

## **Chapter 5**

# **ROUTE PLANNER**

<b>CHAPTER OUTLINE .....</b>	<b>1</b>
<b>5.1 Overview.....</b>	<b>3</b>
<b>5.2 Terminology.....</b>	<b>4</b>
<b>5.3 Key Concepts .....</b>	<b>7</b>
5.3.1 Shortest path problem .....	7
5.3.2 Dijkstra's algorithm.....	7
5.3.3 Network layers.....	7
<b>5.4 Major Data Inputs .....</b>	<b>8</b>
5.4.1 Generating Single-Trip Requests .....	9
5.4.2 Generating the Internal Network.....	10
5.4.2.1 Generating nodes and links in the Internal Network.....	12
5.4.2.2 Generating travel time functions for each link in the Internal Network .....	15
<b>5.5 Major Data Outputs .....</b>	<b>21</b>
<b>5.6 Module Interfaces .....</b>	<b>26</b>
5.6.1 Inputs received from the Activity Generator Module .....	26
5.6.2 Inputs received from the Traffic Microsimulator Module.....	27
5.6.3 Outputs sent to the Traffic Microsimulator Module.....	27
5.6.4 Feedback to the Selector Module .....	28
<b>5.7. Configuration Files.....</b>	<b>33</b>
<b>5.8 Algorithms.....</b>	<b>35</b>
5.8.1 Time-Independent Shortest Path Problem (TISP) .....	35
5.8.1.1 Practical Example for the Time-Independent Shortest Path Problem (TISP) .....	41
5.8.2 Time-Independent Label Constrained Shortest Path Problem (TILSP) .....	46
5.8.2.2 Example of the Time-Independent Label Constrained Shortest Path Problem (TILSP) .....	48
5.8.3 Time-Dependent Label-Constrained Shortest Path Problem (TDLSP) .....	54
<b>5.9 Example of the Route Planner Module.....</b>	<b>57</b>
<b>APPENDIX A .....</b>	<b>64</b>

# ROUTE PLANNER

## Chapter Outline

The preceding chapters have discussed the following:

- The generation of synthetic households from census data at the block group level or the census tract level.
- The development of associated demographic characteristics (e.g., age, gender, income, etc.) for each synthetic household.
- The placement of each synthetic household on a link in the transportation network.
- The assignment of vehicles to each household, including information regarding the vehicle emission type and the initial vehicle location on the network.
- The assignment of a set of activities, their priority, travel mode, and vehicle preference for each household member of the synthetic household for a 24-hour period.
- The determination of the location of each activity that takes place away from the household, by activity type.

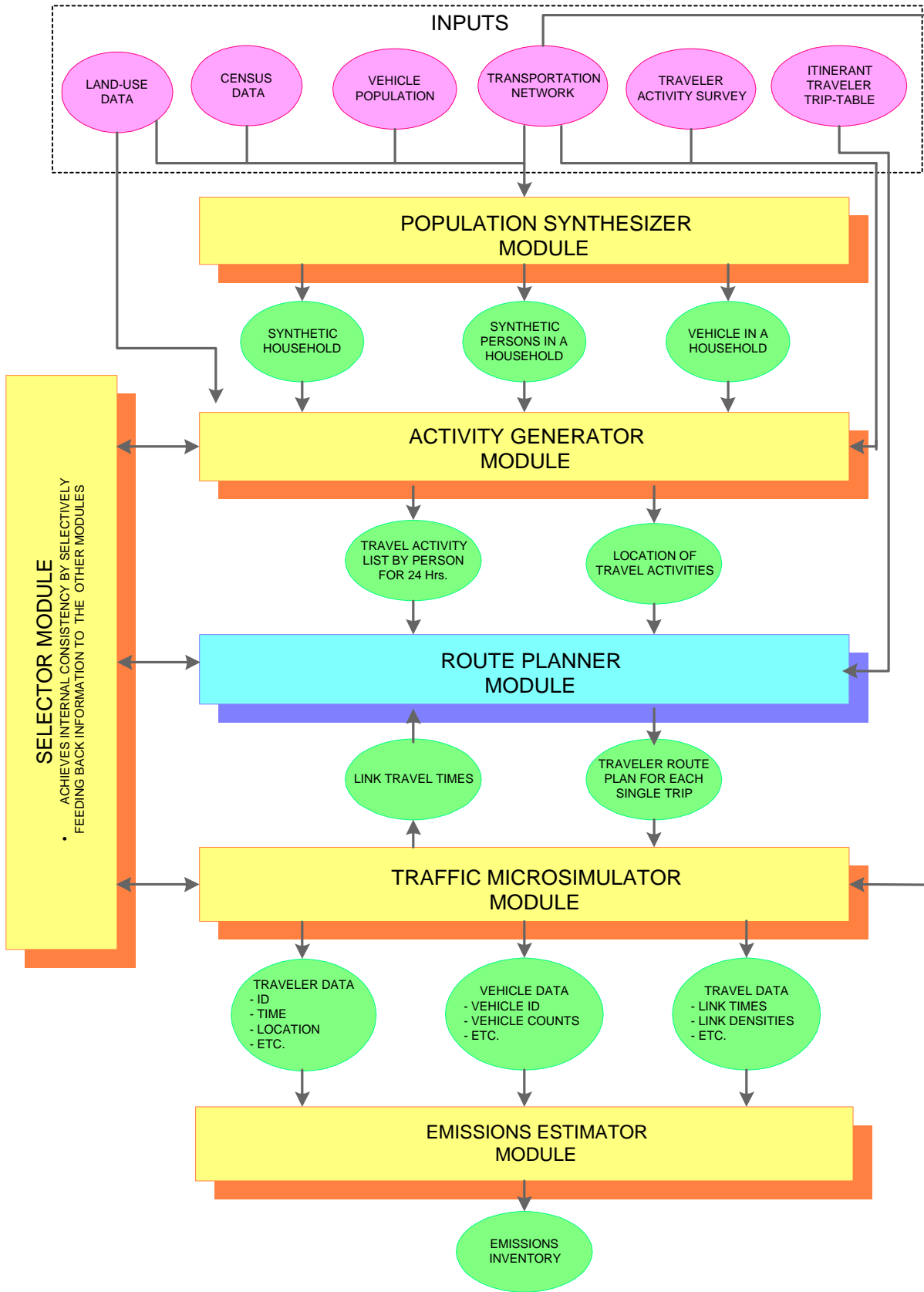
This chapter is concerned with the Route Planner, the third module in the TRANSIMS framework, which develops a time-dependent label-constrained shortest path for each trip executed by a traveler in the system. Figure-5.1 details the inputs and outputs of this module, along with its interactions with the other relevant modules of TRANSIMS. There are several shortest path techniques that vary by complexity, required data, computer processing time, and appropriate application. However, most techniques require some combination of the following inputs:

- A network consisting of nodes and links.
- A travel time function on each link, which could be a time-independent or a time-dependent function. Time-dependent functions account for time-of-day delays resulting from actual travel conditions such as peak-hour congestion.
- A string of admissible mode labels that delineates the permissible travel mode sequences that could be adopted by the user in traveling from the origin to the destination of the trip.

The technique adopted by TRANSIMS to identify a suitable travel route for any user is a variant of Dijkstra's procedure for finding shortest paths, which is suitably modified to accommodate time-dependent travel times and label sequence constraints. The underlying problem is referred to as Time-Dependent Label-Constrained Shortest Path Problem (TDLSP), and is unique to TRANSIMS applications.

This section begins with an overview of the Route Planner Module, a definition of the terminology, some key concepts, the major data inputs and outputs, various module interfaces, and values for the relevant configuration files. Thereafter, three shortest path problems on networks are described, along with the salient concepts of their solution. These problems are as follows:

- 1 Time-Independent Shortest Path Problem (TISP),
- 2 Time-Independent Label-Constrained Shortest Path Problem (TILSP), and
- 3 Time-Dependent Label-Constrained Shortest Path Problem (TDLSP).



**Figure-5.1: TRANSIMS framework**

## 5.1 Overview

The Route Planner Module develops the route plans based on the demand represented in the Activities data file. Each traveler, including itinerant travelers, truck drivers, and transit drivers, has an individual travel plan. The overall travel plan consists of several *single-trip requests*\*. Each single-trip request is created from the activity file, traveler list, mode preference file, *mode string*\*, and vehicle file (see Figure-5.2).

The Route Planner computes a label-constrained, time-dependent shortest path for each single-trip request. This is achieved by the Route Planner based on the construction of an *Internal Network*\*, given the *TRANSIMS network*\*, along with link travel time functions as obtained via a feedback from the Traffic Microsimulator Module.

Once the plans are generated for all the travelers, they are simultaneously fed into the Traffic Microsimulator Module.

\*Please see Section 5.2 for the terminology and for relevant definitions.

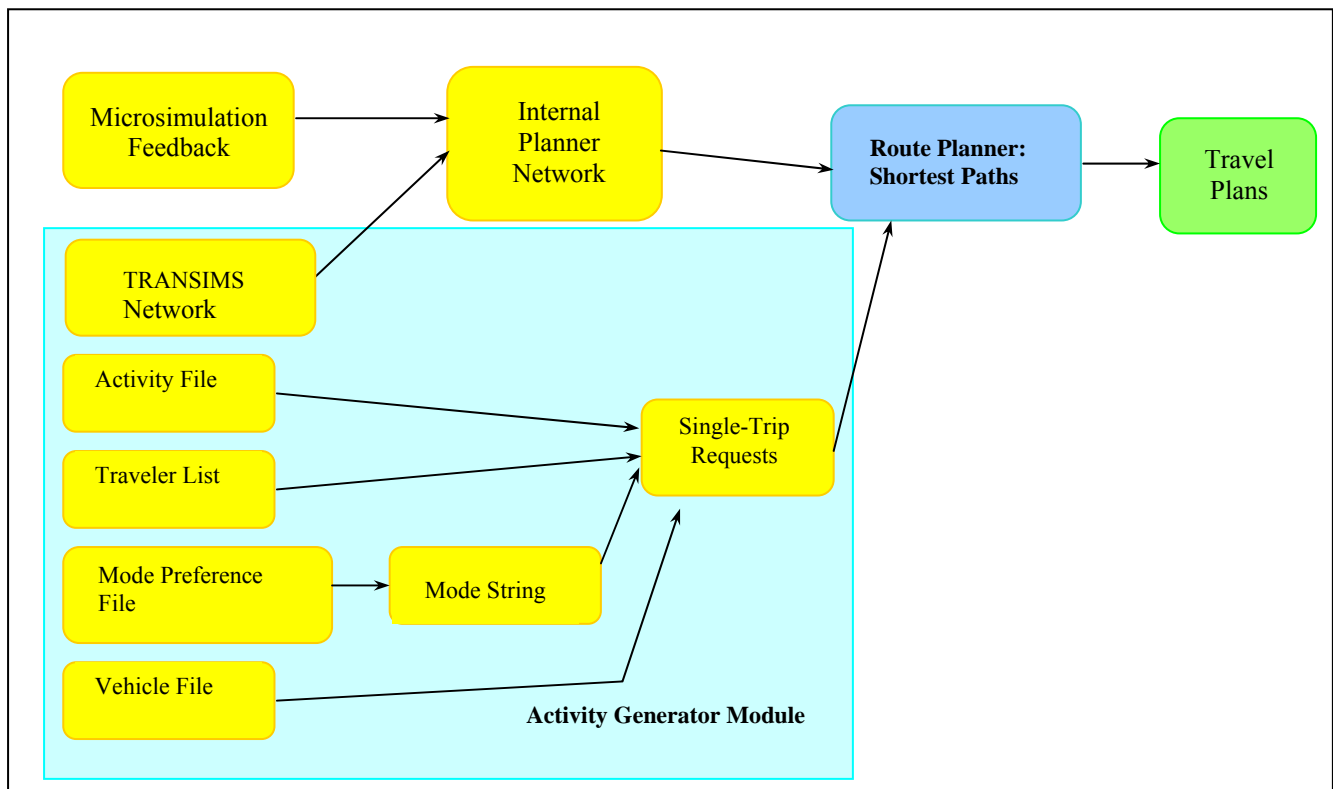
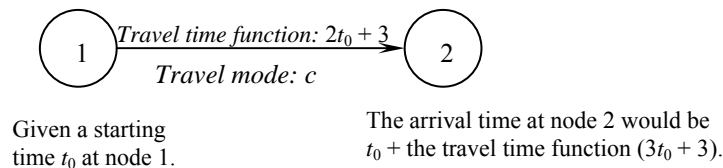


Figure-5.2: Data flow diagram of the TRANSIMS Route Planner

## 5.2 Terminology

- **Single-trip request:** A single-trip request mainly consists of a starting location ( $O$ ), a destination location ( $D$ ), a starting time ( $t_0$ ), a maximum finish time ( $T$ ), and a mode string.
- **Transit:** From the point of view of the Route Planner, a transit vehicle is considered to be any vehicle that makes scheduled stops along a predetermined route, such as, buses, trains, streetcar, etc.
- **TRANSIMS network:** The TRANSIMS network provides information about streets, intersections, signals, parking, activity locations, and transit modes within a road transportation network. This information is used to construct the Internal Planner Network.
- **Internal Network:** The Route Planner translates the TRANSIMS network into a working form, the “Internal Network”, to facilitate the time-dependent, label-constrained routing algorithm. The Internal Network consists of nodes, links, time-dependent travel time function on each link, and the possible travel mode on each link. The travel time function on each link is time-dependent, that is, the travel on a link may incur different travel times at different times of the day. Information regarding the delays on the links is derived from the Traffic Microsimulator output and provided via the Feedback File.
- **Activity location:** An activity location is a place where a traveler’s activities (such as work, home, shopping) can take place.
- **Node:** This is a physical location in the TRANSIMS network, such as an intersection, activity location, bus stop, etc.
- **Link:** This is a unidirectional connection between a pair of nodes. The example below depicts a link between nodes 1 and 2. Every link has a travel time function, a travel mode, and a layer associated with it. For a traveler going from node 1 to node 2 using travel mode “c”(car), with a starting time  $t_0$  and a travel time obtained via a travel time function “ $2t_0 + 3$ ”, which depends on the time  $t_0$  at which the traveler enters the link at node 1, we can represent this information as follows:

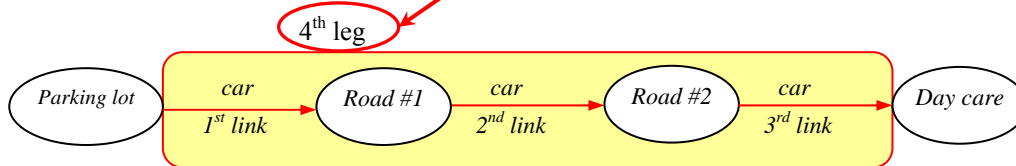
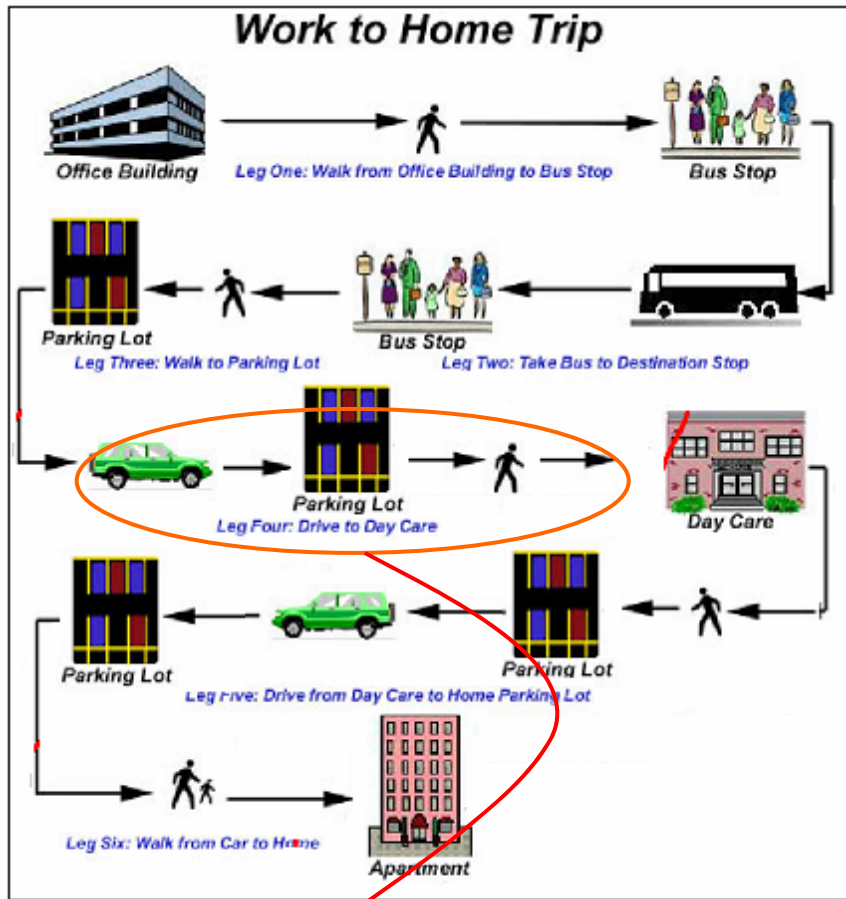


Suppose that the traveler arrives at node 1 at time 4. Then the travel time on this link to node 2 is equal to  $2(4) + 3 = 11$ . Hence, the traveler would arrive at node 2 at time  $4 + 11 = 15$ .

- **Travel time function:** Each link in the Internal Network has a travel time associated with it. Links on the street layer have a travel time for driving on that link. Links on the walk layer have a travel time for walking on that link. Transit links have a travel time for the time

between boarding a transit vehicle at one stop and exiting the vehicle at the following stop. Travel times can either be constant, such as walking times, or dependent on the time of day, such as for driving times.

- **Leg:** A leg describes a traveler's movement through the network. A leg must start and end at an activity location, parking location, or transit stop. There could be only one link or several links within this leg. For example, a trip “*wbwcw*” from office to home as shown in the diagram below is comprised of six legs.
  1. The first leg uses the *walk* mode from the office building to the bus stop. There is only one link within this leg.
  2. The second leg uses the *bus* mode to take the traveler from the origin bus stop to the destination bus stop. There is only one link within this leg.
  3. The third leg uses the *walk* mode from the bus stop to the parking lot. There is only one link within this leg.
  4. The fourth leg uses the *car* mode from the parking lot to the day care. This leg contains three links as shown in diagram below. They are:
    - 1) a link between parking lot and road # 1
    - 2) a link between road # 1 and road # 2, and
    - 3) a link between road # 2 and the day care.
  5. The fifth leg uses the *car* mode from the day care to the home parking lot. There is only one link within this leg, and
  6. The sixth leg uses the *walk* mode from the home parking lot to home.



- **Layer:** A separate layer exists for each mode of travel. For example, a *walk layer* consists of all the streets that can be walked along, and a *transit layer* is comprised of all possible transit modes such as, rail, bus, etc.
- **Stage  $s$ :** Stage  $s$  is an analytical algorithmic description, which designates a step in the procedure where we examine all the nodes that are reachable from the origin node in  $s$  steps.



## 5.3 Key Concepts

### 5.3.1 Shortest path problem

In transportation planning, a traveler is always considered to choose a path that has the minimum travel time from a specified starting location ( $O$ ) to a specified destination location ( $D$ ). The shortest path problem lies at the heart of determining optimal flows in finding the shortest path in a transportation network. Many researchers and practitioners have developed several methods and algorithms for solving this problem. Although the problem itself is quite simple and widely studied, new contributions keep appearing in the scientific literature. TRANSIMS implements Dijkstra's algorithm, one of the most widely used algorithms for solving shortest path problems having nonnegative travel time functions.

### 5.3.2 Dijkstra's algorithm

The network flow literature typically classifies approaches for solving shortest path problems into two groups: label-setting and label-correcting procedures. Both approaches are iterative. The basic label-setting algorithm for nonnegative travel time functions has become known as Dijkstra's algorithm because Dijkstra was among the first of several people to discover it independently.

### 5.3.3 Network layers

The Route Planner conceptually views the network as a set of interconnected, unimodal layers (see Figure-5.3). A separate layer exists for each travel mode (walk, bike, car, bus, rail, trolley, etc.). At certain designated locations in each layer (activity location, parking location, transit stop, etc.), which become nodes in the Route Planner's view of the network, a special link called a *process link* connects one unimodal layer to another. These process links allow intermodal transitions to take place from one layer to another. The layers themselves are constructed from the TRANSIMS network. The travel time for each link in each layer is computed via a link travel time function, which could be time-dependent, or time-independent.

Conceptually, layers are associated with travel modes. There are three major types of layers in the network:

- A walk layer, which consists of all activity locations and all of the streets that can be walked along. However, the parking locations and transit stops that belong to the other two types of layers are only accessible from activity locations in the walk layer via process links.
- A street layer, which consists of all links between intersections. This also includes the parking locations.
- A transit layer, which consists of separate layers for each type of transit vehicle (e.g., a bus layer, a rail layer, etc.). This also includes transit stops and transit routes. Note that each bus route in a bus system is a *separate* layer by itself.

Based on individual traveler preferences and constraints as specified by the activities data file, the Route Planner plans for trips that consist of multiple modal legs (e.g., walk-car-walk, etc.). The process of constructing multiple layers in which each layer can be encoded as a different unimodal

network allows for the efficient computation of trips that are constrained by specified modal sequence requirements.

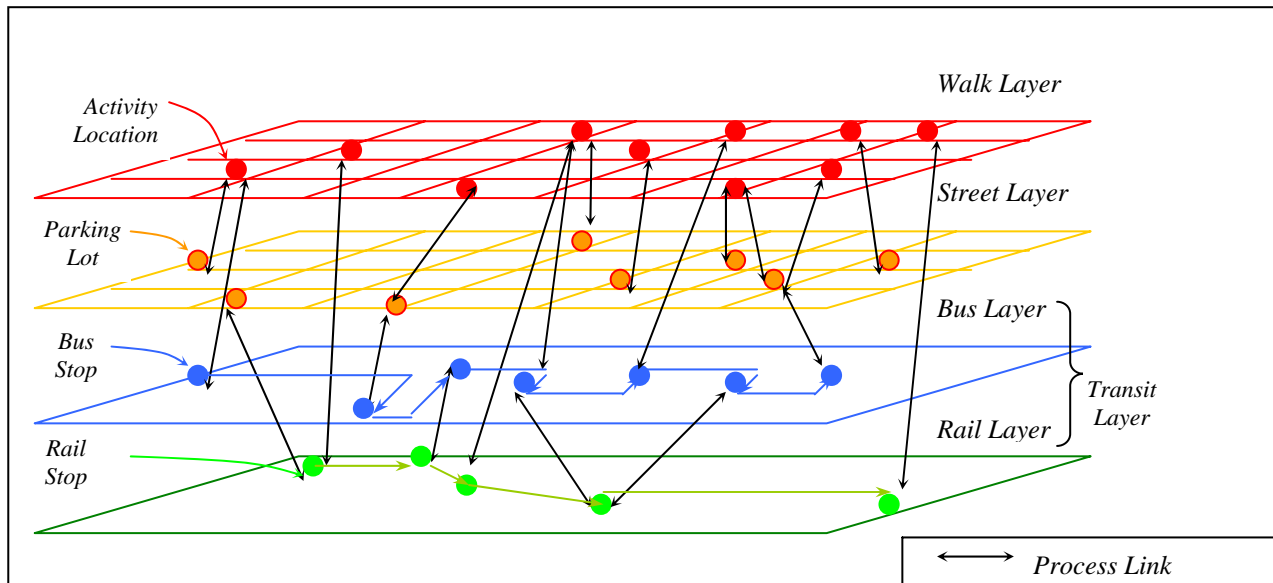


Figure-5.3: A high-level depiction of the various layers used by the Route Planner

## 5.4 Major Data Inputs

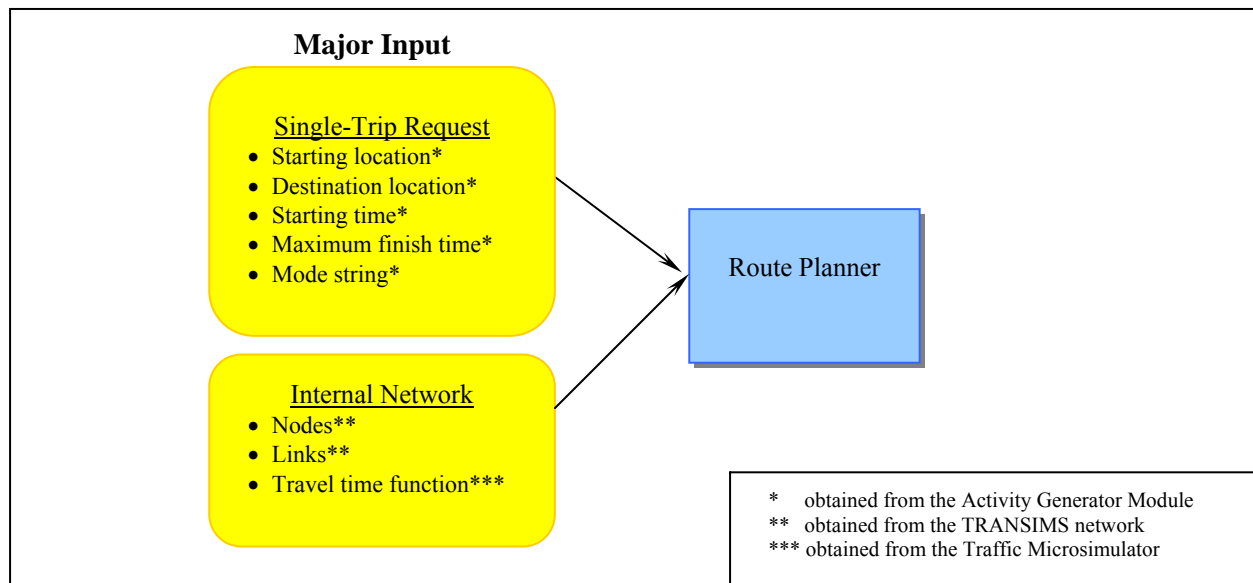


Figure-5.4: The major input specifications for the Route Planner Module

The inputs for the Route Planner Module (as shown in Figure-5.4) are comprised of **single-trip requests**, along with an **Internal Network** consisting of nodes and links, where a node is a physical location in the TRANSIMS network such as an intersection, activity location, or bus stop, and a link is a unidirectional connection between a pair of nodes.

### 5.4.1 Generating Single-Trip Requests

Single-trip requests for each traveler are generated from the TRANSIMS Activities file that is obtained from the Activity Generator Module. The TRANSIMS Activities file (see more details in Section 6.1: Input received from the Activity Generator Module) provides information regarding the non-transportation and transportation activities for each traveler within a household. The Route Planner creates the single-trip requests for each transportation activity, which is interspersed between a pair of non-transportation activities.

Table-5.1 displays an example of the actual data contained in a TRANSIMS Activities file for a household having an ID 13092. This information is obtained from the Activity Generator Module. In this example, there are two people in the household. The first person has an ID 13300, and the second person has an ID 13301. There are three non-transportation activities for each person, which represent the time spent at home and at work. These two individuals have the same type of activities, which relate to going from “home” to “work”, and back to “home” (this is coded in the activity type (**ACTTYP**) column in Table-5.1, where 0 designates home, and 1 designates work). Every activity must be performed because each activity has a 9 priority level. The starting time, ending time, and duration for each activity varies as seen from the table. Although the “home” activities correspond to the same location, which is a house having an ID 845654 (see the **Location** column), the “work” activities are different. The first person goes to work at location ID 833503 via a car (as seen in coded form in the **Mode** column, where 1 designates walk, and 2 designates car), and where the car has an ID 13217 (this is specified in the vehicle ID (**VEHid**) column), while the second person goes to work at location ID 853676 via a car having ID 13218. These two persons travel alone to the “work” activities (this is recorded in the number of other participants in the activity (**N Others**) column).

**Table-5.1: Example of a TRANSIMS Activities file**

HHID	PERID	ACTNO	ACTTYP	PRIORITY	St Time	End T	Dur Tim	Mode	VEHid	N Loc	Location	N Others	OthersID
13092	13300 1 <sup>st</sup> person	1	0 (home)	9	0-0	7.0833- 8.5833	7.0833- 8.5833	1	-1	1	845654 (home)	0	-
13092	13300	2	1 (work)	9	7.5833- 8.0833	14.75- 15.25	6.9167- 7.4167	2	13217	1	833503 (work1)	0	-
13092	13300	3	0 (home)	9	14.75- 16.25	24-24	7.75-9.25	2	13217	1	845654 (home)	0	-
13092	13301 2 <sup>nd</sup> person	1	0 (home)	9	0-0	8.25-9.75	8.25-9.75	1	-1	1	845654 (home)	0	-
13092	13301	2	1 (work)	9	9.1667- 9.6667	16.25- 16.75	6.8333- 7.333	2	13218	1	853676 (work2)	0	-
13092	13301	3	0 (home)	9	16.25- 17.75	24-24	6.25-7.75	2	13218	1	845654 (home)	0	-

From the information given above, Table-5.1 is translated into a format for single-trip requests for each traveler in a household as shown in Table-5.2. The information regarding the single-trip requests as shown in Table-5.2 is used as part of the primary input provided to the Route Planner, aside from the Internal Network itself. Note that the non- transportation activities are also included in Table-5.2 because TRANSIMS continuously records each kind of activity for each member in the household over a period of 24 hours. These outputs for the non-transportation activities are shown as part of the output for the Route Planner. However, the Route Planner runs the shortest path procedure for composing the transportation activities only.

**Table-5.2: Single-trip requests for each traveler corresponding to information from Table-5.1**

Person ID	Activity Number	Starting Location	Destination Location	Starting Time (seconds since midnight)	Maximum Travel Time (second)	Mode String*
13300	1	845654 (Home)	845654 (Home)	0	-	(non-transportation activity)
	2	845654 (Home)	833503 (Work)	30207**	3600***	wcw
	3	833503 (Work)	833503 (Work)	-	-	(non-transportation activity)
	4	833503 (Work)	845654 (Home)	55703	5400	wcw
	5	845654 (Home)	845654 (Home)	-	-	(non-transportation activity)
13301	1	845654 (Home)	845654 (Home)	0	-	(non-transportation activity)
	2	845654 (Home)	853676 (Work)	32636	5100	wcw
	3	853676 (Work)	853676 (Work)	-	-	(non-transportation activity)
	4	853676 (Work)	845654 (Home)	59466	5400	wcw
	5	845654 (Home)	845654 (Home)	-	-	(non-transportation activity)

\* *Mode string* is established by matching mode integer in Table-5. 2 to a mode preference file.

\*\* Randomly selected from 7.0833 hours (25500 seconds) to 8.5833 hours (30900 seconds), which are obtained from End T of ACTNO 1 in Table-5.1.

\*\*\* Obtained from  $(8.0833 - 7.0833) \times 3600 = 3600$  seconds, which is a maximum difference of St Time of ACTNO2 minus the End T of ACTNO1 in Table-5.1.

## 5.4.2 Generating the Internal Network

The Internal Network is comprised of **nodes** and **links**, along with a specification of the **travel time function for each link**. This section describes the two principal parts of this structure, which are:

5.4.2.1 Generating nodes and links in the Internal Network, and

5.4.2.2 Generating travel time functions for each of the links in the Internal Network.

The Route Planner uses information from the TRANSIMS network that records the basic transportation network to create the **nodes and links** for the Internal Network. Table-5.3 displays the required and optional files from the TRANSIMS network that the Route Planner uses to create the nodes and links for the Internal Network.

**Table-5.3: The required and optional files from the TRANSIMS network used by the Route Planner**

<b>File</b>	<b>Description</b>	<b>Key</b>
NET_NODE_TABLE	This file contains information of each node in the TRANSIMS network, which are node ID, the <i>x</i> , <i>y</i> , and <i>z</i> -coordinate of the node.	E
NET_LINK_TABLE	This file contains information of each link, such as, link ID, ID number of the node at the beginning ( <i>A</i> ) and ending ( <i>B</i> ) of the link, number of lanes on the link heading toward to node <i>A</i> and <i>B</i> , etc. Note that there could be only one lane or several lanes within the link.	E
NET_POCKET_LANE_TABLE	A pocket lane is the additional lane that starts and ends <b>within</b> a permanent lane. This file contain information of the pocket lane, such as, ID number of the pocket lane, ID number of the node to which the pocket lane leads, ID number of the link on which the pocket lane lies, etc.	M
NET_PARKING_TABLE	This file contains information of the parking place, which lies on a certain link, such as, ID number of the parking place, ID number of the toward node, which vehicles are traveling to, ID number of the link on which the parking place lies, location of the parking place, etc.	M
NET_LANE_CONNECTIVITY_TABLE	This file contains information of connection on each node, which are ID number of the node, ID number of the incoming and outgoing link, lane number of the incoming and outgoing lane.	M*
NET_SPEED_TABLE	This file contains information of speed limit, free-flow speed, vehicle types to which speed apply, starting, and ending time for the speeds on each link.	O*
NET_LANE_USE_TABLE	This file contains information when a lane has restrictions for certain vehicle types at certain times of the day.	O*
NET_TRANSITS_STOP_TABLE	This file contains information of each transit stop, such as, ID number of the stop, ID number of the node to which vehicles are traveling, ID number of the link on which the stop occurs, location of the stop, type of vehicle for the stop, etc.	O*
NET_TURN_PROHIBITION_TABLE	This file contains information when a particular movement at a node is prohibited at certain times of the day.	O*
NET_BARRIER_TABLE	A barrier is a divider such as a curb or a grade separation that prevents vehicles from moving between two adjacent lanes on a link. This file contains information of each barrier, such as, ID number of the barrier, ID number of the node to which vehicles are traveling, starting position of the barrier, type of the barrier, etc.	O*

Key: E = essential,  
M = needed, but can be generated automatically,  
O = optional, and  
\* = not used by the current TRANSIMS release, but will be used eventually.

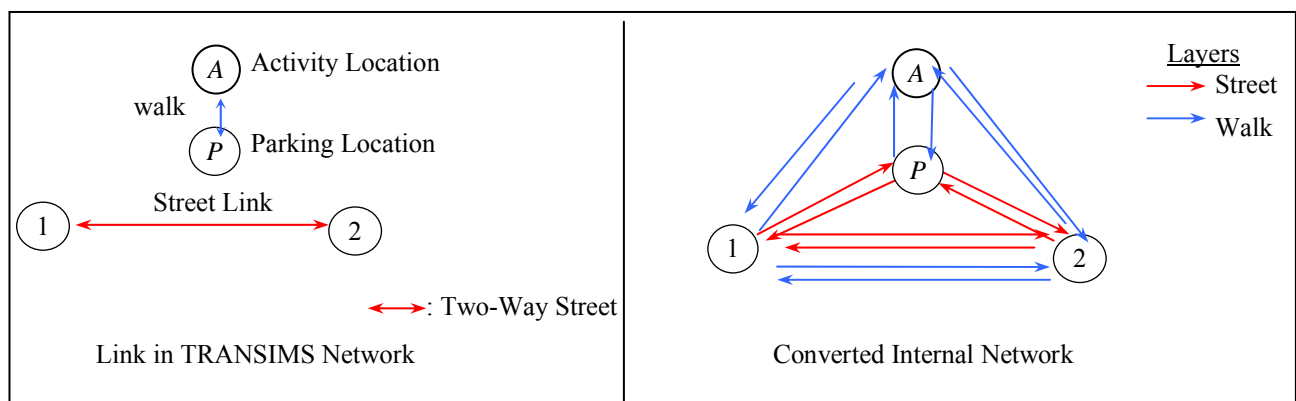
The Route Planner also uses other information from the Traffic Microsimulator Module, such as the travel time or cost on each link of the network, to create the **travel time function** for each link. The reason for creating a separate Internal Network for the Route Planner is to increase the efficiency of the path-finding algorithm.

### 5.4.2.1 Generating nodes and links in the Internal Network

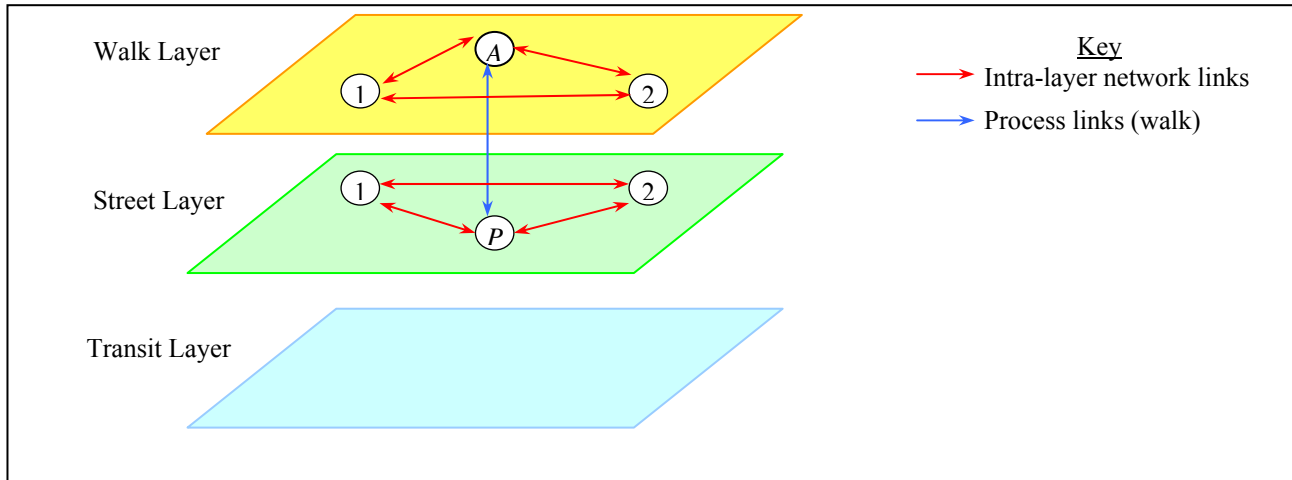
One of the main differences between the TRANSIMS network and the Internal Network is that the links in the Internal Network are all unidirectional, whereas the links in the TRANSIMS network could be bidirectional. Any bidirectional link in the TRANSIMS network is converted to a pair of unidirectional links in the Internal Network, one in each direction.

Each link in the TRANSIMS network can have “accessories” (transit stops, activity locations, parking locations) attached to it. These accessories are represented by new nodes in the Internal Network. Each link containing an accessory is split into two links, one from the start node of the link to the accessory node, and one from the accessory node to the end node of the link. An example that illustrates this transformation is displayed in Figure-5.5. This example does not include a transit layer. The transit layers are described in more detail later on. The left part of Figure-5.5 shows a bidirectional street link in the TRANSIMS network between street nodes 1 and 2 having two accessories, namely, *A*: activity location, and *P*: parking lot for this activity location. Generally, each activity location is attached to a corresponding parking location. The right part of Figure-5.5 shows the constructed Internal Network for this example. Each node must be explicitly connected to appropriate activity locations in the walk-network using process links. Note that there is no need to place a walk-network connection between the street nodes 1 or 2 and a parking node. There are two oppositely-directed unidirectional links on the walk layer because walking can always be performed in both directions.

Figure-5.6 illustrates the layers of the Internal Network corresponding to Figure-5.5. This example has two layers, which are the street layer and the walk layer. **All activity locations are always placed on the walk layer**, while parking locations are placed on the street layer. Conceptually, nodes 1 and 2 appear in two different layers, even though these appearances correspond to the same nodes in TRANSIMS.



**Figure-5.5: A transformation from the TRANSIMS network to the Internal Network**

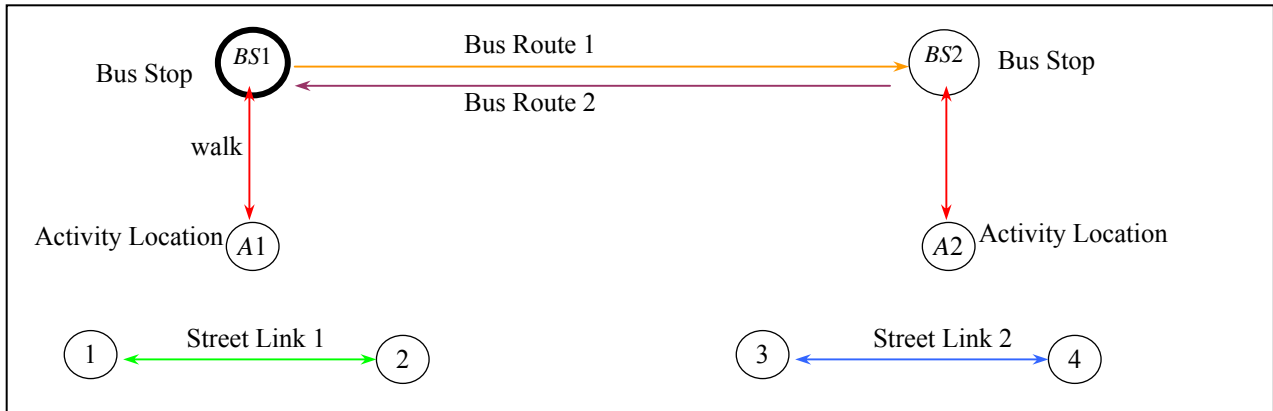


**Figure-5.6: Layers of the Internal Network corresponding to Figure-5.5**

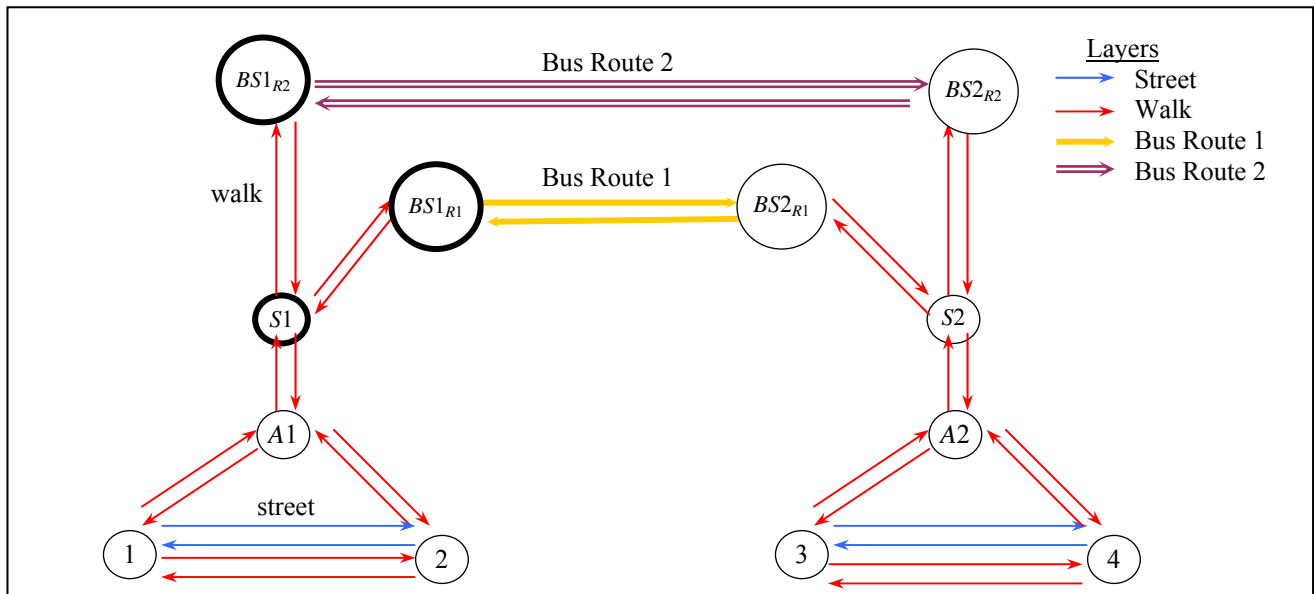
Generally, the Internal Network contains a **transit layer**. This transit layer may be split into many different transportation sub-layers, such as, bus route 1 layer, bus route 2 layer, rail 1 layer, rail 2 layer, etc. The information regarding the transit system comes from the TRANSIMS network's transit stop table, the transit route file, and the transit schedule file. Each transit stop must be explicitly connected to appropriate activity locations in the walk network using process links.

Figure-5.7 depicts the TRANSIMS network representation of two streets with a bus stop and an activity location on each street. There are two bus routes connecting the bus stops. The corresponding Internal Network representation is displayed in Figure-5.8. Note that each bus stop in Figure-5.7 splits into three nodes in Figure-5.8. For example, a bus stop  $BS1$  in Figure-5.7 splits into a node for the bus shelter for passengers ( $S1$ ), a node for the bus-parking place for the bus route 1 ( $BS1_{R1}$ ), and a node for the bus-parking place for the bus route 2 ( $BS1_{R2}$ ). Figure-5.9 illustrates the layers of the Internal Network corresponding to Figure-5.8.

Note that there are five different layers in this Internal Network as shown in Figure-5.9. The street layer containing the intersection nodes, the walk layer containing the activity locations, the bus layer containing the bus stations, and two bus route layers.

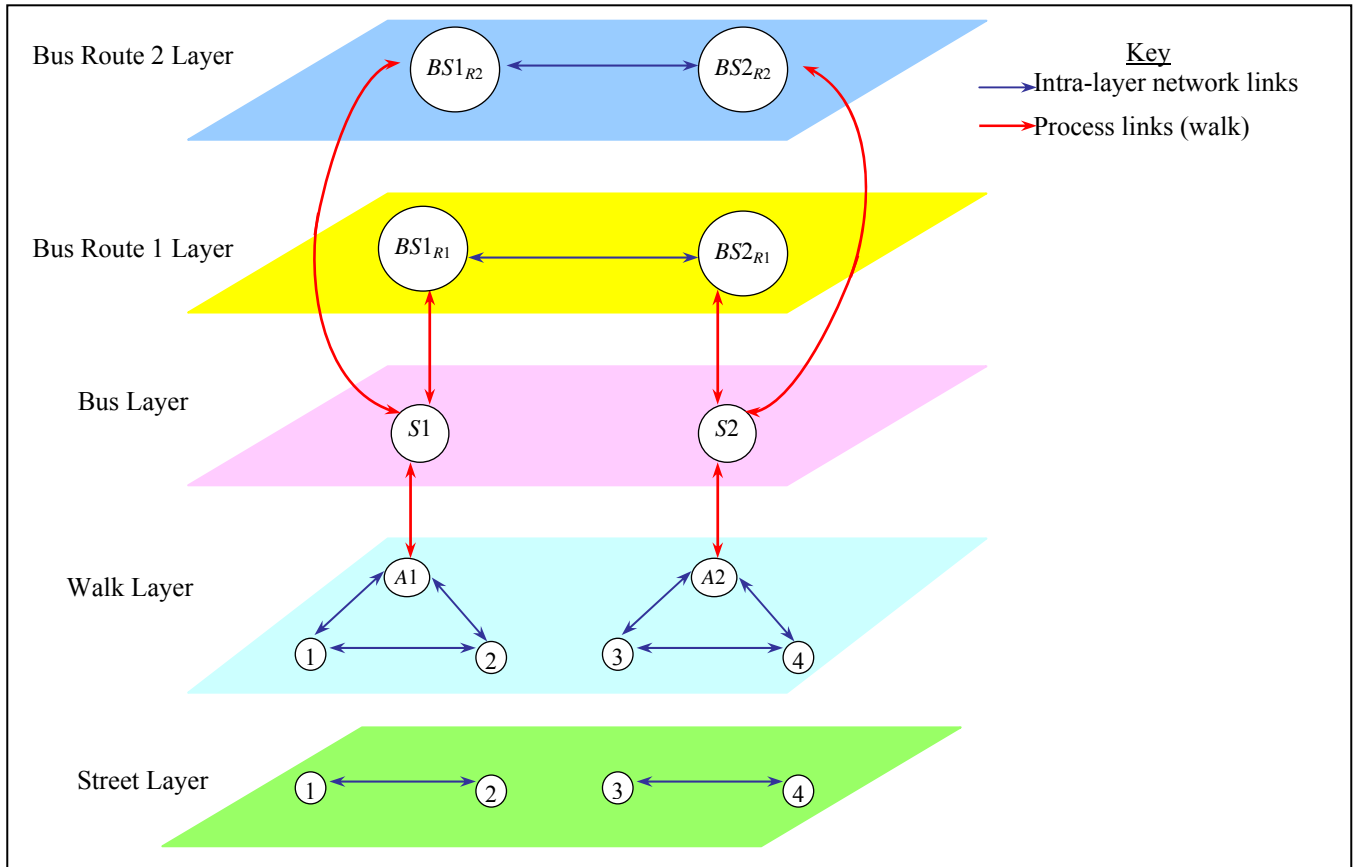


**Figure-5.7: TRANSIMS network representation of two streets with a bus stop and an activity location on each street. There are two bus routes connecting the bus stops**



**Figure-5.8: The Internal Network representation corresponding to Figure-5.7**





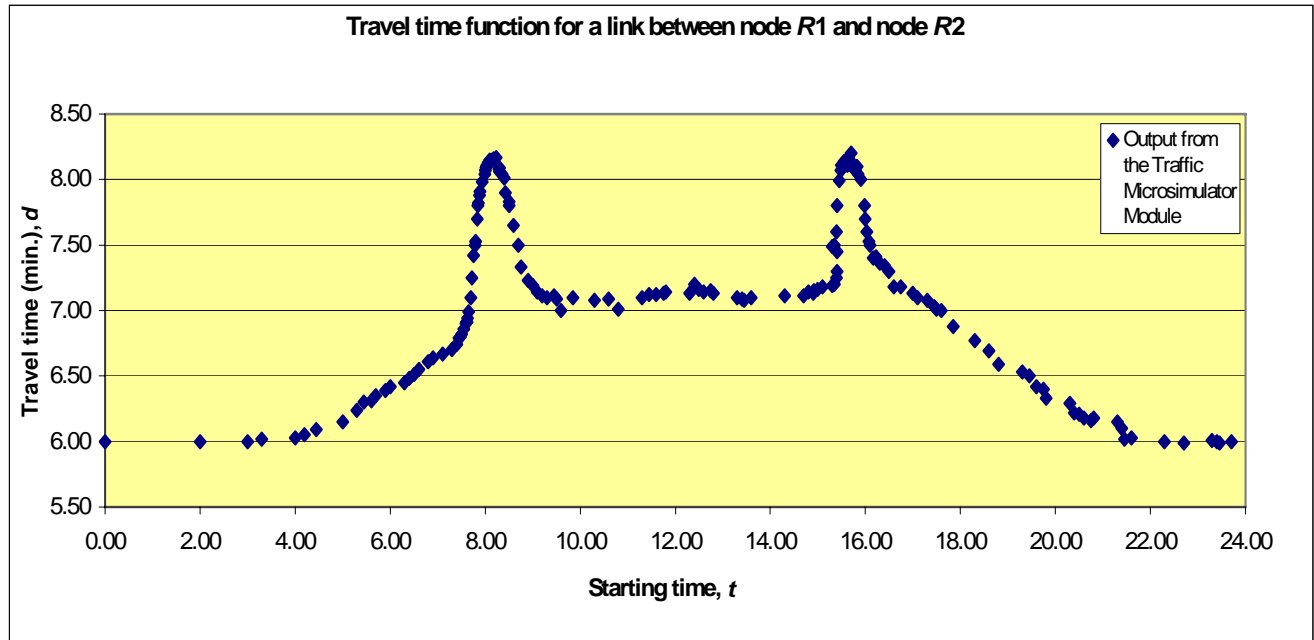
**Figure-5.9: Layers of the Internal Network corresponding to Figure-5.8**

### **5.4.2.2 Generating travel time functions for each link in the Internal Network**

Each link in the Internal Network has a travel time associated with it. Links on the street layer have a travel time for driving on each link. Links on the walk layer have a travel time for walking on each link. Transit links have a travel time equal to the time elapsed between boarding a transit vehicle at one stop and exiting the vehicle at the following stop. Travel times can either be constant, such as walking times, or dependent on the time of day, such as driving times.

Initially, there is no feedback input from the Traffic Microsimulator Module. The free-flow speed and the length of each link are therefore used to generate the travel times on the links by “dividing the link length by the admissible speed”. The Route Planner then finds a shortest path for the various single-trip requests using the given Internal Network representation. Once the plans are generated for all the travelers, they are simultaneously fed into the Traffic Microsimulator Module to initiate the first simulation run.

The output from the simulation runs in the Traffic Microsimulator Module provides more accurate information regarding the travel time functions on the links in the Internal Planner Network on a time-dependent basis. These outputs are plotted on graphs that show the relationship between a starting time and the travel time for each link, which is a time-dependent function. Figure-5.10 illustrates an example of a travel time function over a 24-hour interval for a link defined by a pair of specific nodes (node *R1* and node *R2*), assuming that this link is on a street layer (travel via a car).



**Figure-5.10: An example of a travel time function over a 24-hour interval for a specific link defined by a pair of specific nodes (node *R1* and node *R2*)**

After obtaining the output from the simulation runs in the Traffic Microsimulator Module, an estimate for the travel time function for each link is determined. This travel time function represents the average travel time experienced by the vehicles that traverse the link, averaged over a 15-minute interval. The average travel time is obtained via a linear approximation through a Linear Regression Analysis for the outputs over a 15-minute interval.

Linear Regression Analysis is a statistical method of fitting a line through data to minimize error. With linear regression analysis, model coefficients can be determined to estimate the travel time function for each link in the Internal Network.

In linear regression analysis involving one independent variable  $t$  (the starting time on that link), and one dependent variable  $d$  (the travel time), the relationship that is used to fit  $n$  data points ( $1 \leq i \leq n$ ) is of the following form, when  $a$  and  $b$  are model parameters that need to be estimated:

$$d = a + bt. \quad (1)$$

Mathematical formulas for estimating  $a$  and  $b$  in Equation (1) using a simple linear regression are as follows:

$$b = \frac{\sum_{i=1}^n t_i d_i - \bar{t} \sum_{i=1}^n d_i}{\sum_{i=1}^n t_i^2 - \bar{t} \sum_{i=1}^n t_i} \quad (2)$$

$$a = \bar{d} - b\bar{t} \quad (3)$$

where,

$$\bar{t} = \frac{\sum_{i=1}^n t_i}{n} \quad (4)$$

and

$$\bar{d} = \frac{\sum_{i=1}^n d_i}{n} \quad (5)$$

As an example, consider a portion of Figure-5.10 during an interval of 8.00 a.m. to 8.15 a.m. The corresponding graph is highlighted in Figure-5.11, and the data is shown in Table-5.4.

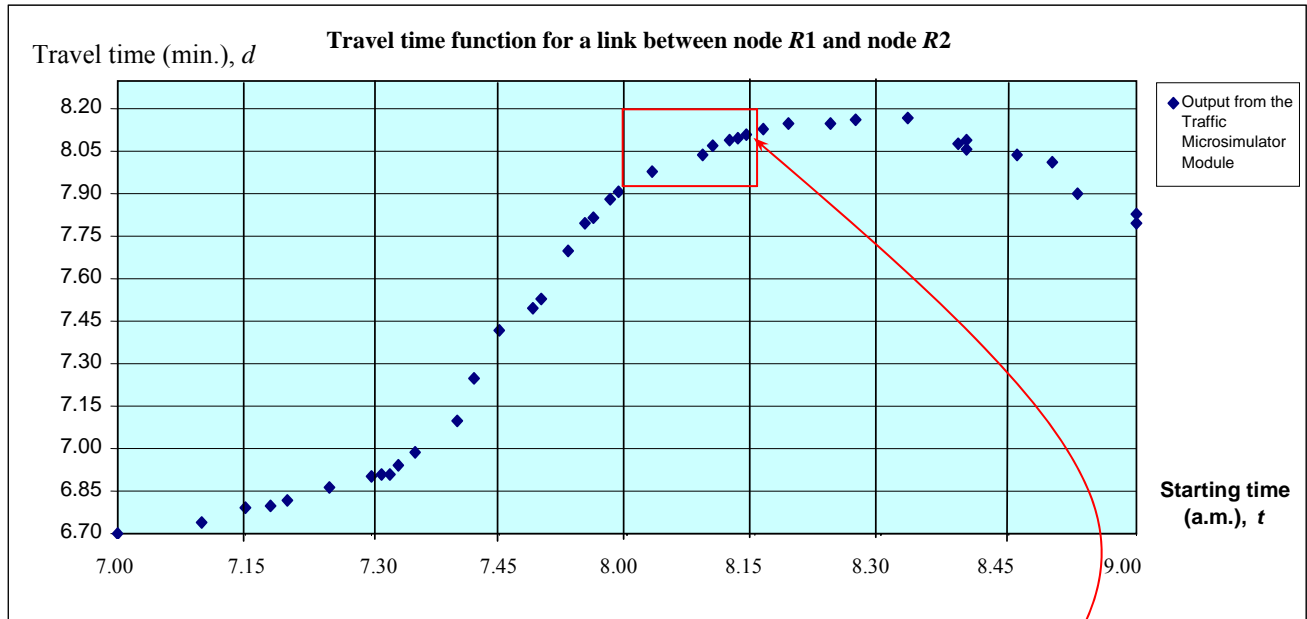


Figure-5.11: Travel time function over 7.00-9.00 a.m. interval corresponding to Figure-5.10

Table-5.4: Data of a travel time function over the interval 8.00 a.m.-8.15 a.m

Data point, $i$	Starting time (a.m.)	Converted starting time into a decimal, $t_i$	Travel time (minute), $d_i$	$t_i d_i$	$t_i^2$
1	8.00	8.00	7.91	63.28	64.00
2	8.04	8.07	7.89	64.37	65.07
3	8.09	8.15	8.04	65.53	66.42
4	8.12	8.20	8.07	66.17	67.24
5	8.13	8.22	8.09	66.47	67.51
6	8.14	8.23	8.10	66.69	67.79
		$\sum_{i=1}^6 t_i = 48.87$	$\sum_{i=1}^6 d_i = 48.19$	$\sum_{i=1}^6 t_i d_i = 392.54$	$\sum_{i=1}^6 t_i^2 = 398.09$

From the data given data in Table-5.4, we compute a linear approximation by using Equations (1) to (5) as follows.

$$\bar{t} = \frac{48.87}{6} = 8.145$$

$$\bar{d} = \frac{48.19}{6} = 8.032$$

$$\begin{aligned}
 b &= \frac{\sum_{i=1}^n t_i d_i - \bar{t} \sum_{i=1}^n d_i}{\sum_{i=1}^n t_i^2 - \bar{t} \sum_{i=1}^n t_i} \\
 &= \frac{392.54 - (8.145 \times 48.19)}{398.09 - (8.145 \times 48.87)} = \frac{0.03245}{0.04385} \\
 &= \mathbf{0.74}
 \end{aligned}$$

$$\begin{aligned}
 a &= \bar{d} - b\bar{t} \\
 &= 8.032 - (0.74) \times (8.145) \\
 &= \mathbf{2.}
 \end{aligned}$$

Thus, using  $d = a + bt$ , we get

$$d = 2 + 0.74t.$$

A plot of the travel time function,  $d = 2 + 0.74t$ , is shown in Figure-5.12.

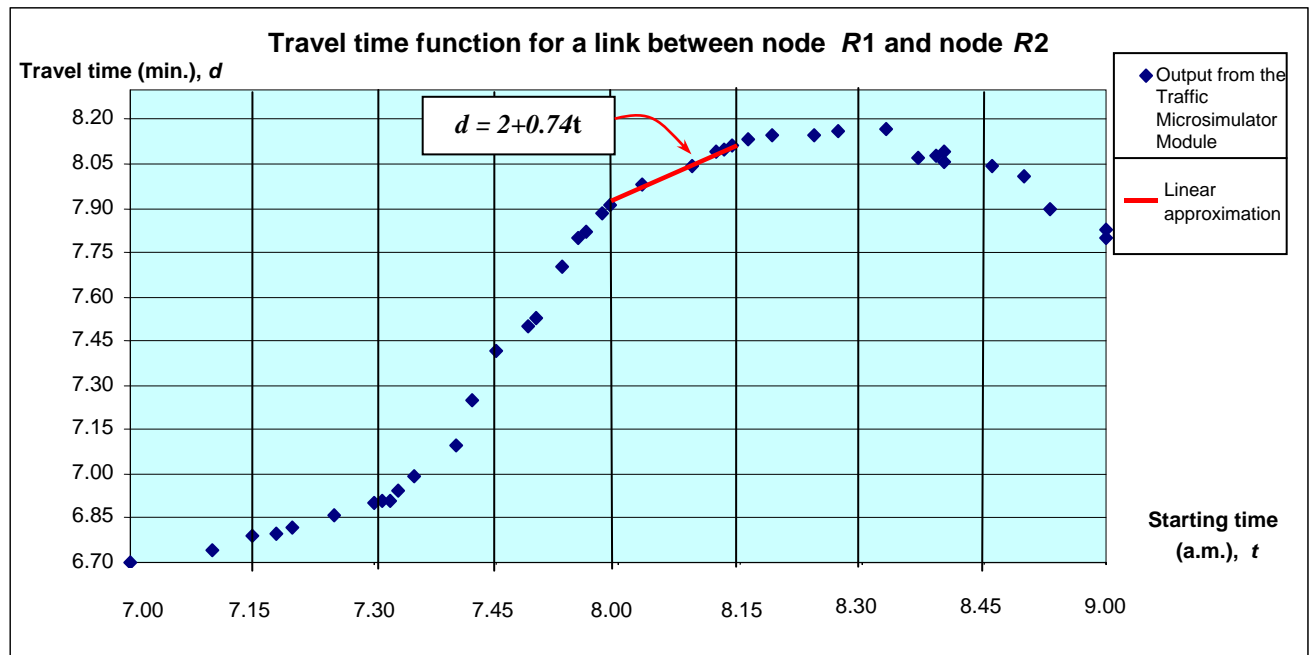
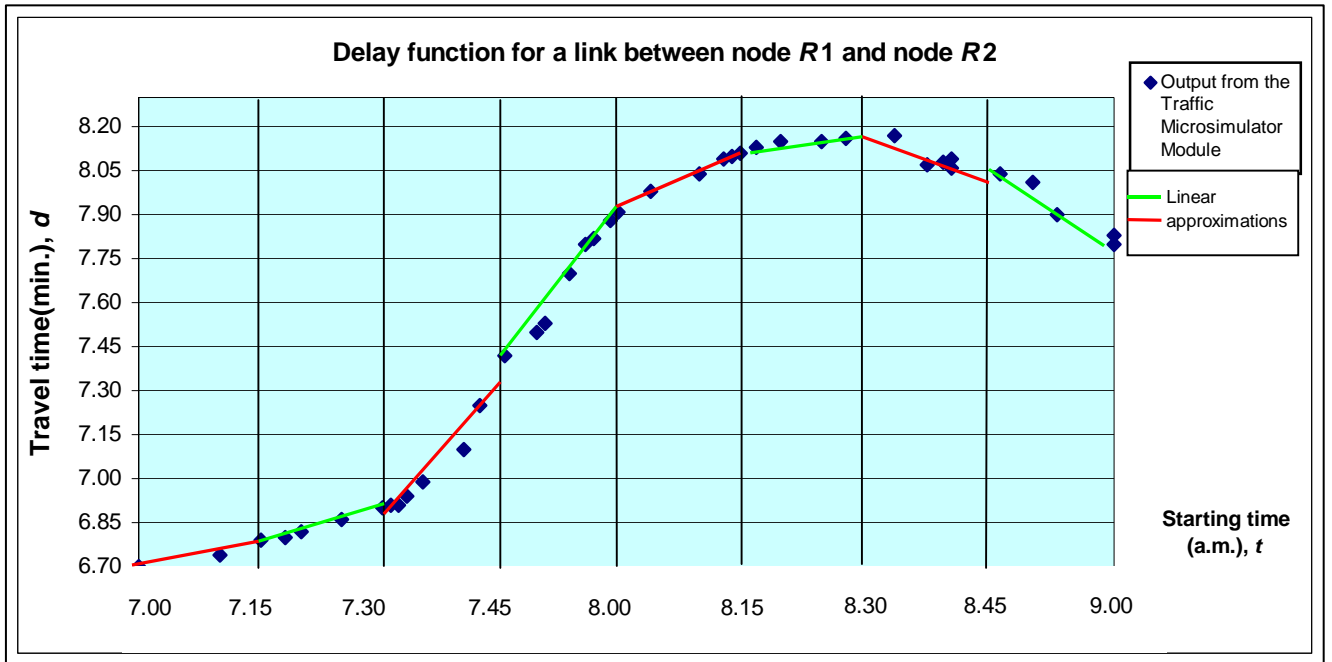


Figure-5.12: A plot of the approximation to the travel time function over the interval 8.00 a.m. – 8.15 a.m.

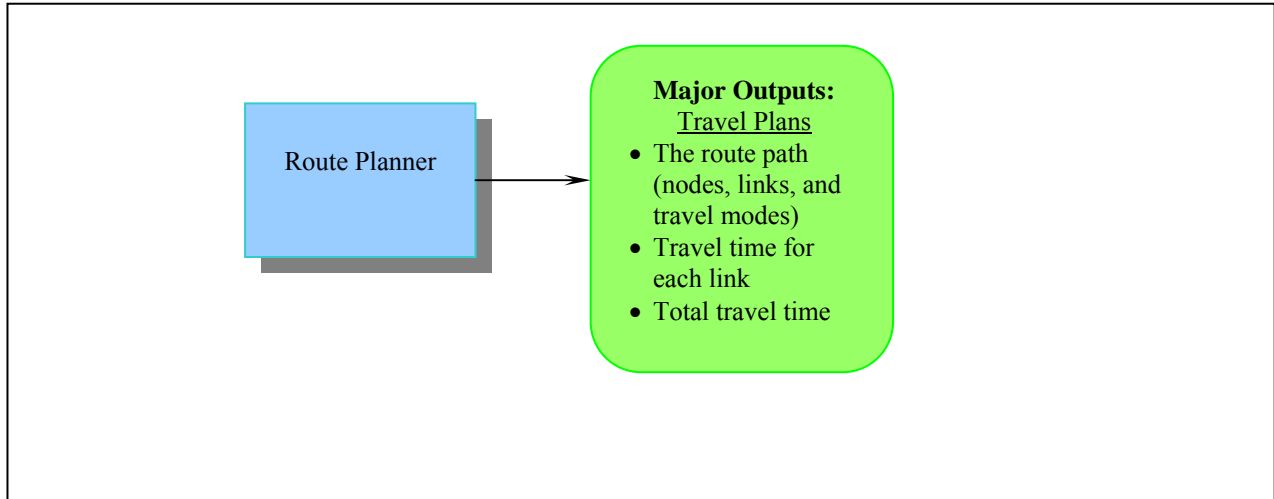
If we recursively use linear regression analysis for all the other 15 minute-intervals, we obtain a piecewise linear approximation for the travel time function over the interval 7.00 a.m. – 9.00 a.m. (as shown in Figure-5.13).



**Figure-5.13: A plot of the approximation to the travel time function over the interval 7.00 a.m. – 9.00 a.m.**

Moreover, to enhance the effectiveness of the Microsimulator-Route Planner feedback, noise can be added to the link travel times. The maximum amount of noise that can be added to the link as a percentage of the link travel time can be specified. If the travel time for a link is  $d_1$  and the specified noise percentage value is  $k$ , the reported travel time will be specified to fall in the interval  $(d_1 - kd_1, d_1 + kd_1)$ .

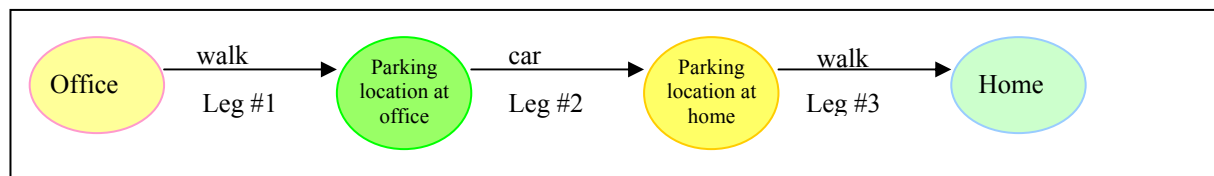
## 5.5 Major Data Outputs



**Figure-5.14: The major outputs from the Route Planner Module**

The major output of the Route Planner is the information about transportation activities for each traveler, which comprises the route path (nodes, links, and travel modes), the travel time for each link, and the total travel time (as shown in Figure-5.14). In addition, TRANSIMS displays the non-transportation activities in the output as well, in order to maintain a record for every activity for each member over the 24-hour horizon.

Generally, the outputs are formatted to show the relevant information for **each leg** on a trip. A leg must start and end at an activity location, a parking location, or a transit stop (notice that there is a special leg for the non-transportation activity that starts and ends at the same location). Figure-5.15 provides a simple illustration of the output for a traveler who has executed a trip from the “office” to “home”. The Route Planner finds a shortest path that is comprised of three legs. Each leg (as shown in Figure-5.15) has specific information attached to it.



**Figure-5.15: Sample route output from the Route Planner Module**

The general format of this information for **each leg** is comprised of the following:

(Traveler ID) (Special attribute) (Trip ID) (Leg ID)  
 (Starting time) (Starting location ID) (Starting accessory type) (Ending location ID) (Ending accessory type)  
 (Duration) (Stop time) (Monetary cost) (Generalized cost function) (Max time flag)  
 (Driver flag) (Travel mode)  
 (Number of tokens)  
 (Vehicle ID) (Number of passengers)  
 (Token ID)

Table-5.5 provides a more detailed description of this format.

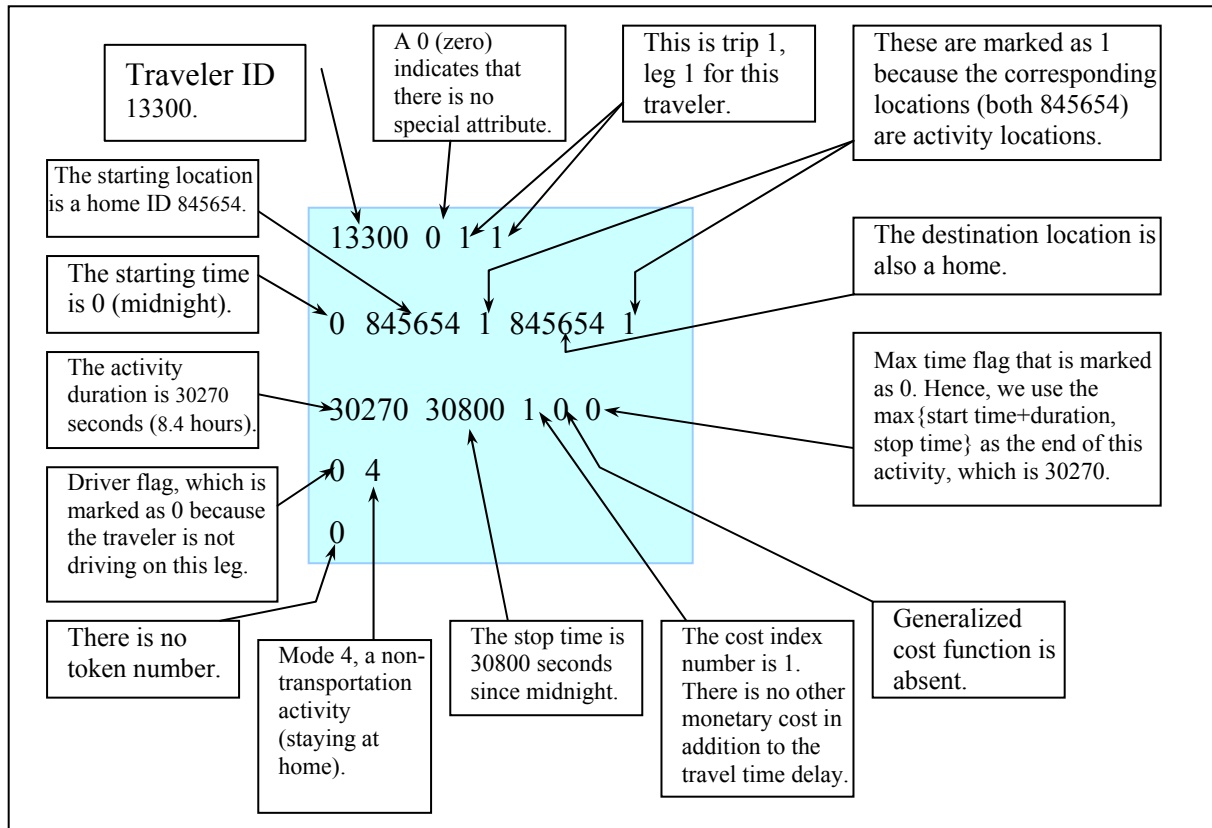
**Table-5.5: Description of the format output for the Route Planner Module**

<b>Name</b>	<b>Description</b>
Traveler ID	Each person is given a unique ID in the population
Special attribute	Available to the user to set as desired. If there is no special attribute, set as 0 (zero).
Trip ID	Number of trips for the traveler: sequentially from 1.
Leg ID	Number of legs within a trip: sequentially from 1.
Starting time	The earliest time the simulation needs to examine this leg. This is generally the starting time (estimated by the planner) for the leg. For a transit leg, however, this represents the arrival time of the passenger at the transit stop, rather than the arrival time of the transit vehicle.
Starting location ID	Denotes the network accessory ID of the starting location for this leg.
Starting accessory type	Denotes the type of accessory of the corresponding location. This is necessary because the IDs are not globally unique over the accessories. This entity should be one of: 1) activity location 2) parking 3) transit stop
Ending location ID	Denotes the network accessory ID of the ending location for this leg.
Ending accessory type	Denotes the type of accessory of the corresponding location. This is necessary because the IDs are not globally unique over the accessories. This entity should be one of: 1) activity location 2) parking 3) transit stop
Duration	In conjunction with Stop time and Max time flag, this specifies how long this leg is expected to take. This value is computed as seconds since midnight.
Stop time	In conjunction with Stop time and Max time flag, this specifies an absolute ending time for this leg. This value is computed as seconds since midnight.
Monetary cost	In addition to travel time delay, process links can also have an associated monetary cost. This can be used to account for parking fees and transit fares. Additionally, transit routes may have zone to zone fares associated with them. Currently, the Route Planner does not use monetary costs (marked as index number 1).
Generalized cost function (GCF)	To more accurately model mode choice, the concept of a GCF has been developed. The GCF allows other factors, besides travel time and monetary cost, to be taken into account when determining a plan for a traveler. These other factors are included in the “cost” of a trip. The importance of the monetary cost of a trip may be modified depending on a traveler’s income. A greater penalty for traveling on congested links can be imposed by calculating the difference between the actual travel time and the free speed delay. Note that transit transfers may impose a higher cost than the actual delay involved. The GCF currently used is simply the travel time (marked as 0).



Name	Description
Max time flag	If true (marked as 1), the end of this activity is best estimated as $\min\{\text{start time} + \text{duration}, \text{stop time}\}$ . Otherwise, the maximum operator is used instead.
Driver flag	This is true (marked as 1) if the traveler is driving a vehicle on this leg, zero otherwise.
Travel mode	This must be one of: 0 = car 1 = transit 2 = pedestrian 3 = bicycle 4 = non-transportation activity
Number of tokens	This is the number of routes within the leg if the travel occurs via the transit mode, or this equals the number of visiting street nodes within the leg if the travel is by car.
Vehicle ID	This indicates the vehicle ID if the travel is by car.
Number of passengers	This is the number of other household members who might participate and use the same transportation mode or activity (e.g., the same car).
Token ID	This ID should be present only when the number of tokens is positive. It indicates the route ID if the travel occurs by transit, or the street node ID if the travel is by car.

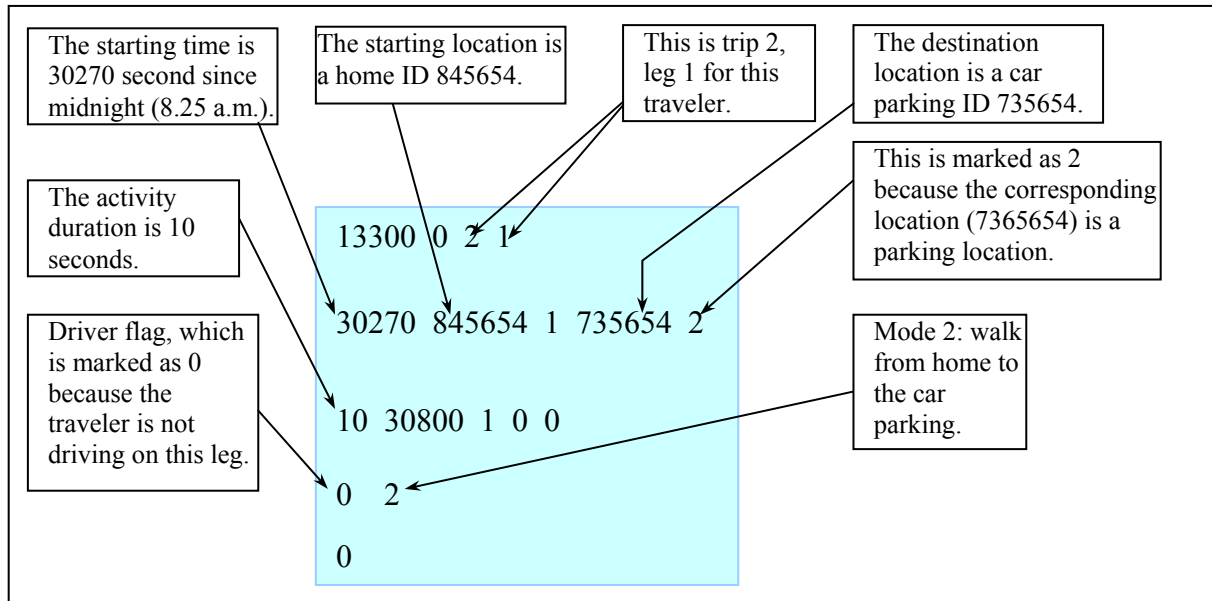
Figures 5.16 to 5.19 illustrate and explain various output examples from the Route Planner for the household activities shown in Table-5.2 for traveler ID 13300.



**Figure-5.16: Example of an output for a non-transportation activity; “staying at home”**

As mentioned above, the Route Planner also displays information in the output for non-transportation activities in order to maintain a record for every activity for each member over 24-hour horizon. Figure-5.16 illustrates the output for a non-transportation activity at home.

Figure-5.17 illustrates the output for a transportation activity; a walk trip from home to parking.



**Figure-5.17: Example of an output for the "walk" mode (walk from home to the parking location)**

Figure-5.18 illustrates the output for a transportation activity; a trip from parking at home to parking at the office. This trip uses a car as a vehicle, and is comprised of travel on five different streets (the number of tokens is 5).



## 5.6 Module Interfaces

An outline for the module interfaces involving the Route Planner Module is displayed in Figure-5.20.

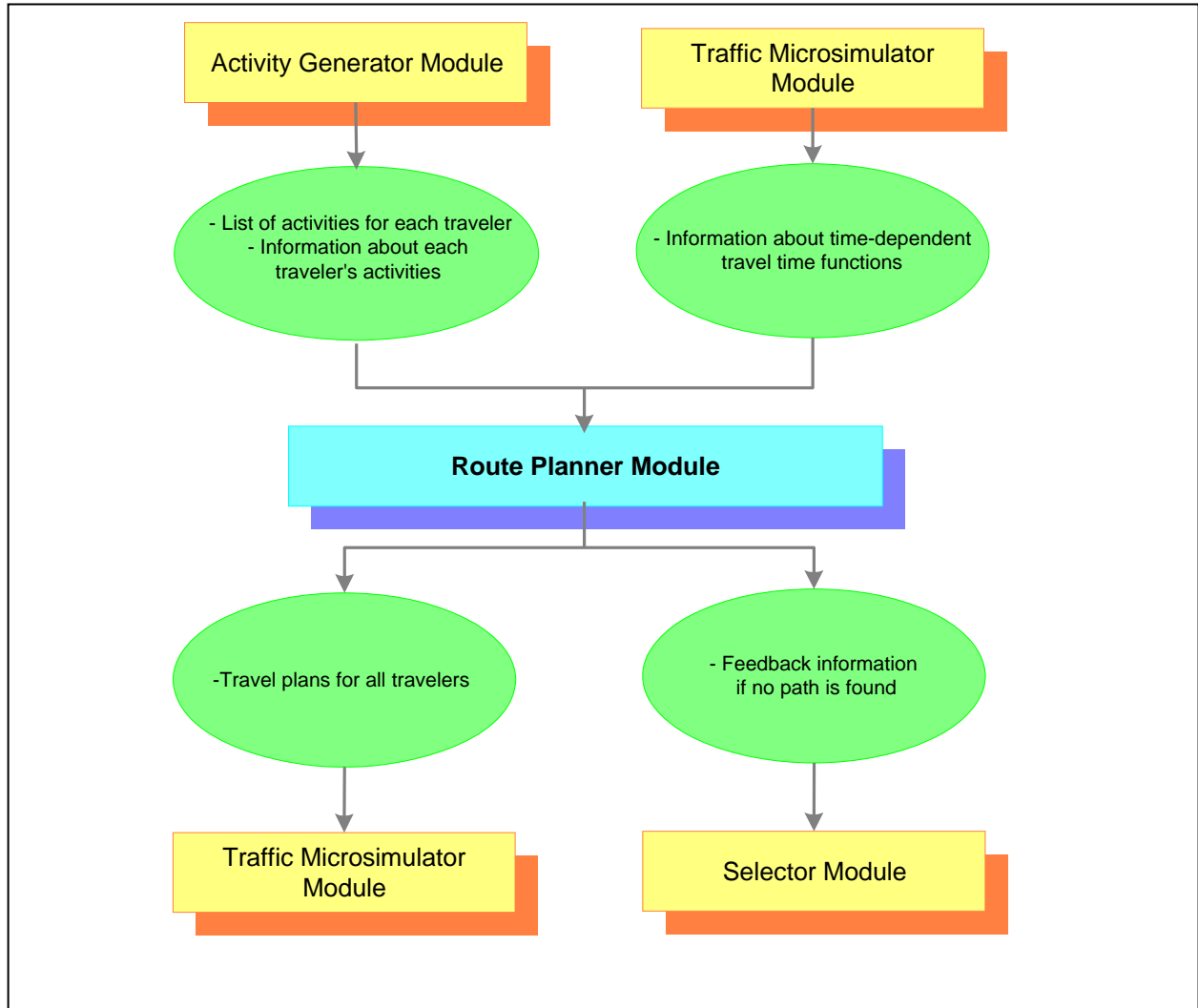


Figure-5.20: Flowchart of the module interfaces for the Route Planner Module

### 5.6.1 Inputs received from the Activity Generator Module

The input derived from the Activity Generator Module is the list of activities for each member of a synthetic household, along with information about each traveler's activity. This information is used to generate the trip request for each traveler. Chapter 4 gives the detailed information about this file.

## 5.6.2 Inputs received from the Traffic Microsimulator Module

The input from the Traffic Microsimulator Module comprises the information about time-dependent travel time functions for all the links in the Internal Network. Table-5.6 lists all files received from the Traffic Microsimulator Module that contain information about travel time functions.

**Table-5.6: The list of files received from the Traffic Microsimulator Module that contains information about travel time functions**

File	Description
ROUTER_LINK_DELAY_FILE	Feedback file from which to read link delays/travel times. If the key is not present or the file does not exist, the free speed delays are used.
ROUTER_WALKING_SPEED	Speed to use when computing travel times for walk links (meters/second). <b>Default 1.0.</b>
ROUTER_BIKING_SPEED	Speed to use when computing travel times for walk links traversed by bicycle (meters/second). <b>Default 4.0.</b>
ROUTER_GET_ON_TRANSIT_DELAY	Delay/travel time encountered when boarding a transit vehicle. <b>Default 3 seconds.</b>
ROUTER_GET_OFF_TRANSIT_DELAY	Delay/travel time encountered when exiting a transit vehicle. <b>Default 4 seconds.</b>

## 5.6.3 Outputs sent to the Traffic Microsimulator Module

Once the shortest travel plans are generated for all the travelers, they are simultaneously fed into the Traffic Microsimulator Module as shown in Table-5.6. The following tables, Tables 5.7 to 5.11, list the additional files sent to the Traffic Microsimulator Module to differentiate the traveler plans by mode and whether the traveler is a car driver or a car passenger.

**Table-5.7: The list of files for a car driver**

File	Description
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.
Number of Passengers	The number of passengers, not including the driver, on this leg.
List of Node IDs	The nodes (in order) through which the driver's route will pass.
List of Passenger IDs	The traveler ID of each passenger to be carried on this leg.

**Table-5.8: The list of files for a car passenger**

File	Description
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.

**Table-5.9: The list of files for a transit driver**

File	Description
Schedule Pairs	Number of (stop ID, depart time) pairs.
Vehicle ID	Each vehicle (with its ID) available in the simulation is listed in the vehicle database.
Route ID	Route IDs are specified in the transit route file. Only one route ID is allowed per leg.
List of Node IDs	The nodes (in order) through which the driver's route will pass.
List of Schedule Pairs	Each pair consists of a stop ID and a depart time. When a transit driver arrives at a transit stop whose ID is given in this list, the driver will remain at that stop until the departure time.

**Table-5.10: The list of files for a transit passenger**

File	Description
Route ID	Traveler will board any transit vehicle whose driver's plan matches this Route ID.

**Table-5.11: The list of files for a pedestrian**

File	Description
List of Node IDs	The nodes (in order) through which the traveler's route will pass.

For non-transportation activities, there are no files sent to the Traffic Microsimulator Module.

#### **5.6.4 Feedback to the Selector Module**

There are currently four types of anomalous activities recognized by the Route Planner: *No Path*, *Invalid Time*, *Invalid Shared Ride*, and *Invalid Shared Ride Time*. Each activity for which an anomaly is detected is fed back to the Selector Module to request new activity characteristics. The four anomalous activities are described below.

1. *No Path*: A *No Path* anomaly takes place when a trip request cannot be satisfied because a path from the source location to the destination location could not be found which obeys the time and travel-mode constraints. Common reasons for this anomaly include no connectivity between the source location and the destination location, and no transit vehicles running after the start time. The *No Path* anomaly includes information about the source and destination locations, the travel mode, and the start time of the transportation leg. When a *No Path* anomaly is detected, no plan is generated, and the remaining activities for this traveler are skipped. Table-5.13 provides the *No Path* files sent to the Selector Module.
2. *Invalid Time*: An *Invalid Time* anomaly occurs when the actual time used by the Route Planner does not fit within the bounds specified by the activity. The start time, end time, and

duration are checked for consistency with respect to the ranges specified for the activity. The *Invalid Time* anomaly includes information about the type of inconsistency, the lower and upper bounds from the activity file, and the actual value used by the Route Planner. When an *Invalid Time* anomaly is detected, a plan is generated for the anomalous activity using the inconsistent times. Table-5.14 provides the *Invalid Time* files sent to the Selector Module.

3. *Invalid Shared Ride*: An *Invalid Shared Ride* anomaly occurs when the driver's activities and passenger's activities do not match up. Currently, only the condition where there are too few driver's activities for the number of passenger's activities is detected. When this anomaly is detected, no plan is generated for the passenger and the rest of the passenger's activities are not planned. The driver's activities are planned as usual. No extra files are output for this anomaly.
4. *Invalid Shared Ride Time*: An *Invalid Shared Ride Time* anomaly takes place when the transportation leg for a passenger shared ride takes longer than the time between the two adjacent activity legs. If the trip extends past the upper bound of the following activity's start time, but not past the following activity's end time, an *Invalid Shared Ride Time* entry is created in the anomalous activity file and the rest of the passengers' trip requests are planned as usual. If the trip extends past the end time of the following activity, an *Invalid Shared Ride Time* entry is created in the anomalous activity file and no further trips are planned for this traveler. The *Invalid Shared Ride Time* anomaly contains the arrival time of the passenger-shared ride-trip, the upper bound of the start time of the following activity, and the end time of the following activity. Table-5.15 provides the *Invalid Shared Ride Time* files sent to the Selector Module.

Table-5.12 provides the common files used for each type of anomaly.

**Table-5.12: Anomalous Activity File (Common Files)**

<b>File</b>	<b>Description</b>
HouseholdID	ID of the anomalous household.
TravelerID	ID of the anomalous traveler.
ActivityID	ID of the anomalous activity.
TripID	ID of the trip generated by this activity.
LegID	ID of the first leg generated by this activity.
ProblemType	Type of anomaly: 1- <i>No Path</i> , 2- <i>Invalid Time</i> , 3- <i>Invalid Shared Ride</i> , and 4- <i>Invalid Shared Ride Time</i> .
Number of data files	Number of remaining files; varies by anomaly type.

**Table-5.13: No Path Files**

<b>File</b>	<b>Description</b>
SourceLocation	Source Location ID of the anomaly trip.
SourceType	Source Location type of the anomaly trip. This entity should be one of: 1-activity location 2-parking 3-transit stop
DestinationLocation	Destination Location ID of the anomaly trip.
DestinationType	Destination Location type of the anomaly trip. This entity should be one of: 1-activity location 2-parking 3-transit stop
Mode	Travel mode of the anomaly trip.
StartTime	Time the anomaly trip should start.

**Table-5.14: Invalid Time Files**

<b>File</b>	<b>Description</b>
TimeType	Type of the anomalous time: 0-Start, 1-End, and 2-Duration.
LowerBound	Distribution of the lower bound.
UpperBound	Distribution of the upper bound.
Actual	The actual value used by the Route Planner.

**Table-5.15: Invalid Shared Ride Time Files**

<b>File</b>	<b>Description</b>
Arrival Time	Arrival time of the passenger-shared ride-trip.
Start Time Bound	Upper bound of the starting time of the activity leg following the passenger-shared ride-trip.
Stop Time	The stop time of the activity leg following the passenger-shared ride-trip.

The following is the example of the anomalous activities recognized by the Route Planner and fed back to the Selector Module. The example has five anomalous activities as shown in the first five lines of the example. Each line is comprised of the code numbers denoting the characteristics of the activity and its anomaly. These code numbers have the format and description as shown in Figure-5.21a.

The simple descriptions of these anomalies, referred to as #WARNING..., are shown below in figure-5.21. These correspond to the errors from the first five lines of the anomalies. They show that the first four anomalies are the *Invalid Time* anomaly and the last one is the *No Path* anomaly.



1	101	2	2	1	2	4	0	12253	12853	12249
2	102	2	2	1	2	4	0	12176	12777	12145
2	102	3	4	1	2	4	0	23112	23711	23748
2	102	3	4	1	2	4	2	62688	63287	62652
8	108	3	-1	-1	1	6	1048	1	1053	1 wtw 46724

#WARNING (1) in [ROUTER]: Traveler 101's activity choices infeasible: Start of activity at time 12249 but Start Time bound is (12253, 12853)

#WARNING (1) in [ROUTER]: Traveler 102's activity choices infeasible: Start of activity at time 12145 but Start Time bound is (12176, 12777)

#WARNING (1) in [ROUTER]: Traveler 102's activity choices infeasible: Start of activity at time 23748 but Start Time bound is (23112, 23711)

#WARNING (1) in [ROUTER]: Traveler 102's activity choices infeasible: Duration of activity at time 62652 but Start Time bound is (62688, 63287)

#WARNING (1) in [ROUTER]: No path found for activity 8 108 3 between 1048 (ActLoc) and 1053 (ActLoc), mode wtw departing 46724

**Figure-5.21: Anomalous activities recognized by the Route Planner and fed back to the Selector Module**

The following is a description of the code numbers used in the example.

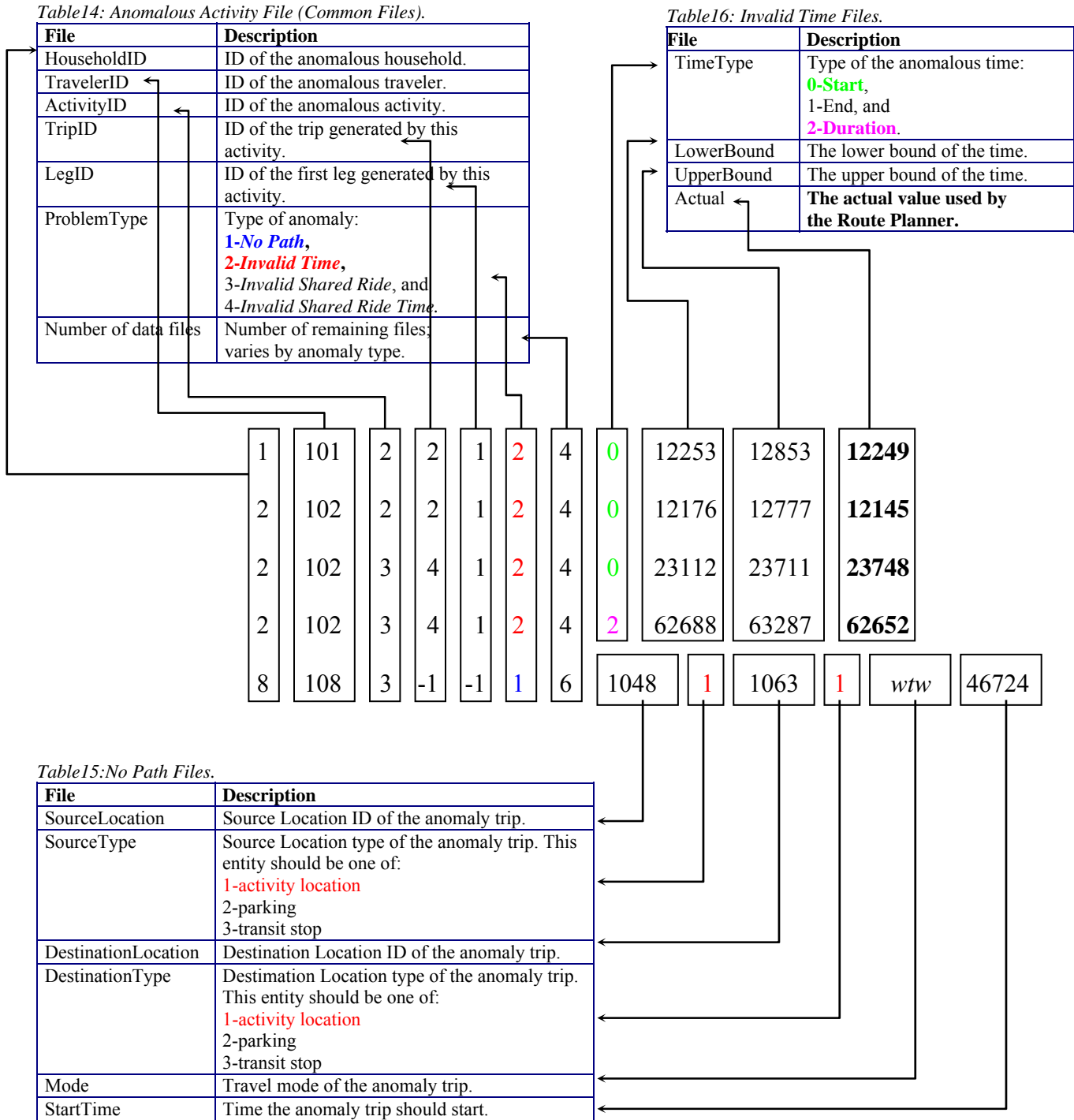


Figure-5.21a: A description of the example for the Route Planner Anomaly outputs

## 5.7. Configuration Files

Table-5.16 lists all files used to run the Route Planner and their descriptions, including the default values. Note that this table does not include files from the TRANSIMS network used to generate the Internal Network. Please see Section 4.2 (*Generating the Internal Network*) for more information on the files from the TRANSIMS network.

**Table-5.16: The list of all the files used by the Route Planner**

<b>Configuration Key</b>	<b>Description</b>
ACTIVITY_FILE	Path to a TRANSIMS activity file. <b>Required.</b>
VEHICLE_FILE	Path to a TRANSIMS vehicle file. <b>Required.</b>
MODE_MAP_FILE	Path to a mode file. <b>Required.</b>
ROUTER_PROBLEM_FILE	Path name to a file in which activities with anomalies as identified by the Route Planner are written. <b>Required.</b>
ROUTER_HOUSEHOLD_FILE	Path to a file containing a list of integral IDs for householders to be planned.
PLAN-FILE	Name of the file where plans should be written. (Overwrites an existing file.) <b>Required.</b>
TRANSIT_ROUTE_FILE	File containing routes of transit vehicles.
TRANSIT_SCHEDULE_FILE	File containing schedules of transit vehicles.
ROUTER_INTERNAL_PLAN_SIZE	Positive integer. Should be enough to accommodate the length (in number of nodes) of the shortest path between any two nodes in the network (and may need to be quite large when multimodal plans are used). <b>Default 400.</b>
ROUTER_MESSAGE_LEVEL	Level of warning messages: - 2 (ERROR) - 1 (PRINT) 0 (SEVERE WARNING) 1 (WARNING) Produces information about possible anomalies encountered by the Route Planner. <b>Default 1.</b>
LOG_ROUTING	Turns on Route Planner logging. This produces information about the status and progress of the Router. <b>Default 0.</b>
LOG_ROUTER_DETAIL	Turns on detailed Route Planner logging. Produces many messages. <b>Default 0.</b>
ROUTER_DELAY_NOISE	Percentage of noise to add to link travel times. <b>Default 0.</b>
ROUTER_SEED	Seed to use for random number generator. If key is set to 0, use process ID. <b>Default 0.</b>
ROUTER_WALKING_SPEED	Speed to use when computing travel times for walk links (meters/second). <b>Default 1.0.</b>

ROUTER_BIKING_SPEED	Speed to use when computing travel times for walk links traversed by bicycle (meters/second). <b>Default 4.0.</b>
ROUTER_GET_ON_TRANSIT_DELAY	Delay/travel time encountered when boarding a transit vehicle. <b>Default 3 seconds.</b>
ROUTER_GET_OFF_TRANSIT_DELAY	Delay/travel time encountered when exiting a transit vehicle. <b>Default 4 seconds.</b>
ROUTER_FILTER_INCLUDE_VEHICLE	Plan vehicle types to include in plan file. <b>Default is to include all vehicle types.</b>
ROUTER_FILTER_EXCLUDE_VEHICLE	Plan vehicle types not included in the plan file. <b>Default is to include no vehicle types.</b> Only one of INCLUDE_VEHICLE and EXCLUDE_VEHICLE can be specified.
ROUTER_FILTER_INCLUDE_MODE	Plan modes to include in plan file. <b>Default is to include all modes.</b>
ROUTER_FILTER_EXCLUDE_MODE	Plan modes not included in plan file. <b>Default is to include no modes.</b> Only one of INCLUDE_MODE and EXCLUDE_MODE may be specified.
ROUTER_LINK_DELAY_FILE	Feedback file from which to read link delays/travel times. If the key is not present or the file does not exist, the free speed delays are used.
<i>ROUTER_NUMBER_THREADS*</i>	Positive integer. Number of worker threads to be used. A value of 0 means no threads will be used. <b>Default 0.</b>
<i>ROUTER_OVERDO*</i>	Nonnegative float. If set to 0, no adjustment is made to the distance estimates. If positive, the search for the shortest path to the origin will be biased in the direction of a straight line to the destination. This will produce non-optimal paths. The paths will still be reasonable, but the heuristic may ignore relatively small congestions on certain links, and this can sacrifice optimality. <b>Default 0.0.</b>
<i>ROUTER_CORR*</i>	Float between 0 and 1. The Route Planner will change the reported length of a link to be equal to its Euclidean length whenever the ratio of the two is less than this value. This is done in order to avoid problems when the Sedgewick-Vitter heuristic is used. <b>Default 0.0.</b>
<i>ROUTER_ZERO_BACKD*</i>	Integer, 0 or 1. <b>Default 0.</b>

*\*These files are used in the Heuristics algorithm for the Route Planner, which are not currently described in this section because of lack of information.*

## 5.8 Algorithms

The algorithm adopted for the Route Planner is a variant of Dijkstra’s procedure for finding shortest paths, which is suitably modified to accommodate time-dependent delays/travel times, and label sequence constraints. The underlying problem is referred to as the “Time-Dependent Label-Constrained Shortest Path Problem (TDLSP)”. Before we consider the TDLSP, we first introduce two other basic shortest path problems, namely, the “Time-Independent Shortest Path Problem (TISP)” and the “Time-Independent Label-Constrained Shortest Path Problem (TILSP)”.

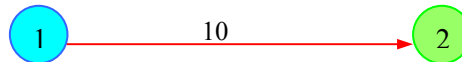
In the subsections identified below, we therefore discuss three types of problems:

- 5.8.1 Time-Independent Shortest Path Problem (TISP),
- 5.8.2 Time-Independent Label-Constrained Shortest Path Problem (TILSP), and
- 5.8.3 Time-Dependent Label-Constrained Shortest Path Problem (TDLSP).

Here we provide a simplified version of the workings of these algorithms. However, a more rigorous and detailed explanation of these algorithms is presented in *Appendix A*.

### 5.8.1 Time-Independent Shortest Path Problem (TISP)

The TISP has a **constant** *travel time*\* on each link within the given network. For example, a link between node 1 and node 2 shown below has a travel time equal to 10 seconds, regardless of the arrival time at node 1.



**Figure-5.22: Constant Link Travel Time from Node 1 to Node 2.**

*\*Note: Generally, a travel time could be replaced by a travel cost, so that the shortest path problem would seek a minimum cost path. Currently, TRANSIMS considers a shortest path problem that minimizes only the total travel time.*

Suppose that we are given a network  $G$  having  $m$  nodes,  $n$  links, along with a travel time  $t_{ij}$  associated with each link  $(i, j)$  in  $G$ , a starting node (the 1<sup>st</sup> node), a destination node (the  $m^{\text{th}}$  node), and a starting time ( $t_1$ ) at the origin node. The shortest path problem is to find a shortest (minimum travel time) path from the starting node to the destination node in  $G$  by implicitly evaluating the various routes between the starting node 1 and all the other  $m$  nodes for that starting time  $t_1$ .

Consider the case when all  $t_{ij} \geq 0$ . In this case, a very simple and efficient procedure, known as Dijkstra’s algorithm, exists for finding a shortest path (from node 1 to node  $m$ ). This method also automatically yields the shortest path from node 1 to all of the other nodes as well.

## Mathematical Terminology and Definitions

The time  $t_i$  denotes an arrival time at node  $i$ . For example,  $t_1$  is the arrival/starting time at the origin node 1, and  $t_m$  is the arrival time at the destination node  $m$ , which represents the ending time of the trip from node 1 to node  $m$ .

$t_{ij}$  is a travel time between the node  $i$  and node  $j$ .

$X$  is a set that contains node 1 (the starting node) and any other nodes in the network  $G$  for which the shortest path has currently been determined, but not node  $m$  (the destination node).

$SE$  is a *scan-eligible* set that contains nodes *adjacent* to the nodes in  $X$ . Note that there exists at most a *single link* between any pair of nodes in  $X$  and  $SE$ .

**Forward star of node  $p$**  is the set of nodes that are adjacent to the node  $p$  and for which there exists an arc from  $p$  to each of the nodes in this set. For example, as shown in the network below, nodes  $Q$  and  $R$  belong to the forward star of the node  $p$  but the node  $S$  is *not* in the forward star of the node  $p$ .

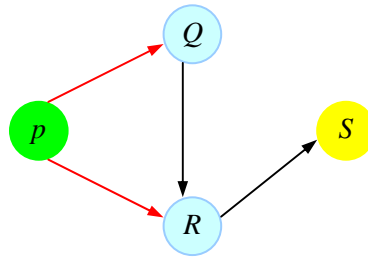


Figure-5.23: Forward Star of Node P.

$(i, j)$  is a link between node  $i$  and node  $j$ .

$(X, SE) = \{(i, j): i \in X, j \in SE\}$ , is the set of possible links from the nodes in  $X$  to the nodes in  $SE$ . (This is called a *cut-set*.)

**DOWN ( $\cdot$ ) label:** if  $(i, j)$  is a link included in the current estimate of the shortest path, we set  $\text{DOWN}(j) = i$ .

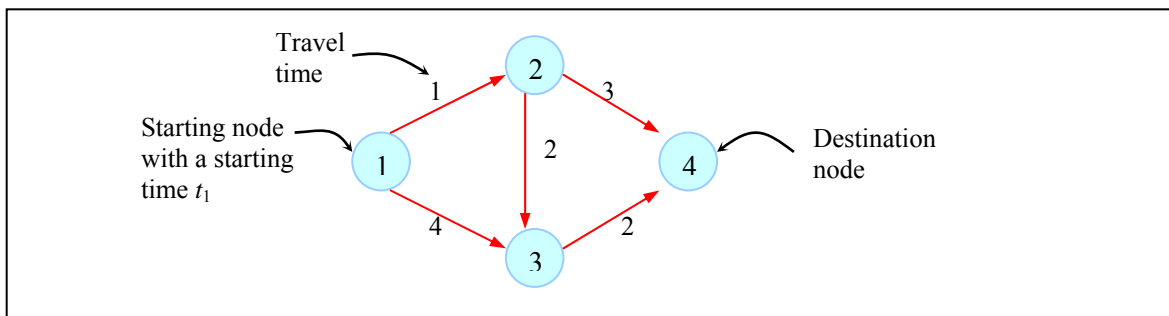


Figure-5. 24: An example of a simple network.

In Figure-5.24, the network has four nodes. Node 1 is the starting node, and node 4 is the destination node. Each link has a time-independent travel time. There are four possible sets  $X$  that contain “node 1” and “any other nodes in the network (nodes 2 and 3) but not node 4”. The four possible sets of this type are  $X = \{1\}$ ,  $X = \{1, 2\}$ ,  $X = \{1, 3\}$ , and  $X = \{1, 2, 3\}$ . Table-5.17 shows the possible sets  $X$ , their corresponding scan-eligible sets  $SE$ , and the cut-sets  $(X, SE)$ . (Note that Dijkstra’s algorithm *does not* require this enumeration, and we only display this here for the sake of illustration.)

Finally, in the (shortest) path  $\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{4}$  from node 1 to 4, we have  $\text{DOWN}(4) = 2$ , and  $\text{DOWN}(2) = 1$ .

**Table-5.17: The possible sets  $X$ , their corresponding scan-eligible sets  $SE$  and the cut-sets  $(X, SE)$**

$X$	$SE$	$(X, SE)$
$X = \{1\}$	$\{2, 3\}$ <i>Note that node 4 is not included because node 4 is not adjacent to node 1 in this network.</i>	$\{(1, 2), (1, 3)\}$
$X = \{1, 2\}$	$\{3, 4\}$	$\{(1, 3), (2, 3), (2, 4)\}$
$X = \{1, 3\}$	$\{2, 4\}$	$\{(1, 2), (3, 4)\}$
$X = \{1, 2, 3\}$	$\{4\}$	$\{(2, 4), (3, 4)\}$

## Dijkstra’s Algorithm

### INITIALIZATION STEP

1. Set a starting time  $t_1$  for the starting node (node 1) as desired.
2. Let  $SE$  initially contain only the starting node,  $SE = \{1\}$ , and let  $X$  be empty.
3. Label node 1 with its starting time  $t_1$ , and label all the other nodes as infinity ( $\infty$ ).

### MAIN ITERATIVE STEP

1. If  $SE$  is empty, then stop; the destination node is unreachable from the starting node. Otherwise, pick the node  $p$  from  $SE$  that has the smallest label  $t_p$  (break ties arbitrarily, but in favor of the destination node). Remove this node from  $SE$  and add it to  $X$ .
2. If  $p$  equals the destination node  $m$ , then stop; the shortest path to node  $m$  is of length  $t_m$ , and can be traced by following the  $\text{DOWN}(\ )$  labels backwards.
3. Else, scan the forward star of  $p$ . For each node  $q$  in this forward star of  $p$ , if  $t_p + t_{pq}$  is less than the current label  $t_q$  of node  $q$ , then re-set  $t_q = t_p + t_{pq}$ , let  $\text{DOWN}(q) = p$ , and let  $SE = SE \cup \{q\}$ .
4. Return to Step 1.

**Note:** At any stage of this algorithm, at the end of Step 3,  $SE$  contains all the nodes that are adjacent to the nodes currently in  $X$ , and the label  $t_j$  of any node  $j$  in  $SE$  equals the minimum over all  $i$  in  $X$  such that  $(i, j)$  belongs to the cut-set  $(X, SE)$  of  $t_i + t_{ij}$ .

The following example illustrates the above method for determining a simple shortest path for the network shown in Figure-5.25.

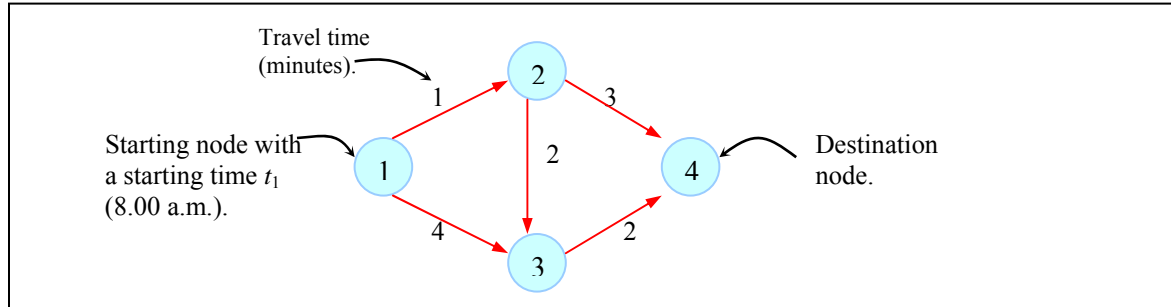


Figure-5. 25: An example of a simple network.

### INITIALIZATION STEP EXECUTIONS

1. Set a starting time at the starting node (node 1) as 8.00 a.m. ( $t_1 = 8$ ).
2. Let the set  $SE$  initially contain only the starting node ( $SE = \{1\}$ ), and let  $X$  be empty ( $X = \emptyset$ ).
3. Label node 1 with its starting time ( $t_1 = 8$ ), and label all other nodes as infinity ( $\infty$ ).

Go to the Main Step.

### MAIN STEP EXECUTIONS

#### Step 1

Note that  $SE = \{1\}$ .

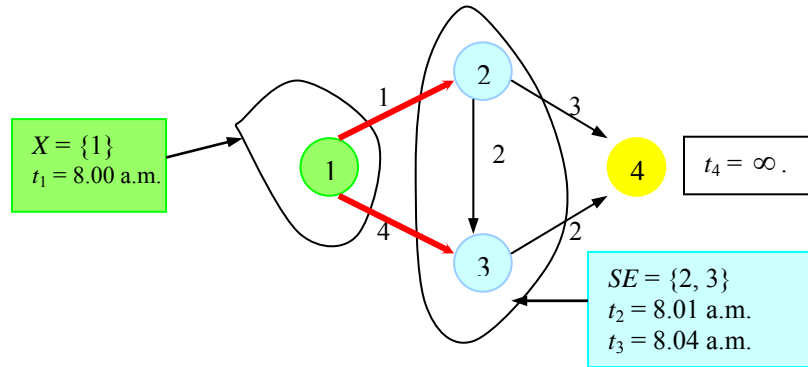
1. Pick the node  $p =$  node 1. Remove the node 1 from  $SE$  ( $SE = \emptyset$ ) and add it to  $X$  ( $X = \{1\}$ ).
2. Scan the forward star of  $p$  (node 1). Here, the nodes 2 and 3 are in the forward star of  $p$  (node 1). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

Table-5.18: Values of  $t_p + t_{pq}$ .

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
1	8.00 a.m.	2	$\infty$	1 minute	8.01 a.m. ( $< \infty$ )
		3	$\infty$	4 minutes	8.04 a.m. ( $< \infty$ )

3. From the table above, all values of  $t_p + t_{pq}$  are less than the current label of node  $q$ . Then set  $t_2 = 8.01$  a.m.,  $\text{DOWN}(2) = 1$ ,  $t_3 = 8.04$  a.m.,  $\text{DOWN}(3) = 1$ , and let  $SE = \emptyset \cup \{2,3\} = \{2, 3\}$ .





**Figure-5.26: Current Shortest Paths from Node 1 to Nodes 2, and 3.**

Note that the red-colored links (  $\rightarrow$  ) show the current shortest paths from node 1 to nodes 2, and 3 at this step.

Step 2

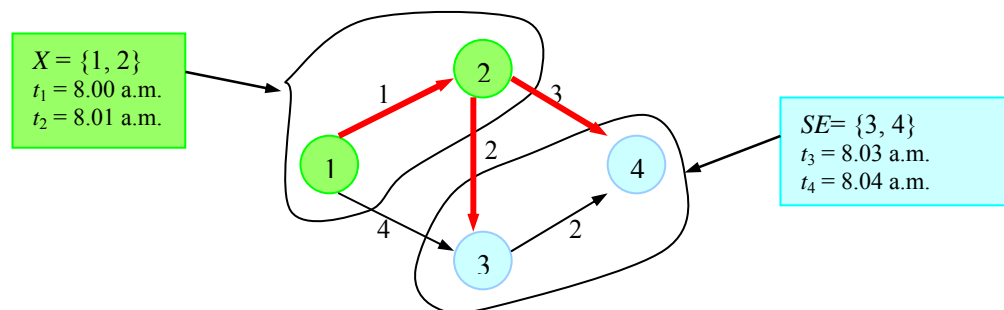
Note that  $SE = \{2, 3\}$ .

1. Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p =$  node 2. Remove the node 2 from  $SE$  ( $SE = \{3\}$ ) and add it to  $X$  ( $X = \{1, 2\}$ ).
2. Scan the forward star of  $p$  (node 2). Here, the nodes 3 and 4 are in the forward star of  $p$  (node 2). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .


**Table-5.19: Values of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
2	8.01 a.m.	3	8.04 a.m.	2 minutes	8.03 a.m. (<8.04 a.m.)
		4	$\infty$	3 minutes	8.04 a.m. (< $\infty$ )

3. From the table above, all values of  $t_p + t_{pq}$  are less than the current label of node  $q$ . Then set  $t_3 = 8.03$  a.m.,  $DOWN(3) = 2$ ,  $t_4 = 8.04$  a.m.,  $DOWN(4) = 2$ , and let  $SE = \{3\} \cup \{3, 4\} = \{3, 4\}$ .



**Figure-5.27: Current Shortest Paths from Node 1 to Nodes 2, 3 and 4.**

Note that the red-colored links (  ) show the current shortest paths from node 1 to nodes 2, 3, and 4 at this step.

Step 3

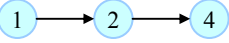
Note that  $SE = \{3, 4\}$ .

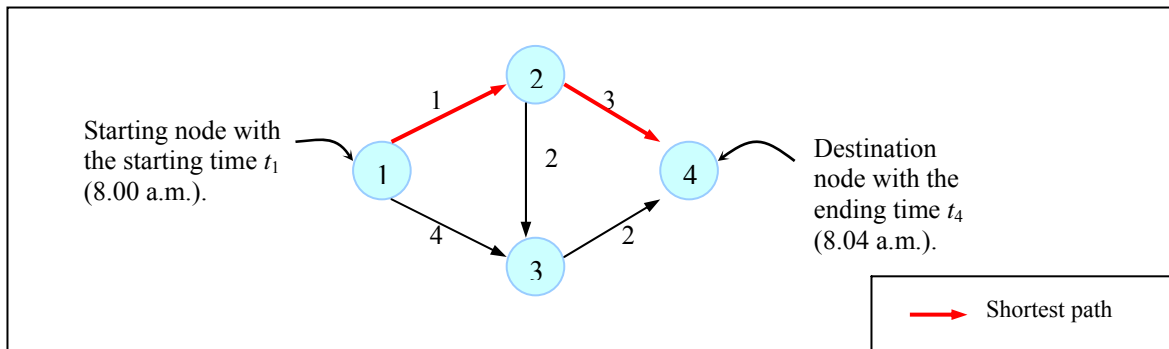
1. Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = \text{node } 3$ . Remove the node 3 from  $SE$  ( $SE = \{4\}$ ) and add it to  $X$  ( $X = \{1, 2, 3\}$ ).
2. Scan the forward star of  $p$  (node 3). Here, the node 4 is in the forward star of  $p$  (node 3). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.20: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
3	8.03 a.m.	4	8.04 a.m.	2 minutes	8.05 a.m. (>8.04 a.m.)

3. From the table above,  $t_p + t_{pq}$  is not less than the current label of node  $q$ . Hence,  $t_4$  and  $\text{DOWN}(4)$  remain the same as in the previous step, which are:  
 $t_4 = 8.04 \text{ a.m.}$ ,  $\text{DOWN}(4) = 2$ . Moreover,  $SE = \{4\}$ .

Note that the set  $SE$  now contains only the node 4, which is the **destination node**. We therefore, stop with the ending time =  $t_4 = 8.04 \text{ a.m.}$  Tracing backwards yields the shortest path  noting that we have  $\text{DOWN}(4) = 2$ , and  $\text{DOWN}(2) = 1$ .



**Figure-5.28: Shortest Path from Node 1 to Node 4.**

The following example illustrates a practical instance of the Time-Independent Shortest Path Problem.

### 5.8.1.1 Practical Example for the Time-Independent Shortest Path Problem (TISP)

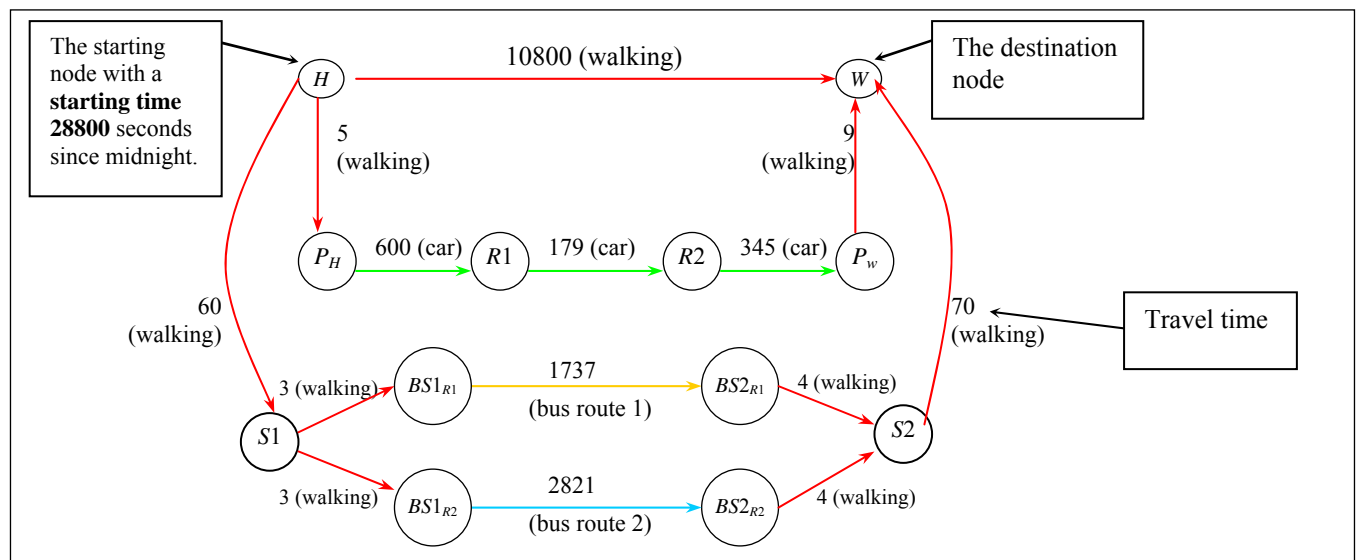
The example shown in Figure-5.29 is used to illustrate a solution of the Time-Independent Shortest Path Problem (TISP) for a network that is represented in the format required by the TRANSIMS Internal Network. The same example will be used for the other two problems with appropriately modified data: the Time-Independent Label Constrained Shortest Path Problem (TILSP), and the Time-Deendent Label-Constrained Shortest Path Problem (TDLSP).

Suppose that we are given a (simple) single-trip request, without label constraints (Table-5.21) for a traveler designated ID 13300, involving a trip starting from “home” and going to “work”.

**Table-5.21: Single-trip request for a traveler**

Person ID	Trip Number	Starting Location	Destination Location	Starting Time (seconds since midnight)
13300	1	845654 (Home)	833503 (Work)	28800 (8.00 a.m.).

Furthermore, suppose that we have constructed the Internal Network as shown in Figure-5.29, having unidirectional links and associated constant travel times as shown therein. For example, a walk link between node *H* (home) and node *W* (work) has a travel time of 10800 seconds and a starting time of 28800 seconds since midnight. Hence, the ending time at *W* along this route equals the starting time plus the walk time, which results in 39600 seconds since midnight.



**Figure-5. 29: The Network for the illustrative example for TISP**

From Figure-5.29, the starting location is node  $H$  (home), and the starting time is  $t_H = 28800$  seconds (since midnight). The destination location is node  $W$  (work location). If this traveler wants to go to work by car, he/she must travel via node  $P_H$  (car parking at home), then proceed to node  $R1$  (via road 1), next to node  $R2$  (via road 2), then to node  $P_W$  (car parking at work), and finally walk to the work place.

If the traveler prefers to go to work by bus, he/she must travel via node  $S1$  (a bus shelter 1 for passengers). This Internal Network has two bus routes. A traveler has to choose between bus route 1 and bus route 2. If they want to travel via bus route 1, they must get on the bus at node  $BS1_{R1}$  (bus-parking place 1 for bus route 1) and then get off the bus at node  $BS2_{R1}$  (bus parking place 2 for bus route 1). If they prefer to travel via bus route 2, they must get on the bus at node  $BS1_{R2}$  (bus-parking place 1 for bus route 2) and then get off the bus at node  $BS2_{R2}$  (bus-parking place 2 for bus route 2). After traveling via one of the bus routes, they must pass node  $S2$  (a bus shelter 2 for passengers), and then walk to the work place.

### INITIALIZATION STEP

1. Set a starting time at the starting node (node  $H$ ) as 8.00 a.m. ( $t_H = 28800$  seconds since midnight).
2. Let the set  $SE$  initially contain only the starting node ( $SE = \{H\}$ ), and let  $X$  be empty ( $X = \emptyset$ ).
3. Label node  $H$  with its starting time ( $t_H = 28800$ ), and label all other nodes as infinity ( $\infty$ ).

Go to the Main Step.

### MAIN STEP EXECUTIONS

#### Step 1

Note that  $SE = \{H\}$ .

1. Pick the node  $p = \text{node } H$ . Remove the node  $H$  from  $SE$  ( $SE = \emptyset$ ) and add it to  $X$  ( $X = \{H\}$ ).
2. Scan the forward star of  $p$  (node  $H$ ). Here, the nodes  $W$ ,  $P_H$ , and  $S1$  are in the forward star of  $p$  (node  $H$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

Table--5.22: Value of  $t_p + t_{pq}$ .

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$H$	28800	$W$	$\infty$	10800	39600 ( $< \infty$ )
		$P_H$	$\infty$	5	28805 ( $< \infty$ )
		$S1$	$\infty$	60	28860 ( $< \infty$ )

3. From the table above, all values of  $t_p + t_{pq}$  are less than the corresponding current label of node  $q$ . Hence, set

$$t_w = 39600, \text{DOWN}(W) = H,$$

$$t_{P_H} = 28805, \text{DOWN}(P_H) = H,$$

$$t_{S1} = 28860, \text{DOWN}(S1) = H, \text{ and let } SE = \emptyset \cup \{W, P_H, S1\} = \{W, P_H, S1\}.$$

Step 2

Note that  $SE = \{W, P_H, S1\}$ .

1. Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = P_H$ . Remove the node  $P_H$  from  $SE$  ( $SE = \{W, S1\}$ ) and add it to  $X$  ( $X = \{H, P_H\}$ ).
2. Scan the forward star of  $p$  (node  $P_H$ ). Here, the node  $R1$  is in the forward star of  $p$  (node  $P_H$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.23: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$P_H$	28805	$R1$	$\infty$	600	29405 ( $< \infty$ )

3. From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set

$$t_{R1} = 29405, \text{DOWN}(R1) = P_H,$$

$$\text{and let } SE = \{W, S1\} \cup \{R1\} = \{W, S1, R1\}.$$

Step 3

Note that  $SE = \{W, S1, R1\}$ .

1. Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = S1$ . Remove the node  $S1$  from  $SE$  ( $SE = \{W, R1\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1\}$ ).
2. Scan the forward star of  $p$  (node  $S1$ ). Here, the nodes  $BS1_{R1}$  and  $BS1_{R2}$  are in the forward star of  $p$  (node  $S1$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.24: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$S1$	28860	$BS1_{R1}$	$\infty$	3	28863 ( $< \infty$ )
		$BS1_{R2}$	$\infty$	3	28863 ( $< \infty$ )

3. From the table above, all values of  $t_p + t_{pq}$  are less than the corresponding current label of node  $q$ . Hence, set

$$t_{BS1_{R1}} = 28863, \text{DOWN}(BS1_{R1}) = S1,$$

$$t_{BS1_{R2}} = 28863, \text{DOWN}(BS1_{R2}) = S1,$$

$$\text{and let } SE = \{W, R1\} \cup \{BS1_{R1}, BS1_{R2}\} = \{W, R1, BS1_{R1}, BS1_{R2}\}.$$

Step 4

Note that  $SE = \{W, R1, BS1_{R1}, BS1_{R2}\}$ .

1. Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . The node  $BS1_{R1}$  and  $BS1_{R2}$  have the smallest label. Let us select one of them, say, the node  $BS1_{R1}$ . Hence,  $p = BS1_{R1}$ . Remove the node  $BS1_{R1}$  from  $SE$  ( $SE = \{W, R1, BS1_{R2}\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1, BS1_{R1}\}$ ).

2. Scan the forward star of  $p$  (node  $BS1_{R1}$ ). Here, the node  $BS2_{R1}$  is in the forward star of  $p$  (node  $BS1_{R1}$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.25: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$BS1_{R1}$	28863	$BS2_{R1}$	$\infty$	1737	30600 ( $< \infty$ )

3. From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set

$$t_{BS2_{R1}} = 30600, \text{DOWN}(BS2_{R1}) = BS1_{R1},$$

$$\text{and let } SE = \{W, R1, BS1_{R2}\} \cup \{BS2_{R1}\} = \{W, R1, BS1_{R2}, BS2_{R1}\}.$$

#### Step 5

Note that  $SE = \{W, R1, BS1_{R2}, BS2_{R1}\}$ .

- Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = BS1_{R2}$ . Remove the node  $BS1_{R2}$  from  $SE$  ( $SE = \{W, R1, BS2_{R1}\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1, BS1_{R1}, BS1_{R2}\}$ ).
- Scan the forward star of  $p$  (node  $BS1_{R2}$ ). Here, the node  $BS2_{R2}$  is in the forward star of  $p$  (node  $BS1_{R2}$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.26: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$BS1_{R2}$	28863	$BS2_{R2}$	$\infty$	2821	31684 ( $< \infty$ )

3. From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set

$$t_{BS2_{R2}} = 31684, \text{DOWN}(BS2_{R2}) = BS1_{R2},$$

$$\text{and let } SE = \{W, R1, BS2_{R1}\} \cup \{BS2_{R2}\} = \{W, R1, BS2_{R1}, BS2_{R2}\}.$$

#### Step 6

Note that  $SE = \{W, R1, BS2_{R1}, BS2_{R2}\}$ .

- Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = R1$ . Remove the node  $R1$  from  $SE$  ( $SE = \{W, BS2_{R1}, BS2_{R2}\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1, BS1_{R1}, BS1_{R2}, R1\}$ ).
- Scan the forward star of  $p$  (node  $R1$ ). Here, the node  $R2$  is in the forward star of  $p$  (node  $R1$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.27: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$R1$	29405	$R2$	$\infty$	179	29584 ( $< \infty$ )

3. From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set

$$t_{R2} = 29584, \text{DOWN}(R2) = R1,$$

$$\text{and let } SE = \{W, BS2_{R1}, BS2_{R2}\} \cup \{R2\} = \{W, BS2_{R1}, BS2_{R2}, R2\}.$$

Step 7

Note that  $SE = \{W, BS2_{R1}, BS2_{R2}, R2\}$ .

- Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = R2$ . Remove the node  $R2$  from  $SE$  ( $SE = \{W, BS2_{R1}, BS2_{R2}\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1, BS1_{R1}, BS1_{R2}, R1, R2\}$ ).
- Scan the forward star of  $p$  (node  $R2$ ). Here, the node  $P_W$  is in the forward star of  $p$  (node  $R2$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.28: Value of  $t_p + t_{pq}$ .**

Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
R2	29584	$P_W$	$\infty$	345	29929 ( $< \infty$ )

- From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set  $t_{P_W} = 29929$ ,  $DOWN(P_W) = R2$ , and let  $SE = \{W, BS2_{R1}, BS2_{R2}\} \cup \{P_W\} = \{W, BS2_{R1}, BS2_{R2}, P_W\}$ .

Step 8

Note that  $SE = \{W, BS2_{R1}, BS2_{R2}, P_W\}$ .

- Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = P_W$ . Remove the node  $P_W$  from  $SE$  ( $SE = \{W, BS2_{R1}, BS2_{R2}\}$ ) and add it to  $X$  ( $X = \{H, P_H, S1, BS1_{R1}, BS1_{R2}, R1, R2, P_W\}$ ).
- Scan the forward star of  $p$  (node  $P_W$ ). Here, the node  $W$  is in the forward star of  $p$  (node  $P_W$ ). For each node  $q$  in this forward star of  $p$ , check if  $t_p + t_{pq}$  is less than the current label of node  $q$ .

**Table--5.29: Value of  $t_p + t_{pq}$ .**

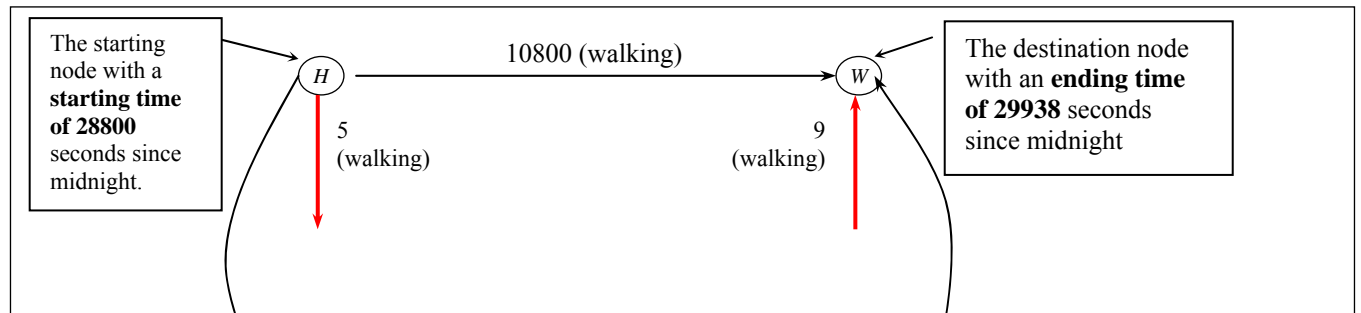
Starting node ( $p$ )	Starting time ( $t_p$ )	Forward star ( $q$ )	Current label of node $q$ ( $t_q$ )	Travel time ( $t_{pq}$ )	starting time + travel time ( $t_p + t_{pq}$ )
$P_W$	29929	$W$	39600	9	29938 ( $< 39600$ )

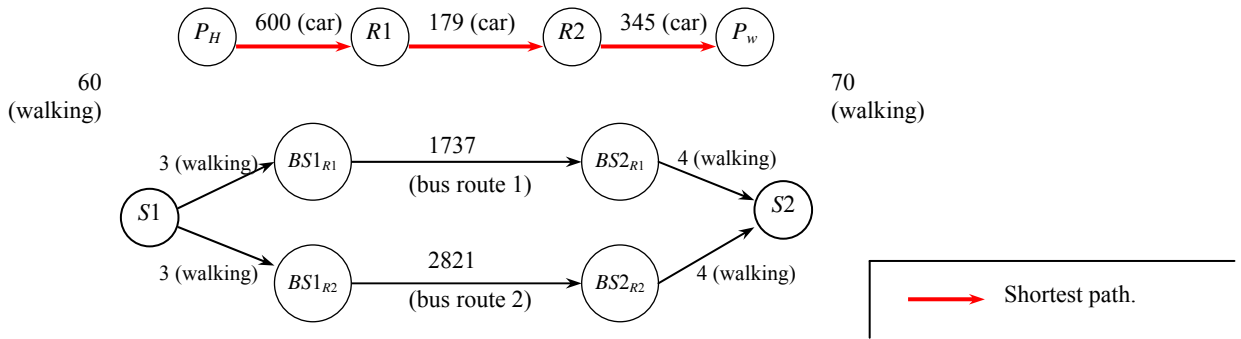
- From the table above,  $t_p + t_{pq}$  is less than the current label of node  $q$ . Hence, set  $t_W = 29939$ ,  $DOWN(W) = P_W$ , and let  $SE = \{W, BS2_{R1}, BS2_{R2}\} \cup \{W\} = \{W, BS2_{R1}, BS2_{R2}\}$ .

Step 9

Note that  $SE = \{W, BS2_{R1}, BS2_{R2}\}$ .

Pick the node  $p$  from  $SE$  that has the smallest label  $t_p$ . Hence,  $p = W$ , which is the **destination node**. We therefore, stop with an ending time of 29938 seconds since midnight. Tracing backwards using the  $DOWN(\cdot)$  labels yields **the shortest path**  $H \rightarrow P_H \rightarrow R1 \rightarrow R2 \rightarrow P_W \rightarrow W$  (as shown in Figure-5.30).





**Figure-5. 30: The shortest path solution for the example**

Next, we consider the case where each link is also ascribed a *label* taken from some *collection of symbols* (such as letters).

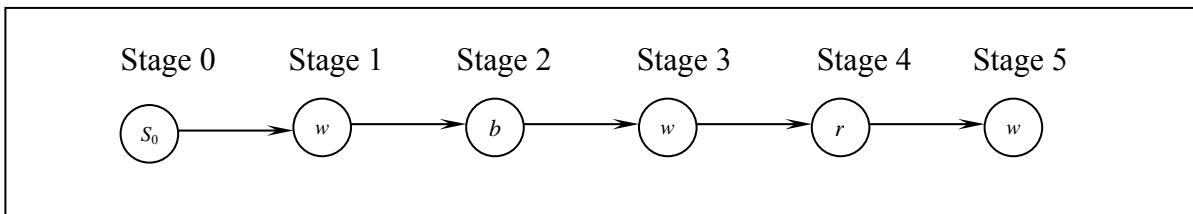
### 5.8.2 Time-Independent Label Constrained Shortest Path Problem (TILSP)

For the Route Planner, a lettered *label* is considered as a “*travel mode*” and a *collection of labels* is considered as a “*mode string*”.

#### Step 1

Examine the admissible mode strings and then construct a corresponding *transition graph*  $G_L$ . The transition graph  $G_L$  is a graph that shows all the possible sequences of the travel modes by which we can reach the destination node from the starting node. Each node in  $G_L$  represents an admissible travel mode. We begin with a single node corresponding to a dummy label  $s_0$  representing *Stage\* 0*. Then *Stage 1* reflects the first admissible travel mode corresponding to the given mode string. Similarly, we designate the stepwise admissible modes for *Stages 2, 3, ...*

For example, if we have an admissible mode string  $wbwrw$  (walk-bus-walk-rail-walk) for a single trip, where these labels indicate the admissible sequence of modes on the path of links, then the corresponding graph  $G_L$  would be as follows:



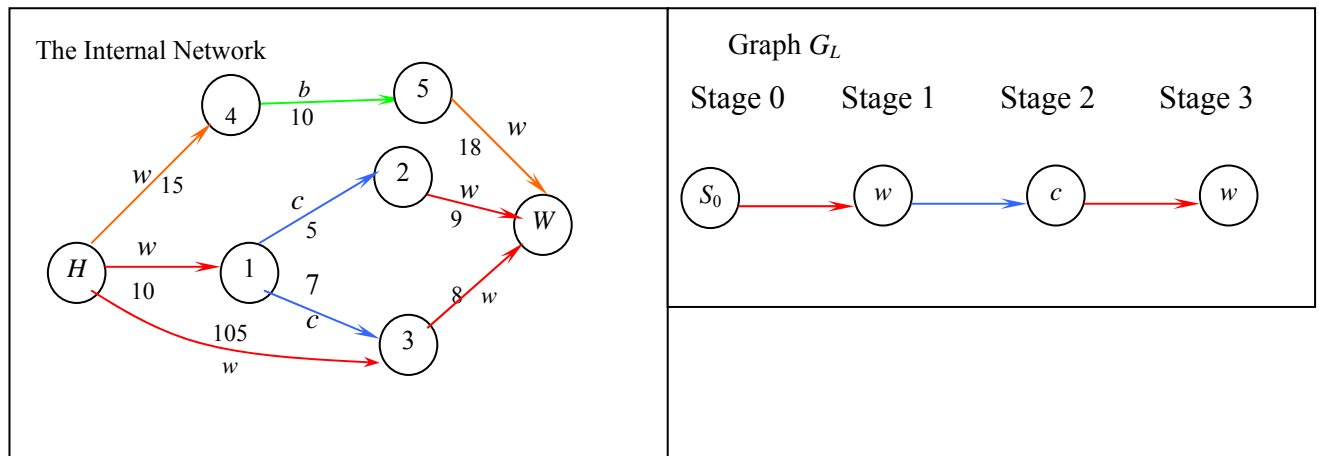
**Figure-5.31: Transition Graph ( $G_L$ ) for Admissible Mode String  $wbwrw$ .**



## Step 2

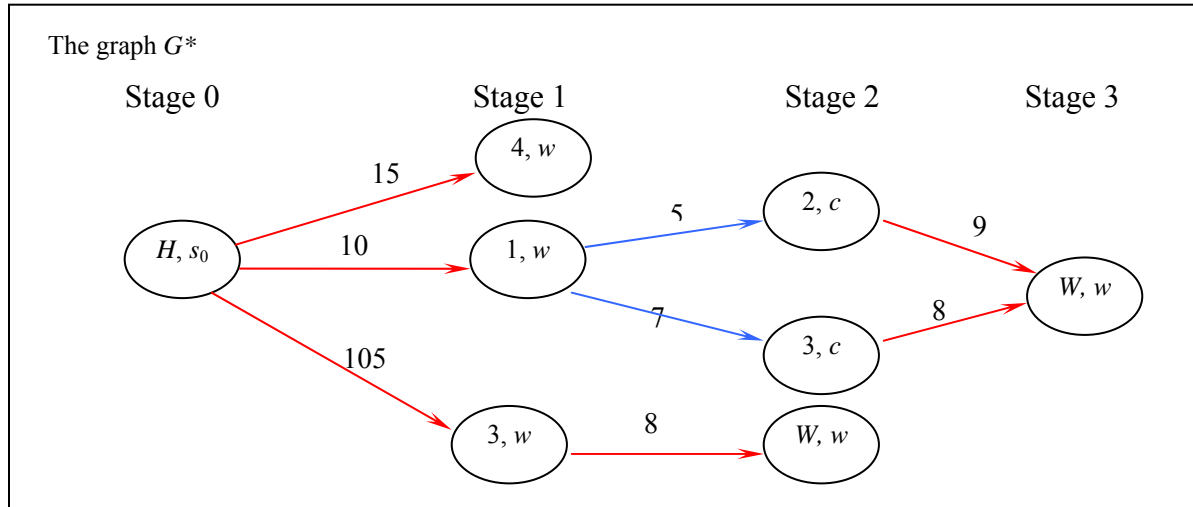
Using the graph  $G_L$  and the Internal Network, we construct a combined graph  $G^*$  on which the actual shortest path problem will be solved. The graph  $G^*$  shows all possible paths for the given single-trip request starting from the origin node and ending at the destination node within the admissible mode string. Beginning with *Stage 0*, the graph  $G^*$  has a node designated as the two-tuple (Starting node,  $s_0$ ). Recursively, we will determine nodes for each subsequent *Stage s* as  $\{(i, l)\}$ : it is possible to come to node  $i$  at *Stage s* via an arc with travel mode  $l$ , for each  $s = 1, \dots, S$ , where  $S$  is the maximum possible number of stages before we either reach the destination or exceed the maximum specified travel time limit. At the last *Stage s = S*, only nodes  $(i, l)$  with  $i$  equal to the terminal node  $W$  are admissible, given that the trip does not exceed the maximum time limit specified. (The latter is computed as the starting time plus the maximum travel time.) Each link that connects a pair of nodes in graph  $G^*$  has a constant travel time associated with it.

As an example, given an Internal Network as depicted below, and given a mode string  $\{w c w\}$ , we can construct the transition graph  $G_L$  as shown in figure-5.32.



**Figure-5.32: Internal Network (left) and Transition Graph ( $G_L$ ) (right) for Mode String  $w c w$ .**

The graph  $G^*$  would then be given as follows:



**Figure-5.33: Combined Graph ( $G^*$ ) for Admissible Mode String  $wcw$ .**

**Step 3**

In the graph  $G^*$ , find a shortest path from the starting node ( $H, s_0$ ) to the destination node at Stage  $S$ , using the earlier described Dijkstra’s TISP algorithm.

**5.8.2.2 Example of the Time-Independent Label Constrained Shortest Path Problem (TILSP)**

Suppose that we are given a single-trip request shown in Table-5.30 for a traveler designated ID 13300, involving a trip starting from “home” and going to “work”. The admissible mode strings are “ $w...wc...cw...w$ ” or “ $w...wb...bw...w$ ”. Note that in this context, as in TRANSIMS, the string  $w...wc...cw...w$ , for example, represents a *sequence* of one or more walk links, followed by one or more car links, and ending with one or more walk links.

**Table-5.30: Single-trip requests for a traveler**

Person ID	Trip Number	Starting Location	Destination Location	Starting Time (seconds since midnight)	Maximum Travel Time (second)	Mode String
13300	1	845654 (Home)	833503 (Work)	28800 (8.00 a.m.).	3000 *	wcw or wbw

\*Note: the maximum travel time is 3000 seconds hence the maximum finish time equals the starting time plus the maximum travel time, which is 31800 seconds since midnight (8.50 a.m.).

Furthermore, suppose that we have constructed the Internal Network as given in Figure-5.34, having unidirectional links, along with associated constant travel times, and **travel mode** labels. This example is the same instance described earlier for TISP, except that the links have mode-labels associated with them such as  $w$ ,  $c$ , and  $b$ , and the problem has an associated label string restriction.

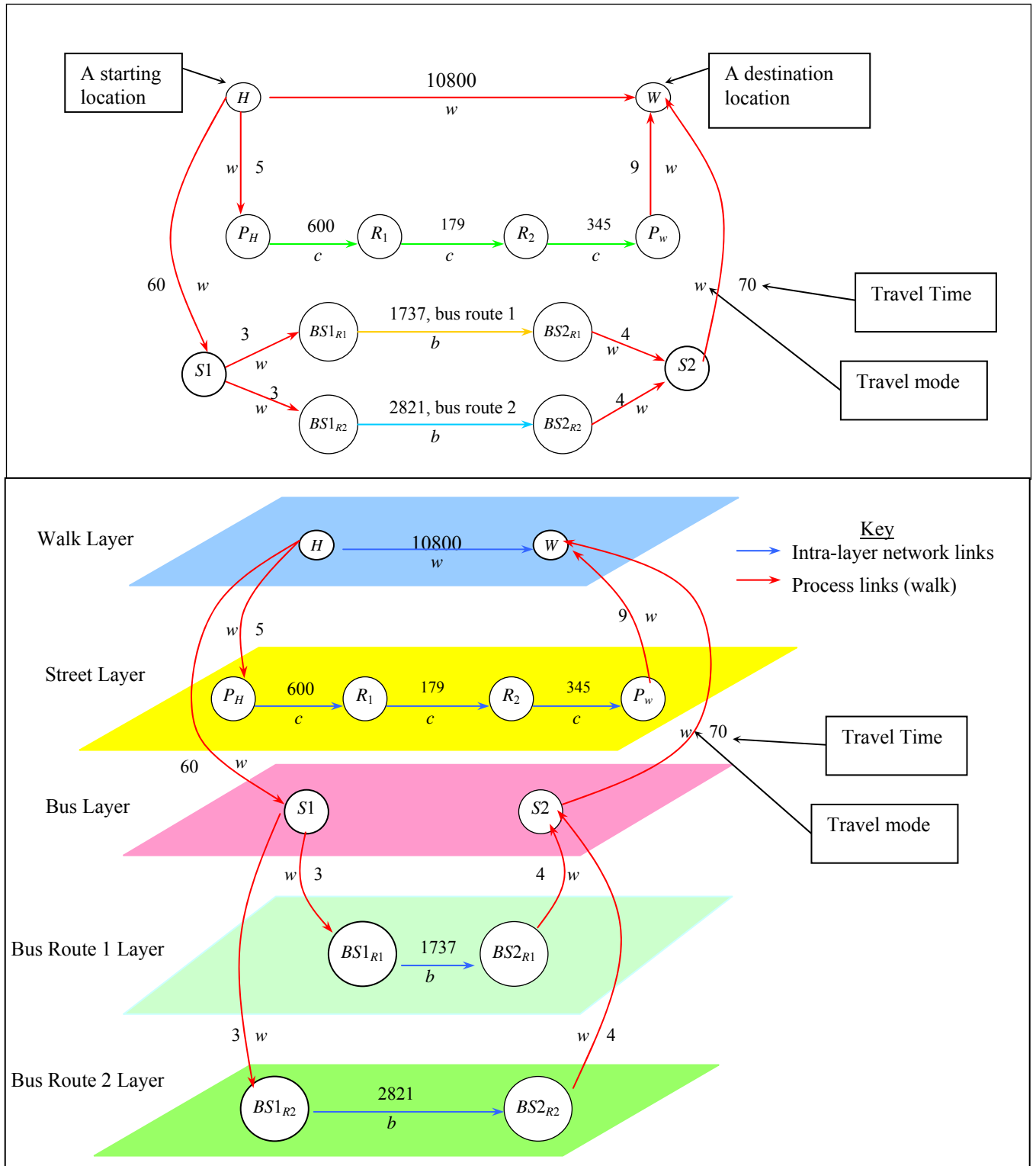
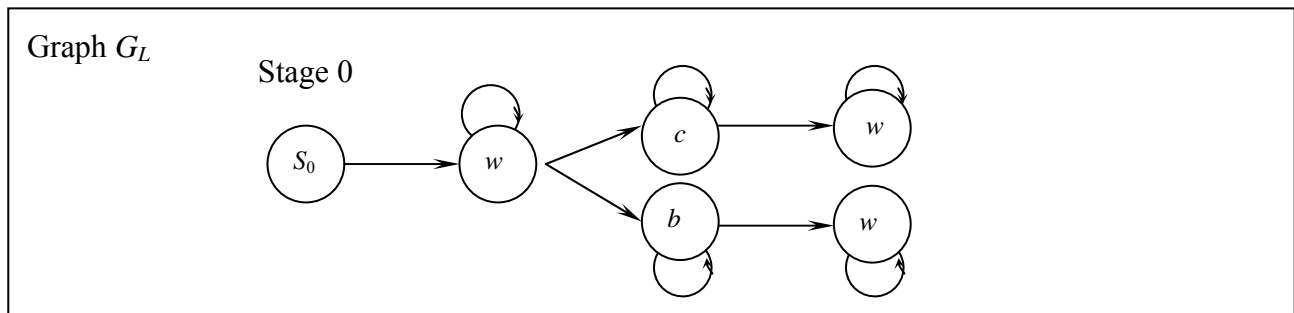


Figure-5.34: Layers of the Internal Network for the example for TILSP

Figure-5.34 depicts the layers of the Internal Network for this example. The starting location ( $H$ ) and the ending location ( $W$ ) are placed on the walk layer. The street layer provides the street network. Here, we have four street nodes ( $P_H$ : car parking at home,  $R_1$ : start of road 1,  $R_2$ : start of road 2, and  $P_W$ : car parking at work). This Internal Network has two bus routes. The bus layer contains the two bus shelters for the passengers ( $S1$  and  $S2$ ). The bus route 1 layer contains a bus route 1 network, which has only one link in our example, from  $BS1_{R1}$  (a bus-parking place 1 for bus route 1) to  $BS2_{R1}$  (a bus-parking place 2 for bus route 1). The bus route 2 layer contains a bus route 2 network, which also has only one link in our example, from  $BS1_{R2}$  (a bus-parking place 1 for bus route 2) to  $BS2_{R2}$  (a bus-parking place 2 for bus route 2).

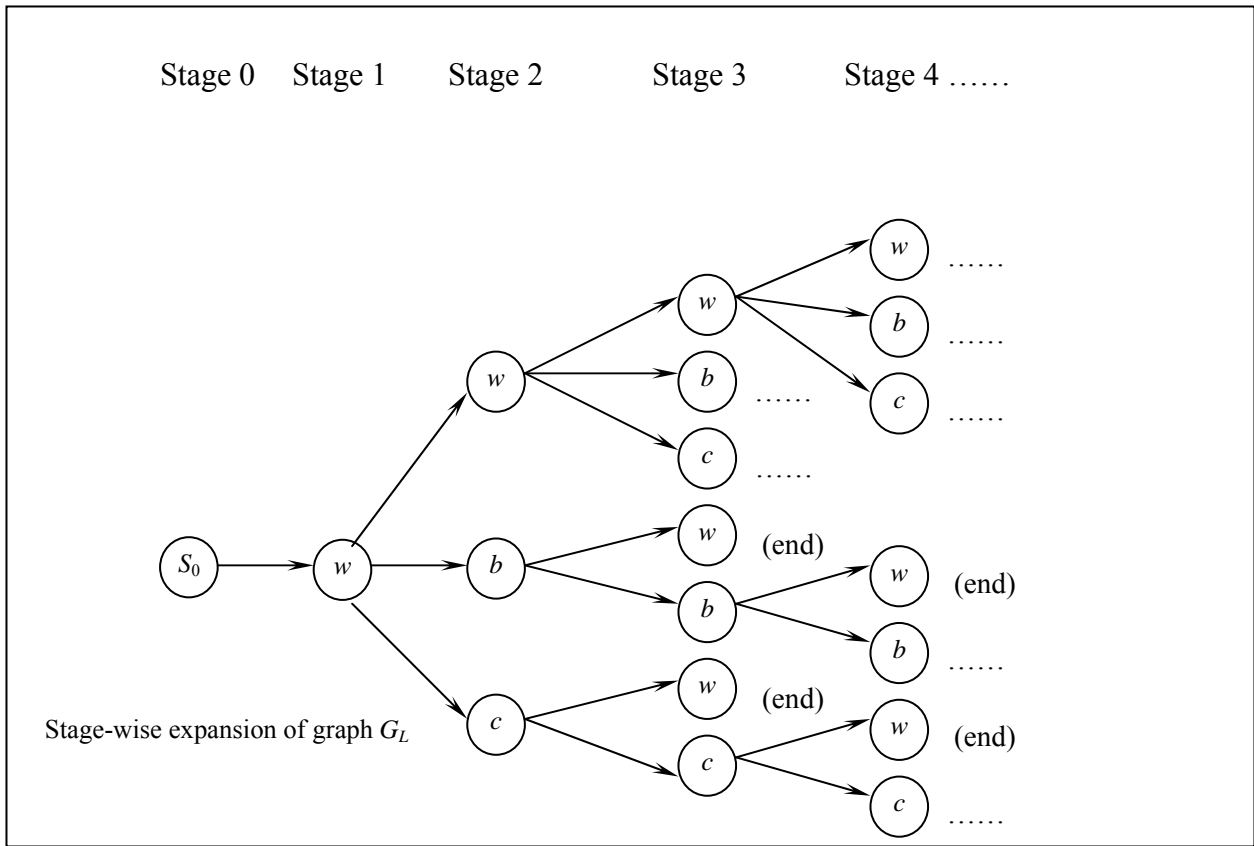
### Step 1

Examining the admissible mode strings “ $w...wc...cw...w$ ” and “ $w...wb...bw...w$ ”, we can construct a corresponding *transition graph*  $G_L$  as follows. We begin with a single node corresponding to a *dummy label*  $s_0$  representing Stage 0. Then, the next transition (Stage 1) is necessarily conducted via a walk link. We might continue to walk over several subsequent stages or transition via a link that represents a car travel or via a link that represents a bus travel. The remainder of  $G_L$  shown below has a similar interpretation.



**Figure-5.35: Transition Graph ( $G_L$ ) for Admissible Mode Strings  $wcw$  and  $wbw$ .**

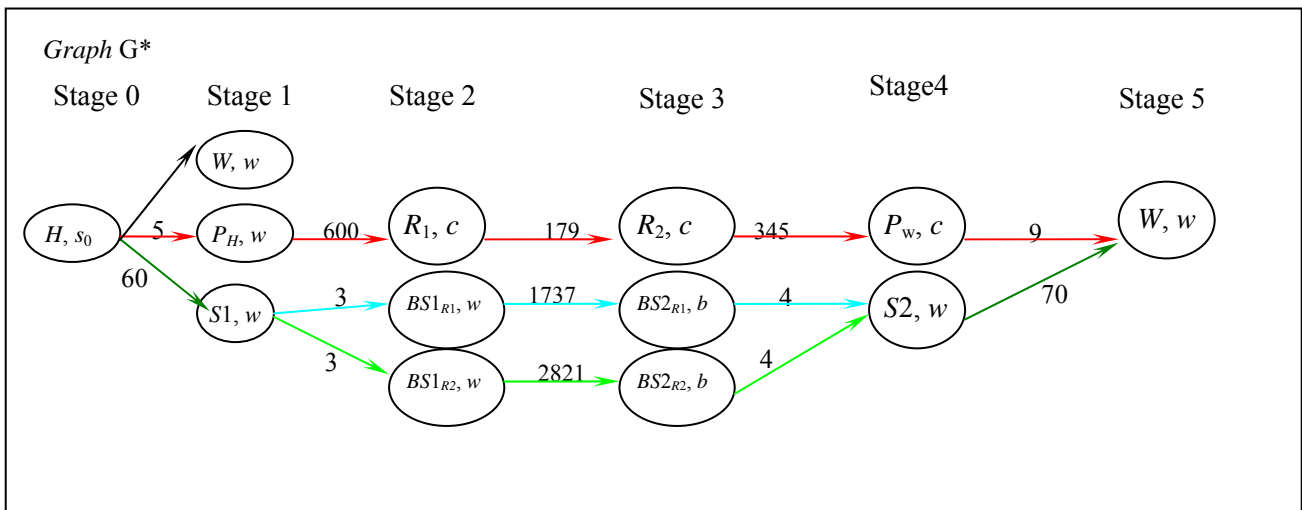
A stage-wise partial blow-up of the graph  $G_L$  is shown below.



**Figure-5.36: Stage-wise Partial Blow-up of Transition Graph ( $G_L$ ) Above.**

**Step 2**

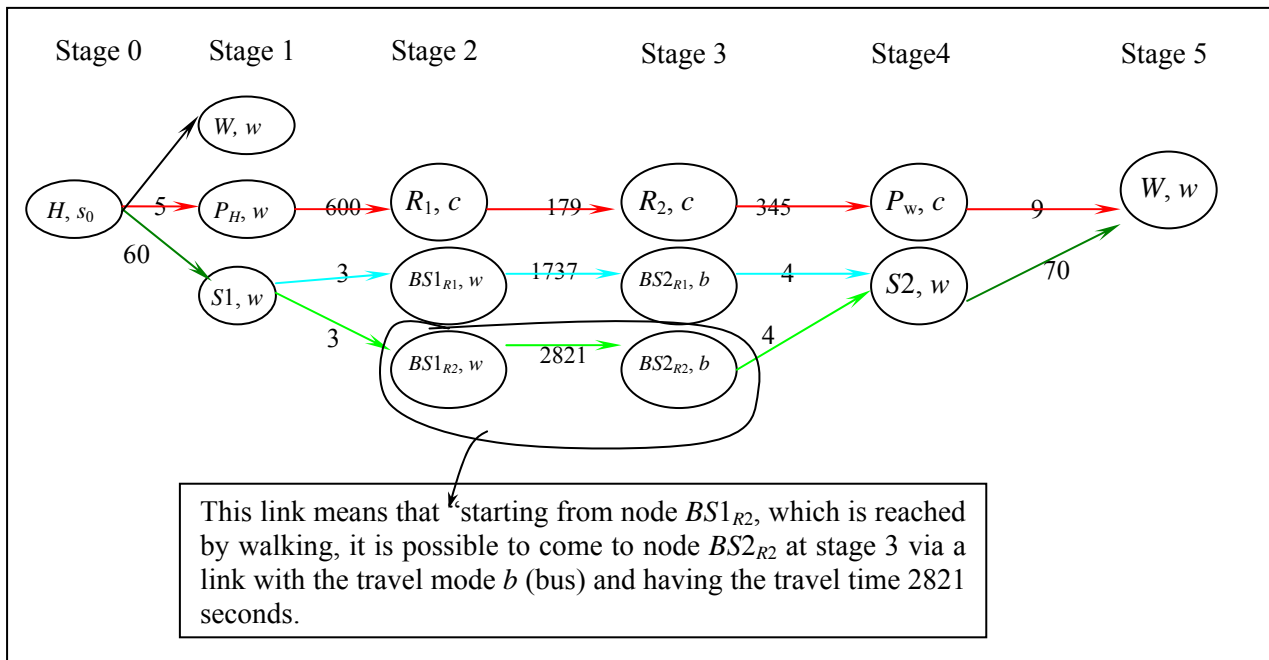
Using the graph  $G_L$  and the Internal Network shown in Figure-5.34, we can construct a combined graph  $G^*$  as described earlier. The actual shortest path problem will be solved on this graph  $G^*$ . Beginning with Stage 0, the graph  $G^*$  has a node  $(H, s_0)$ . Recursively, we determine nodes for each subsequent Stage  $s$  as shown below in order to construct  $G^*$ .



**Figure-5.37: Combined Graph ( $G^*$ ) Solving Shortest Path Problem.**

The graph  $G^*$  contains all possible admissible paths starting from the node  $(H, s_0)$  and ending at node  $(W, w)$  at Stage 5. We can see that there are three possible paths. The first path uses a  $wcccw$ -mode string. The second path uses a  $wwbww$ -mode string. The third path also uses a  $wwbww$ -mode string (on a different bus route). Note that the link from  $(H, s_0)$  to  $(W, w)$  at Stage 1 has no feasible continuation, and this node  $(W, w)$  at Stage 1 is *not* a legitimate terminal node (unless if the string  $www\dots w$  is admissible).

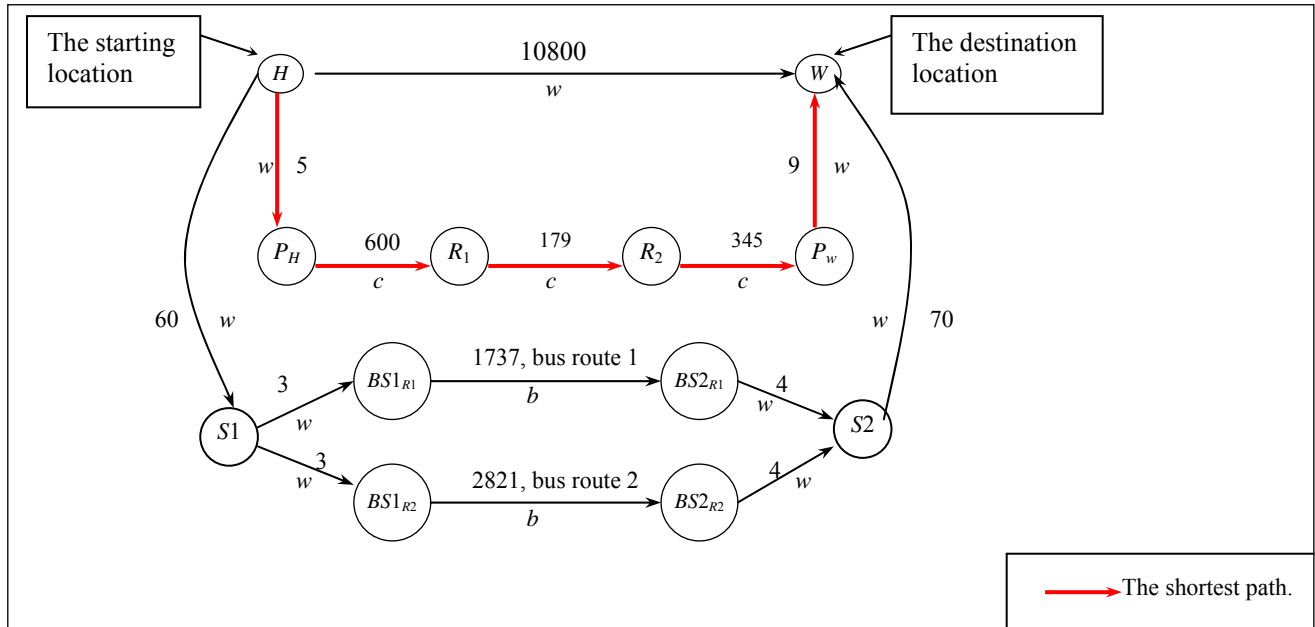
The diagram below offers a specific sample explanation of a feature of the graph  $G^*$ .



**Figure-5.38: A specific sample explanation of a feature of the graph  $G^*$ .**

### Step 3

In the graph  $G^*$ , find the shortest path from the starting node  $(H, s_0)$  to the destination node  $(W, w)$  at Stage 5 using any standard shortest path algorithm. Here we use Dijkstra's algorithm as in the previous example. This yields a shortest path for the Time-Independent Label-Constrained Shortest Path Problem (TILSP), which turns out to be the same as the solution for the previous example (as shown in Figure-5.39).



**Figure-5.39: The shortest path solution for the example for TILSP**

Next, we consider the case where the travel time on a link is dependent on the arrival time on that link.

### 5.8.3 Time-Dependent Label-Constrained Shortest Path Problem (TDLSP)

The algorithm adopted in this case is a variant of Dijkstra's algorithm for finding shortest paths, which is suitably modified to accommodate time-dependent travel times, and label sequence constraints. TRANSIMS constructs  $G^*$  as for Problem TILSP in this case, but instead of constant travel times, the travel times on the links in  $G^*$  are now designated as being time-dependent. The modified form of Dijkstra's algorithm using scan-eligible lists as discussed in Section 5.8.1 is then used. We will describe a different (more effective) viewpoint of this procedure in which  $G^*$  is *not* actually constructed, but is only *implicitly* used. Here, each stage  $s$  will be comprised of further *augmented nodes* of the type  $\{(i, t, l): \text{it is possible to come to node } i \text{ at Stage } s \text{ via an arc with label } l \text{ at time } t\}$ . Let  $N_s$  be the set of all possible nodes  $(i, t, l)$  at Stage  $s$ . The algorithmic process implicitly considers all possible routes within the admissible internal network for the specified string of modes as described below. If no path is found that obeys the mode constraints of the traveler (*Path Anomaly*), or that satisfies the time bound constraint (*Time Anomaly*) or the *Invalid Shared Ride Time anomaly*, then the traveler and the trip request are marked for a *Path/Time/Shared Ride Time Anomaly* feedback to the Selector Module. Otherwise, a shortest route that satisfies the maximum travel time and the label sequence constraints is determined as described below.



**Step 1** (This is the same step discussed in the previous algorithm of Section 5.8.2.)

Examine the admissible mode strings and construct a corresponding *transition graph*  $G_L$ . The transition graph  $G_L$  is a graph that shows all the possible sequences of the travel modes by which we can reach the destination node from the starting node. Each node in  $G_L$  represents an admissible travel mode. We begin with a single node corresponding to a dummy label  $s_0$  representing Stage 0. Then Stage 1 reflects the first admissible travel mode corresponding to the given mode string. Similarly, we designate the stepwise admissible modes for Stages 2, 3, ....

## Step 2

We now use the graph  $G_L$  and the Internal Network to solve for the shortest path problem as follows. In this implicit computation (where  $G^*$  is *implicitly* used), each Stage  $s$  will have augmented nodes of the type  $\{(i, t, l)\}$ : it is possible to come to node  $i$  at Stage  $s$  via an arc with label  $l$  at time  $t$ . We let  $N_s$  denote the set of all possible nodes  $(i, t, l)$  at Stage  $s$ . At the initial stage, we have  $N_0 = \{( \text{the starting node}, \text{the starting time}, s_0)\}$ . Then, we examine the Internal Network and the transition graph  $G_L$  and accordingly determine the set  $N_1$  along with the predecessor labels corresponding to the connections to the possible nodes in  $N_0$ . This continues until the last stage where the set  $N_S$  contains the destination node. Note that the sequences of links must satisfy the mode string, as ensured by examining  $G_L$  in this construction process. Moreover, since we are not interested in pursuing paths having a finish time that exceeds the maximum finish time (computed from the information specified for the single-trip request), we trim off nodes for which the arrival time exceeds the maximum finish time.

The following is an example that illustrates the basic concept of solving the TDLSP problem. Consider the Internal Network shown below, where the time-dependent travel time functions are given against the links along with the mode labels. Suppose that the admissible mode string is specified as  $\{w...wc...cw...w\}$ . Then, we can construct the transition graph  $G_L$  as depicted below.

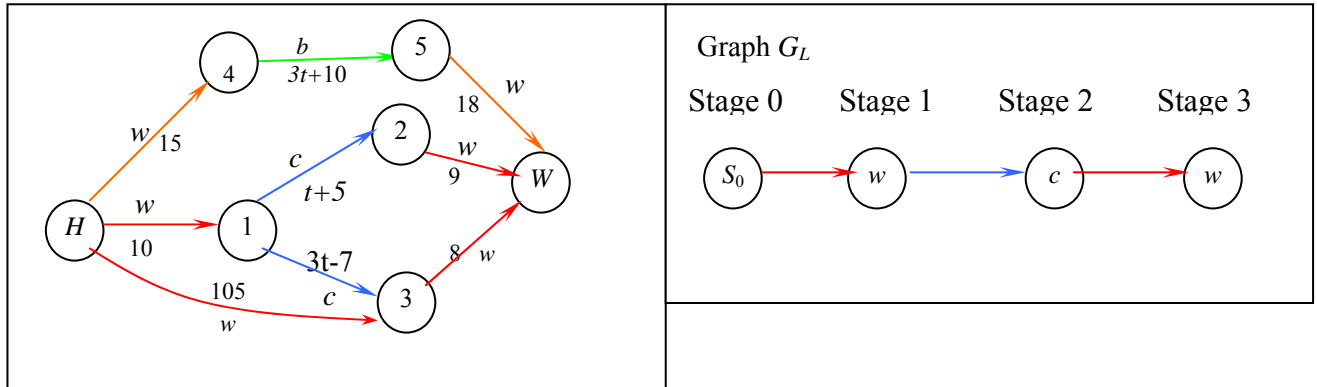
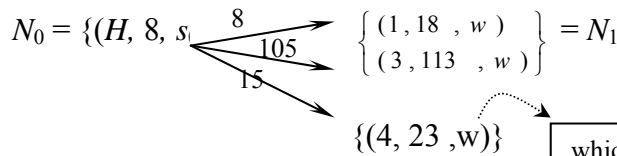


Figure-5.40: Internal Network (left) and Transition Graph ( $G_L$ ) (right) for Mode String  $wcw$ .

The set  $N_s$  for the initial stage  $s = 0$  is  $N_0 = \{(H, 8, s_0)\}$ .

The transitions from the set  $N_0$  to the set  $N_1$ , yields

$s = 1$

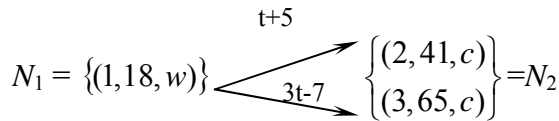


Obtained from the (starting time) + (the travel time) =  $(8+10) = 18$

which is infeasible to consider any further because it is connected to a bus node.

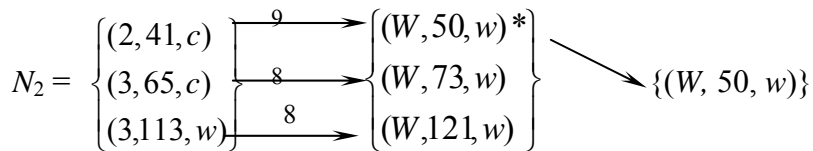
Then at Stage 2, we consider *admissible* transitions from the set  $N_1$  in order to determine the set  $N_2$ , as follows:

$s = 2$



This process continues for the remaining stage,  $s_3$  as follows:

$s = 3$



Terminate with destination node =  $W$ , and with the node  $(W, 50, w)$  as the terminal node of the shortest path. Tracing backwards yields the path  $H \rightarrow 1 \rightarrow 2 \rightarrow W$  (as shown below), having a travel mode  $wcw$  and an ending time of 50.

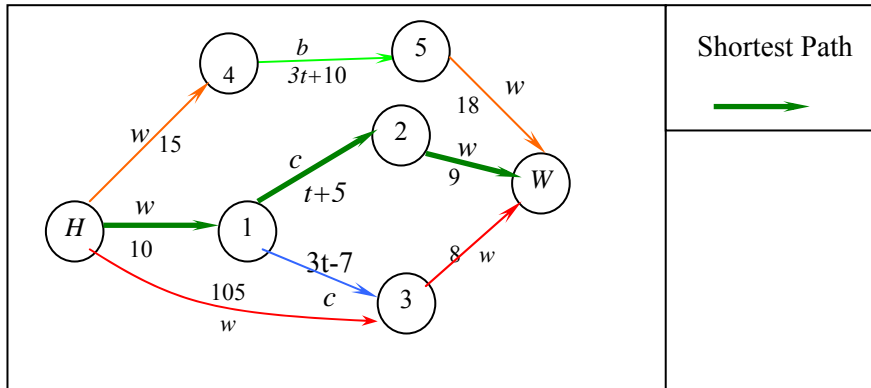


Figure-5.41: The Shortest Path.

An overall practical example in which TDLSP is embedded, is presented next below.

## 5.9 Example of the Route Planner Module

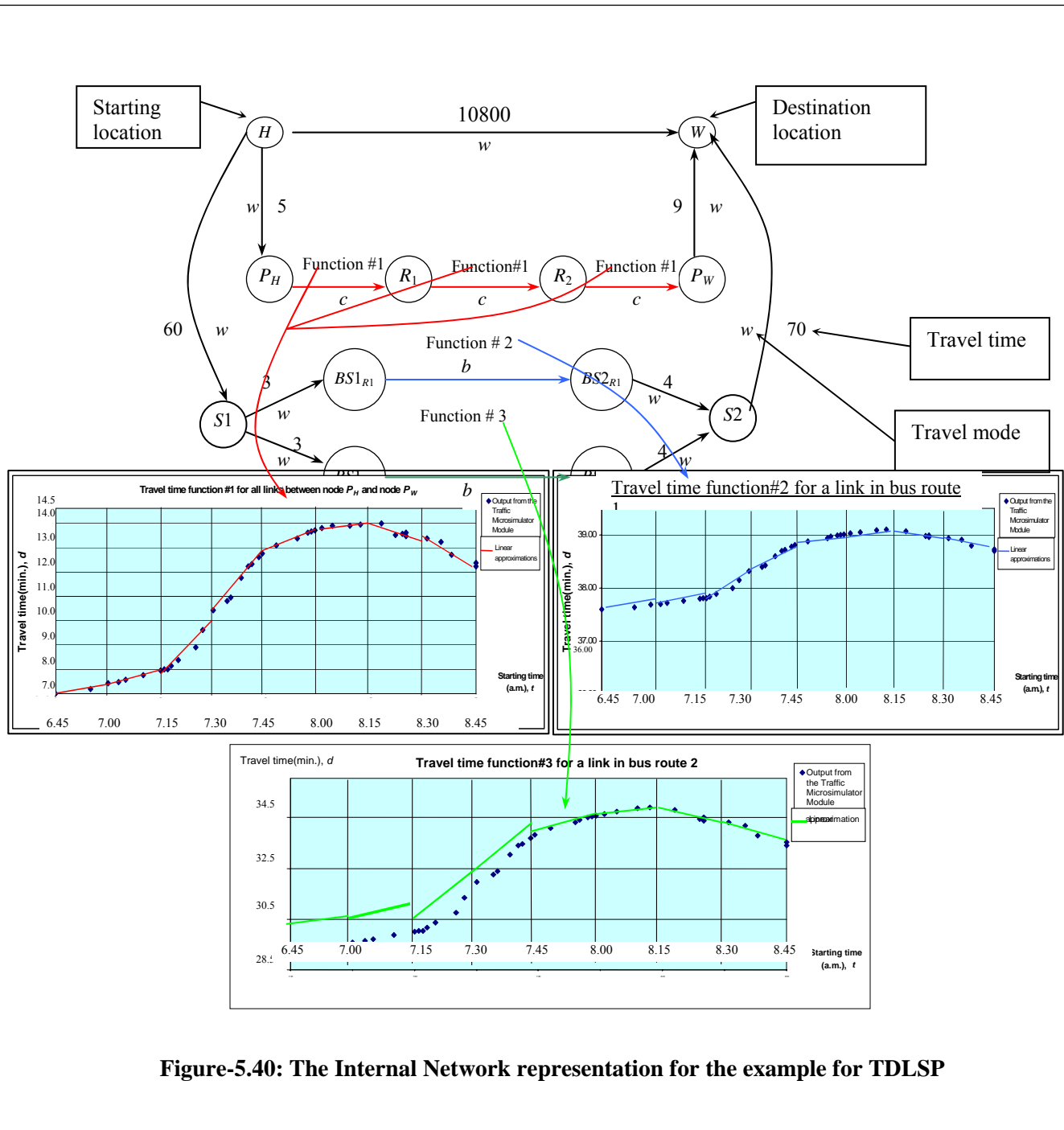
In this example we are using the same single-trip request provided in the earlier example (as shown in Table-5.31) for the traveler designated ID 13300, with admissible mode string “w...wc...cw...w” or “w...wb...bw...w”.

Table-5.31: Single-trip requests for a traveler

Person ID	Trip Number	Starting Location	Destination Location	Starting Time (seconds since midnight)	Maximum Travel Time (second)	Mode String
13300	1	845654 (Home)	833503 (Work)	28800 (8.00 a.m.).	3000 *	wcw or wbw

\*Note: the maximum travel time is 3000 seconds, hence, the maximum **finish time** equals the starting time plus the maximum travel time, which is 31800 seconds since midnight ( 8.50 a.m.).

Also, we are using the same Internal Network as shown in Figure-5.39, except that the links have time-dependent travel time functions  $d_{ij}(t)$ , excluding the walk links, which are time-independent.



**Figure-5.40: The Internal Network representation for the example for TDLSP**

In Figure-5.40, the starting location of the trip is node  $H$  (home), and the destination location is node  $W$  (work location). Figure-5.41 depicts the layers of the Internal Network, where the links have time-dependent travel time functions.

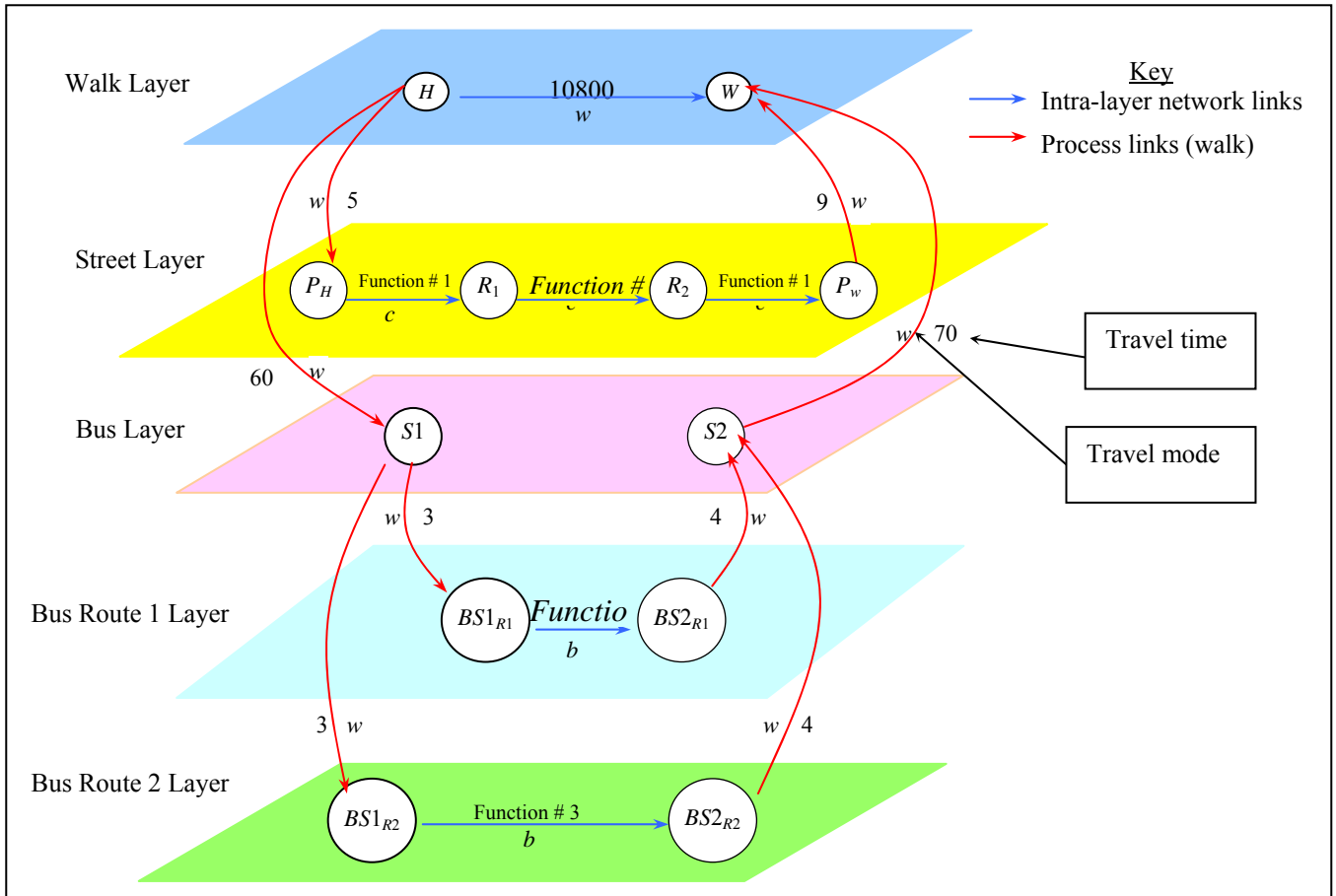
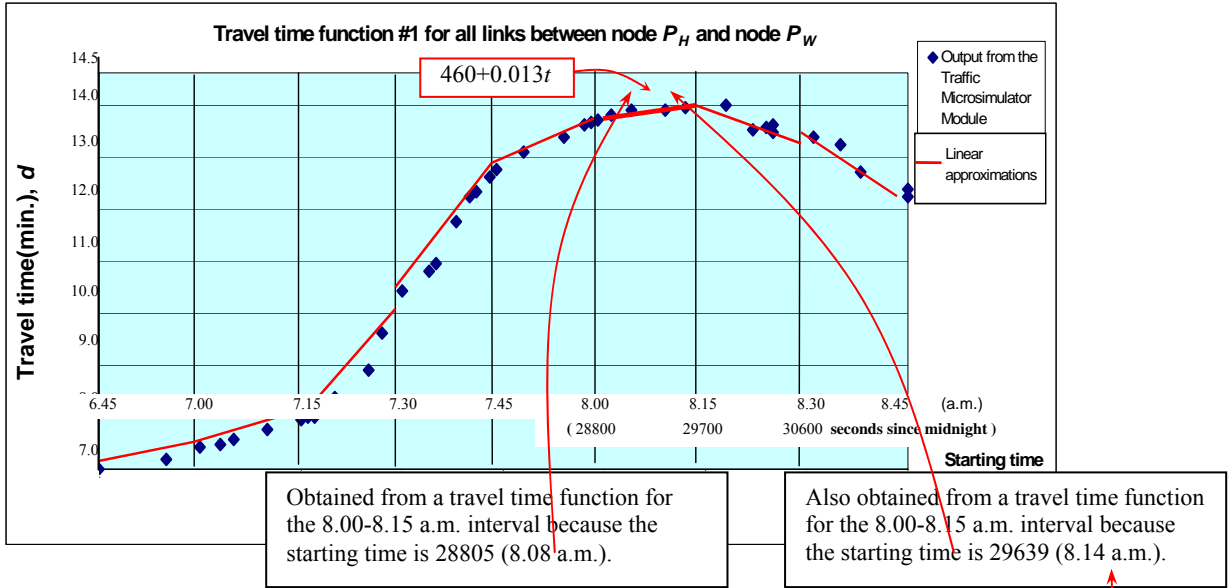


Figure-5. 41: Layers of the Internal Network for the example for TDLSP

The same procedure is used in developing the graph  $G_L$  in TDLSP as described earlier in TILSP except that the links have travel time values dependent on the arrival time at the starting node of that link. To avoid repetition of the procedure for determining graph  $G_L$ , we skip directly to the solution of the actual example.

Initialization:

$$\underline{s = 0} \quad N_0 = \{(H, 28800, s_0)\} \begin{cases} \xrightarrow{10800} (W, 39600, w) \\ \xrightarrow{5} (P_H, 28805, w) \\ \xrightarrow{60} (S1, 28860, w) \end{cases} = N_1$$



$s = 1$

$$N_1 = \left\{ \begin{matrix} (P_H, 28805, w) \\ (S1, 28860, w) \end{matrix} \right\} \begin{matrix} \xrightarrow{460+0.013t} \\ \xrightarrow{3} \\ \xrightarrow{3} \end{matrix} \left\{ \begin{matrix} (R_1, 29639, c) \\ (BS1_{R_1}, 28863, w) \\ (BS1_{R_2}, 28863, w) \end{matrix} \right\} = N_2$$

$s = 2$

$$N_2 = \left\{ \begin{matrix} (R_1, 29639, c) \\ (BS1_{R_1}, 28863, w) \\ (BS1_{R_2}, 28863, w) \end{matrix} \right\} \begin{matrix} \xrightarrow{460+0.013t} \\ \xrightarrow{900+0.05t} \\ \xrightarrow{915+0.04t} \end{matrix} \left\{ \begin{matrix} (R_2, 30484, c) \\ (BS2_{R_1}, 31206, b) \\ (BS2_{R_2}, 30933, b) \end{matrix} \right\}$$

Obtained from a travel time function for the 8.00-8.15 a.m. interval because the starting time is 28863 (8.01 a.m.).

Obtained from a travel time function for the 8.00-8.15 a.m. interval because the starting time is 28863 (8.01 a.m.).

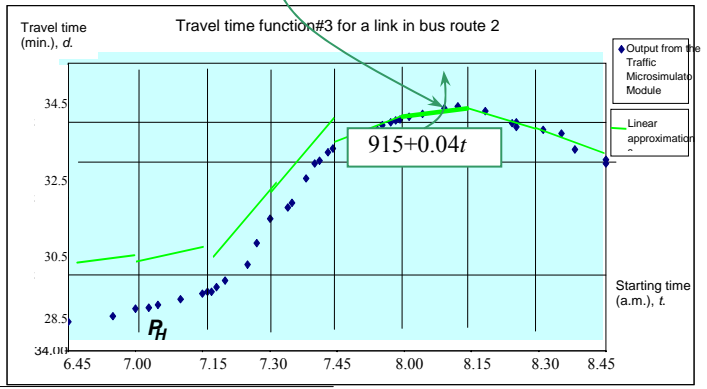
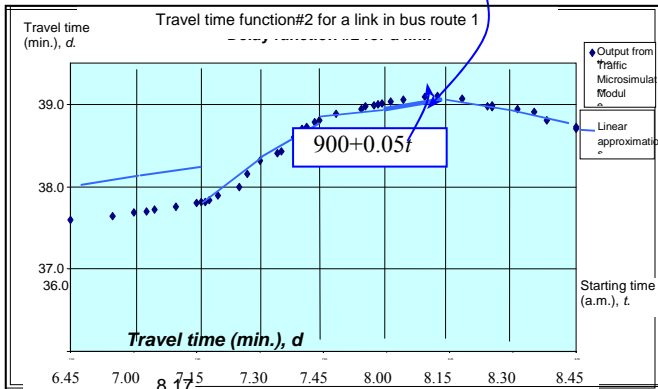
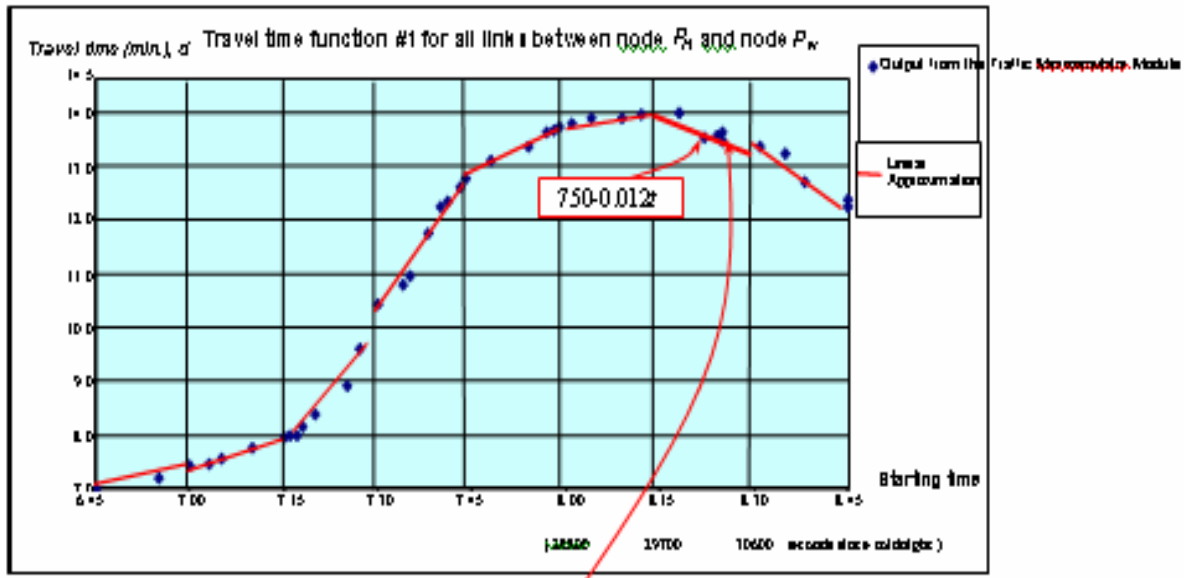
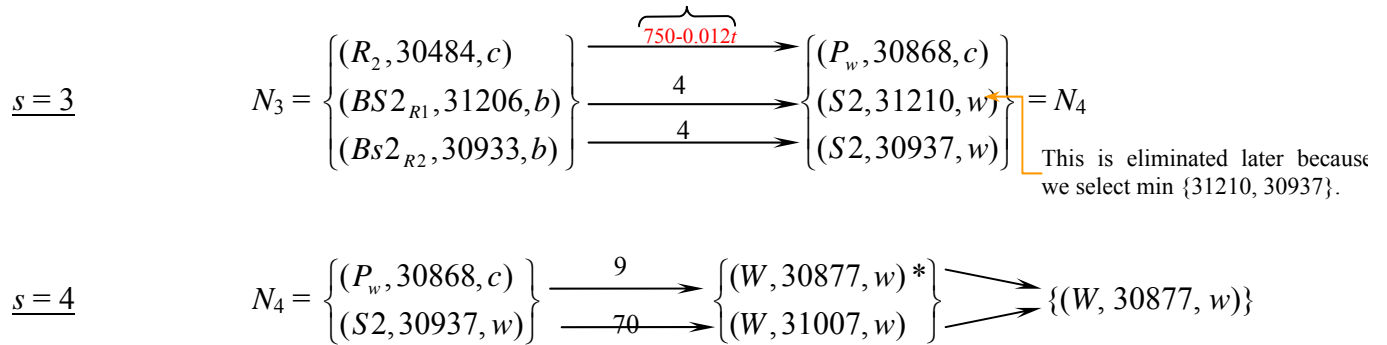


Figure-5.42: Travel Time Functions for All Links between Node  $P_H$  and  $P_W$

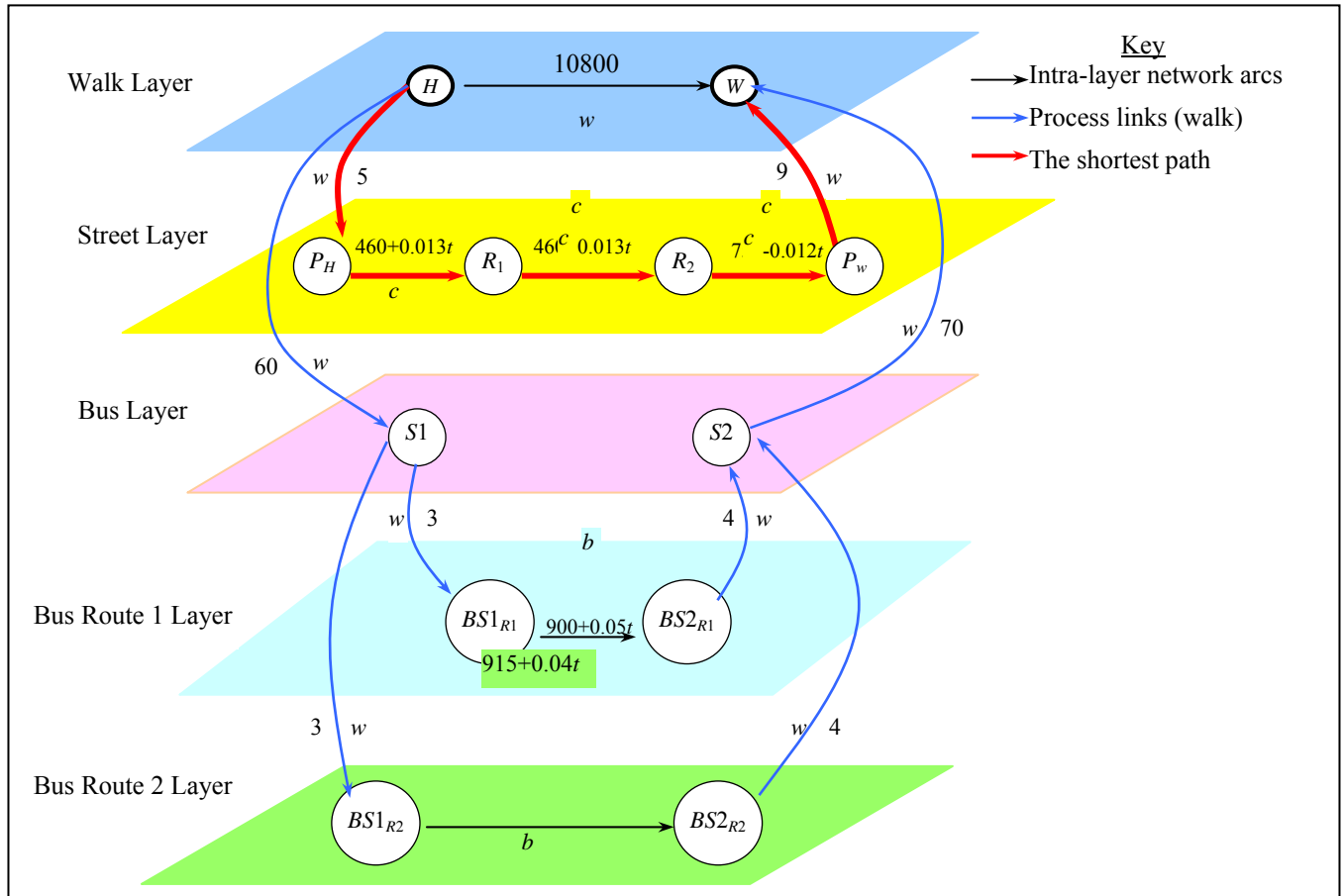


Obtained from a travel time function for the 8.15-8.30 a.m. interval because the starting time is 30484 (8.28 a.m.).

Figure-5.43: Travel Time Function #1 for All Links between Node P<sub>H</sub> and P<sub>W</sub>



s = 5 Terminate with *Destination node* = W, and with the node (W, 30877, w) as the terminal node of the shortest path. Note that there are no paths that exceed the maximum finish time (31800 seconds since midnight). Tracing backwards yields the path H → P<sub>H</sub> → R<sub>1</sub> → R<sub>2</sub> → P<sub>W</sub> → W (as shown in Figure-5.44), having a travel mode wccw and an ending time of t\* = 30877 seconds since midnight, or 8.35 a.m. The total travel time is (30877 – 28800) = 2077 seconds, which is less than the maximum allowable travel time of 3000 seconds.



**Figure-5.44: The shortest path solution for the example for TDLSP**

The shortest path has three legs:

- The first leg is a link between node  $H$  (home) to node  $P_H$  (car parking at home).
- The second leg is comprised of the links between node  $P_H$  and node  $R_1$  (road 1), between node  $R_1$  and node  $R_2$  (road 2), and between node  $R_2$  and node  $P_w$  (car parking at work).
- The last leg is a link between node  $P_w$  and node  $W$  (work).



The output would be formatted as follows.

**For the first leg**

```

13300 0 1 1
28800 845654 1 735654 2
5 31800 1 0 0
0 2
0

```

Annotations:

- Home ID points to 735654
- ID for car parking at home points to 735654
- Walk mode points to 2

**For the second leg**

```

13300 0 1 2
28805 735654 2 7335034 2
2063 31800 1 0 0
1 0
4
13217 0
55654 35653

```

Annotations:

- ID for car parking at work points to 7335034
- Car mode points to 0
- Number of tokens. points to 4
- Car ID 13217, and an indicator (0) that there is no other passenger joining this trip. points to 13217 and 0
- ID for node 1. points to 55654
- ID for node 2. points to 35653

**For the third (final) leg**

```

13300 0 1 3
30868 7335034 2 833503 1
9 31800 1 0 0
0 2
0

```

Annotations:

- Work ID points to 833503

# APPENDIX A

## Time-Dependent Label Constrained Shortest Path Problems (Route Planner Module)

(Adaptation and extension of Section 5.1 of Barrett, Jacob, and Marathe (1998) on regular expression constrained shortest path problems, where the expressions are specified in terms of a nondeterministic finite automaton (NFA).)

### 1. Some Basic Definitions

The following are definitions of certain key computer science terminology used in the TRANSIMS documentation and in Barrett et al. (1998). For the purpose of our discussion and development, the items 1, 2, 3, and 10 below suffice.

1. **Alphabet  $\Sigma$ :** collection of symbols (such as letters).
2.  **$\Sigma^*$ :** collection of all possible finite strings of alphabets.
3. **Language  $L \subseteq \Sigma^*$ :** collection of “words” or strings from  $\Sigma^*$  that are acceptable according to some criteria or rules.

4. **Deterministic Finite Automaton (DFA):**  $(Q, \Sigma, \delta, q_o, F)$ , where

$Q$  = set of finite states for the system

$\Sigma$  = finite input alphabet set

$q_o \in Q$  = initial state

$F \subseteq Q$  = set of final possible states

$\delta : Q \times \Sigma \rightarrow Q$ : transition function that takes a (state, alphabet) combination, and accordingly, transforms to some other (perhaps the same) state.

**Note:** Sometimes  $\delta : Q \times \Sigma^* \rightarrow Q$  is used for more general purposes.

5. **String  $x \in \Sigma^*$  is Accepted by a Deterministic Finite Automaton  $M$ :**

This happens if  $\delta(q_o, x) = p$  for some  $p \in F$ , i.e., starting in the state  $q_o$ , under the operations implied by the string  $x$ , one will transition ultimately to a desired final state.

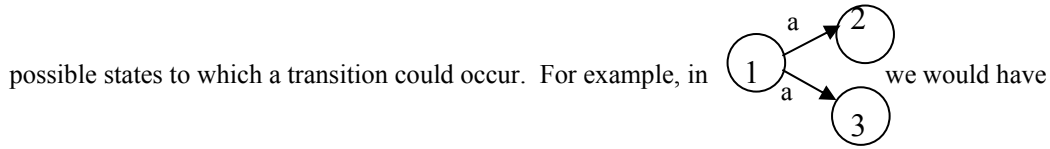
6. **Language Accepted by a Deterministic Finite Automaton  $M$ :  $L(M)$**

$L(M) = \{x \in \Sigma^* : \delta(q_o, x) \in F\}$ , i.e., this is the set of words  $x$  for which a transition from the initial state to some final state is possible.

7. **Regular Language  $L$ :** A language for which there is some deterministic finite automaton  $M$  for which

$L \equiv L(M)$ .

8. **Regular Expressions:** Class of regular languages, i.e., languages accepted by the collection of finite automata.
9. **Nondeterministic Finite Automaton (NFA):** One in which the transition function is a *point-to-set* map, i.e., given a state  $q$  and a label or alphabet  $a$ ,  $\delta(q, a)$  might be a set of



$\delta(1, a) = \{2, 3\}$ . Hence, in an NFA, we have

$$\delta : Q \times \Sigma \rightarrow 2^Q, \text{ the power set of } Q \text{ (set of all subsets of } Q\text{)}.$$

As before, we can generalize  $\delta$  to

$$\delta : 2^Q \times \Sigma^* \rightarrow 2^Q, \text{ where } \delta(P, x) = \bigcup_{q \in P} \delta(q, x).$$

**Note:** Any NFA can be represented by an *equivalent* DFA by allowing the states of the corresponding DFA to be *sets* of the states defined for the NFA.

10. **Graphs: Forward and Reverse Stars:** We assume standard terminology for graphs (see Bazaraa, Jarvis, Sherali (1990), for example). In particular, if  $G(N, A)$  is a graph having a node set  $N$  and an arc set  $A$ , then the forward star  $FS(i)$  for any  $i \in N$  is defined as  $FS(i) = \{j : (i, j) \in A\}$ , and the reverse star  $RS(i)$  for any  $i \in N$  is given by  $RS(i) = \{j : (j, i) \in A\}$ .

## 2. Time-Independent Label-Constrained Shortest Path Problem (TILSP)

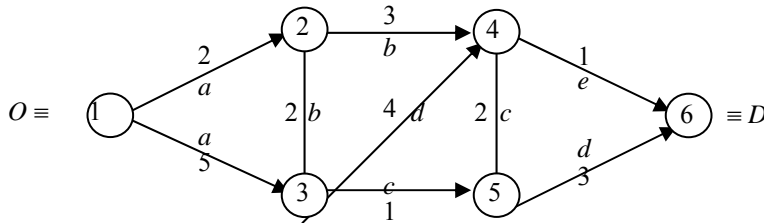
Let us begin by considering the time-independent version of this problem. Suppose that we are given a digraph  $G(N, A)$  where  $N$  and  $A$  are the sets of nodes and arcs of  $G$ , respectively. Let  $O \in N$  be the origin (or starting) node, and let  $D \in N$  be the destination (or final) node. Without loss of generality, assume that the network has been preprocessed such that  $RS(O) = \emptyset$  and  $FS(D) = \emptyset$ , where  $FS(\cdot)$  and  $RS(\cdot)$  respectively denote the forward and reverse stars of any node  $(\cdot)$ . Furthermore, assume that by successively scanning the sets  $FS(\cdot)$  starting at  $O$  we find all the nodes that are reachable from  $S$ , and by successively scanning the sets  $RS(\cdot)$  starting at  $D$  we find all the nodes that can reach  $D$ . Accordingly, we can then assume that  $N$  is comprised only of nodes in the intersection of these two sets, with  $A$  being the associated connecting arcs. Note that this reduction is for algorithmic convenience/efficiency, but not necessary for its operation/application.

In addition to the (constant) delays or travel times  $d_{pq}$  specified on the arcs  $(p, q) \in A$ , suppose further that each arc is also ascribed a label taken from some alphabet  $\Sigma$ , and that we are given a language  $L(R)$  (or simply  $L$ ) defined on a regular expression  $R$ . That is, the set  $L$  is comprised of *words*, or sequences of alphabets, that constitute acceptable sequences of labels on any selected path  $P \in \wp = \{\text{Paths in } G \text{ from node } O \text{ to node } D\}$ . Hence, if  $l(P)$  denotes the word formed by the sequence of labels on the arcs in a path  $P \in \wp$ , the *Time-Independent Label-Constrained Shortest Path Problem (TILSP)* is to find a shortest path  $P^*$  from  $O$  to  $D$  from among all paths in the set  $\wp \cap \{P : l(P) \in L\}$ .

We will first focus on the basic concepts, and subsequently, we will prescribe an efficient algorithm in which the detailed steps and constructs described below are only *implicitly* conceptualized within the implementation. Toward this end, consider the following example.

**Example 1.**

Given a graph  $G$  having arc delays  $d_{pq} \forall (p, q) \in A$  and labels as shown below, suppose that we are required to find a shortest path (not necessarily simple) from node  $O = 1$  to node  $D = 6$  subject to the constraint that the corresponding label sequence (word) should belong to the language  $L$ , where  $L$  is given by one of the following cases: (i)  $L \equiv \{abcd\}$ , or (ii)  $L \equiv \{abcd, abde\}$ , or (iii)  $L \equiv \{abcd, abe\}$ .



**Step 1.** Let node  $O \equiv 1$  be the *state* at *Stage 0*, and define *states* at *Stage s* to be nodes reachable from node 1 in  $s$  steps. Hence, we have,

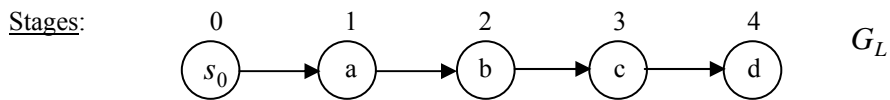
		Stages						
		0	1	2	3	4	5	6
States (nodes)	1		2	3	4	5	6	0
	3			4	5	6		
	5				6			

(**Note:** This need not be pre-generated; as mentioned above, this is only for conceptual purposes at this point, and the relevant information will be utilized implicitly during the solution process developed in Section 3.)

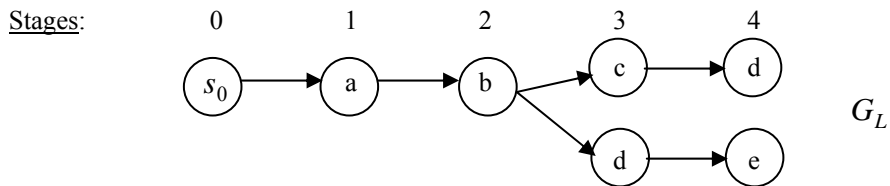
**Step 2.** Examining the admissible labels in  $L$ , construct a corresponding *transition graph*  $G_L$  as follows. Begin with a single node corresponding to a *dummy label*  $s_0$  at Stage 0. Then, recursively for stages  $s = 1, 2, \dots$ , given the graph up to Stage  $s-1$ , extend it to Stage  $s$  by performing the following constructions. Let the nodes of  $G_L$  at Stage  $s$  correspond to the possible distinct labels on the arcs via which we are permitted to reach a node/state in  $G$  at this Stage  $s$ . Next, if the language  $L$  permits an arrival via a label  $l'$  arc in  $G$  at Stage  $s-1$  followed by a traversal of an arc having a label  $\lambda$  to a node in  $G$  at Stage  $s$ , then introduce the arc  $(l', \lambda)$  in  $G_L$  between the corresponding nodes  $l'$  and  $\lambda$  at stages  $s-1$  and  $s$ , respectively, in  $G_L$ . Note that we assume that the language  $L$  contains only acceptable words, i.e., strings that define possible sequences by which we can reach node  $D$  from node  $O$  in that many steps. Furthermore, we assume that  $G_L$  exists such that all chains in  $G_L$  are admissible with respect to  $L$ .

For the three cases of  $L$  specified above, we have the following corresponding transition graphs  $G_L$

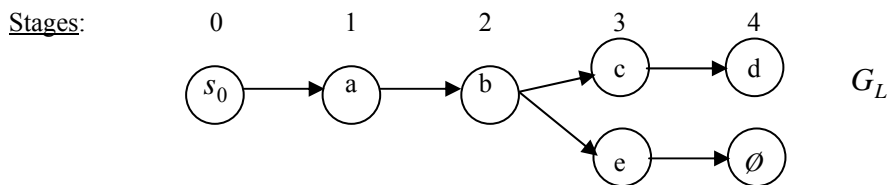
**Case (i)**



**Case (ii)**



**Case (iii)**



Note that as in Case (iii), if the permissible strings or words in  $L$  are of unequal lengths, we use a sequence of “empty” label symbols to extend the shorter words, so that all words in  $L$  become of the same length. Let the *number of stages*  $S$  to be considered equal the length of the maximum admissible string in  $L$  ( $S = 4$  here).

**Step 3.** Using the graphs  $G$  and  $G_L$  (conceptually), construct a graph  $G^*$  having node  $(O, s_0)$  at Stage 0, and having the following nodes for each stage  $s, s = 1, \dots, S$ :

$$\{(i, \lambda) : \text{it is possible to come to node } i \text{ at Stage } s \text{ via an arc with label } \lambda\}$$

where we also have

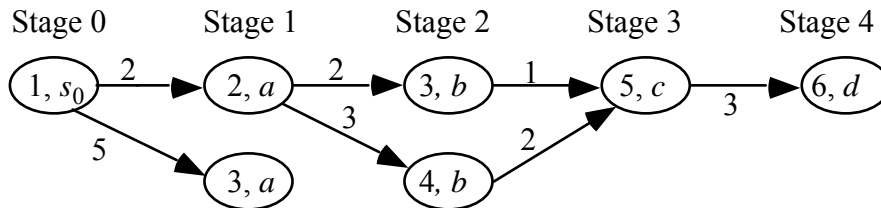
(a) for Stage  $S$ , only nodes  $(i, 1)$  with  $i$  equal to the terminal node  $D$  are admissible, and

(b) the node  $(i, \emptyset)$  is permitted only for  $i$  equal to the terminal node  $D$ . (In our example, node  $D = 6$  is the terminal node.)

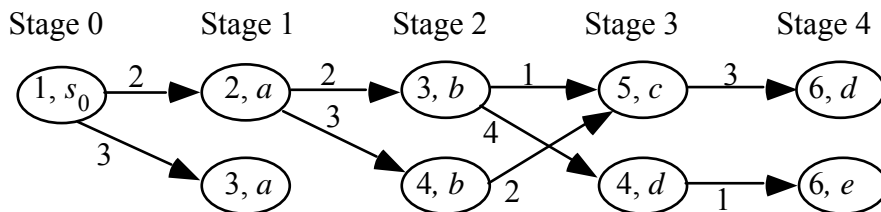
Next, progressing along the stages in order  $s = 1, \dots, S$ , for each stage  $s \in \{1, \dots, S\}$ , consider each node  $(i, 1)$  belonging to this stage, and construct an arc from  $(i', 1')$  at Stage  $s-1$  to this node  $(i, 1)$  having a cost  $d_{i'i}$  if (a) we can come to node  $i$  from node  $i'$  in  $G$  with arc  $(i', i)$  in  $G$  having a label  $\lambda$ , and (b) the arc  $(1', 1)$  exists in  $G_L$  from Stage  $s-1$  to Stage  $s$ . (Note that if  $1 \equiv \emptyset$ , then we must have  $i' \equiv i \equiv D$ , and we take  $d_{i'i} \equiv 0$ .)

For the above three cases, the graph  $G^*$  can be constructed as follows.

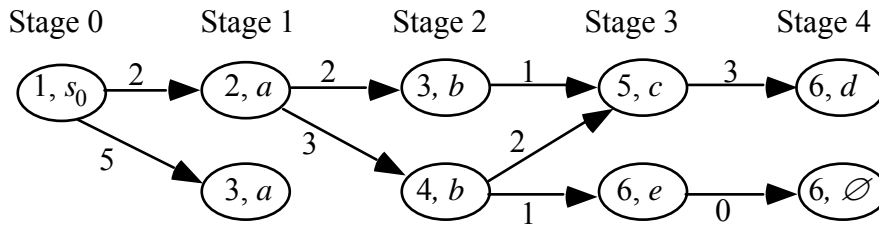
**Case (i)  $G^*$ :**



**Case (ii)  $G^*$ :**



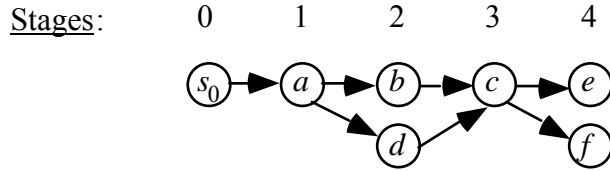
**Case (iii)  $G^*$ :**



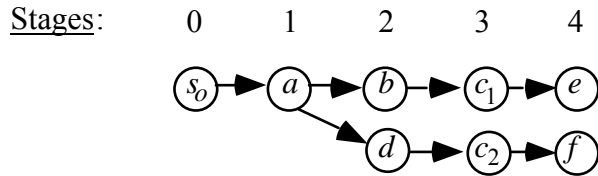
**Step 4.** Find the shortest path from node  $(0, s_0)$  to all the terminal nodes at Stage  $S$  using any standard SP algorithm.

Pick the shortest of these paths and trace the corresponding path and the labels using a backward pass (using the DOWN or predecessor labels in the usual fashion).

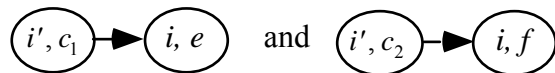
**Remark 1.** Note the importance of asserting the existence of the graph  $G_L$  without a duplication of nodes at each stage in  $G_L$ . For example, if  $L = \{abce, adcf\}$  for some graph  $G$ , then if we use the particular graph shown below as  $G_L$ , it would also imply the admissibility of the crossover strings  $abcf$  and  $adce$  in  $L$ .



In order to avoid this inclusion of inadmissible strings, we would need to define a graph  $G_L$  as follows:



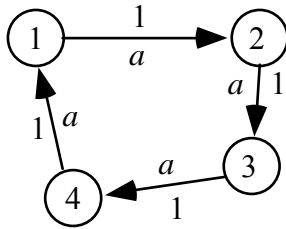
where  $c_1$  and  $c_2$  are duplicates of the label  $c$  at Stage 3. Then, the node and arc definition process for  $G^*$  above would need to be modified to accommodate nodes of the type  $(i, l)$  with  $l = c_1$  or  $c_2$  at Stage 3, along with the corresponding arc connections. For example, at Stage 4, we would consider connections of the following type.



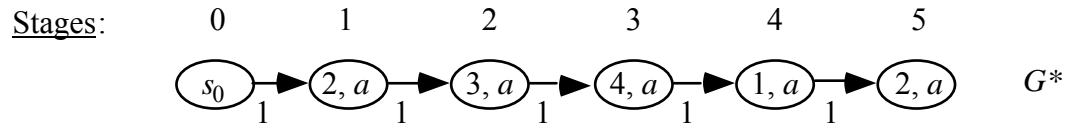
Henceforth, we assume that the label and language definitions conform with the definition of a suitable graph  $G_L$  for which the paths in  $G_L$  correspond to admissible words in  $L$ , and vice versa.  $\square$

Figure 1 presents a flow-chart summarizing the essential steps of a rudimentary algorithm (including an efficient construction of  $G^*$ ) for solving SLCSP.

**Remark 2.** As Barrett et al. point out, a label constrained SP can be *nonsimple*. (The problem of finding shortest *simple* label constrained paths, if they exist, is NP-hard.) For example, consider the graph  $G$  given below, with  $L = \{aaaaa\}$ , and with nodes 1 and 2 as the starting and terminal nodes  $O$  and  $D$ , respectively. The shortest label constrained path is  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 2$ , which is nonsimple.



This path, however, comes from the following shortest *simple* path in  $G^*$ .



Note how nodes can repeat at different stages (e.g. node  $(2, a)$  appears at stages 1 and 5).  $\square$

### 3. Time-Dependent Label-Constrained Shortest Path Problem (TDLSP)

In contrast with the former case of constant travel times, suppose now that the link delays are time-dependent functions.

Specifically, for each  $(p, q) \in A$ , suppose that we have a time-dependent link delay function  $d_{pq}(t)$  that specifies the travel time on link  $(p, q)$ , given a starting time  $t$  at the corresponding tail node  $t$ . This function  $d_{pq}$  might be a general real valued function defined on a continuum of time over some horizon interval  $H$  ( $d_{pq} : H \subseteq R \rightarrow R$ ), or it might be some discretized approximation from some experimental or simulation output analysis, being defined as

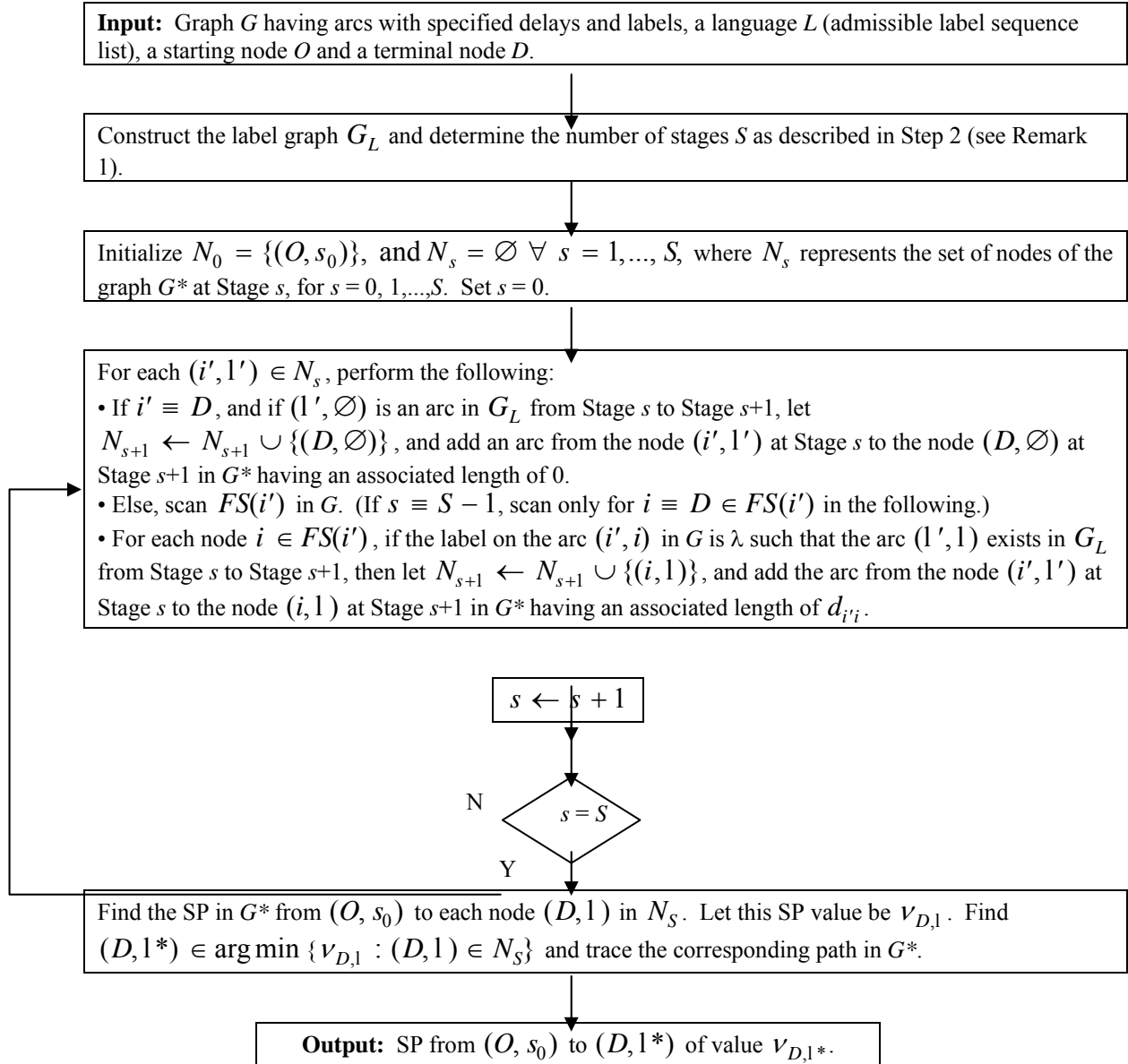
$$d_{pq} : H \equiv \{0, \Delta, 2\Delta, \dots, K\Delta\} \rightarrow \{0, \Delta, 2\Delta, \dots, K'\Delta\}$$



for some discretized time duration  $\Delta$  and integer  $K'$ , and some suitably large integer  $K$  such that the characterization of the delay function beyond the time  $K\Delta$  is not of practical interest.

In either case, let us define

$T$  = some upper bound on the length (total delay) of any acceptable path in the solution to the underlying problem.



**Figure 1.** Flow-Chart for a Rudimentary Procedure to Solve the TILSP Problem.

Note that we use this additional parameter  $T$  to control the degree of exploration of the network and to fathom or weed out the examination of paths that do not have delays below this threshold  $T$ . Also, it is well known that there exists a time-space static equivalent network representation for this problem. In this representation, each node is replicated as  $\{i, t\}$  for all possible values of  $t \in H$  when we could be at node  $i$ . Furthermore, for each  $(i, j) \in A$ , given a possible starting time  $t < T$  at node  $i$ , where  $t \in H$ , this representation would construct an arc  $\{(i, t), (j, t + d_{ij}(t))\}$  having a fixed delay of  $d_{ij}(t)$ , such that  $t + d_{ij}(t) \leq T$ , where note that  $t + d_{ij}(t)$  then also belongs to  $H$  by our assumption.

Similar to the time-independent case, we also have labels associated with each arc taken from some alphabet set  $\Sigma$ , and we are given a language  $L$  defined on regular expressions that is comprised of a set of acceptable words (sequence of arc labels on acceptable paths). Suppose further that as before, we are given (or have constructed) the corresponding graph  $G_L$ . The Time-Dependent Label-Constrained Shortest Path Problem (TDLSP) then is to find a time-dependent shortest path  $P^*$  in  $G$  from  $O$  to  $D$  among all paths  $P$  from  $O$  to  $D$  (i.e.,  $P \in \wp$ ) for which  $l(P) \in L$ , where  $l(P)$  is the sequence of alphabet labels on the path  $P$ .

The procedure we develop dynamically generates a reduced-size time-space network *implicitly*, using a minimal number of time-based node replications, while *simultaneously* finding the TDLSP from  $O$  to  $D$ . This is done within the framework of a dynamic programming routine that is similar in concept to the procedure of Figure 1. Figure 2 describes the proposed procedure. Here, time-expanded replicates  $(p, t, l)$  of each node  $p \in N$  are automatically created only for specific, necessary values of times  $t$ , and labels  $\lambda$  based on possible visitation times and label sequences.

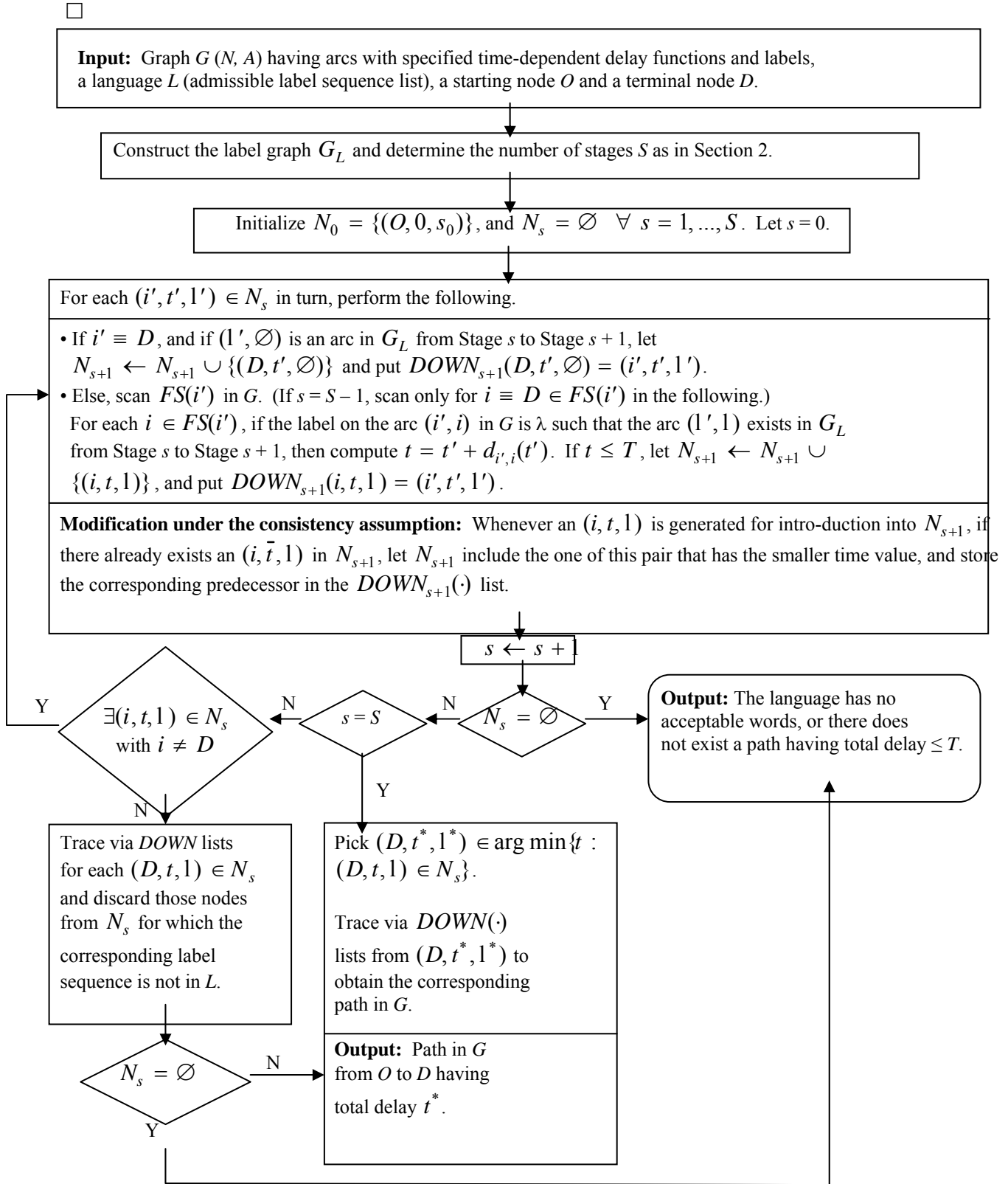
In this process, we develop the node sets  $N_s$  for  $s = 0, 1, \dots, S$  as in Figure 1, but maintain an additional time component for each node, along with the predecessor or  $DOWN_s(\cdot)$  list for each stage  $s$  in order to facilitate a back-tracing of the generated paths. Moreover, since we are not interested in pursuing paths having total delays that exceed  $T$ , and all delays are nonnegative, we trim off nodes for which the delay exceeds  $T$ .

Furthermore, as before, the nodes investigated for the final stage  $S$  correspond only to the terminal node  $D$ . Note that termination might occur prior to stage  $S$  either because no time-dynamic label constrained path is realizable that has a

total delay less than or equal to the specified limit  $T$ , or within such a limit, the nodes at some earlier stage all correspond to the terminal node  $D$ .

**Remark 3.** Observe that when we perform the operation  $N_{s+1} \leftarrow N_{s+1} \cup \{(i, t, 1)\}$ , if the node  $(i, t, 1)$  already exists in  $N_{s+1}$ , we revise its  $DOWN_{s+1}(\cdot)$  index as stated in the procedure; however, we could have left this index the same. Hence, only some alternative equally attractive choice of a partial path and label sequence is being maintained.  $\square$

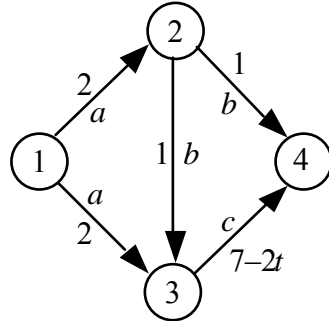
**Remark 4.** Note that the general procedure does not assume the *first-in-first-out* (FIFO) or *consistency* assumption whereby for each link  $(p, q) \in A$ , if we enter the link at an earlier time then we would also exit the link at a relatively earlier time (see Kaufman and Smith (1993)). However, when such an assumption holds true, we do not need to maintain duplicate nodes of the type  $(i, t_1, 1)$  and  $(i, t_2, 1)$  for  $t_1 < t_2$  at any stage; the latter node is dominated by the former and can be dropped from consideration. Figure 2 states this modified rule in the main processing block. In particular, note that for the time-independent case, we have  $d_{pq}(t) \equiv d_{pq} \forall t \in H, \quad \forall (p, q) \in A$ . Hence, the consistency assumption holds true, and the procedure of Figure 2 can be applied under this revision.



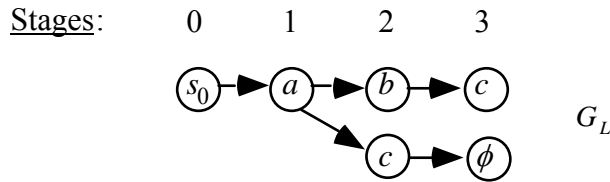
**Figure 2.** Flow-Chart for the Algorithm to Solve the TDLSP Problem.

**Example 2 (Inconsistent or Non-FIFO Delays)**

Consider the following graph  $G$  with delay functions and arc labels as shown.



Note that all the links have time-independent delays, except for link (3, 4) for which  $d_{34}(t) = 7 - 2t$ . Hence, if we arrive at node 3 at time  $t = 2$ , it would put us at node 4 at time  $2 + 3 = 5$ , while arriving at node 3 at a later time  $t = 3$ , would put us at node 4 at time  $3 + 1 = 4$ , i.e., earlier than in the former case. Suppose also that the specified language is  $L = \{abc, ac\}$ , so that the non-label constrained shortest path  $1 \rightarrow 2 \rightarrow 4$  is infeasible. Let us also assume that  $T \geq 5$ , and note that  $S = 3$ . The graph  $G_L$  is given as follows.



The algorithm of Figure 2 would proceed as follows, where the arrows depict the corresponding  $DOWN(\cdot)$  relationships established at each stage.

Initialization:

$$N_0 = \{(1, 0, s_0)\}, \text{ where } O \equiv 1, \text{ and } N_s = \emptyset \text{ for } s = 1, 2, 3.$$

$s = 0$ :

$$N_0 = \{(1, 0, s_0)\} \begin{cases} \rightarrow (2, 2, a) \\ \rightarrow (3, 2, a) \end{cases} = N_1$$

$s = 1:$

$$N_1 = \left\{ \begin{array}{l} (2, 2, a) \\ (3, 2, a) \end{array} \right\} \rightarrow \left\{ \begin{array}{l} (4, 3, b) \\ (3, 3, b) \\ (4, 5, c) \end{array} \right\} = N_2$$

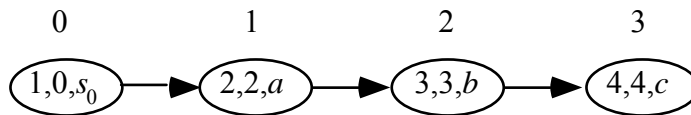
$s = 2:$

$$N_2 = \left\{ \begin{array}{l} (4, 3, b) \\ (3, 3, b) \\ (4, 5, c) \end{array} \right\} \rightarrow \left\{ \begin{array}{l} (4, 4, c) \\ (4, 5, \emptyset) \end{array} \right\} = N_3$$

$s = 3:$

With  $D \equiv 4$ , we identify  $(D, t^*, l^*)$  as  $(4, 4, c)$ . Tracing via  $DOWN(\cdot)$  produces the following path (in reverse order).

Stages:

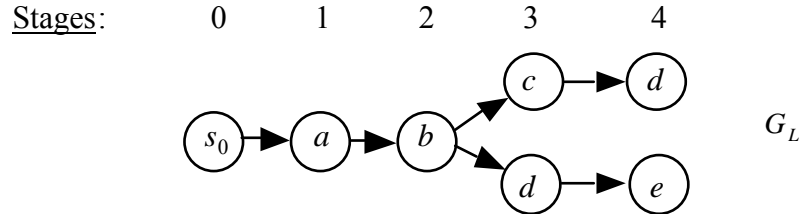


Hence, the path  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$  having a label sequence  $abc$  and total delay of  $t^* = 4$  solves the given TDLSP problem.

**Remark 5.** Note that if we had specified  $T = 4$ , then the node  $(4, 5, c) \in N_2$  would have been suppressed at Stage  $s = 1$ , and so,  $N_3$  would only have inherited  $(4, 4, c)$  at Stage  $s = 2$ . Alternatively, with  $T = 3$ ,  $N_2$  would again have dropped  $(4, 5, c)$  at  $s = 1$ , while we would have obtained  $N_3 = \emptyset$  at  $s = 2$ . The procedure would have terminated with an infeasibility indication. Decreasing  $T$  further would have yielded an infeasibility indication at an earlier stage.  $\square$

**Example 3. (Consistent or FIFO Delays):**

For the sake of illustrating the application of Figure 2 to solve a *time-independent* label constrained shortest path problem, consider Example 1 under the language specification of Case (ii), and assume that  $T = 9$ . The value of  $S$  is 4, and the graph  $G_L$  is shown below.



The procedure of Figure 2 would proceed as follows, noting the modification stated under the consistency assumption.

Initialization:  $N_0 = \{(1, 0, s_0)\}$ , where  $O \equiv 1$ , and  $N_s = \emptyset$  for  $s = 1, \dots, 4$ .

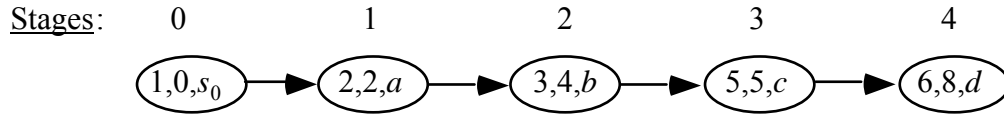
$s = 0$ :  $N_0 = \{(1, 0, s_0)\} \begin{cases} \rightarrow \{(2, 2, a)\} \\ \rightarrow \{(3, 5, a)\} \end{cases} = N_1$

$s = 1$ :  $N_1 = \begin{cases} \{(2, 2, a)\} \\ \{(3, 5, a)\} \end{cases} \begin{cases} \rightarrow \{(3, 4, b)\} \\ \rightarrow \{(4, 5, b)\} \end{cases} = N_2$

$s = 2$ :  $N_2 = \begin{cases} \{(3, 4, b)\} \\ \{(4, 5, b)\} \end{cases} \begin{cases} \rightarrow \{(5, 5, c)\} \\ \rightarrow \{(4, 8, d)\} \\ \rightarrow \{(5, 7, c)\} \leftarrow \text{eliminate} \end{cases} = N_3$

$s = 3$ :  $N_3 = \begin{cases} \{(5, 5, c)\} \\ \{(4, 8, d)\} \end{cases} \begin{cases} \rightarrow \{(6, 8, d)\} \\ \rightarrow \{(6, 9, e)\} \end{cases} = N_4$

$s = 4$ : Terminate with  $(D, t^*, l^*) = (6, 8, d)$ . Tracing backwards using the  $DOWN(\cdot)$  list yields



This yields the optimal TDLS  $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6$  in  $G$ , with the label sequence  $abcd$ , and having a total delay equal to  $t^* = 8$ .

**Remark 6 (Choice of  $T$  and Curtailing Computations).** If we know some feasible path to the TDLS problem, then using its delay as the value of  $T$ , we can reduce the extent of computations performed. This motivates the use of a quick heuristic to derive an initial feasible solution. Furthermore, in large-scale applications, we could partition the graph into sections depending on the choice of  $O$  and  $D$ , and require the nodes within each section to be visited by a certain time. Alternatively, we can specify a threshold value  $T_s$  for each stage  $s$ , and maintain only those  $(i, t, l) \in N_s$  for which  $t \leq T_s \forall s = 1, \dots, S$ . By suitably choosing a progression of  $T_s$ -values, we can curtail the computational effort, although this might mean a loss in finding an exact optimum.  $\square$

We close this section by establishing the complexity of the procedure in Figure 2.

**Proposition 1.** Given a graph  $G(N, A)$ , and given the graph  $G_L$  corresponding to the language  $L$ , let

$r =$  maximum admissible word length in  $L$  (longest simple path in  $G_L$ )

$m =$  maximum number of nodes at any stage in  $G_L$ .

Then under the consistency assumption, the procedure of Figure 2 is of polynomial-time complexity  $O(rm^3 | A |)$ .

**Proof.** The number of stages performed is  $O(r)$ . For each stage  $s$ , any node  $i$  of  $G$  appears in  $(i, t, l) \in N_s$  at most  $m$  times. Scanning the forward star of  $i$  and checking the at most  $m^2$  arcs in  $G_L$  from stage  $s$  to stage  $s + 1$  for each such repetition of  $i$  takes  $O(|FS(i)| m^3)$  time. Summing this over all possible nodes of  $G$  appearing at stage  $s$  gives a total complexity of  $O(m^3 | A |)$  per stage. Hence, the overall process is of complexity  $O(rm^3 | A |)$ , and this completes the proof.  $\square$



**Remark 7.** In the case of non-FIFO link delays, if  $\tau$  is the maximum number of distinct values of times for which it is possible to visit any node within the interval  $[0, T]$  (for integer valued delay functions, we can take  $\tau \equiv T$ ), then the complexity of the algorithm of Figure 2 is pseudopolynomial of order  $O(\tau m^3 | A |)$ .  $\square$

#### 4. Time-Dynamic Chained Activity Route Planner

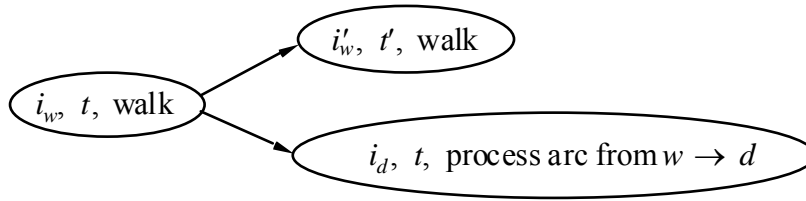
The key utility of the foregoing approach arises in the following context. Consider a certain traveler starting at a home location at a certain time ( $t = 0$ ), and wishing to go to the office via label strings constituting a language of the type  $L = \{(w\dots w, d\dots d, r\dots r, w\dots w), (w\dots w, d\dots d, b\dots b, w\dots w)\}$  where  $w$ ,  $d$ ,  $r$ , and  $b$  respectively represent walking, driving, rail (taking trains), and taking buses. The words can be of various lengths, so long as they are admissible with respect to the overall multi-modal transit network. This latter network can be conceptualized as a layered network as shown in Figure 3, where the starting node  $O$  is represented by the home location and the terminal node  $D$  is represented by the office location node  $D$  on the walking network, and where there exist various *process arcs* (shown dotted) between appropriate possible and desirable location connections from one layer to another.



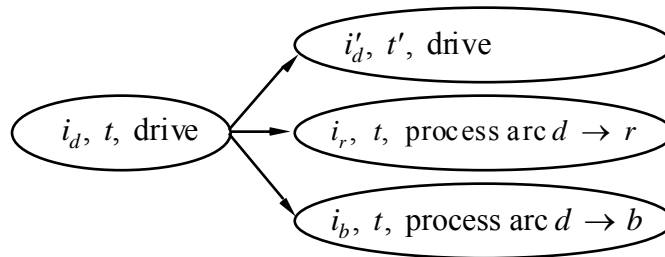
**Figure 3:** Layered Multi-Modal Network.

We can now apply the algorithm of Figure 2 to this network under time-independent or time-dependent conditions, and in the latter case, under a non-FIFO or FIFO (consistency) assumption. For example, given a node  $(i_w, t, \text{walk})$  at some stage  $s$ , representing that we have reached location  $i$  at time  $t$  while walking (in the walking network), we might have

possible connecting arcs of the following type leading to states in stage  $s + 1$ , where  $i_w$  and  $i_d$  respectively represent location  $i$  in the walking and driving networks.



Similarly, in the drive network, we might have the following types of connections from one stage to the next.



The key element here is an efficient implementation of the procedure of Figure 2, using suitable node-set reduction schemes as described in Remark 6.

**Remark 8.** Note that the algorithm discussed thus far assumes that the time delay functions  $d_{ij}(t)$  are known for each link  $(i, j)$  in each of the layered networks. Such information might be gleaned experimentally or via a dynamic traffic assignment simulation process executed using some OD trip matrices. These delay values for each link can be stored in look-up tables (e.g. as estimated travel times on an hourly or half-hourly basis) or be input as analytical statistically determined functions. We can also assume that the FIFO or consistency condition holds, which is true for most practical cases.  $\square$

## REFERENCES

1. Mokhtar S. Bazaraa, Hanif D. Sherali, and John J. Jarvis. *Linear Programming and Network Flows, Second Edition*. John Wiley and Sons, New York, NY (1990).
2. Ravindra K. Ahuja, Thimas L. Magnanti, and James B. Orlin. *Network Flows Theory, Algorithms, and Applications*. Prentice Hall, New Jersey (1993).
3. Frederick S. Hillier, and Gerald J. Lieberman. *Introduction to Operations Research, Fourth Edition*. Holden-Day, CA (1986).
4. John R. Canada, William G. Sullivan, and John A. White. *Capital Investment Analysis for Engineering and Management, Second Edition*. Prentice Hall, New Jersey (1996).
5. Hanif D. Sherali. Time-Dependent Label Constrained Shortest Path Problems (Route Planner Module). *Notes*. Virginia Polytechnic Institute and State University, Blacksburg, VA (2000).
6. Hanif D. Sherali. On the Equivalence between Some Shortest Path Algorithms. *Operations Research Letters*. Vol. 10 (1991) 61-65.
7. Hanif D. Sherali, Kaan Ozbay, and Shivaram Subramanian. The Time-Dependent Shortest Pair of Disjoint Paths Problem: Complexity, Models, and Algorithms. *Networks*. Vol. 31 (1998), 259-272.
8. Los Alamos National Laboratory. TRansportation ANalysis SIMulation System (TRANSIMS) Version: TRANSIMS-LANL-1.0. NM (1999).
9. Los Alamos National Laboratory. TRansportation ANalysis SIMulation System (TRANSIMS) Version: TRANSIMS-LANL-1.1. NM (2000).
10. Kai Nagel, Riko Jacob, and Marathe Madhav. A Computational Study of Routing Algorithms for Realistic Transportation Networks. *TRANSIMS Report Series*. NM (1998).
11. Giorgio Gallo, and Stefano Pallottina. Shortest Path Algorithms. *Annals of Operations Research*. Vol. 13 (1988), 3-79.
12. Shivaram Subramanian. Routing Algorithms for Dynamic, Intelligent Transportation Networks. *Master's Thesis*. Virginia Polytechnic Institute and State University, Blacksburg, VA (1997).
13. Chris Barrett, Riko Jacob, and Marathe Madhav. Formal Language Constrained Path Problems. Los Alamos National Laboratory, NM (1997), 234-243.
14. Ariel Orda, and Raphael Rom. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the Association for Computing Machinery*. Vol. 37 (1990), 607-625.