Mumbai Navigator

Abhiram Ranade* M. Srikrishna† K. Tilak M. Datar

Abstract

Our program *Mumbai Navigator* plans travel within the city of Mumbai using public buses and local trains. It has been available to users over the internet for about 4 years, and the initial experience appears positive.

The program takes as input the starting point of the journey and the desired destination, and generates a plan describing which buses/trains to take, where to change. The expected travel time including the time spent in waiting for buses/trains to arrive is also given. The generated plans are adaptive, i.e. they may prescribe different set of actions depending upon which bus arrives first. Finally, under a reasonably simple model of bus arrivals, it is possible to show that the plans generated by the program require the minimum total travel time in an expected sense.

For generating good plans, we need to have detailed data about the time required to travel from every stop to the next on each route. Such data is available for trains, but typically not for buses. For such cases we need to use some innovative interpolation procedures for generating this data that from the minimal data that we expect will be possible to get from the bus company. We describe our experience in this regard for the city of Mumbai.

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.

Chirqui and Robillard[3] present an algorithm for planning travel under such conditions. We have implemented this algorithm for for the bus and local train system of the city of Mumbai. This is a fairly large transport system with the following figures indicating its size:

- There are about 400 bus routes, with buses plying in both directions on most routes.
- There are about 2000 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).
- There are 96 local train stations including some in the distant suburbs.
- There are more than 30 different patterns of train services.

Our program doesnt consider other modes of public transport (such as taxis) available in the city.

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, and in some European cities where public transport apparently runs punctually[1, 8]. However,

*Department of Computer Science and Engg., Indian Institute of Technology Bombay, Powai, Mumbai 400076, India. `ranade@cse.iitb.ernet.in`

†Persistent Systems Pvt. Ltd., Pune. `mutnuru_msk@yahoo.com`

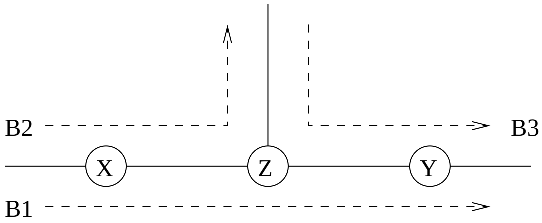

Figure 1: A sample bus network

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 B_1. Suppose B_1 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 *adaptive* plans is clearly more complex than than finding plans when buses/trains run punctually.

1.1 Overview

We begin in Section 2 by describing how Chirqui and Robillard[3] model the travel planning problem for bus networks. This model is clearly an engineering approximation which appears to be quite useful in practice. In Section 3 we describe the algorithm[3] for generating travel plans.

The 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. We expect that in most other cities in India the situation would be quite similar: we will not have detailed timing information, but only coarse information as above. Hence we feel that the strategies we used for generating the detailed data will be of general interest (Section 4).

Chirqui and Robillard proposed the model mainly in the context of buses, and in extending it to trains some additional issues need to be considered. This is discussed in Section 5.

Section 6 describes a session with the program, and includes some screenshots from the execution. Section 7 describes our experience with the program and the feedback we have obtained from users. Section 8 sketches possible extensions and concludes.

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[2] with expected time between arrivals being $1/F(b)$. We assume that the frequency of each route is known to us.

Chirqui and Robillard's model assumes 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:

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

[1] This really depends upon the distribution of bus arrivals.

[2] The reader may wonder whether assuming random arrivals is appropriate for trains. We will consider these issues in Section 5. For now only consider buses.

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.

Travel plans are modelled 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 it is required that the root must be labelled with the 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

$$T(r) = \frac{1}{\sum_i F(b_i)} + \sum_i \frac{F(b_i)}{\sum_i F(b_i)} (R_{b_i}(r, v_i) + T(v_i))$$

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 $R_{b_i}(r, v_1) + T(v_i)$. 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 Chirqui and Robillard's algorithm[3] finds an optimal plan of height H, where H is a parameter set by the user. This is done 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:

(1)

$$T_b(s, h) = \min(T_b(s, h-1), \min_i \{R_b(s, s_i) + T(s_i, h-1)\})$$

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:

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

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 subsets of buses passing through s. There are $2^{|B_s|}$ such subsets, and enumerating them all would be too much work. But this is not necessary, as the following observation shows.

Lemma 1 *Let $B_s = \{b_1, b_2, \ldots, b_m\}$ denote the set of buses passing through stop s. Let $T_{b_i} = T_{b_i}(s,h)$. Assume that $T_{b_1} \leq T_{b_2} \leq \ldots \leq T_{b_m}$ (if not, sort B_s and renumber). Then the optimal plan from s must involve waiting for exactly the buses $b_1, \ldots, b_k$ for some k and ignoring the other buses even if they arrive first.*

We formally prove this Lemma in Appendix A; here we only provide some informal justification. Suppose for the sake of argument that the optimal plan at s involves waiting for some bus b_i but not perhaps b_{i-1}. We know that $T_{b_{i-1}}(s,h) \leq T_{b_i}(s,h)$ and this can be thought of as saying: "if both b_{i-1} and b_i arrive simultaneously, then you expect to reach earlier if at all by taking b_{i-1} rather than b_i". Given this, if b_{i-1} arrives alone there should be no reason to reject it! But this only shows that the optimal plan must also include b_{i-1}.

Thus, we dont need to examine $2^{|B_s|} = 2^m$ different combinations of buses, but only m combinations: every prefix of the sequence $b_1, \ldots, b_m$. The following Lemma characterizes the optimal combination further.

Lemma 2 *Suppose B_s is as above. Let*

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

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 $T_l^* \leq T_{b_{l+1}}$. Then it suffices to wait for buses $B = \{b_1, \ldots, b_l\}$, and ignore the other buses even if they arrive first. In other words:

$$T(s,h) = T_l^*$$

The proof is in Appendix A.

3.1 Algorithm summary

The 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 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 1 that

   (2)
   $$T_b(s_i,h) = \min(T_b(s_i,h-1), R_b(s_i,s_{i+1}) + T_b(s_{i+1},h))$$

   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 Lemmas 1,2. For this, the m buses stopping at s must first be arranged in increasing order of $T_{b_i}(s,h)$. This can be done in time $O(m \log m)$ using sorting. Then all T_i^* 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 $O(HL \log C)$.

In practice H is likely to be at most 2-3, since no traveller would like to change buses more often than that. So it will be seen that the overall complexity is quite small. In fact, in practice, the program generates the plans essentially instantaneously after the user gives the origin and destination.

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 $\sum_{j=s_g}^{s_{g'}-1} d(s_j, s_{j+1})$, 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 couldnt be arbitrarily close (say no closer than some parameter $d_{\min}$), nor arbitrarily far apart (say no farther than some parameter $d_{\max}$). We initialized $d_{\min}$ and $d_{\max}$ to values which are clearly correct, e.g. 10 meters and 100 km respectively. This gives us inequalities of the form $d_{\min} \leq d(s,t) \leq d_{\max}$. 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 $d_{\min}$ and $d_{\max}$ and tried again. In this manner, the feasible solution for the closest $d_{\min}$ and $d_{\max}$ was found and selected as the estimate.

Generating time estimates: We considered it reasonable to assume that bus speeds are bounded, say between $v_{\min} = 10$ kmph and $v_{\max} = 60$ kmph. This gives us inequalities of the form $d(s,t)/v_{\max} \leq R(s,t) \leq d(s,t)/v_{\min}$. 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 dont expect bus speeds to vary too much. We feel that this assumption is reasonable in absence of additional information.

4.1 Inconsistent data

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) didnt 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 didnt 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 dont have to worry about this issue.

4.2 Express Buses

In our initial implementations we did not pay any special attention to express buses (which stop at only a limited number of stops and hence are also called *limited* buses), and this caused a problem which can be called *violation of triangle inequality.*

We explain this with an example. The bus stops

s : `dockyardcolony(kanjurmarg)`

t : `humatalkies(dockyard)`

u : `gandhinagar(kanjurmarg)`

are consecutive on the routes of Bus 425. Bus 425 ltd. on the other hand does not halt at `humatalkies(dockyard)` but goes directly from `dockyardcolony(kanjurmarg)` to `gandhinagar(kanjurmarg)`. Clearly, we must have

$$R(s,u) \leq R(s,t) + R(t,u)$$

i.e. the time for the limited bus should not be more than the time for the ordinary bus. Our initial implementation did not enforce these conditions explicitly – and as the reader will note, they are not implied in any of the conditions described earlier. Indeed, our initial estimation procedure generated estimates which violated the above inequalities sometimes. This would cause ordinary buses to be preferred in planning, and is clearly incorrect.

So we add in such inequalities explicitly into the linear programs wherever the routes of limited buses overlaps with those of ordinary buses.

5 Local Trains

We have included trains in our program by treating them as buses, and by treating stations similarly as bus stops. It maybe argued that trains should expected to run more punctually, and to that extent, our treatment is an engineering approximation. Do note however, that even trains in a city like Mumbai are often delayed (some of the reasons for this are the heavy traffic including long distance train traffic which propagates delays from elsewhere into the suburban local system).

The train system and the bus system needs to be connected together. We accomplish this by using a special *walking connection.* A walking connection from A to B is modelled as a bus with infinite frequency (you can start walking anytime, without waiting!) and with travel time equal to the time required to walk from A to B. Most railway stations have bus stops close by – in such cases we add a walking connection between the two.

6 Implementation and Sample Output

The ideas described above have been implemented as Java programs. The main Navigator program runs as a servlet and can be accessed over the internet through the URL `www.cse.iitb.ac.in/navigator`. Figure 2 shows the main screen as would be seen in a popular browser[3].

The traveller selects the journey starting point by selecting an item under the menu "Origin". The destination is likewise selected from the menu "Destinations". These menus are fairly long (more than 2000 entries for Mumbai), and can be viewed by manipulating the scrollbars by their side. Figure 2 in fact shows that `i.i.t.market` has been chosen as the origin, and `acharyaatrechowk` as the destination. After selecting these, the traveller must click the "Get plan" button, whereupon the plan shown in Figure 3 is produced.

The program first attempts to generate plans involving no bus/train changes. This may often not be possible, as in the present case. Or even when possible, there maybe faster plans if the traveller is willing to change buses/trains. So plans which respectively have at most one change, and two changes, are also printed. the program is currently configured to allow at most 2 changes; but this can be set arbitrarily large.

A travel plan is printed as a sequence of numbered steps corresponding to travel in trains or buses, possibly interspersed with a few unnumbered walking steps. A step, e.g.

```
1.  i.i.t.market to baharcinema by
396ltd., 403ltd.
```

in general has the form "`stop1` to `stop2` by ..." and gives useful ways of travelling from `stop1` to `stop2`. There may be several steps starting at `stop1`, in

[3]Other browsers also work fine.

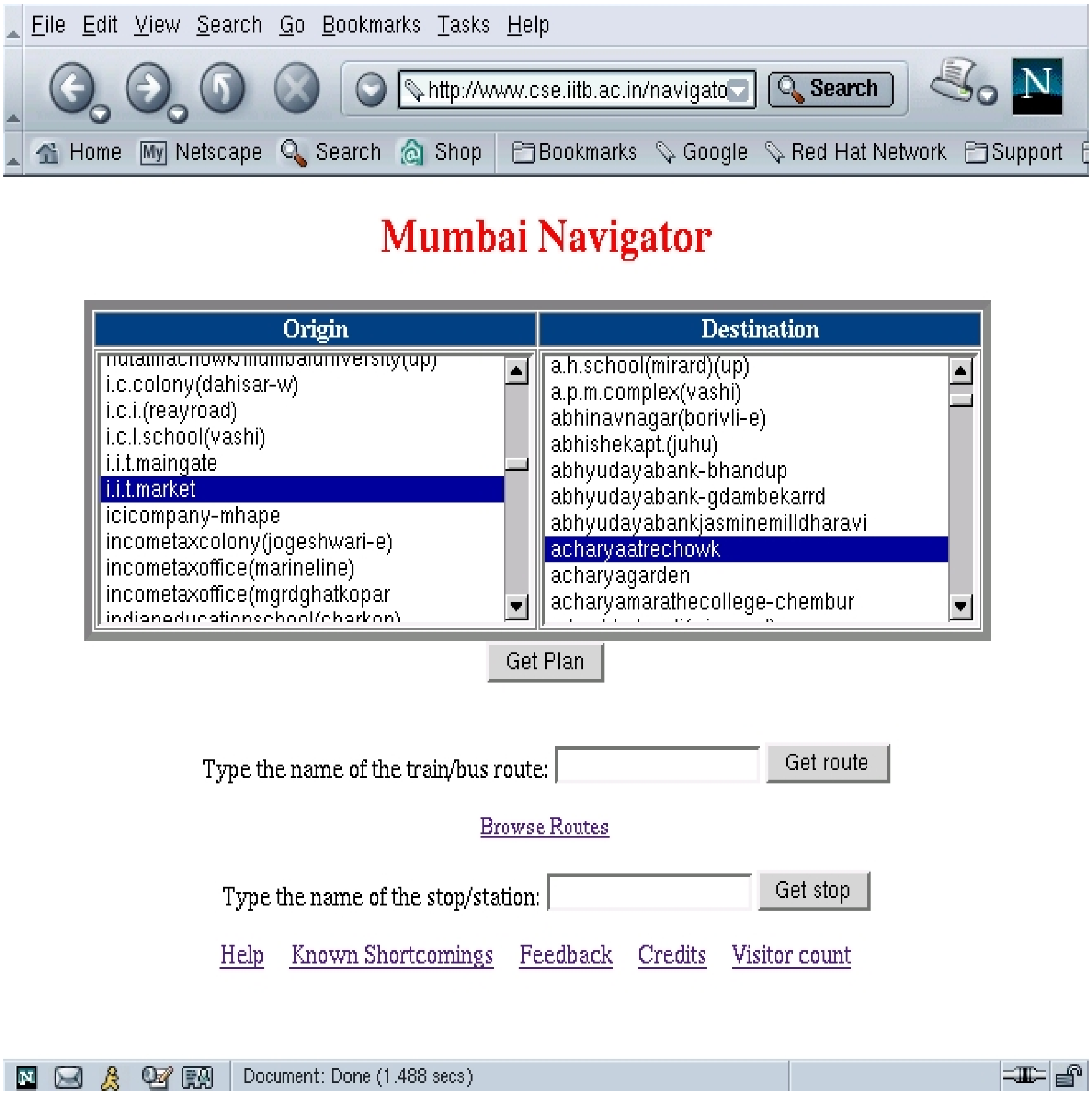

Figure 2: Mumbai Navigator Home Page

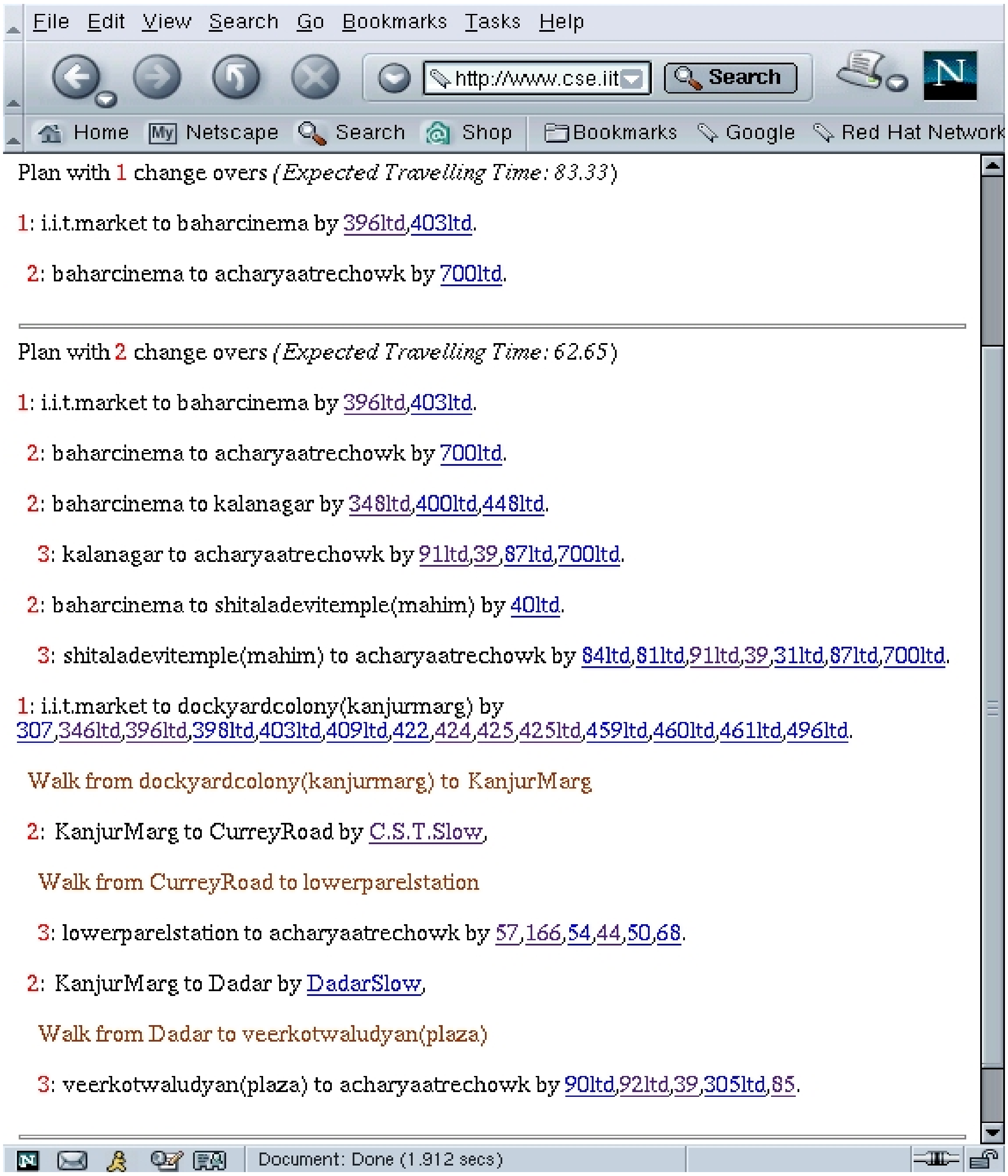

Figure 3: Plan Generated

which case the plan requires you to follow that step for which the appropriate bus arrives earliest.

For example, in the 2 change plan there are two entries in which `i.i.t.market` appears as the first stop – these require travelling either to `baharcinema`, or to `dockyardcolony(kanjurmarg)`. In this case the traveller must get into whichever bus comes first of those listed in the two entries. If one of 396ltd. or 403ltd. arrives first, then it should be taken upto `baharcinema`; if any of 307 through 496ltd. arrives first then it should be taken to `dockyardcolony(kanjurmarg)`. What does the traveller do after getting to `baharcinema`? This can be known by looking for lines of the form "`baharcinema to ...`". There are two such lines in the plan, and as before, the traveller is directed to take whichever bus that comes first out of those listed. If in the first step the traveller reached `dockyardcolony(Kanjurmarg)` then he is advised to "`Walk from dockyardcolony(Kanjurmarg) to KanjurMarg`", where `KanjurMarg` refers to the train station.

7 Experience

Mumbai Navigator has been operational over the internet for about 4 years. For most of this time, the existence of the program was informally known within the I.I.T. campus, and among the students/alumni, but not outside the I.I.T. community. There were several regular users of the program in this community.

In late July 2003, the newspaper Indian Express published a report on the program[6], and subsequently Mumbai Doordarshan also aired a short segment in their Marathi news program on August 5. After this publicity, there have been about 50,000 visitors to the program site. The program homepage also has a link using which visitors can give comments.

The comments have been overwhelmingly positive. Many improvements/suggestions have also been given. The most common one is to include maps. Maps will help in many ways: travellers can give their current position by clicking on the map rather than having to know the nearest bus stop as is the case currently. Also, it will help the traveller to see the routes be displayed on a map. Apart from these, the next important set of comments concerns the need to update the bus data base, which has changed substantially by now.

8 Concluding Remarks

There are three main areas where more work could be done: the basic model itself, the interpolation procedure, and the operational issues of maintaining/enhancing the website.

8.1 A Critique of the basic model

Perhaps the most unfounded assumption in the 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. So while anecdotal experience may indicate that a Poisson approximation is adequate, in the least it is of great theoretical interest to consider distributions. Boyan and Mitzenmacher[2] have recently considered these issues.

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 travelling (i.e. between trips) because of delays in arrival of drivers etc. However, traffic jams and such will delay buses also while travelling. 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.[4] Also note that such delays will affect every plan. Thus the delays incurred while travelling will very likely not affect the choice among different plans.

The model glosses over some other details, but

[4] If such delays are also Poisson distributed, we can actually include them in our model simply by adding a (statically predictable) term to each $R(s,t)$.

these could be modelled 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 modelled 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 nonpeak hours. This could be solved by having a different database for peak and nonpeak 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 closeby. 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 modelled by a bus with infinite frequency and running time the same as the time required to walk between the sheds.

8.2 Data Interpolation and Related Issues:

We have presented a procedure for generating stop-to-stop timings from end-to-end timings and distance estimates based on linear programming. Clearly, other approaches might also be possible, and even better.

The basic problem here is to find a suitable solution to a set of (linear) equations as discussed in Section 4. A natural solution that often works well is the *least squares solution*. The least squares criterion prefers solutions in which all values are as close as possible, over solutions in which some values are much larger than others. This is an interesting avenue for exploring further.

Since our data is noisy, it will be very useful if our preprocessing can help in identifying where the errors might be located. Especially so, since some errors could always be expected during data entry of bus information. Work is also needed here.

8.3 Maintenance and Enhancements to the website

One of the hardest tasks to undertake in an academic environment is maintenance. Unlike program development, which can be done through student projects (as we have), it is hard to find manpower for continuously liasoning with the bus company (BEST in Bombay) and updating the database used by the program. We therefore feel that for long term survival of the website, more support is needed from a governmental or commercial agency.

An obvious enhancement to the site is to include maps. Map based city transport information systems are common in many European cities[1, 8]. We have taken some preliminary steps in this direction[7], but more needs to be done.

Acknowledgements

We are grateful to Datta Ghogare and others at BEST for giving us bus related data, and also patiently answering our questions. We would like to thank Dr. Patkar of MMRDA for comments on a previous version[5] of this paper. Finally we would like to thank Jose R. Correa[4] for bringing to our attention the work of Chirqui and Robillard[3].

We would like to thank Tata Infotech Ltd. for sponsoring the work of the third author.

References

[1] http://www.vvb-fahrinfo.de/fahrinfo. Berlin City Website. In several languages including English.

[2] Justin A. Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Symposium on Discrete Algorithms*, pages 895–902, 2001.

[3] C. Chirqui and P. Robillard. Common Bus Lines. *Transportation Science*, 9:115–121, 1975.

[4] Jose R. Correa, 2001. Personal communication.

[5] M. Datar and A. Ranade. Commuting with Delay Prone Buses. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 22–29, 2000.

[6] City Guide: Use this net friend to go places. Indian Express (Mumbai Newsline), July 26, 2003.

[7] A. Kotwal. Integration of Mumbai Navigator with a G. I. S., 2002. M. Tech. Project Report, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay.

[8] http://www2.vvs.de/vvs.navi_efa. Stutgart City Website. In German.

A Proofs of Lemmas

Proof of Lemma 1: Suppose the optimal plan involves waiting for a set B of buses. 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:

$$\begin{aligned}
&\frac{1+\sum_{b\in B} F(b)T_b}{\sum_{b\in B} F(b)} \\
&\quad\le \frac{1+\sum_{b\in B-\{b_i\}} F(b)T_b}{\sum_{b\in B-\{b_i\}} F(b)} \\
&\quad= \frac{\left(1+\sum_{b\in B} F(b)T_b\right) - F(b_i)T_{b_i}}{\left(\sum_{b\in B} F(b)\right) - F(b_i)}
\end{aligned}$$

Noting that $\frac{x}{y} \le \frac{x-aq}{y-q} \implies \frac{x}{y} \ge \frac{x+ar}{y+r}$ for positive x, y, a, q, r, we see that:

$$\begin{aligned}
&\frac{1+\sum_{b\in B} F(b)T_b}{\sum_{b\in B} F(b)} \\
&\ge \frac{\left(1+\sum_{b\in B} F(b)T_b\right) + F(b_{i-1})T_{b_i}}{\left(\sum_{b\in B} F(b)\right) + F(b_{i-1})} \\
&\ge \frac{\left(1+\sum_{b\in B} F(b)T_b\right) + F(b_{i-1})T_{b_{i-1}}}{\left(\sum_{b\in B} F(b)\right) + F(b_{i-1})}
\end{aligned}$$

since $T_{b_i} \ge T_{b_{i-1}}$. 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.

Proof of Lemma 2: Suppose $k > l$. We show that $T_k^* \ge T_l^*$. We first describe the intuition behind the argument for the case $k = l + 1$. We know that the time $T_{b_{l+1}}$ needed to finish the journey even after b_{l+1} has already arrived is no less than the expected waiting and travelling time T_l^* if we wait for $b_1, \ldots, b_l$. Thus it cannot help to take bus b_{l+1} even if it arrives earliest. In general:

$$\begin{aligned}
T_k^* = \frac{1+\sum_{j=1}^{k} F(b_j)T_{b_j}}{\sum_{j=1}^{k} F(b_j)} \ge \\
\frac{1+\sum_{j=1}^{l} F(b_j)T_{b_j} + T_{b_{l+1}}\left(\sum_{j=l+1}^{k} F(b_j)\right)}{\sum_{j=1}^{l} F(b_j) + \sum_{j=l+1}^{k} F(b_j)}
\end{aligned}$$

Since $T_{b_{l+1}} \le T_{b_{l+2}} \le \ldots \le T_{b_k}$. But for positive numbers w, z, x, a, q, y we know that

$$\frac{w}{z} \ge \frac{x+aq}{y+q} \text{ and } a \ge \frac{x}{y} \implies \frac{w}{z} \ge \frac{x}{y}$$

Thus we get:

$$\begin{aligned}
&\frac{1+\sum_{j=1}^{k} F(b_j)T_{b_j}}{\sum_{j=1}^{k} F(b_j)} \\
&\ge \frac{1+\sum_{j=1}^{l} F(b_j)T_{b_j}}{\sum_{j=1}^{l} F(b_j)}
\end{aligned} \tag{3}$$

by choosing:

$$\begin{aligned}
w &= 1+\textstyle\sum_{j=1}^{k} F(b_j)T_{b_j} \\
z &= \textstyle\sum_{j=1}^{k} F(b_j) \\
x &= 1+\textstyle\sum_{j=1}^{l} F(b_j)T_{b_j} \\
a &= T_{b_{l+1}} \\
q &= \textstyle\sum_{j=l+1}^{k} F(b_j) \\
y &= \textstyle\sum_{j=1}^{l} F(b_j) + \sum_{j=l+1}^{k} F(b_j)
\end{aligned}$$

and observing that $a = T_{b_{l+1}} \ge T_l^* = x/y$. But (3) simply asserts $T_k^* \ge T_l^*$. In a similar manner we can show that this holds even for $k < l$.