Research Article

Floyd-A* Algorithm Solving the Least-Time Itinerary Planning Problem in Urban Scheduled Public Transport Network

Yu Zhang,¹ Jiafu Tang,^{1,2} Shimeng Lv,¹ and Xinggang Luo¹

¹ Department of Systems Engineering, State Key Lab of Synthetic Automation of Process Industries, Northeastern University, Shenyang 110004, China

² College of Management Science and Engineering, Dongbei University of Finance and Economics (DUFE), Shahekou, Dalian 116025, China

Correspondence should be addressed to Jiafu Tang; jftang@mail.neu.edu.cn

Received 26 October 2013; Revised 5 March 2014; Accepted 10 March 2014; Published 27 April 2014

Academic Editor: Cristian Toma

Copyright © 2014 Yu Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We consider an ad hoc Floyd-A* algorithm to determine the a priori least-time itinerary from an origin to a destination given an initial time in an urban scheduled public transport (USPT) network. The network is bimodal (i.e., USPT lines and walking) and time dependent. The modified USPT network model results in more reasonable itinerary results. An itinerary is connected through a sequence of time-label arcs. The proposed Floyd-A* algorithm is composed of two procedures designated as Itinerary Finder and Cost Estimator. The A*-based Itinerary Finder determines the time-dependent, least-time itinerary in real time, aided by the heuristic information precomputed by the Floyd-based Cost Estimator, where a strategy is formed to preestimate the time-dependent arc travel time as an associated static lower bound. The Floyd-A* algorithm is proven to guarantee optimality in theory and, demonstrated through a real-world example in Shenyang City USPT network to be more efficient than previous procedures. The computational experiments also reveal the time-dependent nature of the least-time itinerary. In the premise that lines run punctually, “just boarding” and “just missing” cases are identified.

1. Introduction

When a traveler plans to travel from one place (origin) to another (destination) beginning at a given initial time (or imposed by a deadline) in a real-world urban scheduled public transport (USPT) network, it can be difficult to determine the a priori least-time (LT) itinerary. Typically speaking, the itinerary should specify the USPT services (e.g., metro lines, bus lines) that combine the vehicle trips to take, the roads to walk, and the stops at which to transfer in order to arrive at a destination consuming the least time. The aforementioned least-time itinerary planning problem in an urban scheduled public transport network (LTIP-USPT) is a common decision problem for travelers in a city travelling from an origin to a destination to make a date, attend a conference, participate in a party, and, most often, to go to work.

Unlike the well-studied shortest path problems in static networks, LTIP-USPT is more difficult to address concerning

the following reasons: (i) the topological structures and timetables of the public transport services in a city are prescheduled. (ii) One or more transfers are unavoidable in most cases, resulting in walking time and waiting time as penalties. (iii) In addition to spatial connectivity and distance, travelers must also consider temporal and operational factors. In a word, public-transport travelers, as opposed to private-vehicle travelers, cannot wander in a USPT network freely without any constraints (for more detailed explanation, please see [1]). Moreover, in real-world applications, the solution procedure should be as fast as possible. It is a challenging task because in the time-dependent USPT network, various nodes and lines are interconnected, thus leading to numerous combinations of lines, vehicle trips, walks, and transfers, making a difficult combination optimization problem.

The headway-based public transport services were considered in early researches (see [2–4]). Recent years witness a boom in the development of schedule-based public

transport. Correspondingly, in the transportation research community, the focuses have shifted from headway-based services to schedule-based ones. The main difference comes from the evaluating of each transfer waiting time. The transfer waiting time is typically assumed to be half of the headway concerning headway-based services. However, for schedule-based services, it could be precisely evaluated depending on the combined timetables. Thus the network travel times (especially transfer waiting times) shift from deterministic to time dependent, resulting in the needs of methodological changes for many transportation problems, for example, itinerary planning (see [5–8]) and traffic assignment (see [9–11]). And these methodologies appear more promising and competing against those for the traditional headway-based services. Note that in reality, the “schedule-based lines” are a generalized concept that might include bus lines and metro lines. In this sense, the bimodal (USPT lines and walking) network considered in this paper could also be interpreted as a multimodal one, which is a typical urban public transport network in real world.

There are numerous related works in the existing literatures. Tong and Richardson [12] were among the first to study the scheduled public transport; they developed a branch-and-bound type algorithm for the minimum itinerary. Along this vein, research community extended the itinerary planning problems by introducing different real-world considerations. Horn introduced multimodal transport services, and a Dijkstra-based algorithm was considered for minimizing generalized travel costs [13]. Tan et al. [7] developed a recursive algorithm for finding reasonable paths that satisfy the defined acceptable time criterion and transfer-walk criterion, where travelers’ preferences were required to give. Not only the LT itinerary, but also the k-LT itinerary need to be determined for travelers. Accordingly, Xu et al. [6] and Canca et al. [5] studied the k-shortest path problems in schedule-based transit networks. Androutsopoulos and Zografos considered a dynamic programming based algorithm [14, 15] for the itinerary planning problem in the context where traveler imposes time windows on nodes; multiobjective was also considered. These works have significantly contributed to real-world derived itinerary planning problems for different types of travelers’ requirements. However, their focuses were mainly on the considerations of real-world factors. With regard to the algorithmic efficiency, they only cared about whether a query could be completed within a short time. In query-intensive scenarios, the query should be as fast as possible, which motivated researchers in recent years to examine heuristic methods to speed up computing.

Fu et al. reviewed the heuristic shortest path algorithms for transportation applications [16], noting that A*-based algorithms were widely used. The heuristic A* algorithm was first proposed by Hart et al. [17], typically for the shortest path problems in static networks. The performance of A* algorithm depends primarily on the strategy of estimating the travel time of a partial path. A well-designed strategy leads to a considerable savings of computational time while assuring an optimal solution. Chabini and Lan adapted A*

for the fastest path in deterministic discrete-time dynamic networks [18]; they proposed three estimating strategies. Subsequently, numerous works appeared in the literatures that primarily applied A*-based algorithms to LT path problems in unscheduled time-dependent road networks (see [19, 20]). Chen et al. proposed an A*-based integrated approach combining offline precomputation and online path retrieval for a road navigation system [21]. This work introduced precomputation (see [22, 23]) technology and significantly improved the efficiency. However, determining the LT itinerary in a scheduled network is more challenging than that of unscheduled network. But relatively little related works were found.

In fact, two notable works had studied the itinerary planning problem in scheduled public transport network, especially focusing on the speed-up technologies [24, 25]. They showed that the time-dependent model is superior to the time-expanded model in the sense of itinerary-finding efficiency. A*-based and other strategies are demonstrated to be capable of speeding up computing. Following the previous works, this paper goes beyond in the following aspects: (i) modelling a modified bimodal (i.e., USPT services and walking), time-dependent USPT network. This model is observed to be more applicable in that the results over the modified network model intrinsically have smaller number of transfer times. (ii) An ad hoc Floyd-A* algorithm is developed to solve the LTIP-USPT where transit vehicles are assumed to run punctually. A novel approach to estimating travel time of the partial itinerary is embodied in Floyd-A*. To implement the approach, we generate a slacked network and let the arc travel time be a static tight lower bound of the associated real time-dependent arc travel time. Precomputing technology is also used. The algorithm is proven to be optimal in theory and was demonstrated with a real-world example to be very applicable. The Floyd-A* procedure outperforms the previous procedures. It reduces the averaged computational time by 63.9% compared with a conventional Dijkstra-like procedure. (iii) From the management perspective, an illustrated example reveals the time-dependent nature of the least-time itinerary. In the premise that lines run punctually, the solution aids travelers in avoiding “just missing” cases.

The remainder of this paper is organized as follows. Section 2 formulates the modified USPT network model and itinerary structure, develops the formula for each time-dependent time-label arc, and subsequently formulates the LTIP-USPT followed by hypotheses. In Section 3, we propose an ad hoc Floyd-A* algorithm composed of two procedures, that is, Floyd-based Cost Estimator and A*-based Itinerary Finder. The Cost Estimator precomputes the estimated travel time of destination-ended partial itineraries as heuristic information. The Itinerary Finder heuristically determines the LT itinerary in real-time. Floyd-A* is mathematically proven to be admissible and efficient. Furthermore, an illustrated example is presented in Section 4 that reveals the time-dependent nature of an LT itinerary and provides guidance for travelers in determining the initial time to begin travel. Meanwhile, through both a numerical example and a real-world case, the Floyd-A* procedure is proven to be more efficient than two other conventional procedures, that is,

TABLE 1: A timetable example.

k	n											
	l_1		l_2		l_3		l_4		l_5		l_6	
	n_1	n_2	n_3	n_4	n_1	n_2	n_4	n_5	n_6	n_7	n_1	n_7
1	6:02	6:14	6:25	6:05	6:18	6:30	6:05	6:15	6:02	6:16	6:00	6:12
2	6:08	6:21	6:31	6:15	6:29	6:40	6:10	6:20	6:10	6:23	6:03	6:15
3	6:13	6:25	6:36	6:25	6:39	6:51	6:15	6:24	6:18	6:32	6:06	6:17
4	6:18	6:29	6:39	6:33	6:46	6:58	6:20	6:30	6:26	6:40	6:09	6:20
5	6:22	6:34	6:44	6:41	6:54	7:07	6:23	6:33	6:30	6:45	6:12	6:24
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Dijkstra-like and Plain-A* procedures. Finally, concluding remarks and future works are discussed in Section 5.

2. Formulation of the Least-Time Itinerary Planning Problem in Urban Scheduled Public Transport Network (LTIP-USPT)

In modeling the urban scheduled public transport (USPT) network, both USPT lines and walking modes should be considered. An itinerary should encompass spatial, temporal, and operational features. In an USPT network, a passenger who plans to travel from an origin to a destination at a given initial time will select the USPT lines to take, the roads to walk, and the stops at which to transfer in order to arrive at their destination as quickly as possible. We identify this problem hereafter as LTIP-USPT. To solve a LTIP-USPT, a challenge is to model a more applicable USPT network and a reasonable itinerary structure; these are developed in Sections 2.1 and 2.2, respectively. In Section 2.3, we formulate the LTIP-USPT as the least-time itinerary planning problem in a deterministic bimodel time-dependent scheduled network.

2.1. A Modified Scheduled Network Model. Intuitively, Figure 1(a) shows an example of a physical USPT network, and the associated modified network model $G := (N, A, L)$ is shown in Figure 1(b). The advantage of this model beyond previous scheduled network model can be found in Remark 1. Table 1 gives an associated timetable example.

Let $l := (N_l, A_l, T_l) \in L$ denote the scheduled directed USPT line that operates on the USPT network. The USPT line typically refers to (but not constrained to) the bus line or metro line that runs on fixed road and runs through a predetermined series of nodes N_l based on a timetable T_l . There are numerous *vehicle trips* within a single day. $\theta_{l,k}^n$, element of T_l represents the scheduled time when the k th vehicle trip of l arrives or departs at node $n \in N_l$. Table 1 shows a timetable example, where, for example, $\theta_{1,2}^{n_2} = 6:21$.

Any move along a specific line is not necessarily between two adjacent nodes but may pass through one or more intermediate nodes. Correspondingly, $C_{|N_l|}^2$ arcs can be generated by line l . The set of these arcs is formulated as $A_l = \{(n_i, n_j)_l \mid 0 < \rho_l^{n_i} < \rho_l^{n_j} \leq |N_l|\}$, where ρ_l^n denotes the sequence number that line l passes through node n . Let $\rho_l^n = 0$ if l does not pass

through n . Obviously, for any node n , $\rho_l^n \in \{0, 1, 2, \dots, |N_l|\}$. For example, l_1 of Figure 1 associates with the set of arcs $A_{l_1} = \{(n_1, n_2)_{l_1}, (n_1, n_3)_{l_1}, (n_2, n_3)_{l_1}\}$. In this model, it should be noted that only one arc in A_l , rather than two or more connected arcs, is traversed from the traveler's boarding to alighting a vehicle of line l .

Remark 1. One of the challenges of this problem is the existence of multiple solutions, and it is fairly easily that an algorithm is trapped into some poor local minimums. This Remark elaborates this phenomenon and the solution method. In previous related works, the arcs of a USPT line typically only exist between the adjacent nodes; that is, $A_l = \{(n_i, n_j)_l \mid 0 < \rho_l^{n_i} = \rho_l^{n_j} - 1 < |N_l|\}$. In practice, the disadvantage is shown as follows by an example. Consider the two USPT network models in Figure 2, where Figure 2(a) shows a previous network model and Figure 2(b) a modified one. A traveler goes from n_1 to n_3 starting at 9:00. There are the following two alternative itineraries.

Itinerary 1. Wait 5 minutes and start at origin n_1 , traveling by l_2 directly to the destination n_3 .

Itinerary 2. First travel by line l_1 to n_2 , and wait 5 minutes and then travel by l_2 to n_3 .

They both arrive at destination at 9:50, so they are both the least-time itineraries in theory. But real-world traveler typically prefers Itinerary 1, because Itinerary 2 contains a transfer activity. In the model of Figure 2(a), a label-setting algorithm (see Dijkstra 1959) will obviously choose Itinerary 2, because 9:25 is earlier than 9:30 regarding node n_2 ; that is to say, $\text{arc}(n_1, n_2)_{l_1}$ dominates $\text{arc}(n_1, n_2)_{l_2}$. In comparison, executing a label-setting algorithm in the model of Figure 2(b) will lead to the choice of Itinerary 1. In fact, once the $\text{arc}(n_1, n_3)_{l_2}$ is searched, it will never be dominated, because there is no way to reach n_3 earlier than 9:50. It is observed that the aforementioned phenomenon exists comprehensively in our experiments. The results over the modified network model intrinsically have smaller number of transfer times; thus, it is more applicable.

In most real-world cases, the destination node n_d cannot be reached by using only one line from the origin node n_o , so transfer is necessary. Transferring does not always occur at just one node; a traveler may have to walk some distance

FIGURE 1: A USPT network example.

FIGURE 2: USPT network example.

to another node in order to transfer. The tolerable walking distance is constrained by a constant upper bound D . Thus, the set of walking arcs is formulated as $A_w = \{(n_i, n_j)_w \mid n_i \neq n_j, \text{dist}(n_i, n_j) < D\}$. Using the USPT network of Figure 1 as an example, $(n_3, n_5)_w \in A_w$ while $(n_1, n_2)_w \notin A_w$ because $\text{dist}(n_1, n_2) > D$. For denotation convenience, let USPT service s denote either a line l or the walk w ; that is, $s \in L \cup \{w\}$.

In summary, with regard to USPT network $G := (N, A, L)$, $N = \bigcup_{l \in L} N_l$, and $A = \bigcup_{l \in L} A_l \cup A_w$. In general, a node $n \in N$ in this network represents a bus stop or metro station. An arc $(n_i, n_j)_s \in A$ shows an available move. In any given specific arc, the arc travel time does not always remain constant, which is actually dependent on the initial start time. This makes the USPT network a single-layer, bimodal, and time-dependent network.

2.2. Itinerary Structure and Timing. The Itinerary is represented as a sequence of orderly arcs or nodes in a static network, though there must be some adaptation in a USPT network context. An arc could not describe the temporal factor, so we define the time-label arc in Definition 2. An itinerary in a USPT network could be represented as a sequence of time-label arcs.

Definition 2. A time-label arc (t -arc for short) is defined as a 4-tuple (n_i, n_j, s_i, t_i) link representing a passenger's move from a tail node n_i to a head node n_j by means of a transport service s_i at a given initial time t_i . This representation is legitimate if and only if there exists an available transport service $s_i \in L \cup \{w\}$ for a passenger who is located at node $n_i \in N$ at time

t_i (maybe with some waiting time) to move towards node $n_j \in N$. Using Figure 1 as an example, $t\text{-arc}(n_1, n_2, l_1, 10)$ denotes that a passenger arrives at node n_1 at the initial time 7:10:00 and travels to node n_2 by line l_1 . In addition, subscripts of the service s_i and the initial time t_i are kept consistent with the tail node n_i . If necessary, superscripts are used to distinguish the different services and initial times.

When a passenger travels from n_o to n_d at a given initial time t_o , there can be numerous eligible itineraries. The set of these itineraries is denoted by $P^{t_o}(n_o, n_d)$, whose elements can be represented by a sequence of connective t -arcs shown in

$$p^{t_o}(n_o, n_d) = \{(n_o, n_2, s_o, t_o), (n_2, n_3, s_2, t_2), \dots, (n_{|p|}, n_d, s_{|p|}, t_{|p|})\}, \quad (1)$$

where $|p|$ is the number of t -arcs that compose $p^{t_o}(n_o, n_d)$.

A passenger may be concerned about the total travel time of itinerary $p^{t_o}(n_o, n_d)$, which is the accumulated travel time of each component t -arc. The elapsed travel time therefore acts as the cost (weight) of each t -arc. There are three components of travel time as follows:

- (1) *in-vehicle time*—elapsed during vehicular travel on the line,
- (2) *walking time*—elapsed during walking between two nodes for transfer purposes,
- (3) *waiting time*—elapsed at node waiting for the arriving transfer vehicle.

Let π be an operator that times each t -arc or itinerary. The computing method to time t -arc (n_i, n_j, s_i, t_i) depends on the

associated transport service s_i . If $s_i = w$, the t -arc is traversed by walking. It calculates the fixed walking time cost as shown in

$$\pi(n_i, n_j, w, t_i) = \frac{\text{dist}(n_i, n_j)}{v_{\text{walk}}}. \quad (2)$$

The arrival time at n_j is then easily calculated in

$$t_j = t_i + \pi(n_i, n_j, w, t_i). \quad (3)$$

If $s_i = l$, the t -arc is traversed by line l . Both the in-vehicle time and the waiting time must be considered. Therefore, the associated travel time is not fixed but time dependent, as calculated in

$$\pi(n_i, n_j, l, t_i) = \theta_{l,k}^{n_j} - t_i. \quad (4)$$

Because the passenger will board the first arriving vehicle of the transferred line l , k of Formula (4) is determined by Formula (5); the waiting time and the in-vehicle time in this process are $\theta_{l,k}^{n_i} - t_i$ and $\theta_{l,k}^{n_j} - \theta_{l,k}^{n_i}$, respectively. The corresponding arrival time at n_j is calculated with Formula (6):

$$k = \arg \left\{ \min_k (\theta_{l,k}^{n_i} - t_i \mid \theta_{l,k}^{n_i} - t_i > 0) \right\}, \quad (5)$$

$$t_j = \theta_{l,k}^{n_j}. \quad (6)$$

In any specific USPT network, once the initial time t_i of each t -arc (n_i, n_j, s_i, t_i) is known, the travel time of this t -arc $\pi(n_i, n_j, s_i, t_i)$ and the associated arrival time t_j can be easily calculated. With respect to any itinerary $p^{t_o}(n_o, n_d)$, the first initial time t_o is predetermined by the passenger, and the subsequent times can be calculated recursively by Formula (3) or (6). In other words, the initial time of a specific t -arc is equal to the arrival time of the upstream t -arc. In this case, the travel time of the itinerary formulated in Expression (1) can be written as

$$\begin{aligned} \pi(p^{t_o}(n_o, n_d)) = & \left\{ \pi(n_o, n_2, s_o, t_o) + \pi(n_2, n_3, s_2, t_2) \right. \\ & \left. + \cdots + \pi(n_{|p|}, n_d, s_{|p|}, t_{|p|}) \right\}. \end{aligned} \quad (7)$$

Along with spatial and temporal features, practical operability should also be considered from the passenger's perspective. Some properties of the itinerary that describe operability are given below.

Property 1. Two t -arcs that are traversed by walking cannot be adjacent, due to the hypothesis that a walking distance between two nodes cannot be larger than D . In other words, when $s_i = w$, we have $s_{i+1} \neq w$, where $i = o, 2, 3, \dots, |p| - 1$.

Property 2. During the travel process, if a line has been already used as a transport service, a passenger will not likely reuse this line or its inverted line (see Definition 3) in his/her subsequent travel process. In other words, when $s_i \neq w$, we have $s_j \neq s_i$ and $s_j \neq \bar{s}_i$, where $i, j = o, 2, 3, \dots, |p|$ and $i \neq j$.

Property 3. In reality, a passenger is not likely to travel an itinerary that goes through a specific node twice. Therefore, we have $n_i \neq n_j$, where $i, j = o, 2, 3, \dots, |p|$, d and $i \neq j$.

Definition 3. With regard to a specific line l , there usually exists an *inverted line* $\bar{l}$ that runs on almost the same road segments of l but in inverted directions. l is also the inverted line of $\bar{l}$; that is, $\bar{\bar{l}} = l$. Intuitively, l_3 is the inverted line of l_4 in the USPT network shown in Figure 1 ($\bar{l}_3 = l_4$).

Take the USPT network of Figure 1 as an example, whose associated timetables are provided in Table 1. A passenger arrives at n_2 at 6:10, waits for 4 minutes, takes the first available vehicle trip of l_1 towards n_3 , arrives at 6:25, walks to n_6 using $180 \text{ m}/1.5 \text{ m/s} = 2(\text{min})$, waits for 3 minutes, boards the vehicle on the 5th trip of l_4 at 6:30, and finally arrives at n_7 at 6:45. This itinerary is represented by $p^{10}(n_2, n_7) = \{(n_2, n_3, l_1, 6 : 10), (n_3, n_6, w, 6 : 25), (n_6, n_7, l_4, 6 : 27)\}$ and consumes 35 minutes in total; thus, $\pi(p^{6:10}(n_2, n_7)) = 35(\text{min})$.

2.3. Problem Formulation. In any specific USPT network, a passenger decides to travel from an origin n_o to a destination n_d , at an initial time t_o . The problem is determining a connected itinerary among the large volume of available choices that requires a minimum of travel time. This can be mathematically formulated as follows:

$$\begin{aligned} \min \quad & \pi(p^{t_o}(n_o, n_d)) \\ \text{s.t.} \quad & p^{t_o}(n_o, n_d) \in P^{t_o}(n_o, n_d). \end{aligned} \quad (8)$$

The travel time of the t -arc traversed by walking is fixed, while that traversed by a line is time dependent, thus leading to a time-dependent rather than static USPT network. The LTIP-USPT pertains to the least-time itinerary planning problem in a bimodal, time-dependent scheduled network. The traditional shortest path algorithms do not apply. Through the adaptation of the A* algorithm, Section 3 develops an ad hoc Floyd-A* algorithm to address the LTIP-USPT. The following hypotheses are assumed and summarized as follows.

- (1) Line vehicles run punctually.
- (2) The vehicle capacities are infinite.
- (3) The road network is noncongested.
- (4) The vehicle departs immediately after arriving at a specific node.
- (5) The origins and destinations are all located just at nodes.
- (6) One walking distance cannot be greater than the tolerable upper bound D .

The findings of this research can be widely used; they can assist passengers in arranging their travel and be integrated into traffic assignment models. They can also verify the accessibility of a USPT network and help in the design of timetables, contributing both theoretically and practically.

3. Floyd-A* Algorithm for LTIP-USPT

To solve the LTIP-USPT efficiently, an ad hoc Floyd-A* algorithm is developed that is composed of two procedures, that is, an A*-based Itinerary Finder and a Floyd-based Cost Estimator. The basic scheme of the Floyd-A* algorithm is shown in Figure 3.

The *Cost Estimator* precalculates the estimated travel times of itineraries between any two nodes in a *slacked USPT network*, where static arc travel time is given as the lower bound of the associated time-dependent actual travel time. These values are stored in Table *H*. This is accomplished by a Floyd-based algorithm [26], which is a well-known all-to-all shortest paths algorithm. Once complete, the Cost Estimator is no longer required unless there is an update to the USPT network. The A*-based *Itinerary Finder* makes use of the Table *H* obtained by the Cost Estimator as heuristic information, determining the least-time itinerary. In the case that traveler inputs a triad of (n_o, n_d, t_o) , only the Itinerary Finder conducts a real-time computation. These two procedures are expounded in detail in Sections 3.1 and 3.2, respectively. Section 3.3 mathematically proves its admissibility and analyzes the corresponding computing efficiency by comparing it with Plain-A* and Dijkstra-like procedures.

Remark 4. Speed-up technologies such as “Avoiding Binary Search” and “Further Speedup When Modeling with Train Routes” discussed in the work of Pyrga et al. [25] may further contribute to a higher efficiency. However, this paper only concerns a more efficient A*-based search (also known as goal-directed search), which could coexist with other speed-up technologies to further speed up computing.

3.1. Least-Time Itinerary-Finder Procedure. Assuming that typical readers may not be familiar with the A* algorithm, this searching process will be explained in detail. Given a triad of origin, destination, and initial time (n_o, n_d, t_o) to determine an LT itinerary $p^{t_o}(n_o, n_d) \in P^{t_o}(n_o, n_d)$, the Itinerary Finder expands promising *origin-rooted partial itineraries* (partial itinerary for short) in a node-to-node manner. Beginning with n_o , each *successor* n_{o+} is expanded by searching for each *t*-arc (n_o, n_{o+}, s_o, t_o) in the first round. Each of these *t*-arcs (partial itineraries) may contribute to the LT itinerary. During the second round, we must determine which terminal node of partial itinerary among several candidates is the most promising one.

Let each node n_i be associated with a state denoted by $state(n_i)$. There are three states of node n_i .

- (1) *NEW*: node n_i has not been expanded up to now.
- (2) *OPEN*: node n_i has been expanded and acts as a candidate to expand to another node in the next searching process. That is to say, for each partial itinerary $p^{t_o}(n_o, n_i)$ thus far, $state(n_i) = \text{OPEN}$.
- (3) *CLOSED*: node n_i has been expanded and has already expanding to another node. In other words, for any node $n_j \neq n_i$ that has gone through by any current partial itinerary $p^{t_o}(n_o, n_i)$, $state(n_j) = \text{CLOSED}$.

FIGURE 3: Scheme of Floyd-A* procedure.

As defined above, the nodes associated with the state OPEN are candidates for expanding partial itineraries. For convenience, we use relative time rather than absolute time hereafter. Using the USPT network of Figure 1 as an example, let $n_o = n_1$, $n_d = n_5$ and $t_o = 2$ (minutes after 6:00). Figure 4 combined with Table 2 shows part of the searching process.

We first initialize the state of origin n_1 as OPEN and others as NEW by default (see Figure 4(a)). In the first expansion round (see Figure 4(b)), n_2 , n_3 , and n_7 are expanded by searching for *t*-arcs $(n_1, n_2, l_1, 2)$, $(n_1, n_2, l_2, 2)$, $(n_1, n_3, l_1, 2)$, and $(n_1, n_7, l_5, 2)$. At the same time, n_1 becomes CLOSED; n_2 , n_3 , and n_7 turn from NEW to OPEN. The next paragraph shows that $\{(n_1, n_7, l_5, 2)\}$ is the most promising partial itinerary and $t_7 = 15$; thus, we should continue the second expansion round for the most promising node n_7 , and only n_4 is expanded by searching for a *t*-arc $(n_7, n_4, w, 15)$. This time, n_7 becomes CLOSED and n_4 turns to OPEN (see Figure 4(c)). The searching process continues by similar means. Note that the state of a node may turn from NEW to OPEN, from OPEN to CLOSED, or remain the same. However, a CLOSED node can never re-OPEN (see Theorem 10), for example, $(n_4, n_1, l_2, 16)$ is searched in the 3rd searching round (see Figure 4(d)), but the state of n_1 unconditionally remains CLOSED.

The exposition above focuses on the changing states of nodes during the expansion of partial itineraries. To determine the most promising OPEN node among several candidates, $f'(n_i)$ is defined as the estimated travel time of an LT itinerary $p^{t_o}(n_o, n_i, n_d)$. For each partial itinerary $p^{t_o}(n_o, n_i)$, the terminal node(s) n_i whose $f'(n_i)$ is/are the minimum one(s) among those of all OPEN nodes is/are identified as the most promising one(s). If there is more than one, you may choose the first expanded one.

The actual travel time of the LT itinerary $p^{t_o}(n_o, n_i, n_d)$ can be the summation of two parts calculated as

$$\pi(p^{t_o}(n_o, n_i, n_d)) = \pi(p^{t_o}(n_o, n_i)) + \pi(p^{t_i}(n_i, n_d)). \quad (9)$$

However, it is difficult to calculate $\pi(p^{t_o}(n_o, n_i))$ and $\pi(p^{t_i}(n_i, n_d))$ in real-time within an acceptable computing time. Because of the time-dependence factor, they are not able to be precalculated and stored as fixed values. This is a

TABLE 2: The changing labels of nodes associated with Figure 4.

Node		State(n)	$t_i = g'(n) + t_o$	$g'(n)$	$h'(n)$	Labels $f'(n)$	Pre(n)	Updated
Initialization	n_1	OPEN	2	0	21	21	nil	*√
	n_2	NEW	∞	∞	∞	∞	nil	
	n_3	NEW	∞	∞	∞	∞	nil	
	n_4	NEW	∞	∞	∞	∞	nil	
	n_5	NEW	∞	∞	∞	∞	nil	
	n_6	NEW	∞	∞	∞	∞	nil	
	n_7	NEW	∞	∞	∞	∞	nil	
1st searching round	n_1	CLOSED	2	0	21	21	nil	*
	n_2	OPEN	$\min\{14, 30\} = 14$	12	11.5	23.5	$\{(n_1, n_2, l_1, 2)\}$	*
	n_3	OPEN	25	23	1.5	24.5	$\{(n_1, n_3, l_1, 2)\}$	*
	n_4	NEW	∞	∞	∞	∞	nil	
	n_5	NEW	∞	∞	∞	∞	nil	
	n_6	NEW	∞	∞	∞	∞	nil	
	n_7	OPEN	15	13	10	23	$\{(n_1, n_7, l_5, 2)\}$	*√
2nd searching round	n_1	CLOSED	2	0	21	21	nil	
	n_2	OPEN	14	12	11.5	23.5	$\{(n_1, n_2, l_1, 2)\}$	
	n_3	OPEN	25	23	1.5	24.5	$\{(n_1, n_3, l_1, 2)\}$	
	n_4	OPEN	$15 + 1 = 16$	14	9	23	$\{(n_1, n_7, l_5, 2), (n_7, n_4, w, 15)\}$	*√
	n_5	NEW	∞	∞	∞	∞	nil	
	n_6	NEW	∞	∞	∞	∞	nil	
	n_7	CLOSED	15	13	10	23	$\{(n_1, n_7, l_5, 2)\}$	*
3rd searching round	n_1	CLOSED	2	0	21	21	nil	
	n_2	OPEN	$\min\{14, 51\} = 14$	12	11.5	23.5	$\{(n_1, n_2, l_1, 2)\}$	*√
	n_3	OPEN	25	23	1.5	24.5	$\{(n_1, n_3, l_1, 2)\}$	
	n_4	CLOSED	16	14	9	23	$\{(n_1, n_7, l_5, 2), (n_7, n_4, w, 15)\}$	*
	n_5	OPEN	30	28	0	28	$\{(n_1, n_7, l_5, 2), (n_7, n_4, w, 15), (n_4, n_5, l_3, 16)\}$	*
	n_6	NEW	∞	∞	∞	∞	nil	
	n_7	CLOSED	15	13	10	23	$\{(n_1, n_7, l_5, 2)\}$	

√The node to be COLSED in the next searching round; *The node whose labels are updated in the searching round.

different situation from a static network context. Therefore, $g'(n_i)$ and $h'(n_i)$ are defined to estimate them, respectively. $f'(n_i)$ is their summation, calculated as

$$f'(n_i) = g'(n_i) + h'(n_i). \quad (10)$$

The A*-based Itinerary Finder utilizes the minimum travel time of the partial itinerary $p^{t_o}(n_o, n_i)$, determined to this point as $g'(n_i)$; the strategy for estimating $h'(n_i)$ will be addressed in Section 3.2. To illustrate the process for selecting the most promising node, we again use the USPT network of Figure 1 as an example. A traveler first predetermines $t_1 = 2$. In the first searching round (see Figure 4(b)), $\pi(n_1, n_2, l_1, 2) = 12$ (min), $\pi(n_1, n_2, l_2, 2) = 28$, $\pi(n_1, n_3, l_1, 2) = 23$, and $\pi(n_1, n_7, l_5, 2) = 13$ can be easily determined with Formula (4). One can easily determine that $g'(n_2) = \min\{12, 28\} = 12$, $g'(n_3) = 23$, and $g'(n_7) = 13$. As for the heuristic information yielded by the Cost Estimator, $h'(n_2) = 11.5$, $h'(n_3) = 1.5$, and $h'(n_7) = 10$. Thus, $f'(n_2) = g'(n_2) + h'(n_2) = 12 + 11.5 = 23.5$. Similarly, we have $f'(n_3) = 24.5$ and $f'(n_7) = 23$. Dijkstra-based approaches only consider the performances of origin-rooted partial itineraries and

thus identify $\{(n_1, n_2, l_2, 2)\}$ as the most promising partial itinerary due to $g'(n_2) < g'(n_7) < g'(n_3)$. The A*-based approaches, however, are goal-directed by the heuristics, and the Itinerary Finder selects n_7 as a result of $f'(n_7) < f'(n_2) < f'(n_3)$. Similarly, the second searching round selects the terminal node of $\{(n_1, n_7, l_5, 2), (n_7, n_4, w, 15)\}$, that is, n_4 , as the most promising node, and so forth. Note that in the 3rd searching round, the t -arc $(n_4, n_2, l_2, 16)$ is searched. We have $t_2 = 51$ calculated by Formula (4), meaning that $g'(n_2)$ of OPEN node n_2 will be potentially turned to $51 - t_o = 49$. However, because the previous value of $g'(n_2)$ is 12 and $49 > 12$, the value of $g'(n_2)$ is not updated but remains 12. In another words, the partial itinerary $\{(n_1, n_2, l_1, 2)\}$ dominates $\{(n_1, n_7, l_5, 2), (n_7, n_4, w, 15), (n_4, n_2, l_2, 16)\}$, referring to the partial itinerary from n_1 to n_2 . The destination n_5 is also expanded in this searching round; the associated state turns to OPEN. The searching process will continue, however, until $state(n_5) = \text{CLOSED}$.

Through the scheme that is recursively expanding, comparing, and selecting promising partial itineraries, the algorithm is terminated once the state of destination turns to

FIGURE 4: Partial searching process by Floyd-A*.

CLOSED. If an algorithm is guaranteed to determine an optimal itinerary from origin to destination, we designate it as *admissible*. The Itinerary Finder is proven to be admissible in Section 3.3. The Itinerary Finder places OPEN nodes in an OPEN list and CLOSED nodes in a CLOSED list. If the state of the node cannot be placed either in the OPEN or CLOSED list it is regarded as NEW as default. In a summary of the above analysis, the outline of the Itinerary Finder is presented in Algorithm 1.

If more detailed information (e.g., waiting time, in-vehicle time) is required with respect to a specific arc, this can be obtained by simply adding to $\text{pre}(n_i)$ in the associated iteration.

3.2. Cost-Estimator Procedure for a Tighter Lower Bound. The Itinerary-Finder procedure must be well informed when making a choice to expand partial itineraries. Expanding an unlikely part of an LT itinerary is a waste of computational time, while missing a promising partial itinerary may lead to a failure in determining the LT itinerary. Therefore, the strategy of estimating the travel time of a destination-ended partial itinerary is viewed as the key to improving the efficiency of the Itinerary Finder. Meanwhile, the estimated travel time must

be a lower bound of the real travel time. Note that a tighter lower bound results in higher efficiency.

The travel time of an itinerary is composed of the travel time during walking between two nodes, waiting at nodes for a transfer, and traveling in vehicles. The walking time between two specific nodes is fixed. The waiting time varies in different cases. If fortunate, a traveler can transfer without waiting time. The in-vehicle time depends on the timetable of different lines combined with their different vehicle trips. This paper proposes a strategy to estimate the travel time between two nodes as a tight static lower bound of this real time-dependent value. The basic concept is shown by generating an associated slacked network (see Definition 5) of the USPT network; the minimum travel time of itinerary in this SUSPT network is the associated estimated value in the USPT network.

Definition 5. A slacked USPT network (SUSPT network for short) is defined to share the same topological structure as the USPT network. However, each arc of the SUSPT network is assigned a static travel time as a lower bound of the associated real travel time of the arc in the USPT network. The arc in the SUSPT network is timed by explicitly slacking the associated real travel time by using the following 3 rules.


```

Step 0. (Initialization):
Set OPEN list =  $\emptyset$  and CLOSED list =  $\emptyset$ 
Set  $g'(n_i) = \infty, h'(n_i) = \infty, f'(n_i) = \infty$  and  $pre(n_i) = nil$  where  $n_i \in N$ 
Add  $n_o$  to OPEN list, set  $g'(n_o) = 0$ 
Calculate  $f'(n_o) = g'(n_o) + h'(n_o)$  where  $h'(n_o)$  is pre-calculated by Cost-Estimator
Step 1. (Expanding partial itinerary):
while OPEN list  $\neq \emptyset$  do
    Select any node  $n_i \in \{n_i \mid \min(f'(n_i)), state(n_i) = OPEN\}$ 
    if  $n_i = n_d$  then
        Go to Step 2
    else
        Move  $n_i$  from OPEN list to CLOSED list
    end if
    for all t-arc  $(n_i, n_{i+}, s_i, t_i)$  do
        if  $n_{i+}$  in CLOSED list then
            continue
        end if
        if  $s_i = s_{i-} = w$  then // Set  $s_{o-} = nil$  previously
            continue
        end if
        if  $s_i = s_m$  or  $s_i = \bar{s}_i$  where  $s_m = s_o, s_{o+}, \dots, s_{i-}$  then
            continue
        end if
        Calculate  $\pi(n_i, n_{i+}, s_i, t_i)$  by formula (2) or (4)
        if  $g'(n_i) + \pi(n_i, n_{i+}, s_i, t_i) \geq g'(n_{i+})$  then
            continue
        else if  $n_{i+}$  not in OPEN list then
            Add  $n_{i+}$  to OPEN list
        end if
        Calculate corresponding  $t_{i+}$  by formula (3) or (6)
        Update  $g'(n_{i+})$  with  $g'(n_{i+}) + \pi(n_i, n_{i+}, s_i, t_i)$ 
        Calculate  $f'(n_{i+}) = g'(n_{i+}) + h'(n_{i+})$ 
        Record  $pre(n_{i+}) = \{(n_i, n_{i+}, s_i, t_i)\}$ 
    end for
end while
Step 2. (Reconstructing LT itinerary):
Reconstruct LT itinerary  $p^{t_o}(n_o, n_d)$  by recursively recalling  $pre(n_i)$  where  $n_i = n_d, n_{d-}, \dots, n_{o+}$ 
Return  $p^{t_o}(n_o, n_d)$ 

```

ALGORITHM 1: Procedure of Itinerary Finder (n_o, n_d, t_o) .

Figure 5 shows the associated SUSPT network of the USPT network of Figure 1.

Rule 1. Walking times remain the same.

Rule 2. Ignore all waiting times.

Rule 3. Let the minimum travel time among those traversed by different lines combined with different vehicle trips between two specific nodes be the estimated travel time.

Obviously, there exist no temporal concepts in the static SUSPT network; therefore, let each initial time of t -arc in the SUSPT network be nil . Let π' denote the operator to time the t -arc in SUSPT network. Rule 1 can be reflected in Formula (11). Rules 2 and 3 are interpreted in Formula (12):

$$\pi'(n_i, n_j, w, nil) = \frac{\text{dist}(n_i, n_j)}{v_{\text{walk}}}, \quad (11)$$

$$\pi'(n_i, n_j, l, nil) = \min_k (\theta_{l,k}^{n_i} - \theta_{l,k}^{n_j}). \quad (12)$$

For this problem, typical all-to-all shortest paths algorithms are qualified. This paper chooses a well-known Floyd algorithm. We assume typical readers have already known the algorithm, so there is no detailed exposition here.

The outline of the procedure is shown in Algorithm 2.

Remark 6. In this work, the Itinerary Finder obtains $h'(n_i)$ from table H outputted by the Cost Estimator, while the previous related works substituted $h'(n_i)$ with Formula (13), which was calculated in an online way. The strategy proposed in this work is proved to generate a tighter lower bound and thus leads the Floyd-A* algorithm to be more efficient both in theory and in computation experiments (see Sections 3.3, 4.1, and 4.2):

$$h'(n_i) = \frac{\text{dist}(n_i, n_d)}{v_{\text{max}}}. \quad (13)$$

```

Step 0. (Initialize the SUSPT network):
  for all  $n_i \in N$  do
    for all  $n_j \in N$  do
       $H(i, j) = \min_s \pi'(n_i, n_j, s, nil)$ 
    end for
  end for
Step 1. (Calculate costs of all-to-all shortest paths):
  for all  $n_m \in N$  do
    for all  $n_i \in N$  do
      for all  $n_j \in N$  do
        if  $H(i, j) > H(i, k) + H(k, j)$  then
           $H(i, j) = H(i, k) + H(k, j)$ 
        end if
      end for
    end for
  end for

```

ALGORITHM 2: Procedure of Cost Estimator.

FIGURE 5: Associated SUSPT network of the USPT network of Figure 1.

3.3. Admissibility and Efficiency Analysis. The admissibility and efficiency of the Floyd-A* algorithm are discussed in this section. Hart et al. [17] established how to determine the admissibility of an A* algorithm, which is primarily affected by the travel time estimating strategy of the destination-ended partial itineraries shown in Lemma 7. On this basis, Theorem 8 establishes the admissibility of the Itinerary Finder.

Lemma 7. *If $h'(n_i) \leq \pi(p^{t_i}(n_i, n_d))$, then A* is admissible [17].*

Theorem 8. *The Cost Estimator guarantees that the Itinerary Finder is admissible.*

Proof. The Itinerary Finder is A*-based, where $h'(n_i)$ is computed by the Cost Estimator. To prove Theorem 8, we learn from Lemma 7 that it is equivalent to prove that the Cost Estimator guarantees each $h'(n_i) \leq \pi(p^{t_i}(n_i, n_d))$.

Let $p^{t_i}(n_i, n_d) = \{(n_i, n_{i+}, s_i, t_i), \dots, (n_{d-}, n_d, s_{d-}, t_{d-})\}$ be a destination-ended partial itinerary of the actual LT itinerary $p^{t_o}(n_o, n_d)$. Therefore,

$$\pi(p^{t_i}(n_i, n_d)) = \pi(n_i, n_{i+}, s_i, t_i) + \dots + \pi(n_{d-}, n_d, s_{d-}, t_{d-}). \quad (14)$$

$p^{nil}(n_i, n_d) = \{(n_i, n'_{i+}, s'_i, nil), \dots, (n'_{d-}, n_d, s'_{d-}, nil)\}$ denotes the LT itinerary in the SUSPT network. Note that the topological structures of $p^{t_i}(n_i, n_d)$ and $p^{nil}(n_i, n_d)$ are not necessarily the same.

If $s_i = w$, then for any t -arc (n_i, n_j, w, t_i) , $\pi'(n_i, n_j, w, nil) = \pi(n_i, n_j, w, t_i)$;

$$\text{otherwise } s_i = l, \pi'(n_i, n_j, l, nil) = \min_k (\theta_{l,k}^{n_j} - \theta_{l,k}^{n_i}). \quad (15)$$

In contrast, $\pi(n_i, n_j, l, t_i)$ can be calculated by Formula (4) combined with Formula (5); that is,

$$\pi(n_i, n_j, l, t_i) = \theta_{l,k_0}^{n_j} - t_i, \quad (16)$$

where

$$k_0 = \arg \left\{ \min_k (\theta_{l,k}^{n_j} - t_i \mid \theta_{l,k}^{n_j} - t_i > 0) \right\}. \quad (17)$$

Obviously,

$$\begin{aligned} \pi'(n_i, n_j, l, nil) &= \min_k (\theta_{l,k}^{n_j} - \theta_{l,k}^{n_i}) \leq \theta_{l,k_0}^{n_j} - \theta_{l,k_0}^{n_i} \\ &\leq \theta_{l,k_0}^{n_j} - t_i = \pi(n_i, n_j, l, t_i). \end{aligned} \quad (18)$$

In summation, for any t -arc (n_i, n_j, s_i, t_i) , $\pi'(n_i, n_j, s_i, nil) \leq \pi(n_i, n_j, s_i, t_i)$.

Furthermore,

$$\begin{aligned} h'(n_i) &= \pi'(p^{nil}(n_i, n_d)) \leq \pi'(p^{t_i}(n_i, n_d)) \\ &= \pi'(n_i, n_{i+}, s_i, nil) + \dots + \pi'(n_{d-}, n_d, s_{d-}, nil) \\ &\leq \pi'(n_i, n_{i+}, s_i, t_i) + \dots + \pi(n_{d-}, n_d, s_{d-}, t_{d-}) \\ &= \pi(p^{t_i}(n_i, n_d)). \end{aligned} \quad (19)$$

Thus, Theorem 8 is proven. $\square$

If Inequality (20) is satisfied, we deem this a *consistency assumption* for the Itinerary Finder. The definition of this

assumption helps to explain why the Itinerary Finder never re-OPENS a CLOSED node. The explanation can be found in Theorem 10:

$$\pi(n_i, n_j, s_i, t_i) + \pi'(p^{nil}(n_j, n_d)) \geq \pi'(p^{nil}(n_i, n_d)). \quad (20)$$

Lemma 9. Assuming that the consistency assumption is satisfied, A^* needs never to re-OPEN a CLOSED node [17].

Theorem 10. The Cost Estimator assures that the Itinerary Finder needs never to re-OPEN a CLOSED node.

Proof. To prove Theorem 10, we learn from Lemma 9 that it is equivalent to prove that the estimating strategy proposed in the Cost Estimator satisfies the consistency assumption:

$$\begin{aligned} \text{Let } p^{nil}(n_j, n_d) &= \{(n_j, n_{j+}, s_j, nil), \dots, (n_{d-}, n_d, s_{d-}, nil)\}, \\ \text{and } p^{nil}(n_i, n_d) &= \{(n_i, n_{i+}, s_i, nil), \dots, (n'_{d-}, n_d, s'_{d-}, nil)\}. \end{aligned} \quad (21)$$

One can prove that $\pi'(n_i, n_j, s_i, nil) \leq \pi(n_i, n_j, s_i, t_i)$. Thus,

$$\begin{aligned} &\pi(n_i, n_j, s_i, nil) + \pi'(p^{nil}(n_j, n_d)) \\ &= \pi(n_i, n_j, s_i, t_i) + \pi'(n_j, n_{j+}, s_j, nil) \\ &\quad + \dots + \pi'(n_{d-}, n_d, s_{d-}, nil) \\ &\geq \pi'(n_i, n_j, s_i, nil) + \pi'(n_j, n_{j+}, s_j, nil) \\ &\quad + \dots + \pi'(n_{d-}, n_d, s_{d-}, nil) \\ &\geq \pi'(n_i, n_{i+}, s_i, nil) + \dots + \pi'(n'_{d-}, n_d, s'_{d-}, nil) \\ &= \pi'(p^{nil}(n_i, n_d)). \end{aligned} \quad (22)$$

In other words, the consistency assumption is satisfied. Theorem 10 is thus proven. $\square$

Previous related works had developed two variants of Itinerary-Finder procedure, that is, the *Plain-A** procedure in which the value $h'(n_i)$ is revised by calculating in Formula (13) and the *Dijkstra-like* procedure in which $h'(n_i)$ is replaced by constant 0. Similarly, it is not difficult to prove that the Dijkstra-like and Plain-A* both satisfy the consistency assumption and are thus admissible. We show the comparison among the Floyd-A* and the two procedures as follows.

Lemma 11. Consider the set of lower bounds verifying the consistency assumption. If a node is selected by the A^* algorithm for a given lower bound, then this node will be selected by the A^* algorithm using any smaller lower bound [18].

Let N_{DA} , N_{SA} , and N_{DL} denote the sets of expanded nodes by Floyd-A*, Plain-A*, and Dijkstra-like, respectively. According to Lemma 11, Theorem 12 refers to their relationships.

Theorem 12. $N_{DA} \subseteq N_{SA} \subseteq N_{DL}$.

Proof. To prove Theorem 12, one can equivalently prove that the estimated travel time values of Floyd-A*, Plain-A*, and Dijkstra-like are each a smaller lower bound of the real cost than the next; that is, $\pi(p^{t_i}(n_i, n_d)) \geq \pi'(p^{nil}(n_i, n_d)) \geq \text{dist}(n_i, n_d)/v_{\max} \geq 0$.

Theorem 8 has proved $\pi(p^{t_i}(n_i, n_d)) \geq \pi'(p^{nil}(n_i, n_d))$, and it is obvious that $\text{dist}(n_i, n_d)/v_{\max} \geq 0$ because $\text{dist}(n_i, n_d)$ and $v_{\max}$ are both positive.

We therefore need only prove that

$$\pi'(p^{nil}(n_i, n_d)) \geq \frac{\text{dist}(n_i, n_d)}{v_{\max}}, \quad (23)$$

where $p^{nil}(n_i, n_d) = \{(n_i, n_{i+}, s_i, nil), \dots, (n_{d-}, n_d, s_{d-}, nil)\}$.

$\pi'(n_i, n_j, s_i, nil)$ can be calculated as $\text{len}(n_i, n_j, s_i, nil)/v(n_i, n_j, s_i, nil)$.

With regard to $\text{dist}(n_i, n_d)/v_{\max}$, the numerator $\text{dist}(n_i, n_d)$ is obviously not greater than the real distance of any itinerary from n_i to n_d , and the denominator $v_{\max}$ is not less than any velocity observed by walking, bus, and metro. Then,

$$\begin{aligned} &\pi'(p^{nil}(n_i, n_d)) \\ &= \pi'(n_i, n_{i+}, s_i, nil) + \dots + \pi'(n_{d-}, n_d, s_{d-}, nil) \\ &= \frac{\text{len}(n_i, n_{i+}, s_i, nil)}{v(n_i, n_{i+}, s_i, nil)} + \dots + \frac{\text{len}(n_{d-}, n_d, s_{d-}, nil)}{v(n_{d-}, n_d, s_{d-}, nil)} \\ &\geq \frac{\text{len}(n_i, n_{i+}, s_i, nil) + \dots + \text{len}(n_{d-}, n_d, s_{d-}, nil)}{v_{\max}} \\ &\geq \frac{\text{dist}(n_i, n_d)}{v_{\max}}. \end{aligned} \quad (24)$$

Thus, the theorem is proven. $\square$

Corollary 13. $|N_{DA}| \leq |N_{SA}| \leq |N_{DL}|$.

Under the premises of Theorem 12, Corollary 13 can be easily determined, meaning that the total number of expanded nodes from the Dijkstra-like, Plain-A*, and Floyd-A* algorithms are each no less than the next. Correspondingly, their efficiencies increase orderly.

In summation, the searching scopes of the three procedures intuitively seem to be as shown in Figure 6 and their efficiencies are shown as tested in Sections 4.1 and 4.2.

4. Example Illustration and Analysis

A numerical example and a real-world USPT network instance are presented to demonstrate the suitability and efficiency of the proposed Floyd-A* algorithm, as well as the instructive significance for travelers. For this purpose, the experiments are composed of four parts. Section 4.1 shows the efficiency of the Floyd-A* algorithm through a numerical example, in comparison with the two other conventional procedures, that is, the Dijkstra-like and Plain-A* procedures. A real-world instance is tested to demonstrate applicability and efficiency of the Floyd-A* algorithm when solving large-scale

FIGURE 6: Searching scope of three procedures.

network instance, which is given in Section 4.2. The experiments on time-dependent nature of the least-time itinerary and the phenomenon “just missing” and “just boarding” cases are presented in Sections 4.3 and 4.4, respectively. The experiments ran in a MATLAB environment on an HP Compaq 8280 Elite CMT PC with Intel Core i5-2400 CPU @3.1G Hz and 4GB memory (RAM).

The USPT network of the numerical example shown in Figure 7 is formed by 30 nodes and 103 arcs. There are 10 lines, including 2 metro lines and 8 bus lines, where $\bar{l}_4 = l_8$; 10 corresponding timetables are also provided. Node n_5 is traversed by l_1 , l_2 , and l_3 ; each of another ten nodes is simultaneously traversed by 2 lines. Specific data are omitted due to the limited space.

4.1. High Efficiency of the Floyd-A* Algorithm. An itinerary planning assistant is capable of determining the LT itinerary through real-time querying. Efficiency is the ultimate goal. For testing, 1000 triads of n_o , n_d , and initial time t_o are randomly generated with the distance between each pair of n_o , n_d no less than 5000 meters; all pairs are connectable. Note that in the generating process, cases exist where no itinerary from n_o to n_d was found. Therefore, the Floyd-A* algorithm is capable of verifying the connexity of a USPT network. Given each triad (n_o , n_d , and t_o), each of the three procedures (i.e., Floyd-A*, Plain-A*, and Dijkstra-like aforementioned in Section 3.3) are used to solve the LTIP-USPT. The Dijkstra-like and Plain-A* procedures are traditional methods for solving these types of problems. We utilize two indicators, one averaged the running time during the calculating of the LT itinerary by a specific procedure, and the other averaged the expansion times of nodes during the searching process. Using the performance of Dijkstra-like procedure as a reference, the relative reductions of the two indicators are shown in Table 3 as well. In addition, the results outputted by different procedures in a specific instance are exactly the same. Considering that the three procedures are all admissible, we learn from Table 3 that Floyd-A* procedure reduced the running time by 33.3% and the expansion times of nodes by 61.58% compared with the Dijkstra-like procedure, while the two corresponding values were 12.84% and 25.34% savings

FIGURE 7: An USPT network example for illustration.

from the Plain-A* procedure. Floyd-A* procedure proposed in this paper is superior to both the Plain-A* and the Dijkstra-like conventional procedures in terms of efficiency.

4.2. Applicability of the Floyd-A* Algorithm for Real-World Instance. To verify the applicability and efficiency of the Floyd-A* procedure in a real-world network, we implement and test the three procedures in a Visual Studio 2010 environment on the aforementioned PC, using the real-world public transport data of Shenyang City, the central city of northeastern China. The main urban zone of Shenyang City has a size of more than 700 square kilometers, and a population of more than 5 million until the year 2010. There are totally 446 directed USPT lines, which are composed of 2 metro lines and 444 bus lines. The modeled Shenyang City USPT network (within the main urban zone) is formed by 2812 nodes (after aggregating) and 184178 arcs. Similar to the experiments performed in Section 4.1, 1000 triads of n_o , n_d , and initial time t_o are randomly generated; the performances are shown in Table 4. The real-world LTIP-USPT can be solved by the Floyd-A* procedure in a more efficient way; it reduces the averaged running time by 63.9% compared with that solved by the Dijkstra-like procedure. Therefore, we concluded that the Floyd-A* procedure is significantly superior to the previous related work, that is, both the Plain-A* and the Dijkstra-like procedures, with reference to efficiency.

In reality, faced with such a large network, local citizens and tourists are difficult to determine an optimal itinerary without an itinerary planning system. To benefit the travelers, the Floyd-A* algorithm module is implemented and embedded in a Shenyang City Public Transport Query System, shown in Figure 8. The system is implemented in a Visual Studio 2010 environment, combined with the geography information system TransCAD. In a case that a traveler wants the least-time travel from the Bainaohui Stop to the Wanquan Park Stop given the initial starting time 9:10, the system returns the solution that the traveler should cost 24 minutes

TABLE 3: Efficiency comparison of three procedures for LTIP-USPT in Figure 7.

Procedures	Running time (ms)	Item		
		Relative reduction of running time (%)	Expanding times of nodes (sec.)	Relative reduction of expansion times (%)
Dijkstra-like	2.0422	0	53.64	0
Plain-A*	1.78	12.84	40.049	25.34
Floyd-A*	1.3621	33.3	20.611	61.58

TABLE 4: Efficiency comparison of three procedures for LTIP in Shenyang City USPT network.

Procedures	Running time (ms)	Item	
		Relative reduction of running time (%)	
Dijkstra-like	43.5	0	
Plain-A*	34.2	21.4	
Floyd-A*	15.7	63.9	

(including in-vehicle time and waiting time) traveling from the Bainaohui Stop to the EPA Stop by Line 222, walking 1 minute to another EPA Stop, and finally arriving at the Wanquan Park Stop by Line 118 in 15 minutes. Note that the two EPA Stops are geographically different but close. The interface and the LT itinerary of the example are given as shown in Figure 8. It appears to be applicable and efficient after numerous experiments. It finally turns out that Floyd-A* can potentially be used into many large-scale real-world USPT networks for LT itinerary planning use. Note that it also has the potential to be applied in interurban context, given that all services are schedule-based.

4.3. Time-Dependent Nature. In a static public transport network that does not consider a timetable, it is obvious that, given an origin and destination pair, the optimal itinerary (also referred as path) will consider objectives such as the least transfer time and the lowest financial expense. In other words, the solution does not depend on the departure time, while the situation is different when considering a timetable.

In the case of a specified origin and destination, when given different initial time t_o , the proposed computation method returns a different LT itinerary $p^{t_o}(n_o, n_d)$ and corresponding travel time $\pi(p^{t_o}(n_o, n_d))$. Using the USPT network of Figure 7 as an example, $n_o = n_1$ and $n_d = n_{24}$ are pre-determined; when given a different initial time, for example, $t_o = 55$ and $t_o = 60$, the itinerary $p^{55}(n_1, n_{24})$ and itinerary $p^{60}(n_1, n_{24})$ found with the Itinerary-Finder procedure are LT itineraries in these two cases, respectively. These results are shown in Figure 9, where the horizontal axis represents the time of day and the vertical axis represents the accumulated travel distance of the itinerary. The circles represent nodes and the links are explained in the legend. It is not difficult to see that the slope of the link represents the corresponding velocity and the curve must be monotonically increasing:

FIGURE 8: An example of system interface of LT itinerary planning in USPT network.

FIGURE 9: Two LT itineraries with different initial times.

$p^{55}(n_1, n_{24}) = \{(n_1, n_3, l_1, 55), (n_3, n_{15}, w, 70), (n_{15}, n_{18}, l_6, 71.3), (n_{18}, n_{24}, w, 84)\}$ costs 33.1 minutes, traveling 12336 meters; the itinerary $p^{60}(n_1, n_{24}) = \{(n_1, n_2, l_1, 60), (n_2, n_{10}, l_5, 69), (n_{10}, n_{24}, w, 95)\}$ costs 37.6 minutes, traveling 11974 meters.

The *Spatial itinerary* is defined as an itinerary with the temporal factors deleted. The spatialitineraries of itineraries $p^{55}(n_1, n_{24})$ and $p^{60}(n_1, n_{24})$ are represented as $sp_1(n_1, n_{24}) = \{(n_1, n_3, l_1), (n_3, n_{15}, w), (n_{15}, n_{18}, l_6), (n_{18}, n_{24}, w)\}$ and $sp_2(n_1, n_{24}) = \{(n_1, n_2, l_1), (n_2, n_{10}, l_5), (n_{10}, n_{24}, w)\}$, respectively, as shown in Figure 10. If we neglect the waiting time at transfer, the static itinerary $sp_1(n_1, n_{24})$ intuitively appears more likely to cost less time than $sp_2(n_1, n_{24})$ because about half the

FIGURE 10: Two static itineraries.

distance of $sp_1(n_1, n_{24})$ is traversed by the metro, which is much faster than a bus and the total distances of the two itineraries are very close. How can $sp_2(n_1, n_{24})$ sometimes cost less time than $sp_1(n_1, n_{24})$, for example, when $t_o = 60$?

To answer this question, the corresponding itineraries of $sp_1(n_1, n_{24})$ and $sp_2(n_1, n_{24})$, both given an initial time of 60, are compared in Figure 11. The waiting time of the former itinerary is 3.7 minutes longer than the latter one, while the value of total travel time is only 2.4 minutes longer. Similar results can be found in other cases. Therefore, we conclude that the complex timetables that lead to waiting times during transfers are variable and almost uncontrollable, primarily resulting in the time-dependent nature of an LT itinerary in a USPT network. Obviously, these results could not be determined without considering timetables.

4.4. Just Missing and Just Boarding. Recall that the USPT lines are assumed to run punctually. In this premise, the phenomena of “just missing” and “just boarding” can be evaluated with the proposed algorithm. Again, we let $n_o = n_1$ and $n_d = n_{24}$. When $t_o = 98$ (a “just boarding” case) and $t_o = 98 + \xi$ (a “just missing” case), the approach determines LT itineraries $p^{98}(n_1, n_{24})$ and $p^{98+\xi}(n_1, n_{24})$, respectively, as shown in Figure 12. There is no waiting time at n_1 associated with $p^{98}(n_1, n_{24})$ while with $p^{98+\xi}(n_1, n_{24})$ the traveler must wait for 6 minutes to board the vehicle of l_1 . Furthermore, we learn that when t_o varies in a continuous interval $(98, 104]$, the solutions are nearly the same, only differing in waiting time at n_o . For example, $p^{98+\xi}(n_1, n_{24})$ costs 40 minutes to get to n_{24} , while it only costs 34 minutes in $p^{104}(n_1, n_{24})$. They both arrive at n_{24} at 138 but experience different waiting times at n_1 . If these results are preknown by the traveler, he might adjust earlier to $t_o = 98$ and take 32 minutes to arrive at n_d or postpone to $t_o = 104$. In fact, the waiting time for transfer between two lines is almost uncontrollable, depending entirely on complex timetables, while the time spent at n_o is controllable, depending on both t_o and the timetable. These findings can significantly help travelers determine an ideal initial time to begin travel by meeting a “just boarding” case and avoiding a “just missing” case, thereby saving time. These two categories of special cases could not be determined without considering a timetable.

FIGURE 11: Two different itineraries with the same initial time.

FIGURE 12: “Just missing” and “just boarding” cases.

5. Conclusions and Future Work

This paper has presented an ad hoc Floyd-A* algorithm to determine the least-time itinerary from origin to destination in an urban scheduled public transportation network when given initial time to start the travel. A modified representation of the USPT network and the travel itinerary was proposed. The itinerary in a bimodal, time-dependent USPT network was composed of time-label arcs, whose timing method was explicitly specified. Traveler operability was given to constrain the itinerary structure. A methodology to estimate travel time between two nodes as a sufficiently tight static lower bound of the corresponding real travel time was proposed as a key contribution for high efficiency. The ad hoc Floyd-A* procedure was mathematically proven to be correct and more efficient than the Plain-A* and Dijkstra-like procedures, which appeared in previous related works. Meanwhile, through an illustrated example and a real-world example, we showed that the Floyd-A* algorithm appears to be very suitable and efficient for LTIP-USPT. These results reflect the time-dependent nature of the least-time itinerary in a scheduled network and can serve as guidance for travelers in predetermining an ideal initial time by meeting “just boarding” cases while avoiding “just missing” cases in the

premise that lines run punctually. The efficiency performance was numerically tested to be superior to both Plain-A* and Dijkstra-like procedures.

It should be noted that the proposed approach for LTIP-USPT is primarily based on the hypothesis that vehicles of lines run in absolute compliance with their timetables. However, it is evident that early arrivals or delays can occur at each node during vehicle trips, and the uncertainties are dynamically revealed. One way to mitigate these uncertain effects is to consider the stochastic optimization or robust optimization methodologies. Floyd-A* may not be suitable for direct application at that time, but its adaptation (maybe with a parallel program) should be considered. These topics certainly constitute a motivation for future works.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research is financially supported by the National Natural Science Foundation of China (71021061), the Fundamental Research Funds for the Central Universities (N090204001, N110404021, N110204005) and the National College Student Innovative Experimental Project of China. The authors also gratefully acknowledge the insightful comments and suggestions made by the anonymous referees.

References

- [1] H. Bast, "Car or public transport—two worlds," *Efficient Algorithms*, vol. 5760, pp. 355–367, 2009.
- [2] R. B. Dial, "Transit pathfinder algorithm," *Highway Research Record*, vol. 205, pp. 67–85, 1967.
- [3] H. Spiess and M. Florian, "Optimal strategies: a new assignment model for transit networks," *Transportation Research Part B*, vol. 23, no. 2, pp. 83–102, 1989.
- [4] S. C. Wong and C. O. Tong, "Estimation of time-dependent origin-destination matrices for transit networks," *Transportation Research B*, vol. 32, no. 1, pp. 35–48, 1998.
- [5] D. Canca, A. Zarzo, P. L. Gonzalez-R, E. Barrena, and E. Algaba, "A methodology for schedule-based paths recommendation in multimodal public transportation networks," *Journal of Advanced Transportation*, vol. 47, no. 3, pp. 319–335, 2013.
- [6] W. Xu, S. He, R. Song, and S. S. Chaudhry, "Finding the K shortest paths in a schedule-based transit network," *Computers & Operations Research*, vol. 39, no. 8, pp. 1812–1826, 2012.
- [7] M.-C. Tan, C. O. Tong, S. C. Wong, and J.-M. Xu, "An algorithm for finding reasonable paths in transit networks," *Journal of Advanced Transportation*, vol. 41, no. 3, pp. 285–305, 2007.
- [8] R. Huang, "A schedule-based pathfinding algorithm for transit networks using pattern first search," *GeoInformatica*, vol. 11, no. 2, pp. 269–285, 2007.
- [9] A. Nuzzolo, U. Crisalli, and L. Rosati, "A schedule-based assignment model with explicit capacity constraints for congested transit networks," *Transportation Research C: Emerging Technologies*, vol. 20, no. 1, pp. 16–33, 2012.
- [10] Y. Hamdouch, H. W. Ho, A. Sumalee, and G. Wang, "Schedule-based transit assignment model with vehicle capacity and seat availability," *Transportation Research B: Methodological*, vol. 45, no. 10, pp. 1805–1830, 2011.
- [11] M. H. Poon, S. C. Wong, and C. O. Tong, "A dynamic schedule-based model for congested transit networks," *Transportation Research B: Methodological*, vol. 38, no. 4, pp. 343–368, 2004.
- [12] C. O. Tong and A. J. Richardson, "A computer model for finding the time-dependent minimum path in a transit system with fixed schedules," *Journal of Advanced Transportation*, vol. 18, no. 2, pp. 145–161, 1984.
- [13] M. E. T. Horn, "An extended model and procedural framework for planning multi-modal passenger journeys," *Transportation Research B*, vol. 37, no. 7, pp. 641–660, 2003.
- [14] K. N. Androustopoulos and K. G. Zografos, "Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network," *European Journal of Operational Research*, vol. 192, no. 1, pp. 18–28, 2009.
- [15] K. G. Zografos and K. N. Androustopoulos, "Algorithms for itinerary planning in multimodal transportation networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 175–184, 2008.
- [16] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: state of the art," *Computers & Operations Research*, vol. 33, no. 11, pp. 3324–3343, 2006.
- [17] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [18] I. Chabini and S. Lan, "Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, no. 1, pp. 60–74, 2002.
- [19] G. Nannicini, D. Delling, D. Schultes, and L. Liberti, "Bidirectional A* search on time-dependent road networks," *Networks*, vol. 59, no. 2, pp. 240–251, 2012.
- [20] M. Yu, Y. Ni, Z. Wang, and Y. Zhang, "Dynamic route guidance using improved genetic algorithms," *Mathematical Problems in Engineering*, vol. 2013, Article ID 765135, 6 pages, 2013.
- [21] Y. Chen, M. G. H. Bell, and K. Bogenberger, "Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 1, pp. 14–20, 2007.
- [22] A. V. Goldberg, "Point-to-point shortest path algorithms with preprocessing," *Lecture Notes in Computer Science*, vol. 4362, pp. 88–102, 2007.
- [23] J. Maue, P. Sanders, and D. Matijevic, "Goal-directed shortest-path queries using precomputed cluster distances," *Journal of Experimental Algorithmics*, vol. 14, article 2, 2009.
- [24] M. Muller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, "Timetable information: models and algorithms," *Algorithmic Methods for Railway Optimization*, vol. 4359, pp. 67–90, 2007.
- [25] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Efficient models for timetable information in public transportation systems," *ACM Journal of Experimental Algorithmics*, vol. 12, article 2.4, 2008.
- [26] R. W. Floyd, "Algorithm 97: shortest path," *Communications of ACM*, vol. 5, no. 6, p. 345, 1962.