \) between consecutive bus stops \( s, t \) on every route. In the second phase, we generate estimates for \( R(s, t) \). Both phases use linear programming.

Generating distance estimates: We considered it reasonable to assume that adjacent bus stops couldn't be arbitrarily close (say no closer than some parameter \( d_{\text{min}} \)), nor arbitrarily far apart (say no farther than some parameter \( d_{\text{max}} \)). We initialized
$d_{\text{min}}$ and $d_{\text{max}}$ to values which are clearly correct, e.g. 10 meters and 100 km respectively. This gives us inequalities of the form $d_{\text{min}} \leq d(s,t) \leq d_{\text{max}}$. We added these to the equations given under stage length data above and checked if resulting linear program was feasible. If the linear program was feasible, then we narrowed the gap between $d_{\text{min}}$ and $d_{\text{max}}$ and tried again. In this manner, the feasible solution for the closest $d_{\text{min}}$ and $d_{\text{max}}$ was found and selected as the estimate.

**Generating time estimates:** We considered it reasonable to assume that bus speeds are bounded, say between $v_{\text{min}} = 10$ kmph and $v_{\text{max}} = 60$ kmph. This gives us inequalities of the form $d(s,t)/v_{\text{max}} \leq R(s,t) \leq d(s,t)/v_{\text{min}}$. As before we try to tighten these bounds while ensuring that the linear program remains feasible. Then we use the solution obtained for tightest program possible – this corresponds to saying that within a city we don’t expect bus speeds to vary too much. We feel that this assumption is reasonable in absence of additional information.

### 4.1 Problems due to data inconsistencies

It turned out that the data given to us by BEST also had inconsistencies, i.e. the system of equations given to us (without the inequalities we inserted) didn’t have a solution. We tried to get these resolved by reporting them back to BEST and requesting correct data; however because of various reasons this didn’t always work. So we had to resort to strategies to work around inconsistencies.

Our approach was to incorporate an error variable into each equation. We then added in the inequalities as described above and attempted to solve the resulting program, this time looking for a solution with small total error. We looked at the individual error values and this on occasions alerted us to obvious and correctable errors in the data (e.g. the end to end timing for some route being an absurdly small value). But in the end we were not able to correct all errors and had to live with some inconsistencies. We expect that these are all teething troubles, soon enough we will get consistent data from BEST so that we don’t have to worry about this issue.

![Figure 2: Sample session with Mumbai Navigator](image)

### 5 Implementation and Sample Output

We have coded the algorithm and strategies described above (about 2500 lines of C). The user interface is currently text-based, though a menu-driven Java implementation is also almost ready. The C version has been tested substantially, and we plan to release it to general users shortly. Figure 2 shows a sample session with the program, which we call Mumbai Navigator.

In the sample session the program is asked to plan a journey analogous to the example in Figure 1. The journey is to start at I. I. T. Market and end at Vashi Bus Station (typed by the user after the prompt "$->")

The program can generate plans of arbitrary height, the output only shows plans of height 1 (0 changeovers) and height 2 (1 changeovers). For each plan the total travel time is given, as well as its breakup into the waiting time for the first bus and the time for the rest of the journey. Notice the relatively large waiting time in the 0 changeover plan (30.5 minutes).

The program print outs the generated plan tree in a depth first manner, with indentation to aid readability. Some amount of common subexpression elimination is also done. For example, if $v$ and $w$
are the children of node $u$ in the plan tree with $S(v) = S(w)$, then we are guaranteed that the subtree beneath $v$ and $w$ are identical. Thus in the printout, we only give one of the trees, say the one below $v$, and set the new label for $(u, v)$ to be the concatenation of the old labels $B(u, v), B(u, w)$.

Each step of the printed plan is of the form

\[ i: \text{stop1 to stop2 by busno1, busno2, ...} \]

Here $i$ indicates the level number (root is 1). Thus in the plan shown, there are 3 lines indicating the action at the root: wait for buses 459 ltd. through 396 ltd., buses 336 or 392, or bus 710 limited. If you get into bus 710 (last line) you can directly take it to Vashi Bus Station. If you get into one of the buses 459 limited through 396 limited, then change at Gandhinagar into buses 511 limited or 710 limited which will take you to Vashi Bus Station. Else if at I.I.T. market you got into 336 or 392, you would take it to Hindustanco and then change to 511 limited or 710 limited.

Our program allows the users to discourage changeovers by setting a parameter called changeover penalty. This is simply the additional time deemed spent in making each changeover. In the printout shown the changeover penalty was set to 0. If for example we set it to 22 minutes the program would have rejected the plans involving a changeover as suboptimal, since in the present case the time saved with the change over is less than 22 minutes.

In addition, the program incorporates features such as bus stop name completion, so the valid completions of whatever the user types are printed by the program and selected if there is a unique completion.

6 A Critique of our basic model

Perhaps the most unfounded assumption in our model is that of Poisson arrivals of the buses. This is only an engineering approximation: the bus company clearly must make efforts to keep buses on a route running reasonably regularly (with fixed separation between consecutive trips perhaps). Yet, commuter experience shows that bus arrivals are hardly periodic, a bus with official frequency 30 minutes often does not arrive for an hour or longer, and occasionally, even two buses of the same route might arrive in quick succession.

The operational reason for assuming Poisson arrivals is that this provides a rule for estimating waiting time when we are waiting for arrival of a bus on any of a given set of routes. If we can be given a suitable rule for estimating the waiting time for other distributions, we expect that should be adequate for our purposes. We speculate that the combining rule for other distributions will be very similar to the one we use.

Our assertion that "it usually takes a predictable amount of time (for a bus) to reach its destination once you have climbed aboard, but the amount of time you spend waiting for it is random" is also an engineering approximation. A bus maybe delayed when it is not traveling (i.e. between trips) because of delays in arrival of drivers etc. However, traffic jams and such will delay buses also while traveling. Note however, that the delay seen while waiting for a bus is the cumulative delay (say from the start of the day), while the increase observed in the running time for single journey is the incremental delay experienced in that portion of the journey alone. Thus we expect it to be small.\footnote{If such delays are also Poisson distributed, we can actually include them in our model simply by adding a (statically predictable) term to each $R(s, t)$.}

Also note that such delays will affect every plan. Thus the delays incurred while traveling will very likely not affect the choice among different plans.

Our model glosses over some other details, but these could be modeled if we had more information. For example, our program assumes that you can always get into a bus when it arrives, i.e. it always has space. This could be modeled by using a bus frequency that was smaller than the published frequency – the difference would have to depend upon your knowledge of how full the buses run. Further, our program only has one frequency associated with each bus, whereas bus frequencies drop off at non-peak hours. This could be solved by having a different database for peak and non peak hours. Another simplification made in our program is that if several buses halt at a given bus stop, the physical locations (i.e. bus stop sheds) where they halt are the same, or at least close by. This assumption is unlikely to be satisfied in several nodal bus stops. If we knew the precise shed structure then we could model the sheds as different logical bus stops. The walking time between sheds could be modeled by a bus with infinite frequency and running time the same as the time...
required to walk between the sheds.

7 Future Work

Our program could be extended in several ways.

Other modes of transport need to be included. Most cities (including the city of Mumbai) have several commuter train lines, with the bus service also serving train stations. A complete journey planner which includes trains and buses would be very useful.

Currently, our program does not consider travel cost. It would be useful to include this. For example, common commuter queries could be: “Tell me the cheapest way to go to X from Y.”, or “Tell me the fastest way under Rs. 20”. Or some combination of time and cost might also be minimized.

Our program only plans journeys between bus stops. Thus a commuter must know the name of the bus stops near his origin and destination. It would be very useful to provide a graphical input system in which the user indicates the origin and the destination on a map, and the system tells him the bus stops. A complete system would also have to be knowledgable about landmarks in the city.