Path-Planning Algorithms for Public Transportation Systems

Chao-Lin Liu, Tun-Wen Pai, Chun-Tien Chang, and Chang-Ming Hsieh

Abstract—**Computing travel plans for desired trips in public transportation systems is not exactly the same as finding a shortest driving path in a given area. Path planning in the context of public transportation systems must consider the** *route constraint* **that public vehicles serve on particular paths and that passengers cannot order the drivers to change the bus routes. Explicit representation of the route constraint helps us to design efficient algorithms that focus on viable routes for computing travel plans of interest. This paper presents two strategies for capturing the route constraint. The first strategy employs** *connectivity matrices***, and applies special properties of matrices for quickly identifying feasible travel plans for the desired trips. The second strategy uses** *hubs* **where many service routes concentrate for computing travel plans. Our algorithms perform very well in field tests.**

Keywords—**Intelligent Transportation Systems, Advanced Public Transportation Systems, Path Planning Algorithms, Constrained Programming**

I. Introduction

Advanced public transportation systems are an important component of intelligent transportation systems for reducing traffic demands. Reducing traffic demands helps to reduce traffic jams and gasoline consumption, thereby alleviating the air pollution problems. We report algorithms for providing bus information to tourists and commuters to encourage people to use public transportation systems.

Transportation authorities take a variety of measures to encourage people to share rides in urban areas. The Singapore government legislatively bans a portion of registered vehicles from entering cities on specific days. Many cities in Japan and the United States set up priority routes exclusively for public and high-occupancy vehicles. Providing transfer information is another effective measure for promoting the public transportation systems.

It is not always easy for people to use the public transportation systems to travel around a metropolis. For instance, in the New York City, the problem is not about availability of services, but about finding a travel plan for the desired trip. Finding travel plans for non-recurrent trips from a wide selection and combination of bus, subway, and train routes is not an easy job even for local people, let alone tourists. Demands for non-recurrent trips may occur when there are special events and when services for the commuters' customary travel plans are interrupted or detoured under unexpected situations.

One may tackle the path-planning problems with standard search algorithms, such as the label-setting and label-correcting algorithms [1, 9]. Planning can be carried out at the stop level where the algorithms find the best next stop, starting from the origin of the trip.[1] Public transportation systems, however, exhibit a feature useful for computing travel plans: Public buses, subway, and traditional trains follow selected routes during service. Transferring between routes costs time and money, so people typically prefer travel plans that require lesser transfers. Therefore, it may be better to cope with the path-planning problems at the service-route level rather than at the stop level. Moreover, planning at the route level can be more efficient as we show shortly.

We define some terms used in our algorithms in Section II. Section III introduces matrices for representing the route constraint, and presents an algorithm that applies the route constraint to computing travel plans. We also show properties of matrices useful for planning service routes in public transportation systems. Section IV discusses another path-planning algorithm that takes advantage of transfer hubs. We then go over some design issues for real-world applications of our algorithms, including prioritizing multiple travel plans, and conclude with a brief discussion.

Liu is with the Department of Computer Science of National Chengchi University, Taiwan. E-mail: chaolin@nccu.edu.tw; Pai and Chang are with the Department of Computer Science of National Taiwan Ocean University, Taiwan. E-mail: twp@cs.ntou.edu.tw; and Hsieh is with the Bureau of Transportation, Taichung City Government, Taiwan.

[1]When attempting to find a travel plan from the origin to the destination, there is no immediate need to distinguish bus routes, subway routes, and train routes. Hence, we will not distinguish routes and stops served by different types of vehicles until we need to prioritize travel plans.

II. Route Information

We illustrate the definitions of the basic terms used in our algorithms in a simplified context that is shown in the following figure. Each black circle represents a group of nearby stops that serve the marked locations, and directed lines represent service routes. For clarity, we do not distinguish stop names and location names until later in this paper.

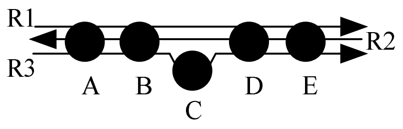


Fig. 1. Route information assignment

We assign a number to each different service route. Service routes, e.g., $R1$ and $R3$, that do not serve the same set of stops will be assigned different numbers. Service routes, e.g., $R1$ and $R2$, that serve the same set of stops but not in the same order will also get different numbers.

We assign an ordinal number to each stop on a route. Smaller numbers will be assigned to stops that are served earlier on a bus route, and we use $K(r, s)$ to denote the number assigned to the stop s on the route r. When a route r does not serve a stop s, we set $K(r, s) = 0$. Assume that the route number of $R1$, $R2$, and $R3$ are 1, 2, and 3, respectively. If $K(1, \mathsf{A}) = l$, $K(2, \mathsf{A}) = m$, and $K(3, \mathsf{A}) = n$, then we will also have $K(1, \mathsf{B}) = l + 1$, $K(2, \mathsf{B}) = m - 1$, and $K(3, \mathsf{B}) = n + 1$. Our algorithms use these numbers to determine whether one may travel from one stop to another on a particular route.

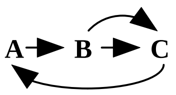
Our algorithms employ two functions that relate bus stops and bus routes. The algorithms need to know the set of service routes, $SR(s)$, that serve a given bus stop s. For instance, *SR*(B) $= \{1, 2, 3\}$ and *SR*(C) $= \{3\}$ in Figure 1. The algorithms also need to know the bus stops that are served by two routes, say $r1$ and $r2$, and we denote these common stops by *CS*$(r1, r2)$. For instance, *CS*$(1, 2) = \{$A, B, D, E$\}$. We can compute *SR*(s) and *CS*$(r1, r2)$ from $K(r, s)$:

$$\begin{aligned} \mathit{SR}(s) &= \{r | K(r, s) > 0\} \qquad \text{and} \\ \mathit{CS}(r1, r2) &= \{s | K(r1, s) > 0 \text{ and } K(r2, s) > 0\}. \end{aligned}$$

III. Path Planning with the Route Constraint

A. Adjacency Matrices

We represent the connectivity among locations in transportation networks with adjacency matrices [1]. We designate each location in the network a unique number. The value of a cell $M_{i,j}$ of an adjacency matrix M is set to the number of direct ways that we may travel from a location i to j. The following figure illustrates how we represent a simple network with an adjacency matrix T. We use $T_{1,1} = 0$, $T_{1,2} = 1$, and $T_{1,3} = 0$ to respectively encode the facts that we cannot go from A to A directly, that we can go from A to B directly, and that we cannot go from A to C directly.



$$T = \begin{array}{c|ccc} & \mathbf{A} & \mathbf{B} & \mathbf{C} \\ \hline \mathbf{A} & 0 & 1 & 0 \\ \mathbf{B} & 0 & 0 & 2 \\ \mathbf{C} & 1 & 0 & 0 \end{array} \quad T^2 = \begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad T^3 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Fig. 2. A bus route segment and its adjacency matrices

Figure 2 shows the square and cube of the connectivity matrix T, and we can verify that each cell in these matrices indicates the number of ways that we may travel from one location to another by respectively two and three transitions. In fact, we can show by induction that the k^{th} power of the matrix T encodes the connectivity of the network when we take k transitions, assuming that we employ the following standard method for matrix multiplication.

$$T_{i,j}^{k+1} = \sum_l T_{i,l}^k * T_{l,j} \tag{1}$$

Property 1: Let T represent the adjacency matrix for a public transportation network. $T_{i,j}^k$ will encode the number of ways to go from location i to j by k transitions.

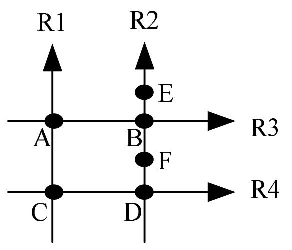
Proof. By construction, we have set values in the adjacency matrix, T, to the number of ways to go from one location to another. Assuming that the values of the k^{th} power of T represent the number of ways to go from one location to another by k transitions, we show that the $(k+1)^{th}$ power of T must represent the number of ways to go from one location to another by $k+1$ transitions. In (1), $T_{i,l}^k$ represents the number of ways to go from location i to an intermediate location l by k transitions, and $T_{l,j}$ the number of ways to go from l to j by one transition. Therefore, $T_{i,l}^k * T_{l,j}$ is the number of ways to go from i to j via a particular l by $k+1$ transitions. As a result, $\sum_l T_{i,l}^k * T_{l,j}$ is equal to the total number of ways to travel from location i to j by $k+1$ transitions. ■

Applying this property, the problem of determining if there are ways to travel from the origin O to the destination D is to find a k such that $T_{i,j}^k > 0$. The path that corresponds to the smallest k among all such ks requires the least number of transitions between locations. Although adjacency matrices are useful, they are not good enough for tackling path-planning problems in public transportation systems for the potentially humongous computational costs.

B. Connectivity Matrices

We extend the fundamental idea of adjacency matrices to representing *connectivity among service routes* in public transportation systems. Let i and j be the numbers of two service routes. We set a cell, say $T_{i,j}$, of a *connectivity matrix* T to the number of stops in *CS*(i, j). Namely, $T_{i,j}$ is the number of ways that we can transfer from route i to j. We set $T_{i,i}$ to 0, although route i and itself have common stops. Analogous to adjacency matrices, connectivity matrices are related to the number of alternatives for transferring between routes.

However, this property demands a careful inspection because representing connectivity among service routes is not exactly the same as representing connectivity among locations. Consider the network shown in the following figure. We assume that we



$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 2 & 2 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{bmatrix}$$
$$\mathrm{T} \qquad\qquad\qquad \mathrm{T}^2$$

Fig. 3. Connectivity matrices

may transfer from one route to another at intersections. The information encoded in $T_{1,4}$ can be misleading in that it suggests that we may directly transfer from route $R1$ to $R4$ no matter where we catch a bus on route $R1$. In fact, we *cannot* transfer directly from $R1$ to $R4$ once we pass C on $R1$. Also, that $T_{1,2}^2 = 2$ suggests that we may travel from any locations served by $R1$ to those served by $R2$ by two transfers. This is not exactly right. Although we may go to E from C by transferring from $R1$ to $R3$ at A and from $R3$ to $R2$ at B, we cannot go to F from C via $R1$, $R3$, and then $R2$. Furthermore, notice that we cannot transfer from $R2$ to $R1$ at all, but we have $T_{2,1}^2 = 2$.

The key is that being able to transfer from a route X to another route Y does not guarantee that we may travel from all locations served by X to all locations served by Y. Computing the powers of connectivity matrices with Equation (1) ignores the service directions of the routes and locations of stops on the routes, and consequently exaggerates the number of ways that one may travel from one location to another.

We can make the information encoded in T^2 precise by calculating T^2 with an improved mechanism. In reality, there is exactly one way to transfer from route i to j via route r for each unique pair of stops $s \in$ *CS*(i, r) and $t \in$ *CS*(r, j) such that the following condition holds.

$$K(r, s) < K(r, t) \tag{2}$$

Applying (2), we obtain the following T^2 that precisely prescribes what we could do by two transfers in Figure 3.

$$T^2 = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix} \tag{3}$$

Notice that we need to check the feasibility of travel plans even if we compute connectivity matrices precisely. There are three possible ways to transfer from route i to j via r in the following figure, i.e., transferring at location pairs (B, G), (B, H), and (D, H). Not all of them may constitute feasible travel plans for all needs to travel from locations served by i to locations served by j. If we are to travel from A to H, all three alternatives

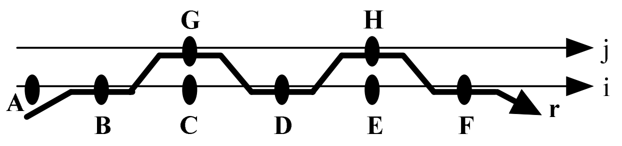


Fig. 4. A difficult example for computing $T^2_{i,j}$

will work. Only one of these alternatives is right for travelling from C to H because we cannot travel backwards from C to B on i, as is implied by $K(r, \mathsf{B}) < K(r, \mathsf{C})$. Consequently, when the connectivity matrix T^2 suggests that we may transfer from one service route to another, we need to double check the K values of the origin, the destination, and the transfer locations to make sure the travel plan is feasible.

Our algorithms also employ T^3 for the path-planning problem. Applying (1) to compute T^3 from T^2 and T is again problematic, and the networks shown in the following figure illustrate the problems. We have one way to transfer from i to $r2$ by two transfers and one way to transfer from $r2$ to j, but *no* way to transfer from i to j via $r1$ and $r2$ by three transfers.

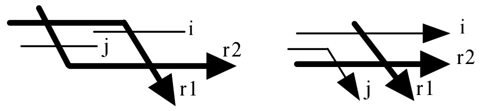


Fig. 5. Difficult examples for computing $T^3_{i,j}$ and $CR_2(i,j)$

Since we would like to set $T^3_{i,j}$ to the number of ways to transfer from route i to j by three transfers, we should increase $T^3_{i,j}$ by 1 for each unique way to transfer from route i to j via a pair of routes, say $r1$ and $r2$. Hence, we should increase $T^3_{i,j}$ by 1 for each unique combination of routes $r1$, $r2$, and stops $x \in CS(i, r1)$, $y \in CS(r1, r2)$, and $z \in CS(r2, j)$ such that the following condition holds.

$$K(r1, x) < K(r1, y) \text{ and } K(r2, y) < K(r2, z) \tag{4}$$

Using this criterion, we make $T^3_{i,j} = 0$ for both cases shown in Figure 5.

Analogous to T^2, a precisely computed T^3 may still overestimate the actual ways to travel from a stop to another via three transfers. In fact, that $T^k_{i,j} = \alpha$ implies that there are *at most* α ways to transfer from stops served by route i to stops served by j by exactly k transfers. Taking advantage of these observations, we can prove the following property that is useful for planning service routes in designing service routes for a public transportation system. The proof is analogous to that for Property 1.

Property 2: Let T represent the connectivity matrix for a public transportation system. Assume that there is no service route running from a location x to another location y. If there exists an integer k such that $\sum_{l=1}^{k} T^l_{i,j} = 0$ for all $i \in SR(x)$ and $j \in SR(y)$, then travelling from x to y requires no less than k transfers.

If we do not want people to transfer more than twice to travel between locations, we must at least make such a k smaller than 3 for all location pairs served by the public transportation system.

The connectivity matrices capture the route constraint on how one may travel from the origin to the destination. Using this information, our search algorithms reduce the amount of search work that would be needed otherwise. Overestimating ways of transferring between routes may offset the improvement in efficiency, but this will not make our algorithms slower than traditional uninformed search algorithms. We apply the connectivity matrices and *connecting-route* functions to path-planning problems next.

C. Path-Planning Algorithm

For a pair of routes, i and j, our algorithms employ connecting-route functions $CR_1(i,j)$ and $CR_2(i,j)$ that relate i and j to route numbers of the routes that provide ways to transfer from i to j. $CR_1(i,j)$ represents the route numbers of the routes that directly run from some stops on route i to some stops on route j. When $T^2_{i,j} > 0$, the algorithm checks $CR_1(i,j)$ for detail information about how to transfer from route i to j. We compute $CR_1(i,j)$ as follows:

$$\{r | T_{i,r} * T_{r,j} > 0 \text{ and } \min_{s \in CS(i,r)} K(r,s) < \max_{t \in CS(r,j)} K(r,t)\}. \tag{5}$$

Applying (5) to the example shown in Figure 3, $CR_1(1,2)$ will be set to $\{3,4\}$, and both $CR_1(1,1)$ and $CR_1(2,1)$ will be set to empty sets. Notice that the second condition in (5) ensures that $CR_1(i,j)$ does carry correct information. If we drop this condition, we would obtain $CR_1(1,1) = \{3,4\}$ and $CR_1(2,1) = \{3,4\}$. It is easy to verify that $CR_1(1,1) = \{3,4\}$ suggests impractical travel plans and that $CR_1(2,1) = \{3,4\}$ suggests impossible travel plans.

$CR_2(i,j)$ relates routes i and j to a pair of routes $(r1, r2)$ that will connect some stops on i to some stops on j. Specifically, we can transfer from i to $r1$, from $r1$ to $r2$, and from $r2$ to j. We can compute $CR_2(i,j)$ precisely with (4), and only those pairs $(r1, r2)$ that meet the requirement specified in (4) are included in $CR_2(i,j)$.

C.1 The Algorithm

We present the algorithm for finding travelling plans that require no more than three transfers next. In the following algorithm, we have assumed that $i \in SR(O)$ and $j \in SR(D)$.

```
Algorithm PathPlanning(Route Information, Origin $O$,
    Destination $D$)
  1. If $O = D$, there is no need to commute.
  2. Direct connection:
     For any route $r \in SR(O) \cap SR(D) \neq \emptyset$, if $K(r,O) <$
     $K(r,D)$, we can go from $O$ to $D$ by $r$.
  3. One transfer:
     If $T_{i,j} \geq 1$, $K(i,O) < K(i,s)$ and $K(j,s) < K(j,D)$
     for a stop $s \in CS(i,j)$, we can take $i$ at $O$, transfer
     from $i$ to $j$ at $s$, and get to $D$ by $j$.
  4. Two transfers:
     If $T^2_{i,j} \geq 1$ and there exist stops $s \in CS(i,r)$ and $t \in$
     $CS(r,j)$ for a route $r \in CR_1(i,j)$ such that
       (a) $K(i,O) < K(i,s)$,
       (b) $K(r,s) < K(r,t)$, and
       (c) $K(j,t) < K(j,D)$,
     then we can take $i$ at $O$, transfer from $i$ to $r$ at $s$, trans-
     fer from $r$ to $j$ at $t$, and get to $D$ by $j$.
  5. Three transfers:
     If $T^3_{i,j} \geq 1$ and there exist stops $x \in CS(i,r1)$, $y \in$
     $CS(r1,r2)$, and $z \in CS(r2,j)$ for a pair of routes
     $(r1,r2) \in CR_2(i,j)$ such that
       (a) $K(i,O) < K(i,x)$,
       (b) $K(r1,x) < K(r1,y)$,
       (c) $K(r2,y) < K(r2,z)$, and
       (d) $K(r2,z) < K(r2,D)$,
     then we can take $i$ at $O$, transfer from $i$ to $r1$ at $x$,
     transfer from $r1$ to $r2$ at $y$, transfer from $r2$ to $j$ at $z$,
     and get to $D$ by $j$.
```

The first step checks if the origin and the destination are the same location, and is included for completeness of the algorithm. The second step examines if both the origin and the destination are served by a service route r. If yes, we have to examine if this route runs from the origin to the destination.

The third step looks for travel plans that require one transfer from a route i to a route j. If $T_{i,j}$ is positive, then we may transfer from route i to j. We must then check the location of the transfer stop s, and make sure that it is possible to travel from O to s by route i and from s to D by route j by comparing the involved K values.

The fourth and the fifth steps look for travel plans that require two and three transfers, respectively. To find plans that require k transfers, we make sure that $T^k_{i,j}$ is positive, look into $CR_{k-1}(i,j)$ for possible connecting routes, and examine if there are locations where we can transfer. In Figure 3, $T^2_{1,2}$ indicates that we can transfer from route $R1$ to $R2$ by two transfers. The algorithm determines that we can travel from C to E via A and B. The algorithm will also find that we cannot travel from C to F via A and B because $K(2,\mathsf{B}) \not< K(2,\mathsf{F})$.

The algorithm relies on the K values of stops for checking the feasibility of travel plans. The assignment of K values implicitly assumes that all service routes are directed and acyclic. There is no need to compare K values of two stops on a route r when the path of r forms a loop, since we can go between any stops on such a route.

Using the same design principle, we can extend the algorithm to find travel plans that require more than three transfers. Hence, the algorithm is *complete* in the sense that it is guaranteed find a travel plan suitable for the desired trip when there is one [9]. In addition, because the algorithm starts its search for travel plans from those that require lesser transfers, the algorithm is also *optimal* in the sense that it can find the travel plans that require the least number of transfers.

Property 3: *PathPlanning* is both complete and optimal for searching travel plans that require the least number of transfers.

C.2 Implementation Issues

Notice that we only check whether $T^2_{i,j}$ and $T^3_{i,j}$ are positive at steps 4 and 5, respectively. Whenever these numbers are positive, the algorithm checks if there are feasible travel plans by comparing K values of locations that are involved in the plans. Since $T^k_{i,j}$ overestimates the actual number of ways to transfer from route i to j by exactly k transfers, we are not required to compute exact values in T^k when we are preparing T^k for the algorithm. We will only make the algorithm run slower than it should be, even if we make $T^k_{i,j}$ positive when it is actually zero. The algorithm will not find any travel plan after comparing the K values. With the same token, the information contained in $CR_2(i,j)$ does not have to be precise. Imprecise $CR_2(i,j)$ may make the algorithm spend more time on conducting futile comparisons between K values, but will not make it report infeasible travel plans. These observations allow us to prepare $CR_2(i,j)$, T^2, and T^3 for the algorithm with simple though imprecise methods.

We may approximate $CR_2(i,j)$ with the help of $CR_1(\cdot)$. A pair of routes $(r1,r2)$ connects routes i and j only if the following condition holds.

$$r1 \in CR_1(i,r2) \text{ and } r2 \in CR_1(r1,j) \tag{6}$$

Therefore, we can approximate $CR_2(i,j)$ by setting it to the set of $(r1,r2)$ such that $r1$ and $r2$ satisfy (6), although we can make up artificial examples to show that (6) is just a necessary condition for $(r1,r2)$ to be included in $CR_2(i,j)$. Using (6), we will respectively set $CR_2(i,j)$ to $\{(r1,r2)\}$ and $\emptyset$ for the left and right cases in Figure 5, but $CR_2(i,j)$ should be an empty set for both cases.

The easiest way to approximate T^2 and T^3 is to compute these two matrices with standard definition for matrix multiplication (1). For the public transportation system we are working on, there are about 560 routes. Computing the powers of a 560×560 matrix is not difficult, and saving both T^2 and T^3 requires no more than 630 kilobytes of storage space.

There does exist a simple and more precise method for approximating $T^2_{i,j}$. The following condition ensures that there is at least one way to transfer from route i to j via r.

$$\min_{s \in CS(i,r)} K(r,s) < \max_{t \in CS(r,j)} K(r,t) \tag{7}$$

If we add the quantity $T_{i,r} * T_{r,j}$ to the summation for computing $T^2_{i,j}$ when (7) holds, T^2, for the example in Figure 3, will be the same as (3). We can make up artificial examples, such as the one shown in Figure 4, to illustrate that T^2 computed with the restriction of (7) may exaggerate the number of ways for transferring between routes. For this particular case,

$\min_{s \in CS(i,r)} K(r,s) = K(r,\mathsf{B})$ and $\max_{t \in CS(r,j)} = K(r,\mathsf{H})$, so (7) holds. As a result, we would add the quantity $T_{i,r} * T_{r,j} = 3 * 2 = 6$ to $T^2_{i,j}$, even though there are only three ways to transfer from i to j via r. Nevertheless (7) can be effective in some special cases. We can prove that (7) allows us to compute $T^2_{i,j}$ precisely if no routes in the public transportation system overlap with each other at two or more separately continuous segments.

Whether one needs to employ the aforementioned approximation methods in implementing the path-planning algorithm depends on the requirements of the system. If the route database is changing frequently and if travel plans recommended by the algorithm must reflect such changes immediately after the raw data is modified, a precise update of $CR_1(r1, r2)$ and the connectivity matrices may not be desirable for complex public transportation systems. A complete update of route-information functions may take too much time. Employing approximation techniques to quickly update the functions should better meet the need for on-line systems. For such special cases, it may be better for the system to adopt approximation strategies during the day, and conduct a precise data update during the night.

IV. Path Planning with Hubs

In a metropolis, there are local business centers where a lot of service routes concentrate. These *hubs* provide a great chance for people to transfer between different routes. For instance, we may prefer travel plans that transfer at hubs at steps 3 to 5 in *PathPlanning*. In a preliminary report [7], we outlined an algorithm that compute travel plans with hubs, and we provide a clearer algorithm here.

We promote a stop to a hub in the route-information database based on relative importance of stops. Bus stops served by more than 15 routes are selected as hubs. Stops of subway and traditional train systems are automatically considered as hubs. To simplify the task of path planning, we assign at least two hubs on every service route. If a route has less than two hubs, we will designate selected stops served by the route as hubs. This extra selection is based on the total number of routes that serve the stops, and we select the stop served by the most number of routes as a hub. At this moment, our system has about 170 hubs that are selected from about 2500 ordinary stops.

Technically speaking, the route map has two levels in the database. The base level contains all stops, and the upper level only hubs. The algorithm needs to know ways for connecting hubs. This is not a difficult task because usually there are direct or one-transfer service routes between major hubs. If difficult situations do occur, we can rely on human knowledge or apply the *PathPlanning* algorithm for the task. The base level will be used for finding travel plans that require zero or one transfer.

For other cases, the algorithm looks for travel plans with hubs. To achieve this, the algorithm finds ways to travel from the origin to its *nearest departing hubs* and ways to travel from the *nearest arriving hubs* to the destination. We can then use the information for connecting hubs to create complete travel plans. A stop s and any of its nearest hubs, h, must be served by a common route $r \in SR(s)$. A nearest departing hub and a nearest arriving hub of s on a route r is respectively the hub h that minimizes a positive $K(r,h) - K(r,s)$ and $K(r,s) - K(r,h)$. The algorithm will use the set of all nearest departing hubs, denoted *NDHS*(s), and the set of all nearest arriving hubs, denoted *NAHS*(s), of a stop s. In preparation of the route database, we ensure that *NDHS*(s) and *NAHS*(s) are not empty for any stop s. To this end, we may need to promote terminal stops of some service routes to hubs. The algorithm follows.

```
Algorithm PathPlanning2(Route Information, Origin $O$,
    Destination $D$)
  1.-3. Same as those in PathPlanning
  4.  Transfer via hubs: Recommend a path from $O$ to $s \in$
      $NDHS(O)$, from $s$ to $t \in NDHS(D)$, and from $t$ to $D$.
```

The last step subsumes the functionality of all steps for finding travel plans that require more than one transfer in *PathPlanning*. Since *NAHS*(s) and *NDHS*(s) are not empty for any stop s, there is at least one hub for going to D and one hub for leaving O. Given that our database will contain information for travelling between any hub pairs, we can find at least one travel plan for any desired trip. Therefore this algorithm is complete. However, this algorithm may not find the travel plan that requires the least transfers. This is because that *PathPlanning2* may recommend a travel plan that requires transfer at a hub when there is a better indirect travel plan that requires transfer at a non-hub stop for the desired trip. Hence, we have the following property.

Property 4: *PathPlanning2* is complete for path planning for public transportation systems.

V. Application Considerations

A. Prioritizing Travel Plans

A path-planning algorithm needs to prioritize travel plans when there are multiple ways for the desired trip. Common factors for comparing travel plans include number of transfers, monetary costs, expected travel time, and seat availability. Different categories of travellers may weigh one factor more than others [2]. For instance, people who are unfamiliar with the cities may strongly prefer travel plans that require the least transfers for avoiding the burden of locating bus stops.

We can integrate the connectivity matrices with standard shortest-path algorithms to find the best travel plans. Traditional algorithms that are built on the principle of dynamic programming compare travel plans extending from the origin toward the destination. If the algorithm searches travel plans at the stop level, it may spend time on stops that appear to be promising initially yet will not lead to the destination. We can remedy this drawback by consulting the connectivity matrices, and let the search algorithms explore only stops that may lead to the destination.

Our algorithms may also cooperate with standard shortest-path algorithms. It is easy to make our algorithms suspendible whenever they find travel plans at any step, thereby favoring travel plans with lesser transfers. Heuristics can then be applied to prioritize the travel plans already found. For instance, one may prefer subway service to bus service for the same trip. If the historical or real-time travel times between stops are available, we can compute the expected travel times for travel plans found at a step, and prioritize alternative travel plans accordingly.

To this end, we may employ our *PathPlanning* algorithm as an embedded function in a standard shortest-path algorithm.

This algorithm consults *PathPlanning* for next stops to explore during the search process. Since *PathPlanning* can distinguish what next stops can lead the travellers from the origin to the destination, it only returns the viable intermediate stops to the standard shortest-path algorithm. When travellers' preferences are time independent, we can use any traditional algorithms, such as the Dijkstra's algorithm, in the integrated algorithm. There are several algorithms proposed for coping with time-dependent preferences. Kaufman and Smith have an algorithm for time-dependent and constant preferences [5]. Wellman and his colleague have algorithms for time-dependent and probabilistic preferences [10, 11], and Liu and Wellman further extend the techniques for time-pressed problems [8]. These algorithms build on the concepts of *consistency* and *stochastic consistency* for dealing with time-dependent preferences.

B. User Interface

Realistic bus-information provision systems must have a good user interface to deal with naming problems of bus stops. Although we may provide a graphic user interface, we cannot assume all users will locate their origins and destinations on a digital map. Any user-friendly system must prepare to accept queries that use text input. Such a demand requires our system to manage the mapping between text inputs for location names and stop names gracefully.

Popular street names such as Main Street may be used as stop names in multiple cities in the metropolis. Therefore, we need to cope with the problem of different stops carrying the same stop names. We solve such homographic problems by annotating extra geographic information with the stop names. When encountering an ambiguous query, our system asks for clarification.

The user interface also needs to cope with synonym problems. It is possible for bus stops that surround a big site, such as the Central Park in the New York City, to have different names. When users ask about how to go to the Central Park, a good system should be able to return a travel plan that terminates at a stop that is geographically close to the Park, although the stop might not be named as Central Park.

In addition, some users may ask for methods for travelling between two areas, as opposed to between specific stops. For instance, one may want to know how to travel from the Jefferson Square to the Franklin Square in San Francisco, or from Oakland to San Francisco by public transportation systems. Our system needs to work with a geographic information system to infer the relationship between names of large areas and names of specific stop names. Our system also needs to adopt heurisitcs to show "major" connections between distant locations, rather than showing many complex detail connections.

VI. Conclusions

Planning explicitly with the route constraint and hubs provides us a chance to find travel plans more efficiently than planning at the stop level. Connectivity matrices capture the route constraint by encoding the possibilities of transferring among routes. With this information, the *PathPlanning* algorithm focuses on viable routes to search for feasible travel plans. Categorizing stops into regular-stop and hub classes allows us to tackle more complex queries efficiently. This hierarchical structure is similar to the *hierarchical encoded map views* technique used for finding shortest paths in large areas [4]. We have implemented *PathPlanning2* for providing information service in the Internet (http://iris.cs.ntou.edu.tw). Timing statistics collected from multiple field tests indicate that our algorithm can compute satisfactory travel plans within a couple of seconds.

We are integrating this path-planning service with a service management system for public transportation systems. Service management systems, e.g., [3], allow system administrators to manage information about the public transportation system. Using automatic bus location techniques [6], the management system may also relay real-time bus locations to people who are waiting at the bus stops. This information may help travellers to select buses when there are multiple choices, and may help travellers to alleviate anxiety when waiting for buses.

Acknowledgments

This work was supported in part by Grants NSC-89-2213-E-004-007 and NSC-89-2515-S-019-001 from the National Science Council of Taiwan.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] S. Bae. An advanced public transportation systems application: Feasibility study of bus passenger information systems operational test in the town of Blacksburg. In *Proceedings of the Sixth Vehicle Navigation and Information Systems Conference*, pages 408–413, 1995.

[3] P. J. Elkins. Service management systems for public transport–the German approach. In *Proceedings of the IEE Colloquium on Vehicle Location and Fleet Management Systems*, pages 401–410, 1993.

[4] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transaction on Knowledge and Data Engineering*, 10(3):409–432, 1998.

[5] D. E. Kaufman and R. L. Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *IVHS Journal*, 1(1):1–11, 1993.

[6] N. Koga. Public transportation priority system using optical bus detectors. In *Proceedings of the Second International IEEE Conference on Intelligent Transportation Systems*, pages 135–138, 1999.

[7] C.-L. Liu, T.-W. Pai, and C.-T. Chang. IRIS: Integrated route information service for multimodal public transportation systems. In *Proceeings of Taiwan's International Conference & Exhibition on Intelligent Transportation Systems 2000*, pages 186–196, 2000.

[8] C.-L. Liu and M. P. Wellman. Using stochastic-dominance relationships for bounding travel times in stochastic networks. In *Proceedings of the Second International IEEE Conference on Intelligent Transportation Systems*, pages 55–60, 1999.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[10] M. P. Wellman, M. Ford, and K. Larson. Path planning under time-dependent uncertainty. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 532–539, 1995.

[11] P. R. Wurman and M. P. Wellman. Optimal factory scheduling using stochastic dominance A*. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 554–559, 1996.