Best-Path Planning for Public Transportation Systems

Chao-Lin Liu

Abstract—**The author examines methods for a special class of path planning problems in which the routes are constrained. General search algorithms assume that we can move around in the traffic network freely, so they extend the partial paths from the very last location to each of its neighbors to form more partial paths. The best partial paths are then selected to expand, unless the selected partial path happens to be a solution. Without proper guidance, this strategy may lead to inefficient planning algorithms when the way one can move around in the networks is constrained. This scenario could happen in public transportation systems where passengers cannot order drivers to change the routes of public buses to meet individual travel needs.**

A few recently proposed path-planning algorithms for public transportation systems capture the route constraints by matrices. Although they work for some applications, they are not perfect for cooperating with traditional algorithms for best-path planning. Applying special properties of matrix multiplication, the author also employs matrices for capturing the route constraints. The author improves previous designs, and come up with the so-called *Q matrices* that serve well in the A* algorithm for best-path planning under route constraints.

Keywords—**Intelligent Transportation Systems, Advanced Public Transportation Systems, Path Planning under Constraints, Matrix Applications, A* Algorithm, Search**

I. Introduction

Standard search algorithms, such as the Dijkstra's and the A* algorithms, find the best path by expanding, comparing, and selecting promising partial paths that emerge from the origin toward the destination of the desired trip [1, 5, 13]. We determine whether a partial path is promising based on the objective function for the particular application, e.g., the shortest or the fastest paths for the trips. The literature has seen a wide variety of planning algorithms for these classes of applications in different contexts. In this paper, we discuss applications of matrices to the path planning problems when we are not permitted to move absolutely freely in a given area.

The standard algorithms search for the best solution by gradually expanding the paths. For instance, in order to find the best path from the intersection L_5 to another intersection L_9 in Figure 1, a typical search algorithm will consider five partial paths, each from L_5 to one of its neighbors. All unselected partial paths and the newly generated partial paths are put into a pool. Next the algorithm chooses the best current partial path to expand, based on the estimations of the merits of the partial paths. The algorithm iteratively chooses the best partial path to expand, and stops when the chosen partial path happens to lead the traveler from the desired origin to the desired destination.

Searching the best path in this style may work well for situations where we can choose the route freely such as a driving scenario. This assumption about autonomy does not always hold however. For example, traveling by the public transportation systems is clearly a counterexample to such an assumption. As a passenger, we cannot demand the driver to take the path

This paper appeared in the *Proceedings of the Fifth International IEEE Conference on Intelligent Transportation Systems*, 834-839, Singapore, 3-6 September 2002. Chao-Lin Liu is with the Department of Computer Science of National Chengchi University, Taiwan. E-mail: chaolin@nccu.edu.tw.

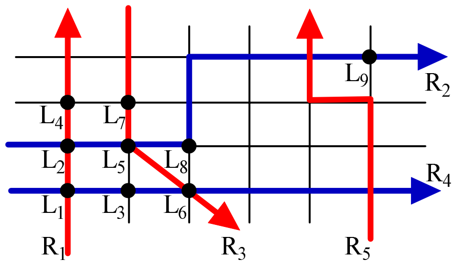

Fig. 1. A simple grid-style network with five service routes

that best meets our travel purposes. Consider the service routes in Figure 1. The partial paths from L_5 to either L_3 or L_7 should not be generated for consideration at all because there is no way to travel that way by public buses. More importantly, if transferring will result in significant costs, $L_5 \rightarrow L_6$ should not be considered as good as $L_5 \rightarrow L_8$ because the former would need three transfers on the way to L_9. Introducing a mechanism to explicitly model the route constraints and predict the need to transfer will improve the efficiency of the best-path planning algorithms for public transportation systems.

A good mechanism should demonstrate two important characteristics: *simple* and *cooperative*. To construct a route-information service that answers people's queries, we can collect information about the stops served by each service route, the service direction, and schedule of each service route in the public transportation systems. Given such raw data, a good mechanism should be able to serve, perhaps after some simple pre-compilation of the raw data. Also, we would like to have a route-information service that is able to find the best path where "best" is determined based on multiple decision factors according to travelers' preferences. Therefore, we should be able to easily integrate the new mechanism for capturing the route constraints with existing algorithms for best-path planning.

In this paper we review the algorithms proposed by Liu and his colleague in the past IEEE ITS conference, discuss why none of their algorithms conform to the simple and cooperative criteria, and propose one new mechanism that meets the criteria. Section II formalizes the problem that we will address and the notation that we will use. Section III discusses an approach that applies the concept of hierarchical planning to implicitly model the route constraints. We designate some special locations as hubs where transferring between routes are convenient. The planning algorithms attempt to find a satisfactory solution based on pre-computed paths among pre-selected hubs [10, 11]. For comparison purpose, Section IV reviews two methods that explicitly capture the route constraints by matrices in details [11]. Section V examines the previously proposed methods, and argues that it is not easy to integrate them with standard algorithms for computing best paths of more complex objective functions. We then look into a better strategy that also employs matrices. This new design avoids the difficulties, and we demonstrate its applications to path planning problems using the A* algorithm. Finally, we wrap up this paper with a brief summary and comparison.

II. Problem definition and notation

We address the path planning problem in which the travelers, which could be passengers in public transportation systems or data packets in computer networks, cannot change their routes absolutely freely. The purpose is to apply matrices to capture the constraints so that we conduct path planning more efficiently.

We use a few terms for describing aspects of constrained routes. First we assume that there is a set of locations to be served by a set of directed service routes. We assign to each location in the area of interest a unique identification number, and denote this set of numbers by S. Similarly we assign to each route serving the area a unique identification number, and let this set of numbers be R. To accommodate the existence of one-way streets and simplex communication lines, we assume that all service routes are directed. We can use two separate routes to model a two-way service route. For the simple grid shown in Figure 1, we could have assigned 24 unique numbers to the intersections, and 5 unique numbers to the service routes.

We also assign an ordinal number $K(\mathtt{r},\mathtt{s})$ to each location, $\mathtt{s}$, served by a route, $\mathtt{r}$. The K value of the departure terminal of each route is 1. If a route $\mathtt{r}$ does not serve a location $\mathtt{s}$, then $K(\mathtt{r},\mathtt{s}) = 0$. Hence, $K(\mathtt{r},\mathtt{s1}) > K(\mathtt{r},\mathtt{s2}) > 0$ implies that route $\mathtt{r}$ serves from $\mathtt{s2}$ to $\mathtt{s1}$. Based on the K function, we define some useful functions.

1. For a location $\mathtt{s}$, the set of <u>s</u>ervice <u>r</u>outes that serve $\mathtt{s}$ is defined as follows.

$$\mathit{SR}(\mathtt{s}) = \{\mathtt{r} | K(\mathtt{r},\mathtt{s}) > 0, \mathtt{r} \in R\}$$

2. The set of <u>c</u>ommon <u>s</u>tops served by two routes $\mathtt{r1}$ and $\mathtt{r2}$ is defined as follows.

$$\mathit{CS}(\mathtt{r1},\mathtt{r2}) = \{\mathtt{s} | K(\mathtt{r1},\mathtt{s}) > 0 \text{ and } K(\mathtt{r2},\mathtt{s}) > 0,\ \mathtt{s} \in S\}$$

3. For two locations $\mathtt{s}$ and $\mathtt{t}$, the following function returns the set of <u>d</u>irect <u>service</u> routes by which we can travel from $\mathtt{s}$ to $\mathtt{t}$ without transfers.

$$\mathit{DService}(\mathtt{s},\mathtt{t}) = \{\mathtt{r} | K(\mathtt{r},\mathtt{t}) > K(\mathtt{r},\mathtt{s}) > 0, \mathtt{r} \in R\}$$

The algorithms discussed in this paper employ some of these three functions. Since these functions depend solely on the route information, we can compute and save the results in a database to speed up the path planning algorithms. When the storage space is a concern, we may also compute these functions on the fly at the expense of computation time.

III. Hub-based planning

In previous work, Liu et al. propose a path planning algorithm that employs pre-selected hubs for path planning under route constraints [10]. This hub-based planning method smack of recent interests in speeding up path-query services by pre-computing partial solutions for the area of interest, e.g., [7]. *Hubs* are locations where several service routes concentrate, so they provide very good opportunities for service requesters to transfer from one route to another.

At the design stage, the system designers collect information about the service routes, and determine the standards for categorizing ordinary locations and hubs. Since hubs are typically served by several routes, it is feasible to employ a simple algorithm to compute and store the best path between any pair of hubs.

At run time, the planning algorithm attempts to find a direct route from the origin to the destination first. If not successful, the algorithm tries to find a one-transfer solution for the desired trip. If not successful again, the algorithm finds a travel plan from the origin to a nearby hub $\mathtt{H1}$ and a travel plan from a hub $\mathtt{H2}$, that is near the destination, to the destination. Since the system already knows how to travel between two hubs, it combines the partial solutions to form a complete travel plan for the desired trip. The skeleton of the algorithm follows.

Algorithm 1 (HPlanning) *Let $\mathtt{O}$ and $\mathtt{D}$ denote the origin and destination, respectively.*
1. Trivial cases: if $\mathtt{O} = \mathtt{D}$, show an appropriate message and return a null plan.
2. Simple cases: if $\mathit{DService}(\mathtt{O},\mathtt{D})$ is not empty, return any service route in the set.
3. One-transfer cases: if $\exists \mathtt{i} \in \mathit{SR}(\mathtt{O})$, $\exists \mathtt{j} \in \mathit{SR}(\mathtt{D})$, and $\exists \mathtt{s} \in \mathit{CS}(\mathtt{i},\mathtt{j})$ such that both $\mathit{DService}(\mathtt{O},\mathtt{s})$ and $\mathit{DService}(\mathtt{s},\mathtt{D})$ are not empty, there is a one-transfer solution.
4. Non-trivial cases: Find a travel plan via hubs.

Although it is easy to implement a working system based on this idea, the approach does have some weaknesses. Using the pre-computed travel plans between hubs might cause inconvenience when part of these plans becomes out of services unexpectedly. In this situation, we would have to call the planning algorithm to find alternative travel plans for affected hubs. Most of all, the designers must set the standards for what locations should be treated as hubs. This step makes the approach less flexible than one may expect, as the standards would depend on the current services which may change over time. If the services do change a lot, we would have to redesign the system, which would require human intervention to determine the standards for hubs again. This also means that we need to set the standards every time we need to have a path planning system for a new application context. Furthermore, the selection could become very complicated if we would like to guarantee a solution for any desired trips, considering the idiosyncracy of the actual services.

IV. Matrices for path planning

For both theoretical and practical purposes, we would like to find an approach that minimizes, if not nullifies, human intervention when we apply the approach to a new context. We discuss methods that employ matrices for automatic provision of path-query services in this section.

A. Adjacency matrices

Adjacency matrices are typical examples for representing graphs with matrices, e.g., [1]. We set $A_{\mathsf{i},\mathsf{j}}$ to 1 only if locations i and j are both adjacent and connected by a common service route serving from i to j. Assume that we assign i to L_i in Figure 1. The adjacency matrix for L_1, L_2, L_3, and L_4 is shown below.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

The powers of A carry special implications. For instance, $A^2_{1,4} = 1$ implies that we can travel from L_1 to L_4 via another location, although L_1 and L_4 are not adjacent as indicated by $A_{1,4} = 0$. In fact, one can prove that $A^n_{i,j}$ encodes the information about whether we can travel from location i to j by passing $(n-1)$ intermediate locations.

Despite such an interesting property, adjacency matrices are not directly applicable to the path planning problems. Computing multiple powers of the adjacency matrix at run time can be very costly in terms of both computational time and resources, yet offers little information about the exact travel plans.

B. Connectivity matrices

In stead of using adjacency matrices directly, we can employ the connectivity matrices for path planning [11]. We designate an ordinal number to each service route, and set the cell $C_{\mathtt{i},\mathtt{j}}$ in a *connectivity matrix* C to 1 if one can transfer from route `i` to `j`. By default, $C_{\mathtt{i},\mathtt{i}}$ is set to zero for all `i`. Like adjacency matrices, the powers of connectivity matrices encode whether we can transfer from one service route to another. However, this application of connectivity matrices requires validity checking.

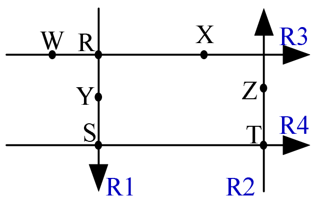

Fig. 2. An area with four directed service routes

The following connectivity matrix and its square for the area shown in Figure 2 illustrate the applications and caveat. In these matrices, we assign 1, 2, 3, and 4 to routes `R1`, `R2`, `R3`, and `R4`, respectively.

$$C = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad C^2 = \begin{bmatrix} 2 & 2 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{bmatrix} \quad (2)$$

For simple cases such as the first and the second cases stated in Algorithm 1, we do not need to use connectivity matrices. Otherwise, we use the matrices for connectivity information. For (S, Z)*, since there is no direct service, the algorithm checks which routes serve these locations, and finds that both `R1` and `R4` serve S and that `R2` serves Z. Also, the facts that $C_{1,2} = 0$ and $C_{4,2} = 1$ suggest that it is possible to go from S to Z by `R4` and `R2`. This possibility must be verified by checking whether we can go from S to a transfer location, in this case T, and to Z, and the answer is yes. For (Y, X), although $C_{1,3} = 1$, we cannot travel from Y to R and then to X because of the service direction of `R1`.

*For simplicity, we will use a pair of locations to show the (origin, destination) pair from now on.

For more complex travels, such as (Y, Z), we can rely on higher powers of the connectivity matrix. Locations Y and Z are served solely by routes `R1` and `R2`, respectively. Since Y and Z are not served by a common route and $C_{1,2} = 0$, we examine the value of $C^2_{1,2}$, and find that it is possible to satisfy the trip by transferring from `R1` to `R2` by two transfers at S and T.

Note that connectivity matrices offer only the possibility of whether we might transfer from one service route to another. There are two things that we need to take care of. First, the powers of connectivity matrices might include misleading information. For instance, it is not easy to interpret why $C^2_{2,1} = 2$ in (2). This is due to the facts that we can transfer from `R2` to `R3`(and `R4`) and from `R3`(and `R4`) to `R1`. Although it seems true that there are two methods to transfer from `R2` to `R1` here, this information is incorrect and becomes disorienting when we are working on the path planning problems in the area to the left of `R2`. The computation of powers of matrices do not take service directions into consideration, so $C^2_{2,1} = 2$ does not reflect the fact that we cannot travel westwards on `R3` and `R4`. Therefore, when the connectivity matrices show that it is possible to transfer between a pair of routes, we always need to double check the feasibility.

The other problem is about data maintenance. For instance, though $C^2_{1,2} = 2$ correctly encodes that there are two ways to transfer from `R1` to `R2`, the matrices do not tell us how to achieve the desired transfer. Thus, the algorithm needs to compute exactly how one can transfer from one route to another at system design time. These so-called *connecting route* functions are important ingredients for speeding up the route-information service system. For instance, we have $\mathit{CR}_1(\mathtt{R1}, \mathtt{R2}) = \{\mathtt{R3}, \mathtt{R4}\}$ for the area shown in Figure 2. The subscript i to *CR* indicates the set is for $(i+1)$-transfer cases, e.g., $\mathit{CR}_1(\mathtt{r1}, \mathtt{r2}) = \{r | r \in R, \exists \mathsf{x} \in \mathit{CS}(\mathtt{r}, \mathtt{r1})$ and $\exists \mathsf{y} \in \mathit{CS}(\mathtt{r}, \mathtt{r2})$ such that $K(\mathtt{r}, \mathsf{x}) < K(\mathtt{r}, \mathsf{y})\}$. The path planning algorithm follows.

Algorithm 2 (CPlanning) *Let O and D denote the origin and destination, respectively. Assume that* $\mathtt{i} \in \mathit{SR}(O)$ *and* $\mathtt{j} \in \mathit{SR}(D)$.
1.-2. Same as those in HPlanning.
3. One transfer: If $C_{\mathtt{i},\mathtt{j}} > 0$, *check if* $\exists s \in \mathit{CS}(\mathtt{i}, \mathtt{j})$ *such that both* $\mathit{DService}(O, s)$ *and* $\mathit{DService}(s, D)$ *are not empty. If yes, there is a one-transfer solution.*
4. Two transfers: If $C^2_{\mathtt{i},\mathtt{j}} > 0$, *check if* $\exists r \in \mathit{CR}_1(\mathtt{i}, \mathtt{j})$ *such that* $\exists s \in \mathit{CS}(\mathtt{i}, r)$, $\exists t \in \mathit{CS}(r, \mathtt{j})$, $\mathit{DService}(O, s) \neq \emptyset$, $\mathit{DService}(s, t) \neq \emptyset$, *and* $\mathit{DService}(t, D) \neq \emptyset$. *If yes, there is a two-transfer solution.*

In principle, this algorithm can be extended to identify travel plans that require any number, say n, of transfers. We just need to prepare C^n and $\mathit{CR}_{n-1}(\mathtt{i}, \mathtt{j})$ for all `i` and `j`. Therefore the algorithm is *complete* in the sense that it can find a travel plan between any location pairs as long as the solution exists. If the goal is to find a travel plan that minimizes the number of transfers, this algorithm is also *optimal* because it can stop whenever it identifies a solution at an early step.

Connectivity matrices have connection with the quality of services. Assume that X and Y are two locations that are not served by a common route. Let $M_{\mathsf{X,Y}}$ be the minimal n such that $C^n_{i,j} \neq 0$ for all $i \in SR(\mathsf{X})$ and $j \in SR(\mathsf{Y})$. Then, we can prove that the least number of transfers necessary for traveling from X to Y is no smaller than $M_{\mathsf{X,Y}}$ [11]. If there exists a pair of X and Y that has a large $M_{\mathsf{X,Y}}$, we may want to add or modify the service routes to reduce this M value for upgrading the services between X and Y.

Connectivity matrices are a better tool for path planning under route constraints than hubs and adjacency matrices. Most importantly, the construction of Algorithm 2 can be fully automatic. All we need is the routing information for individual routes. Unlike the approach discussed in Section III, there is no need for extra human intervention except the data collection stage. Also, since the number of service routes is intrinsically much smaller than the number of locations, it is much easier to compute and store connectivity matrices than adjacency matrices in terms of both computation time and storage space.

V. Planning with transition matrices

Although Algorithm 2 leads to fully automatic realization of a path planning system, it is not easy to apply the algorithm to best-path planning where the merits of a path include factors other than the number of transfers. A common example is computing the best path whose traveling costs include stochastic travel times [6, 8, 14].

As one may have noticed, Algorithm 2 biases against paths that require more transfers. Although paths requiring lesser transfers are more likely to be faster paths, there could also be exceptions. To guarantee the optimality of the solution, we could not afford to blindly ignore paths that require multiple transfers. On the other hand, we might need connectivity matrices of very high dimensions to determine the potential merits of not very viable travel plans. Although this is achievable, it could be very computationally expensive to do so.

Using the M matrix to guide the best-planning algorithm is more viable than using the connectivity matrices. The M matrix contains the lower bounds of the number of transfers necessary for traveling between two stops. However, as we will see next that the *transition and Q matrices* provide more exact information for best-path planning.

A. Transition matrices

We assign an ordinal number to each location in consideration. Then, we assign the cell $T_{\mathrm{i,j}}$ of the *transition matrix* T to be the number of direct routes one may commute from location i to location j. We set $T_{i,i}$ to 0 by default. The T shown below is the transition matrix for locations L_1 through L_6 in Figure 1.

$$T = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

We can prove, by induction, that the n^{th} power of Tencodes the number of ways that we can travel from a location i to another location j by $(n-1)$ transfers, for $n \geq 2$. By definition, $T_{\mathrm{k,j}}$ is the number of direct routes that we can travel from k to j. Also, the standard definition of the n^{th} power of T is

$$T^n_{i,j} = \sum_k T^{n-1}_{i,k} T_{k,j}. \tag{4}$$

Using proof by induction, we can assume that $T^{n-1}_{i,j}$ is the number of routes that we can travel from i to j by $(n-2)$ transfers, since this is the case for $n = 2$. Now, the first term within the summation in (4), $T^{n-1}_{i,k}$, is the number of ways that we can travel from i to an intermediate location k by $(n-2)$ transfers, and the second term is the number of direct ways that we can travel from k to j. Multiplying these two quantities would give us the total number of ways for traveling from i to j and transfer at a particular k by $(n-1)$ transfers. Therefore, summing over all possible k gives us the total number of ways to travel from i to j by $(n-1)$ transfers.

Similar to connectivity matrices, transition matrices have applications for service planning. Define the cell $Q_{i,j}$ of the Q matrix as follows.

$$Q_{i,j} \equiv \text{ the minimal } n \in [1,\infty) \text{ such that } T^n_{i,j} \neq 0$$

It is easy to see that $Q_{i,j}$ encodes the minimal number, i.e., $Q_{i,j}-1$, of transfers necessary for traveling from i to j. Therefore, if $Q_{i,j}$ is very large for some (i, j) pairs, we probably need to improve the service for these location pairs. For practical networks, we can expect that $Q_{i,j}$ will not be a large number. Therefore computing and updating the Q matrix should not be a challenging task. Moreover, the calculation of the Q matrix can be carried out at system design time before we put Q into work.

$Q_{i,j}$ is clearly superior to $M_{X,Y}$ defined in Section IV-B. $M_{X,Y}$ is a lower bound of the minimal number of transfer necessary for traveling from X to Y. In contrast, $Q_{\mathsf{X,Y}}$ unambiguously pinpoints the minimal number of transfers for the same trip.

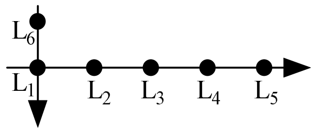

Fig. 3. Locations served by a common route

For locations served by a common route, the transition matrices may seem to exaggerate the number of ways of traveling between these locations. Consider the very simple case shown in Figure 3 where five locations are served by a common route. The square and cubic of the transitivity matrix for these locations are shown below.

$$T^2 = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad T^3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 \end{bmatrix} \tag{5}$$

The fact that $T^2_{1,5} = 3$ may seem weird at first. This is because that, at least in principal, we can get on and off the same bus at L_2 for traveling from L_1 to L_5. Similarly, one can do such "transfers" at L_3 and L_4. Hence, $T^2_{1,5} = 3$ indicates that there are three different one-transfer ways to travel from L_1 to L_5. The facts that $T^3_{1,5} = 3$ and $T^3_{6,5} = 3$ have analogous interpretations.

This property will not affect the usefulness of the Q matrix. It is easy to see and show that, when $T^k_{i,j}$ is an exaggerated value, there must be a $\kappa < k$ such that $T^\kappa_{i,j} \neq 0$. For instance, $T^2_{1,5} = 3$ is an exaggerated value, but we have $T_{1,5} = 1$, so $Q_{1,5} = 1$.

B. Planning under route constraints

Since $Q_{i,j}$ encodes the minimal number of transfers necessary for traveling from location i to j, we can apply the Q matrix to design a path planning algorithm as follows. Notice that there is no need to check feasibility of travel plans as one may need in *CPlanning* because the Q matrix provides decisive connection information.

Algorithm 3 (TPlanning) *Let o and d denote the numbers assigned to the origin and destination, respectively.*
1. Trivial cases: if $\mathsf{o} = \mathsf{d}$, show an appropriate message and return a null plan.
2. Direct: if $Q_{o,d} = 1$, return any service in DService(o, d).
3. One transfer: if $Q_{o,d} = 2$, there must be a location m such that $Q_{o,m} = 1$ and $Q_{m,d} = 1$. Combine any route in DService(o, m) and any route in DService(m, d) to obtain a one-transfer plan.
4. Two transfers: if $Q_{o,d} = 3$, there must be different locations $\mathsf{m1}$ and $\mathsf{m2}$ such that $Q_{o,m1} = 1$, $Q_{m1,m2} = 1$, and $Q_{m2,d} = 1$. Combine one route from each of DService$(\mathsf{o}, \mathsf{m1})$, DService$(\mathsf{m1}, \mathsf{m2})$, and DService$(\mathsf{m2}, \mathsf{d})$ to obtain a two-transfer plan.

Similar to Algorithms 1 and 2, the leading two steps take care of trivial and simple cases. At step 3, we check whether or not $Q_{\mathsf{o},\mathsf{d}} = 2$. If true, there is at least one intermediate location m where we can transfer from the route that serves o to the route that serves d. We check for $Q_{\mathsf{o},\mathsf{m}} = 1$ and $Q_{\mathsf{m},\mathsf{d}} = 1$ because we are sure that there cannot be any direct service from the origin to the destination when $Q_{\mathsf{o},\mathsf{d}} = 2$, so we can satisfy the desired trip only by combining one leg of the trip from each of *DService*(o, m) and *DService*(m, d). Following this design principle, we can add more steps, such as the fourth step, to cope with cases in which two or more transfers are necessary. Therefore, this algorithm is complete and optimal in the sense that it will always find the travel plan that requires the least transfers. This *TPlanning* algorithm is better than *CPlanning* because *TPlanning* requires less checking at run time.

C. Integration with fastest-path algorithms

The Q matrix is not just useful for searching paths that require the least number of transfers. It works with traditional search algorithms naturally. We demonstrate the applicability of the Q matrices to fastest-path planning problems by integrating them into the A* algorithm. In the standard A* algorithm, we employ a heuristic function to estimate the merits of search nodes [13]. In particular, this heuristic function should not overestimate the actual cost from the state being evaluated to the goal state. Such heuristic functions are called *admissible*, and they lead to the optimality of the A* algorithm.

Applying the A* algorithm to path planning problems, we need to select a heuristic function for estimating the traveling cost from a location to the destination. The preferences depend well upon the applications and individual travelers. For public transportation systems, individuals may consider such factors as fare, seat availability, easiness and times of transfers, and so on [2]. Among these factors, the necessity of transferring is directly related to the route constraints.

Therefore, we need to forecast the need to transfer when we compare the goodness of an intermediate location, even when this need will not take effect at the current location. We can rely on the Q matrices for this capability. Let O and D respectively denote the origin and destination of the desired trip, and L_i be any intermediate location. Assume that the current partial path is $\mathsf{O} \rightarrow \mathsf{L}_1 \rightarrow \cdots \rightarrow \mathsf{L}_n$. The evaluation function for this partial path is $f(n) = g(n) + h(n)$, where $g(n)$ is total costs for traveling from O to L_n, and $h(n)$ is the heuristic estimation for future costs if this partial path is extended to reach the actual destination. Hence $h(n)$ is the part of the evaluation function that must consider the transfers that must occur in the future. Let n and d be the ordinal number assigned to L_n and D, respectively. The quantity $Q_{n,d}$ will encode the minimal number of transfers necessary for traveling from L_n to the destination, thereby providing a perfect basis for determining the effects of route constraints on the heuristic estimation $h(n)$.

Consider the example in Figure 1 again, where we assign i to L_i. We assume that it is possible to travel between any location by public buses in the area had we drawn the whole area, and that Q_{xy} is very large if it appears that we cannot go from x to y in the grid. Furthermore, for this illustration, we assume that one would like to travel from L_1 to L_9, and that transferring is very costly. At L_1, we would see that $Q_{2,9} = 1$ and $Q_{3,9} = 3$, so $\mathsf{L}_1 \rightarrow \mathsf{L}_2$ is preferable to $\mathsf{L}_1 \rightarrow \mathsf{L}_3$ in terms of number of transfers. Next, *without* the guidance of the Q matrix, a search algorithm might consider that $\mathsf{L}_1 \rightarrow \mathsf{L}_2 \rightarrow \mathsf{L}_4$ is better than $\mathsf{L}_1 \rightarrow \mathsf{L}_2 \rightarrow \mathsf{L}_5$ because the latter requires an immediate transfer. However, with $Q_{5,9} = 1$ and a large $Q_{4,9}$, the search algorithm will evaluate these alternatives more correctly. Also, the facts that $Q_{8,9} = 1$ and $Q_{6,9} = 3$ will be helpful for appropriate comparison between $\mathsf{L}_1 \rightarrow \mathsf{L}_2 \rightarrow \mathsf{L}_5 \rightarrow \mathsf{L}_8$ and $\mathsf{L}_1 \rightarrow \mathsf{L}_3 \rightarrow \mathsf{L}_6$.

In addition to using the Q matrix to support the comparison among partial paths, we use the K values to support the task of partial path generation. Take the task of expanding $\mathsf{L}_1 \rightarrow \mathsf{L}_2 \rightarrow \mathsf{L}_5$ for example. One could have generated five new partial paths, each from L_5 to one of its neighbors. Using the facts that $K(\mathsf{R2}, 5) > K(\mathsf{R2}, 2)$ and $K(\mathsf{R3}, 5) > K(\mathsf{R3}, 7)$, the algorithm will not generate two invalid paths from L_5 to L_2 and L_7.

Notice that the information provided by the Q matrices is superior to that provided by the M matrices which we discussed at the end of Section IV-B. The M matrices provide a lower bound of the number of needed transfers, so they can be useful for defining an admissible heuristic. The Q matrices, in contrast, provide the exact minimum number of needed transfers, so they are superior for defining a tighter admissible heuristic.

Computing the fastest path is not an easy task. It is well known that, when the link travel times are time dependent, traditional search algorithms cannot be applied directly [6]. Kaufman and Smith find that the Dijkstra's algorithm remains applicable if the transportation network demonstrates the feature that leaving an origin later will not make one arrive at the destination earlier [8]. Wellman et al. extend the applicability of the A* algorithm to transportation networks in which leaving an origin later will not increase the probability of arriving the destination at an earlier time [14]. Liu and Wellman further extend this work to situations where we only afford to compare the goodness of paths roughly by bounds of the distribution of travel times [12]. Bander and White develop an interruptible A* algorithm that employs more heuristic information for path planning, and this algorithm is applicable under time constraints [3]. The Q matrices presented here can be used to guild these algorithms for fastest-path planning when the routes are constrained.

VI. Related work

Researchers have noticed that the needs of transferring among routes require special care for computing cost functions of candidate travel plans. If not carefully taken care of, transferring costs may make standard search algorithms unable to find the best travel plans. Dial discusses the necessity and methods for inclusion of *transfer penalty* in the computation of route costs [4]. Knocz et al. employ matrices for representing connectivity among routes in [9] where the matrices are called *Transit Route Connectivity Matrices*. Despite the similarity between the TRC matrices and our transition matrices, we apply the matrices in different ways. Our algorithm explicitly computes powers of the transition matrices and other related data for better utilization of the static route information, thereby achieving a more efficient planning algorithm. In principle, the *TPlanning* algorithm can easily find travel plans that demand several transfers in complex service networks. As a result, we believe that it is relatively easier for us to integrate our algorithm with the A* algorithm for path planning in time-dependent stochastic transportation networks. The TRC matrices are more general than the transition matrices, however, in that routes are considered "connected" if stops served by two routes are within a preselected walking distance. In general, walking can be treated as a service route with special transfer costs and potentially longer link travel times, and our algorithms can take walking as an alternative accordingly.

VII. Conclusions

The transition and Q matrices clearly dominate the hub-based and the other matrix-based approaches for path planning under route constraints. *HPlanning* embraces the concept of hubs for implicitly modeling the route constraints. *CPlanning* and *TPlanning*, on the other hand, employ matrices for explicitly capturing the route constraints. They embrace, respectively, the connectivity matrices that model the connectivity among service routes and the transition matrices that model the connectivity among locations. In addition to their applications for path planning problems, we can apply the connectivity and transition matrices to service planning problems. We identify matrices, M and Q respectively, that help us to gauge the quality of service in the network.

Both *CPlanning* and *TPlanning* are better than *HPlanning* partially because they do not need extra human intervention for implementing a path-query service. *TPlanning* is considered to be even better than *CPlanning*. The former requires less amount of feasibility checking at run time, although both are good for computing travel plans that require the least transfers. The Q matrices are better than the M matrices for best-path planning. The Q matrices provides the tightest lower bound on the transfer costs. Therefore, the Q matrices offer a very good solution to path planning with route constraints using the A* algorithm.

Finally, although we have presented the planning algorithms for public transportation systems, the basic ideas can be applied to any routing problems where the service routes are constrained.

Acknowledgements

The author would like to thank the anonymous reviewers for invaluable comments on this paper. 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*, 408–413, 1995.

[3] J. L. Bander and C. C. White, III. A new route optimization algorithm for rapid decision support. In *Proceedings of the Second Vehicle Navigation and Information System Conference*, 709–728, 1991.

[4] R. B. Dial. Transit pathfinder algorithm. *Highway Research Records*, 205:67–85, 1967.

[5] E. W. Dijkstra. A note on two problems in connextion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] R. W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation Science*, 20(3):182–188, 1986.

[7] 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.

[8] 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.

[9] N. Koncz, J. Greenfeld, and K. Mouskos. A strategy for solving static multiple-optimal-path transit network problems. *ASCE Journal of Transportation Engineering*, 122(3):218–225, 1996.

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

[11] C.-L. Liu, T.-W. Pai, C.-T. Chang, and C.-M. Hsieh. Path-planning algorithms for public transportation systems. In *Proceedings of the Fourth International IEEE Conference on Intelligent Transportation Systems*, 1061–1066, 2001.

[12] 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*, 55–60, 1999.

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

[14] 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*, 532–539, 1995.