CHAPTER 7

COMPUTATIONAL RESULTS

In this section, we test the performance of our algorithm for solving the time-dependent label-constrained shortest path problem (as specified in Figure 5) and the heuristic methods of Section 6. We use the C++ programming language to conduct our implementation. The test runs were made on a 450 MHz Pentium II with 128 MB of RAM, 4.02 GB of hard drive capacity computer. The transportation network used is called the Bignet network. This network is a part of TRANSIMS's test network, and was created to model a portion of the transportation system in Portland, Oregon (TRANSIMS, version 1.1, 2000). The network contains approximately 25,000 households, 3,853 nodes, 7,441 links within an area of 18×18 square kilometers. Because of computational limitation, we were able to use at most 1,000 nodes, 1,900 links within the same region. The travel modes considered within this network are *walk*, *bus*, *rail-transit*, and *car*. The various problem instances generated are transportation trips within the network, and are specified by their starting locations, destination locations, starting times, maximum finish times, and travel modes. These instances were obtained from the Portland, Oregon, Activity and Travel Survey of 1994/95. The portion of the survey we used is comprised of 1,000 households for a total of 2,258 individuals, resulting in 4,516 transportation activities. Therefore, we have 4,516 time-dependent label-constrained shortest path problems to be solved.

Below, we provide a description of the test network and our assumptions, along with a discussion on design objectives, design evaluations, and design parameters. We then provide computational results and analyses for the aforementioned procedures, and compare the performance of the exact approach versus the various heuristic methods.

7.1 Test Network Description and Necessary Assumptions

Figure 26 shows the overall layout of the network, which is partitioned into nine zones based on land-use information. The network is comprised of five types of land-use areas, namely, the Heavy Commercial (downtown area), the Light Commercial, the Heavy Industrial, the Residential, and the Mixed Residential/Commercial areas. There are four bridges across the river that are located centrally within the network. Surrounding the downtown area (Zone 7) is a light-rail route, which extends over the northern area and along the northern side of the river. A freeway parallels the light rail route in the northern area.

There is one heavy industrial area (Zone 9). This area has no homes, but is the workplace for a significant fraction of the population. The downtown zone, similar to the heavy industrial zone, has no homes, but is the workplace for much of the population. In addition, there is a shopping and recreational destination that is used by a great segment of the population. Surrounding the downtown area at the northern and southern side of the river are two light commercial zones (Zones 3 and 6). These have the same features as the downtown area, except that the activities performed in these zones are far fewer. Covering most of the land areas are residential zones including the mixed residential/commercial zones (Zones 1, 2, 4, 5, and 8). Also, within the mixed residential/commercial zones (Zones 2, 4, 5, and 8) lie most of the schools in the network.

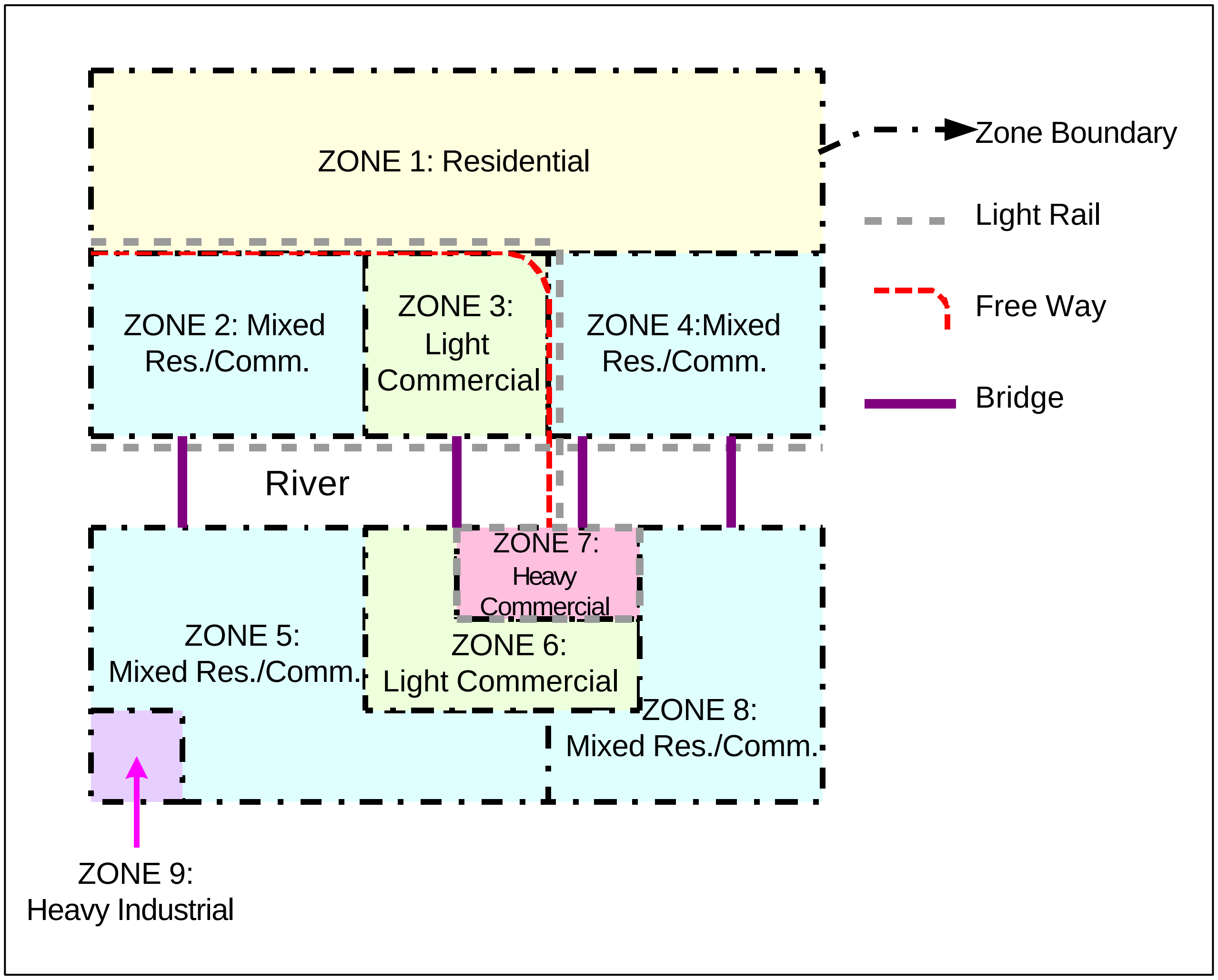


Figure 26: Land Use in the Bignet Network (*not to scale*).

The various notation used for describing the time-dependent label constrained shortest path problem (TDLSP) have the following connotation within the context of the above transportation test network.

Node: This is a physical location in the transportation network, such as a street intersection, activity location (identified as either a starting node or terminal node), household location, school location, work place, shopping mall, bus stop, rail stop, car parking, etc. The starting node and terminal node for each trip are obtained from the Activity and Travel Survey. Each node is ascribed a unique ID and (*x*, *y*)-coordinate. The coordinate is given in meters measured from the (0, 0)-coordinate, which is the southwest-most node in the network. Information on nodes' IDs and their coordinates are provided in a Node Table.

Arc: This is a (unidirectional) connection between a pair of nodes. It has an associated travel mode, which can be *walk*, *bus*, *rail transit*, or *car*. Each arc is ascribed a unique ID, along with the ID of the node at the beginning of the arc (NODEA), the ID of the node at the end of the arc (NODEB), its length (measured in meters), the speed limit (in meters per second) on the arc, and the travel mode (as described earlier). This information for each arc is provided in a Link Table. Each non-walk mode arc has a **time-dependent travel time** which, for the sake of simplicity, is specified in closed-form in terms of the arrival time at the tail node i (w_i), the length of the arc, its speed limit, and the zonal land-use data in which the arc lies. The assumed time-dependent travel time function $c_{ij}(w_i)$ is provided below for the arc connecting nodes i and j.

$$c_{ij}(w_i) = a(w_i)\times w_i + \left(\frac{\text{length of the arc}}{\text{speed limit}}\right)\times \text{daily time index}(w_i) \times \text{zonal index} \qquad (1)$$

where

$a(w_i)$ is a positive or negative rate (slope) that is varied within ranges, depending on the arrival time w_i. The particular relationship used is shown in Figure 27. (This pattern is used for all arcs in the network.) We partition the 24-hour interval into eight intervals, given by [10 PM-4 AM), [4 AM-7:30 AM), [7:30 AM-8:30 AM), [8:30 AM-9:15 AM), [9:15 AM-3:30 PM), [3:30 PM-4 PM), [4 PM-4:30 PM), and [4:30 PM-10 PM). Each interval has the corresponding slope as shown in the figure.

The daily time index(w_i) is a function (≥ 1) of the arrival time whose values vary within ranges as specified in Figure 27. During rush hour, for example in the interval [7:30 AM-8:30 AM), there is a higher daily time index than during non-rush hour. During the late night interval [10 PM-4 AM), we take the daily time index-parameter equal to 1, and also let $a(w_i) = 0$, so that the travel time is then simply based on the arc's length, its speed limit, and the zonal index only.

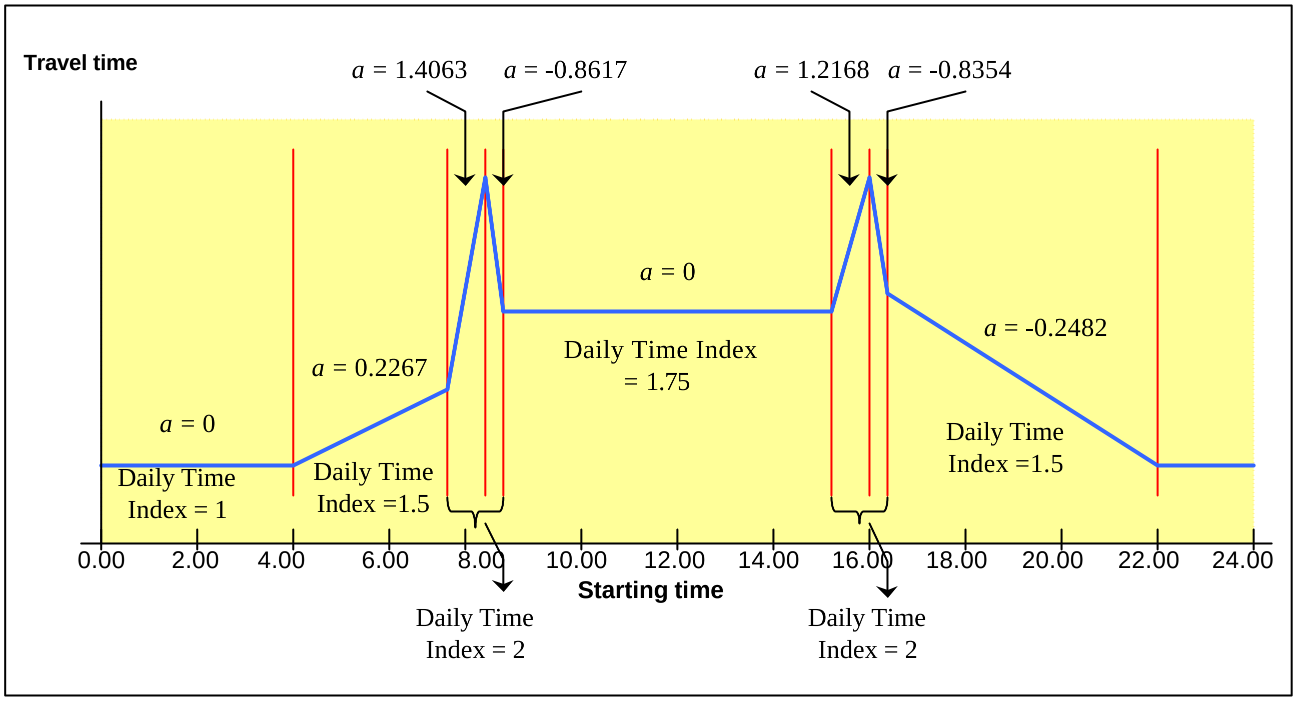


Figure 27: Pattern of Travel Time Function for Every Arc in the Network.

The zonal index is an index (≥ 1) which is used to inflate the travel time when the arc lies in a heavy commercial zone, as opposed to a residential zone, for instance. The zonal index for each land use zone is provided in Table 6.

Table 6: Zonal Index for each Land-Use Zone.

Zone	Zonal Index
Zone 1: Residential	1.0
Zones 2, 4, 5, and 8: Mixed Res./Comm.	1.2
Zones 3 and 6: Light Commercial	1.5
Zone 7: Heavy Commercial	2.0
Zone 9: Heavy Industrial	1.0

For an arc in the **walk** network, the travel time does not vary with the arrival time. In this case, the travel time is computed as the arc's length divided by the walking speed, which is set at **1** meter per second for all individuals in the population.

To scan the forward-star $FS(i)$ for any node i, we use the Lane Connectivity Table, which provides the IDs of the outgoing arcs from each node (OUTLINK), and the incoming arcs to the node (INLINK). Furthermore, the Car Parking and the Transit Stop Tables provide information regarding changes in travel mode, by specifying which nodes are a car parking, bus stop, or light rail stop node. The current version of TRANSIMS does not specify the car-parking location for each household. Hence, we assume that the **closest** such location to the household is its car-parking location. Because every node in the network, is defined in terms of its (x, y)-coordinate in meters, including those for household and car parking locations, we can calculate the distance between any pair of household location and car-parking location to determine the nearest parking. This data on the (x, y)-coordinates for every node is also useful in implementing the heuristic methods of Section 6. In this latter case, for the heuristic methods (i) - (iii), we assume that the average estimated velocity of travel v used in Equation (6.2) is 20 meters per second, based on the specified average speed for cars.

The upper bound *T* on an acceptable total travel time for each trip is obtained from the output of the Activity Generator Module, which is originally determined from the Activity and Travel Survey.

7.2 Design Objectives

The design objectives of the C++ code are to accomplish the following.

1) To find an **optimal solution** for each TDLSP problem using the **exact** TDLSP algorithm.
2) To find a solution using each of the heuristic methods in order to curtail the search, and to compare their relative performance with respect to quality of solution and speed.
3) To prescribe a strategy for implementing either an exact or a heuristic method depending on the problem size and structure based the experiments conducted in 2) above.
4) To conduct sensitivity analysis experiments using different values of the various parameters as prescribed in Section 7.4 below.

7.3 Design Evaluations

The indicators that are used to evaluate the performance of the exact algorithm and the heuristic methods include the following.

1) Quality of solution obtained (numerically evaluated by comparing the total travel time with the travel time obtained for the optimal solution).
2) Computational CPU processing time (in seconds on a 450 MHz Pentium II with 128 MB of RAM, 4.02 GB of hard drive capacity computer).
3) Ease of implementation.
4) Extensibility of the algorithm and methods for solving other variants of the shortest path problem.

7.4 Design Parameters

In order to conduct an empirical study on the selection of the various parameters used in the heuristic methods, we considered the parameter values as specified in Table 7.

Table 7: Parameter Values for the Heuristic Methods.

	Parameter	Value
Heuristic method (i)	b	1.00
Heuristic method (ii)	q	0.10
		0.25
		0.50
		1.00
		2.00
Heuristic method (iii)	a	0.10
		0.25
		0.50
		0.75
		1.00
Heuristic method (iv)	$(g\ , y\ )$	(1.10, 0.75)
		(1.10, 0.85)
		(1.25, 0.50)
		(1.25, 0.75)
		(1.25, 0.85)
		(1.50, 0.50)
		(1.50, 0.75)
		(1.50, 0.85)
		(1.75, 0.25)
		(1.75, 0.50)
		(1.75, 0.75)
		(1.75, 0.85)

Moreover, we designed the program to record information when it curtails a search at any node *i*, i.e., when $w'_i + b_i\, d(i, t) \geq T$ (obtained from (6.1b)). Note that this curtailment needs to be tracked only for the heuristic **methods (i) - (iii)** because for method (iv), the curtailment is done *a priori* based on the defined ellipsoidal regions and freeway as stated in (6.10).

7.5 Computational Results and Analysis

The TDLSP algorithm and the heuristic methods were tested for 4,516 time-dependent label-constrained shortest path problems obtained as specified above from the 1994/95 Activity and Travel Survey of Portland, Oregon. The **CPU processing time** (in seconds on a 450 MHz Pentium II with 128 MB of RAM, 4.02 GB of hard drive capacity computer) and the **quality of the solution** (calculated as the solution value divided by the optimal solution's travel time) were tabulated for each problem instance and parameter value. For the sake of illustration, the 4,516 problems are classified into three types of the trips, based on whether the trips are between HOME and WORK, between HOME and SCHOOL, and OTHER trips. Furthermore, each type of trip is classified into groups depending on the zone interchanges as specified in Table 8. Accordingly, there are 23 classes of problems. In addition, based on the assumed admissible mode strings for these 4,516 transportation activities, there are **nine** major mode strings as shown in Table 9. Furthermore, Table 5 presents the particular admissible mode strings implemented for each of the 23 classes of problems.

Table 8: Types of Travel Activities Classified into Crossing Zones.

Trip Type I: Trips between HOME and WORK	**Trip Type II: Trips between HOME and SCHOOL***	**Trip Type III: OTHER Trips**
Problem Class 1: Trip between Zones **1** and **9**	**Problem Class 11**: Trip between Zones **1** and **2**	**Problem Class 16**: Trip between Zones **1** and **9**
Problem Class 2: Trip between Zones **1** and **7**	**Problem Class 12**: Trip between Zones **1** and **4**	**Problem Class 17**: Trip between Zones **1** and **7**
Problem Class 3: Trip between Zones **1** and **3**	**Problem Class 13**: Trip between Zones **1** and **5**	**Problem Class 18**: Trip between Zones **1** and **Light Comm.** zones (Zones 3, and 6)
Problem Class 4: Trip between Zones **1** and **6**	**Problem Class 14**: Trip between Zones **1** and **8**	**Problem Class 19**: Trip between Zone **1** and **Mixed** zones
Problem Class 5: Trip between Zone **1** and **Mixed** Res./Comm. zones (Zones 2, 4, 5, and 8)	**Problem Class 15**: Trip within **Mixed** zones	**Problem Class 20**: Trip between **Mixed** zones and **9**
Problem Class 6: Trip between **Mixed** zones and **9**		**Problem Class 21**: Trip between **Mixed** zones and **7**
Problem Class 7: Trip between **Mixed** zones and **7**		**Problem Class 22**: Trip between **Mixed** Zones and **Light Comm.** zones
Problem Class 8: Trip between **Mixed** zones and **3**		**Problem Class23**: Trip within **Mixed** zones
Problem Class 9: Trip between **Mixed** Zones and **6**		
Problem Class 10: Trip within **Mixed** zones		

*** Note:** we assume that **Mixed** Res./Comm. zones (Zones 2, 4, 5, and 8) are the only zones containing **schools** in the network.

Table 9: Admissible Mode Strings Implemented in the Network and Their Notations.

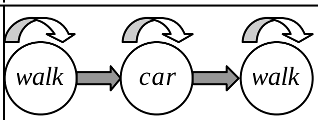
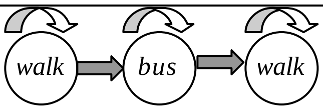
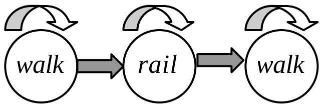
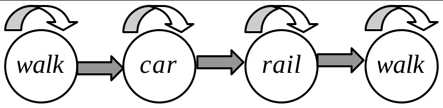
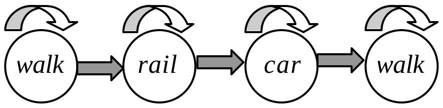
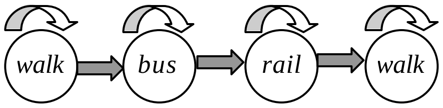
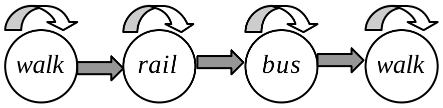
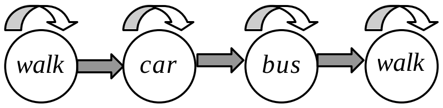
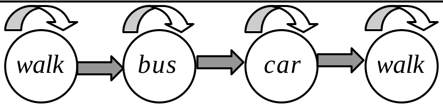
	Admissible Mode String	**Notation**
1		*wcw*-mode
2		*wbw*-mode
3		*wrw*-mode
4		*wcrw*-mode
5		*wrcw*-mode
6		*wbrw*-mode
7		*wrbw*-mode
8		*wcbw*-mode
9		*wbcw*-mode

Table 10: *Admissible Mode Strings for each Class of Problems.*

Problem Class	**Admissible mode strings**
1 and 16 (Trip between Zones 1 and 9)	*wcw, wbw, wcbw, wbcw* *
2 and 17 (Trip between Zones 1 and 7)	(all nine mode strings)
3, 4, and 18 (Trip between Zones 1 and Light Comm. zones)	(all nine mode strings)
5 and 19 (Trip between Zone 1 and Mixed zones)	(all nine mode strings)
6 and 20 (Trip between Mixed zones and 9)	*wcw, wbw, wcbw, wbcw* *
7 and 21 (Trip between Mixed zones and 7)	(all nine mode strings)
8, 9, and 22 (Trip between Mixed Zones and Light Comm. zones)	(all nine mode strings)
10, 15, and 23 (Trip within Mixed zones)	(all nine mode strings)
11 (Trip between Zones 1 and 2)	(all nine mode strings)
12 (Trip between Zones 1 and 4)	(all nine mode strings)
13 (Trip between Zones 1 and 5)	(all nine mode strings)
14 (Trip between Zones 1 and 8)	(all nine mode strings)

* Note that for any trip associated with Zone 9, there is no rail mode because there is no rail route through this region.

7.5.1 Computational Results

Table 11 presents the test results obtained from the TDLSP algorithm of Chapter 4, versus the four heuristic methods which were assigned suitable parameter values as stated in the table. An examination of these results reveals that at an average, the heuristic methods yield optimal solutions 27% of the time, with the overall average quality solution being about 1.078 (within 7.8% of optimality). The reason that the heuristic methods do not always yield optimal solutions is that the curtailment of search for a node to be added to the set NEXT sometimes cuts off nodes that might have led to an optimal solution. The CPU times are **decreased** dramatically by 30%. The proportions for the mode strings used by the heuristic method solutions, and the level of the terminal node, have roughly the same values as those for the exact algorithm. These results and comparisons therefore provide a reasonable validation of the heuristic methods. The interesting thing is that the average level or depth away from any starting nodes is 241 for the exact algorithm, and 242 for the heuristic methods, which is approximately only 25% of the total number of nodes in the network (1,000 nodes).

Table 11: Overall Results.

			Exact algorithm			**Heuristic Methods***				
Trip Type	Problem Class	Total no. of trips	Avg. CPU time (s/trip)	Avg. no. iterations or the level of the terminal node, *l*(*t*)	Avg. % of mode strings used	Avg. CPU time (s/trip)	Avg. no. iterations, *l*(*t*)	Avg. % of mode strings used	Avg. % of heuristic methods that yielded opt. solns.	Avg. soln. quality**
I	1	379	52.718	315	*wcw*56%, *wbw*39%	39.031	325	*wcw*57%, *wbw*40%	23	1.092
	2	316	39.267	229	*wcw*44%, *wrw*37%, *wbw*15%	27.888	214	*wcw*44%, *wrw*35%, *wbw*17%	25	1.084
	3	208	27.190	149	*wcw*49%, *wrw*35%, *wbw*11%	19.511	156	*wcw*48%, *wrw*32%, *wbw*14%	30	1.042
	4	284	42.081	251	*wcw*41%, *wrw*39%, *wbw*12%	31.198	258	*wcw*44%, *wrw*37%, *wbw*11%	26	1.055
	5	311	40.497	219	*wcw*48%, *wbw*36%, *wbrw*14%	29.913	203	*wcw*49%, *wbw*35%, *wbrw*14%	27	1.085
	6	298	44.891	279	*wcw*55%, *wbw*44%	33.949	266	*wcw*54%, *wbw*45%	25	1.061
	7	293	37.564	211	*wcw*38%, *wrw*38%, *wbw*14%, *wbrw*10%	27.205	224	*wcw*40%, *wrw*37%, *wbw*12%, *wbrw*11%	28	1.082
	8	176	39.691	226	*wcw*42%, *wrw*28%, *wbw*19%, *wbrw*10%	29.287	237	*wcw*43%, *wrw*25%, *wbw*20%, *wbrw*11%	28	1.099
	9	185	38.002	219	*wcw*43%, *wrw*30%, *wbw*15%, *wbrw*10%	27.100	209	*wcw*44%, *wrw*30%, *wbw*13%, *wbrw*11%	26	1.090
	10	108	40.257	278	*wcw*38%, *wbw*29%, *wrw*18%, *wbrw*14%	29.631	271	*wcw*39%, *wbw*27%, *wrw*19%, *wbrw*14%	25	1.088
II	11	217	26.583	158	*wcw*45%, *wbw*41%	16.513	172	*wcw*43%, *wbw*43%	31	1.077
	12	226	29.106	262	*wcw*42%, *wbw*40%, *wrw*17%	18.253	280	*wcw*44%, *wbw*41%, *wrw*15%	32	1.088
	13	220	43.097	294	*wcw*46%, *wbw*45%	32.377	297	*wcw*49%, *wbw*45%	27	1.084
	14	153	49.027	287	*wcw*44%, *wbw*46%	36.021	271	*wcw*44%, *wbw*47%	27	1.083
	15	135	40.097	271	*wcw*37%, *wbw*30%, *wrw*18%, *wbrw*13%	28.594	284	*wcw*38%, *wbw*29%, *wrw*18%, *wbrw*12%	26	1.093
III	16	149	53.245	327	*wcw*54%, *wbw*40%	37.829	328	*wcw*55%, *wbw*42%	23	1.087
	17	217	39.891	211	*wcw*43%, *wrw*36%, *wbw*17%	25.114	217	*wcw*41%, *wrw*36%, *wbw*18%	26	1.068
	18	117	31.081	206	*wcw*47%, *wrw*40%, *wbw*10%	20.239	210	*wcw*47%, *wrw*40%, *wbw*10%	28	1.065
	19	113	40.992	224	*wcw*49%, *wbw*34%, *wbrw*15%	26.803	236	*wcw*50%, *wbw*35%, *wbrw*14%	27	1.072
	20	104	41.037	188	*wcw*57%, *wbw*42%	27.143	178	*wcw*59%, *wbw*40%	25	1.065
	21	131	38.510	222	*wcw*37%, *wrw*38%, *wbw*13%, *wbrw*12%	23.766	231	*wcw*38%, *wrw*35%, *wbw*13%, *wbrw*13%	26	1.067
	22	95	38.159	225	*wcw*42%, *wrw*29%, *wbw*17%, *wbrw*11%	23.220	228	*wcw*45%, *wrw*27%, *wbw*12%, *wbrw*10%	27	1.088
	23	81	40.197	282	*wcw*38%, *wbw*28%, *wrw*19%, *wbrw*14%	24.942	280	*wcw*40%, *wbw*28%, *wrw*17%, *wbrw*15%	25	1.086
Total=**4516** trips			Avg.= **39.703** s.	**241**		Avg.= **27.632** s.	**242**		**Average= 27%**	**1.078**
		STD	6.794	46		5.931	46		2.21%	0.014

* For the heuristic method (ii) we used q = 0.25, for method (iii) we used a = 0.25, and for method (iv) we used g = 1.25 and y = 0.75. These parameter values were selected based on the results in Section 7.5.3, where these values yielded the **best solutions** in the most **effective CPU time**. The descriptions on how to select these values are provided in the corresponding sections for each of the methods.

** Avg. soln. quality is given by the average value of the **ratios** of the heuristic solutions to the optimal solutions for each problem number.

7.5.2 Comparison of Each Heuristic Method and the Exact Algorithm

Table 12a presents results for each heuristic method based on the parameter values that yielded the best heuristic performance with respect to the solution quality and the CPU time. Specifically, for method (ii) we used q = 0.25, for method (iii) we used a = 0.25, and for method (iv) we used g = 1.25 and y = 0.75.

Table 12a: Comparison of the Solution Quality for the Various Heuristic Methods.

Trip Type	Problem Class	**Avg. % of runs that yielded opt. solns.**					**Avg. soln. quality**				
		Method (i)	Method (ii)	Method (iii)-1*	Method (iii)-2**	Method (iv)	Method (i)	Method (ii)	Method (iii)-1	Method (iii)-2	Method (iv)
I	1	24	12	21	17	26	1.085	1.097	1.087	1.099	1.092
	2	28	12	22	16	22	1.090	1.099	1.091	1.098	1.041
	3	29	14	25	13	19	1.012	1.048	1.055	1.052	1.045
	4	27	15	24	13	21	1.034	1.078	1.049	1.063	1.053
	5	27	13	24	14	22	1.091	1.099	1.094	1.093	1.047
	6	26	9	25	16	24	1.029	1.113	1.038	1.039	1.084
	7	25	11	23	16	25	1.065	1.104	1.071	1.118	1.050
	8	26	11	22	15	26	1.098	1.119	1.098	1.096	1.083
	9	26	11	24	15	24	1.093	1.097	1.096	1.098	1.064
	10	25	10	24	16	25	1.068	1.101	1.058	1.148	1.065
II	11	27	12	27	13	21	1.034	1.080	1.052	1.043	1.176
	12	27	13	23	14	23	1.048	1.117	1.050	1.091	1.134
	13	26	13	24	14	23	1.055	1.125	1.084	1.106	1.050
	14	26	14	25	13	22	1.092	1.092	1.056	1.118	1.055
	15	25	10	26	16	23	1.094	1.108	1.099	1.106	1.057
III	16	26	13	22	15	24	1.012	1.075	1.069	1.178	1.102
	17	28	13	21	16	22	1.071	1.112	1.081	1.047	1.031
	18	26	11	24	15	24	1.034	1.092	1.045	1.082	1.070
	19	28	13	23	13	23	1.070	1.103	1.019	1.104	1.066
	20	26	10	26	14	24	1.013	1.108	1.018	1.085	1.100
	21	25	10	24	16	25	1.033	1.106	1.015	1.158	1.025
	22	27	13	22	15	23	1.090	1.094	1.092	1.098	1.066
	23	25	9	27	15	24	1.081	1.104	1.086	1.097	1.060
Average		**26**	**12**	**24**	**15**	**23**	**1.061**	**1.099**	**1.065**	**1.096**	**1.070**
STD		1.22	1.67	1.72	1.24	1.68	**0.030**	**0.017**	**0.027**	**0.035**	0.034

* Method (iii)-1 is the heuristic method (iii) using **Equation (6.7a).**

** Method (iii)-2 is the heuristic method (iii) using **Equation (6.7b).**

The results reveal that the heuristic method (i) yields the best solutions (based on the average values of the solution quality) followed in order by methods (iii)-1, (iv), (iii)-2, and (ii). Method (i) consistently adopts the minimum value of b , which is 1. Hence, the term $b_i d(i,t)$ in Equation (6.1b), is always lowest for this method as compared with that for the other heuristic methods. Consequently, method (i) curtails the least, and therefore, its CPU time is greater than that for the other methods as seen in Table 12b. The other heuristic methods curtail more of the nodes to be added to the set NEXT, and hence yield smaller CPU times, while sacrificing solution quality (in the same order as for decreasing solution times). It is interesting to note that method (iii)-1 which starts off with a value of b = 1.4 > 1, and rapidly reduces it via an exponential decay function to 1 comes closet to method (i) in solution quality, and results in somewhat decreased solution times. As a point of interest, we provide in Table 12c results for method (i) (average solution quality and CPU time) for the case of b = 0.9. The results are not competitive enough to recommend this strategy because the case when b = 0.9 yields larger CPU times than for the case b =1.0 which turn out to be almost equal to that for the exact algorithm, while the quality of solutions are only slightly better (4.3% of optimality) than that of the case when b = 1.0 (6.1% of optimality).

Table 12b: Comparison of Heuristic Methods in Computational Time.

Trip Type	Problem Class	Total no. of trips	Avg. CPU time (s/trip)					
			Exact algorithm	Heuristic Methods				
				Method (i)	Method (ii)	Method (iii)-1	Method (iii)-2	Method (iv)
I	1	379	52.718	51.903	28.902	44.903	41.904	27.541
	2	316	39.267	37.157	20.049	32.208	24.084	25.941
	3	208	27.190	23.841	12.494	20.781	17.491	22.947
	4	284	42.081	39.883	19.092	33.837	27.149	36.028
	5	311	40.497	39.557	23.483	29.469	26.106	30.948
	6	298	44.891	44.160	31.044	34.495	31.673	28.371
	7	293	37.564	37.549	20.123	29.340	25.106	23.907
	8	176	39.691	31.178	25.690	33.686	29.398	26.483
	9	185	38.002	32.167	21.605	31.216	28.015	22.496
	10	108	40.257	36.037	23.933	33.316	28.371	26.496
II	11	217	26.583	22.180	11.471	15.707	13.156	20.049
	12	226	29.106	21.718	14.639	18.214	16.127	20.567
	13	220	43.097	41.289	23.979	35.207	30.124	31.284
	14	153	49.027	36.371	26.838	41.617	36.332	38.948
	15	135	40.097	36.587	21.487	30.547	25.214	29.134
III	16	149	53.245	50.647	27.537	43.129	41.841	25.989
	17	217	39.891	30.023	18.244	29.143	24.594	23.567
	18	117	31.081	23.207	11.639	21.634	18.257	26.456
	19	113	40.992	31.754	23.958	30.818	24.031	23.456
	20	104	41.037	29.274	21.266	29.951	27.441	27.784
	21	131	38.510	29.451	15.574	26.474	24.182	23.147
	22	95	38.159	30.321	16.591	25.129	21.493	22.567
	23	81	40.197	33.341	20.184	25.155	23.561	22.469
		Average	**39.703**	**34.330**	**20.862**	**30.260**	**26.333**	**26.373**
		STD	**6.794**	**8.134**	**5.429**	**7.327**	**7.126**	4.646

Table 12c: Results for method (i) for the case of b = 0.9.

Trip Type	Problem Class	Avg. CPU time (s/trip)			Avg. soln. quality	
		Exact algorithm	Method (i) when b = 1.0	Method (i) when b = 0.9	Method (i) when b = 1.0	Method (i) when b = 0.9
I	1	52.718	51.903	52.899	1.085	1.058
	2	39.267	37.157	40.127	1.090	1.065
	3	27.190	23.841	27.161	1.012	1.013
	4	42.081	39.883	42.007	1.034	1.024
	5	40.497	39.557	40.553	1.091	1.064
	6	44.891	44.160	44.817	1.029	1.065
	7	37.564	37.549	37.571	1.065	1.038
	8	39.691	31.178	39.019	1.098	1.071
	9	38.002	32.167	36.633	1.093	1.066
	10	40.257	36.037	39.094	1.068	1.041
II	11	26.583	22.180	24.451	1.034	1.015
	12	29.106	21.718	27.890	1.048	1.021
	13	43.097	41.289	42.285	1.055	1.032
	14	49.027	36.371	46.072	1.092	1.065
	15	40.097	36.587	39.851	1.094	1.064
III	16	53.245	50.647	51.643	1.012	1.012
	17	39.891	30.023	38.981	1.071	1.044
	18	31.081	23.207	28.917	1.034	1.021
	19	40.992	31.754	38.015	1.070	1.043
	20	41.037	29.274	38.713	1.013	1.013
	21	38.510	29.451	37.803	1.033	1.032
	22	38.159	30.321	36.617	1.090	1.063
	23	40.197	33.341	37.701	1.081	1.054
	Average	**39.703**	**34.330**	**38.644**	**1.061**	**1.043**
	STD	**6.794**	**8.134**	**6.917**	**0.030**	**0.021**

7.5.3 Experiments on Selecting Parameter Values for the Different Heuristic Methods

In this section, we provide detailed experimental results on the performance of the various heuristic techniques using different parameter values.

For the heuristic method (i), which is the Standard Based Case, there is no further experimentation because the value of b is fixed at unity. The corresponding results are as given in Tables 12a and 12b. For the other methods, we conducted experiments to study the variation in performance with respect to different parameter values as described in Table 7.

Table 13 presents results on using different values of q for the heuristic method (ii), which is the **Network Sectioning Technique**. Recall that here, we partition the given network between the starting and terminal nodes into **three** sections. Each node in the network is assigned different b-value depending on the section in which it lies. The minimum weight of 1 is assigned to nodes that lie in the section that defines the relative vicinity of the terminal node. The other sections inherit b-values dependent on the choice of the parameter q.

The results reveal that the values $q = 0.1$ or 0.25 yield the best solutions (as seen from the average values reported in Table 13. Actually, these two q values yield roughly the same values of "% opt" and "ASQ" but the CPU times are substantially different. Hence, it is a good compromise to choose $q = 0.25$.

Table 13: Results Based on Various Values of the Parameter q for the Heuristic Method (ii): Network Sectioning Technique.

Trip Type	Problem Class	$q = 0.10$		$q = 0.25$		$q = 0.50$		$q = 1.00$		$q = 2.00$	
		% opt*	ASQ**	% opt	ASQ	% opt	ASQ	% opt	ASQ	% opt	ASQ
I	1	38	1.058	37	1.097	25	1.158	0	1.168	0	1.173
	2	43	1.119	42	1.099	15	1.148	0	1.159	0	1.176
	3	45	1.077	44	1.048	11	1.091	0	1.121	0	1.082
	4	41	1.047	41	1.078	18	1.114	0	1.134	0	1.205
	5	43	1.119	38	1.099	19	1.139	0	1.166	0	1.176
	6	50	1.104	42	1.113	8	1.159	0	1.178	0	1.208
	7	43	1.082	43	1.104	14	1.149	0	1.173	0	1.209
	8	46	1.064	45	1.119	9	1.166	0	1.194	0	1.249
	9	44	1.095	43	1.097	13	1.147	0	1.161	0	1.193
	10	43	1.099	43	1.101	14	1.148	0	1.166	0	1.181
II	11	47	1.073	35	1.08	18	1.122	0	1.137	0	1.148
	12	44	1.083	40	1.117	16	1.162	0	1.192	0	1.226
	13	47	1.059	40	1.125	13	1.181	0	1.228	0	1.293
	14	38	1.079	37	1.092	25	1.145	0	1.152	0	1.201
	15	49	1.145	44	1.108	7	1.153	0	1.172	0	1.144
III	16	46	1.082	41	1.075	13	1.123	0	1.138	0	1.127
	17	42	1.075	40	1.112	18	1.163	0	1.177	0	1.215
	18	47	1.112	46	1.092	7	1.132	0	1.158	0	1.144
	19	47	1.131	42	1.103	11	1.155	0	1.162	0	1.136
	20	43	1.122	43	1.108	14	1.147	0	1.174	0	1.193
	21	48	1.066	47	1.106	5	1.148	0	1.175	0	1.221
	22	44	1.045	40	1.094	16	1.142	0	1.164	0	1.199
	23	46	1.137	46	1.104	8	1.152	0	1.168	0	1.168
	Average	**44**	**1.090**	**42**	**1.099**	**14**	**1.145**	**0**	**1.166**	**0**	**1.186**
	STD	**3.09**	**0.029**	**3.08**	**0.017**	**5.31**	**0.019**	**0**	**0.022**	**0**	**0.044**

	$q = 0.10$	$q = 0.25$	$q = 0.50$	$q = 1.00$	$q = 2.00$
Average CPU time	**24.165**	**<u>20.862</u>**	**18.049**	**17.416**	**16.852**
STD	**5.672**	**5.429**	**5.306**	**5.273**	**5.201**

* % opt represents "Avg. % of runs that yielded optimal solutions."

** ASQ represents "Average Solution Quality."

For the heuristic method (iii), we assign b-values for each node in the network based on its level or depth away from the starting node. Note that we have proposed **two** relationships for prescribing the values of b as a function of the level in this method. One is an exponential decay function (with a limiting minimum value of 1), and other is a linear relationship. We explore these sub-methods separately, but use the same of the parameter a values equal to 0.10, 0.25, 0.50, 0.75, and 1.00.

For method (iii)-1, we use Equation (6.7a) which is redisplayed below:

$$b_i = \max\left\{1,\ 0.9 + 0.5e^{-l\, l(i)}\right\}\ \forall\, i \in N, \text{ where } l = \frac{-\ln(0.2)}{a\ n}. \tag{6.7a}$$

The corresponding results are shown in Table 14a. For method (iii)-2 we use Equation (6.7b) which is redisplayed below:

$$b_i = \max\left\{1, 1.4 - \left(\frac{0.4}{a\ n}\right) l(i)\right\}\ \forall\, i \in N. \tag{6.7b}$$

The corresponding results are shown in Table 14b.

Table 14a: Results for Different a Parameter Values for the Heuristic Method (iii)-1: Level-Based Technique ***using equation (6.7a).***

Trip Type	Problem Class	a = 0.10		a = 0.25		a = 0.50		a = 0.75		a = 1.00	
		% opt	ASQ	% opt	ASQ	% opt	ASQ	% opt	ASQ	% opt	ASQ
I	1	36	1.077	32	1.087	20	1.119	8	1.133	4	1.135
	2	36	1.064	29	1.091	23	1.125	9	1.143	3	1.181
	3	37	1.061	37	1.055	21	1.087	5	1.098	0	1.106
	4	37	1.043	36	1.049	18	1.084	8	1.086	1	1.095
	5	40	1.033	37	1.094	18	1.131	5	1.154	0	1.227
	6	38	1.029	35	1.038	20	1.067	4	1.074	3	1.106
	7	40	1.048	32	1.071	20	1.102	5	1.113	3	1.168
	8	42	1.072	32	1.098	19	1.136	5	1.147	2	1.206
	9	40	1.113	33	1.096	19	1.126	5	1.147	3	1.132
	10	36	1.035	32	1.058	24	1.094	6	1.099	2	1.143
II	11	37	1.072	33	1.052	25	1.089	5	1.093	0	1.098
	12	35	1.064	33	1.05	23	1.086	8	1.092	1	1.095
	13	36	1.043	30	1.084	26	1.113	6	1.133	2	1.192
	14	36	1.064	36	1.056	19	1.093	8	1.099	1	1.094
	15	36	1.073	36	1.099	19	1.138	7	1.151	2	1.179
III	16	36	1.047	32	1.069	20	1.101	8	1.112	4	1.131
	17	39	1.042	37	1.081	19	1.123	5	1.131	0	1.188
	18	33	1.035	33	1.045	22	1.075	8	1.083	4	1.115
	19	37	1.037	37	1.019	18	1.049	7	1.052	1	1.057
	20	37	1.038	37	1.018	18	1.048	6	1.048	2	1.046
	21	35	1.042	33	1.015	20	1.044	8	1.046	4	1.044
	22	36	1.049	34	1.092	22	1.134	6	1.146	2	1.198
	23	37	1.058	33	1.086	22	1.124	7	1.132	1	1.154
	Average	**37**	**1.054**	**34**	**1.065**	**21**	**1.099**	**6**	**1.109**	**2**	**1.134**
	STD	**2.01**	**0.019**	**2.38**	**0.027**	**2.33**	**0.029**	**1.44**	**0.034**	**1.36**	**0.052**

	a = 0.10	a = 0.25	a = 0.50	a = 0.75	a = 1.00
Average CPU time	**34.903**	**<u>30.260</u>**	**27.096**	**25.857**	**24.012**
STD	**7.562**	**7.327**	**7.206**	**7.196**	**7.092**

For this case1 of method (iii), the results reveal that when a = 0.1 or 0.25, we obtain the best quality solutions. The solutions for these two values of a are not significantly different in the sense of "% opt" and the average solution quality (ASQ), but the CPU times (see Table 14a) are substantially different. Hence, it is a good compromise to select a = 0.25.

The results for method (iii)-2 are given in Table 14b. The results reveal the same pattern in the proportion "% opt" as for the previous method (exponential relationship), but the exponential relationship yields much **better solution quality** (as seen from the average solution quality values) for all values of a. The exponential relationship (6.7a) always yields a smaller value of b for each node, and especially so for the intermediate nodes in the origin-destination path before the level $a\,n$. Due to the ascribed values of b, this case has a **lesser** chance of cutting off a node to be added to the set NEXT, and hence is more likely to preserve an optimal path. On the other hand, for the same reason, the exponential relationship consumes greater CPU time.

Table 14b: Results for Different a Parameter Values for the Heuristic Method (iii)-2: Level-Based Technique ***using equation (6.7b).***

Trip Type	Problem Class	a = 0.10		a = 0.25		a = 0.50		a = 0.75		a = 1.00	
		% opt	ASQ	% opt	ASQ	% opt	ASQ	% opt	ASQ	% opt	ASQ
I	1	36	1.046	35	1.099	20	1.153	6	1.162	3	1.223
	2	38	1.062	35	1.098	19	1.153	5	1.162	3	1.227
	3	36	1.044	35	1.052	21	1.093	6	1.101	2	1.159
	4	36	1.084	34	1.063	18	1.111	9	1.117	3	1.118
	5	37	1.087	36	1.093	19	1.136	6	1.154	2	1.158
	6	35	1.059	34	1.039	20	1.079	7	1.087	4	1.087
	7	36	1.144	34	1.118	21	1.167	5	1.188	4	1.174
	8	36	1.131	35	1.096	20	1.136	6	1.153	3	1.143
	9	37	1.128	36	1.098	19	1.152	5	1.163	3	1.112
	10	35	1.157	33	1.148	21	1.202	7	1.217	4	1.257
II	11	36	1.071	36	1.043	17	1.088	8	1.093	3	1.098
	12	34	1.072	33	1.091	21	1.139	8	1.152	4	1.205
	13	34	1.073	33	1.106	21	1.161	8	1.166	4	1.182
	14	36	1.093	35	1.118	19	1.162	7	1.192	3	1.205
	15	35	1.111	34	1.106	19	1.151	9	1.172	3	1.207
III	16	33	1.118	32	1.178	21	1.229	9	1.259	5	1.314
	17	37	1.062	36	1.047	19	1.093	6	1.129	2	1.111
	18	34	1.068	34	1.082	21	1.133	6	1.143	5	1.187
	19	36	1.121	36	1.104	17	1.154	8	1.169	3	1.155
	20	36	1.09	35	1.085	19	1.127	7	1.146	3	1.159
	21	33	1.062	33	1.158	21	1.224	8	1.238	5	1.345
	22	35	1.064	34	1.098	20	1.146	8	1.165	3	1.232
	23	36	1.132	35	1.097	19	1.144	7	1.156	3	1.161
	Average	**36**	**1.090**	**34**	**1.096**	**20**	**1.145**	**7**	**1.160**	**3**	**1.183**
	STD	**1.28**	**0.033**	**1.16**	**0.035**	**1.27**	**0.039**	**1.28**	**0.042**	**0.89**	**0.065**

Average CPU time	**29.671**	**26.333**	**23.586**	**22.376**	**21.707**
STD	**7.274**	**7.126**	**7.089**	**7.041**	**7.003**

Next, we experimented with the parameters g and y for the Ellipsoidal Region Technique of method (iv). For the sake of presentation and illustration, we show the results of "% opt" and the average solution quality (ASQ) separately. Note that the results are obtained from the average of **all the 4,516 problems**, which means that the results for **each cell** in the following tables is averaged over all the 4,516 problems used for our computational experiments.

Table 15a presents the results for "% opt." The results indicate that, for an ellipsoidal region having **too short** a major axis a (as expressed by $g \leq 1.10$), or too short a minor axis b (as expressed by $y \leq 0.50$), we obtain a significant loss of optimality. Both results are obtained for the cases when $g \geq 1.25$ and $y \geq 0.75$. The same logic is reflected in Table 15b related to the solution quality. Table 15c displays the corresponding average CPU times for the (g, y) combinations. These results indicate that when g =1.25 and y =0.75, we obtain good quality solutions (not too different from the solutions obtained for greater g and y values), while consuming a much lesser CPU time. Hence, we select g =1.25 and y =0.75 as the parameter values for the heuristic method (iv).

Based on the foregoing analysis, it is interesting to compare the solution quality and the CPU effort for method (iv) when we use (g, y) = (1.25, 0.75) to define **rectangular regions** for the nodes between the starting node and its nearest freeway entrance, and that for nodes between the terminal node and its nearest freeway exit, instead of ellipsoidal regions. The results are shown in Table 15d.

Table 15a: Avg. % Opt for Various Parameter Values g and y for the Heuristic Method (iv): Ellipsoidal Region Technique.

		g			
		1.10	**1.25**	1.50	1.75
y	0.25	1% opt	1% opt	1% opt	3% opt
	0.50	1% opt	16% opt	19% opt	20% opt
	0.75	14% opt	**23% opt**	24% opt	25% opt
	0.85	16% opt	23% opt	24% opt	25% opt

Table 15b: Average Solution Quality (ASQ) for Various Parameter Values g and y for the Heuristic Method (iv): Ellipsoidal Region Technique.

		g			
		1.10	**1.25**	1.50	1.75
y	0.25	1.483	1.408	1.230	1.207
	0.50	1.369	1.169	1.121	1.116
	0.75	1.235	**1.070**	1.063	1.059
	0.85	1.148	1.068	1.060	1.059

Table 15c: Average CPU times (s/trip) for Various Parameter Values g and y for the Heuristic Method (iv): Ellipsoidal Region Technique.

		g			
		1.10	**1.25**	1.50	1.75
y	0.25	24.223	24.102	25.348	26.305
	0.50	24.317	24.769	26.127	27.501
	0.75	25.099	**26.373**	30.671	32.673
	0.85	25.613	29.873	32.128	35.057

Table 15d: Detailed Results for the Alternative Method (iv) with Rectangular Accesses versus the Regular Method (iv) and the Exact Algorithm.

Trip Type	Problem Class	**Avg. CPU time (s/trip)**			**Avg. soln. quality**	
		Exact algorithm	Regular Method (iv)	Alternative Method (iv)	Regular Method (iv)	Alternative Method (iv)
I	1	52.718	27.541	30.451	1.092	1.088
	2	39.267	25.941	28.450	1.041	1.040
	3	27.190	22.947	26.038	1.045	1.043
	4	42.081	36.028	39.206	1.053	1.049
	5	40.497	30.948	33.485	1.047	1.044
	6	44.891	28.371	30.284	1.084	1.082
	7	37.564	23.907	26.789	1.050	1.049
	8	39.691	26.483	28.456	1.083	1.078
	9	38.002	22.496	24.356	1.064	1.063
	10	40.257	26.496	28.687	1.065	1.064
II	11	26.583	20.049	22.401	1.176	1.170
	12	29.106	20.567	22.978	1.134	1.128
	13	43.097	31.284	33.567	1.050	1.046
	14	49.027	38.948	41.670	1.055	1.051
	15	40.097	29.134	31.859	1.057	1.053
III	16	53.245	25.989	28.867	1.102	1.097
	17	39.891	23.567	25.672	1.031	1.031
	18	31.081	26.456	28.691	1.070	1.068
	19	40.992	23.456	25.782	1.066	1.062
	20	41.037	27.784	28.963	1.100	1.098
	21	38.510	23.147	25.671	1.025	1.025
	22	38.159	22.567	24.785	1.066	1.063
	23	40.197	22.469	24.698	1.060	1.055
	Average	**39.703**	**26.373**	**28.774**	**1.070**	**1.067**
	STD	**6.794**	4.646	4.772	0.034	0.033

Finally, we provide results on tracking the curtailment for each of the relevant heuristic methods (i) - (iii). According to the previous results, we choose the (best) parameter values for each method. For method (i) b was fixed at unity, for method (ii) we used q = 0.25, and for methods (iii)-1 and (iii)-2 we used a = 0.25. The results of tracking the curtailments are shown in Tables 16 to 18b.

For method (i), define the following:

Recall the statements (6.1a) and (6.1b):

if $w_i' \equiv (w_k + c_{ki}) < w_i$ (6.1a)

and if $w_i' + b_i\, d(i, t) < T.$ (6.1b)

N_1 = number of times that the statement (6.1a) holds,

N_2 = number of times that the statements (6.1a) *and* (6.1b) hold (which is equal to the number of updates performed by the heuristic),

$(N_1 - N_2)$ = number of curtailments,

% curtailments = $\frac{(N_1 - N_2)}{N_1} \cdot 100\%$.

Table 16 displays the % curtailments for method (i). Observed that at an average, 14% of nodes were curtailed (under the case b = 1), that would otherwise have been added to the list NEXT.

Table 16: Tracking Curtailment Results for Heuristic Method (i).

Trip Type	Problem Class	% of curtailments
I	1	14
	2	17
	3	13
	4	12
	5	15
	6	14
	7	12
	8	16
	9	15
	10	13
II	11	15
	12	13
	13	14
	14	13
	15	14
III	16	15
	17	17
	18	13
	19	14
	20	13
	21	12
	22	16
	23	14
Average		**14**
STD		**1.47**

For the Network Sectioning Technique (method (ii)), as shown in Table 17, in accordance to what we might expect, the curtailments occur mostly within the first section. Surprisingly, there is still some percentage of curtailments occurring within the final (the third) section.

Table 17: Tracking Curtailment Results for Heuristic Method (ii).

Trip Type	Problem Class	% of curtailments occurring in section r		
		$r = 1$	$r = 2$	$r = 3$
I	1	63	29	8
	2	67	27	6
	3	68	28	4
	4	64	29	7
	5	67	28	5
	6	63	30	7
	7	68	27	5
	8	66	28	6
	9	69	27	4
	10	67	28	5
II	11	69	26	5
	12	68	27	5
	13	64	29	7
	14	62	30	8
	15	67	27	6
III	16	64	29	7
	17	69	26	5
	18	69	26	5
	19	67	27	6
	20	65	28	7
	21	67	28	5
	22	69	27	4
	23	68	27	5
	Average	**66**	**28**	**6**
	STD	**2.21**	**1.18**	**1.21**

For the heuristics (iii)-1 and (iii)-2, as shown in Tables 18a and 18b, respectively, the curtailments occur mostly within the first **250 steps away** from any starting node. The results make sense because we specified a = 0.25, hence, after the level 250, the weight b_i is equal to 1, resulting in a lesser chance of curtailment. Comparing the results of Tables 18a and 18b, we see that method (iii)-2 has a **higher** percentage of curtailments for each interval of level.

This can be explained along the same lines as the analysis corresponding to Tables 12a and 12b.

Table 18a: Tracking Curtailment Results for Heuristic Method (iii)-1.

Trip Type	Problem Class	% of the curtailments occurring within level:				
		$0 \le l(i) < 100$	$100 \le l(i) < 250$	$250 \le l(i) < 500$	$500 \le l(i) < 750$	$750 \le l(i) \le 1000$
I	1	55	36	5	2	2
	2	57	37	3	2	1
	3	58	39	2	1	0
	4	56	37	4	2	1
	5	57	36	4	2	1
	6	56	35	5	2	2
	7	57	37	3	2	1
	8	57	36	4	2	1
	9	57	38	3	1	1
	10	56	36	5	2	1
II	11	58	40	2	0	0
	12	58	39	2	1	0
	13	56	36	4	2	2
	14	55	36	5	2	2
	15	56	38	4	1	1
III	16	54	37	5	2	2
	17	57	36	5	1	1
	18	58	37	3	1	1
	19	56	37	4	2	1
	20	55	38	4	2	1
	21	57	38	3	1	1
	22	56	39	3	1	1
	23	57	36	4	2	1
	Average	**56**	**37**	**4**	**2**	**1**
	STD	**1.08**	**1.29**	**1.01**	**0.59**	**0.60**

Table 18b: Tracking Curtailment Results for Heuristic Method (iii)-2.

Trip Type	Problem Class	% of the curtailments occurring within level:				
		$0 \le l(i) < 100$	$100 \le l(i) < 250$	$250 \le l(i) < 500$	$500 \le l(i) < 750$	$750 \le l(i) \le 1000$
I	1	58	37	3	1	1
	2	60	39	0	1	0
	3	60	39	1	0	0
	4	59	38	2	1	0
	5	61	37	1	1	0
	6	59	37	2	1	1
	7	59	40	1	0	0
	8	60	38	1	1	0
	9	59	40	1	0	0
	10	60	36	3	1	0
II	11	59	40	1	0	0
	12	59	40	1	0	0
	13	60	38	2	0	0
	14	59	37	2	1	1
	15	59	39	2	0	0
III	16	60	38	2	1	0
	17	59	39	2	0	0
	18	60	39	1	0	0
	19	61	38	1	0	0
	20	59	39	1	1	0
	21	59	40	1	0	0
	22	59	40	1	0	0
	23	59	38	1	1	1
	Average	**59.4**	**38.5**	**1.4**	**0.5**	**0.2**
	STD	**0.73**	**1.20**	**0.73**	**0.51**	**0.39**