Chapter 5

Results

This chapter reports on the results of the initial solution algorithm, as programmed in *MATLAB*. The chapter begins with a discussion on how comparative data sets were established for computational purposes, along with a motivation for modifying existing data sets. The results are presented and discussed, focusing on the contribution of the *time window compatibility* on both the computational burden, and the quality of the initial solution.

5.1 The basic Solomon sets

Solomon [46] discusses the generation of data sets for the *Vehicle routing and scheduling problems with time window constraints* (VRPSTW), and indicates that the design of these data sets highlight several factors that affects the behavior of his routing and scheduling heuristics. The corresponding six data sets, referred to as $R1$, $R2$, $C1$, $C2$, $RC1$, and $RC2$, are often used and referred to in literature.

5.1.1 Geographical distribution

The data used for the customer coordinates and demands are based on the work of Christofides *et al.* [13], and are classified into one of three categories:

- Randomly distributed customers (denoted by an R prefix)
- Clustered customers (denoted by a C prefix)
- Semi-clustered customers (denoted by an RC prefix)

By semi-clustered is implied a random mix of both randomly distributed and clustered customers.

5.1.2 Scheduling horizon

The length of the route-time is regarded as a capacity constraint and, along with the vehicle capacities, limit the number of customers serviced by a specific vehicle. The short scheduling horizon problems are denoted by a "1" as a suffix. The problems denoted by a "2" suffix, on the other hand, have a large scheduling horizon, and along with the vehicle capacities, allow a larger number of customers to be serviced by a single vehicle.

5.2 Test data

Solomon's data sets have become a benchmark for vehicle routing problem variants, although the sets are often used with some modification due to the specific variant's particularity. The published data sets [47] assume a homogeneous fleet, indicate a given vehicle capacity, and assume infinite availability of vehicles. Furthermore the sets only indicate a single time window for each geographically distributed customer.

The aim of the algorithm developed in this dissertation is to test the feasibility of integrating multiple time windows, a heterogeneous fleet, and double scheduling into an initial solution heuristic. As none of these three specific variants are addressed by the Solomon data sets, it was necessary to generate a unique data set to accommodate, and integrate, the problematic variants, yet still have resemblance to the familiar benchmarks in literature. The problem of developing an appropriate data set meant addressing both multiple time windows, and a heterogeneous fleet. Double scheduling is a function of the algorithm, and does not require any manipulation of the data sets.

5.2.1 Incorporating multiple time windows

It originally seemed appropriate to use two of the Solomon sets for a specific class of problem, each with 100 customers and a single, unique time window, to create a new set of 100 customers with multiple time windows. It turned out to be futile, as the time windows for different data sets of the same classification, as presented by Solomon [47], had very similar, if not exactly the same, time windows. The extended data sets presented by Homberger [24] are developed in the same manner as Solomon's sets, but have sets with 200, 400, 600, 800, and 1000 customers.

For each of the six problem classes, an extended set with 200 customers from Homberger's sets is used to create a new set with 100 customers, but with two time windows. Table 5.1 illustrates an excerpt from the data used to create a new set for the $C1$ class. In the Homberger data sets, the *cus-*

tomer number, *x-coordinate*, *y-coordinate*, *demand*, *service time*, and the *start-* and *end times* of a single time window, are given. Table 5.1 indicates

Table 5.1: Constructing data sets with multiple time windows

Customer				**Time window 1**		**Time window 2**	
(i)	x	y	...	e_i^1	l_i^1	e_i^2	l_i^2
⋮							
4	4	28	...	616	661	128	195
5	25	26		128	179	142	197
6	86	37		478	531	754	814
7	1	109		616	680	583	647
8	6	135		351	386	1011	1077
9	32	79	...	655	721	950	1003
⋮							

the case where the single time windows of customers 101 through 200 have been used as *second* time windows for customers 1 through 100. For example, the time window for customer 4 is given as $(616, 661)$, measured in minutes from $0:00am$. Customer 104's time window is given as $(128, 195)$ in the original data set. Customer 104's time window now becomes the second time window for customer 4, while all other data about customers 101 through 200 is disregarded. Observe, however, that the two time windows specified for customer 5 overlap, and do not yield two unique time windows as is the case for customer 4.

Procedure 1 indicates the procedure used to manipulate the time windows from multiple sets into a single data set. Where time windows overlap,

Procedure 1 Creating a data set with multiple time windows

if either $e_i^1 > l_i^2$ or $e_i^2 > l_i^1$ **then**
  Number of time windows is 2
  $E_i^1 = \min\{e_i^1, e_i^2\}$
  $L_i^1 = \min\{l_i^1, l_i^2\}$
  $E_i^2 = \max\{e_i^1, e_i^2\}$
  $L_i^2 = \max\{l_i^1, l_i^2\}$
else
  Number of time windows is 1
  $E_i^1 = \min\{e_i^1, e_i^2\}$
  $L_i^1 = \max\{l_i^1, l_i^2\}$
end if

the new, single time window, is defined to start at the opening of the earlier

time window, and end at the closing of the later time window. After the manipulation of the data, the start of the first time window for customer i is denoted by E_i^1, and the end of the time window by L_i^1. Where only one time window exist, no second time window is specified. Alternatively, the start of the second time window for customer i is denoted by E_i^2, and the end of the time window by L_i^2.

5.2.2 Incorporating a heterogeneous fleet

Liu and Shen [31, 32] propose a specific fleet structure with the introduction of their insertion-based savings heuristic for a heterogeneous fleet. The proposed cost structure sees the cost of a vehicle more than doubles when its capacity doubles. Although Dullaert *et al.* [19] challenges the cost structure presented by Liu and Shen, they did not propose a new cost structure. It was therefor considered appropriate to use the given fleet composition as indicated in Table 5.2. It should be noted that Liu and Shen assumed an infinite

Table 5.2: Heterogeneous fleet data

Problem class $R1$		
vehicle	**capacity**	**cost**
A	30	50
B	50	80
C	80	140
D	120	250
E	200	500

Problem class $R2$		
vehicle	**capacity**	**cost**
A	300	450
B	400	700
C	600	1200
D	1000	2500

Problem class $C1$		
vehicle	**capacity**	**cost**
A	100	300
B	200	800
C	300	1350

Problem class $C2$		
vehicle	**capacity**	**cost**
A	400	1000
B	500	1400
C	600	2000
D	700	2700

Problem class $RC1$		
vehicle	**capacity**	**cost**
A	40	60
B	80	150
C	150	300
D	200	450

Problem class $RC2$		
vehicle	**capacity**	**cost**
A	100	150
B	200	350
C	300	550
D	400	800
E	500	1100
F	1000	2500

number of each of the vehicles types. To accommodate the infinite number

of vehicles into the data sets generated for the algorithm, the number of vehicles for each type is said to be the number of vehicles needed to service the demand of all customers, if *only* that type of vehicle was available.

5.3 Results

Six data sets, one for each class of problem, were generated. The algorithm was coded in *MATLAB 6.5 release 13*. The algorithm was executed on three similar *Pentium IV 1.6GHz* computers, each with *256Mb RAM*, with each class being executed at least once on each of the computers. For each class of problem, five runs of the algorithm were executed. The results presented in this chapter is in each case the average of the five runs. Appendix C contains the solution details with respect to the specific customers assigned to the various routes, the orphaned customers (if any), as well as the total scheduling distance.

A summary of the initial solutions generated by the proposed algorithm is presented in Table 5.3. The influence of the problem characteristics (depicted by the problem class) can be appreciated when comparing the average number of customers per route for the type 1 and type 2 classes. The type 1 classes, with a narrow scheduling horizon, has an average of 2.13 customers per route, while the type 2 classes with longer scheduling horizons have, on average, 9.38 customers per route.

Table 5.3: Summary of computational results

Problem class	**Average CPU time** (seconds)	**Number of Tours**	**Number of Routes**	**Total scheduling distance** (kilometers)
$R1$	3180	18	71	11260
$R2$	31730	5	7	8722
$C1$	2170	11	22	7330
$C2$	65650	4	6	7240
$RC1$	1120	26	48	8706
$RC2$	6960	12	19	7748

It should be noted, however, that the average CPU time is extremely high. The reason for the computationally expensive CPU results is twofold:

- The algorithm was coded in *MATLAB* with the specific intent of the candidate to learn the software package. The code structure may therefore be inefficient, with numerous opportunity for code optimization.

- The file format in which *MATLAB* was executed is a non-compiled **.M* file, which is, from a computational point of view, significantly slower than a compiled **.dll* file. The decision to not compile was influenced by compiling software availability at the time of testing the algorithm.

Although there is ample opportunity to improve the algorithms technical performance, the results, in terms of number of tours, number of routes per tour, as well as the actual scheduling distance, proves to have significant contribution to the field of vehicle routing problem algorithms. Figures 5.1 through 5.6 indicate the cumulative CPU time for each class of problem.

5.4 Evaluating the contribution of TWC

It is important to evaluate the contribution that the proposed *time window compatibility* has on the results of the algorithm. For this purpose, a comparative *control algorithm* is created. The control algorithm differs only in two respects from the proposed algorithm:

- It does not evaluate nodes for *time window compatibility* when calculating the insertion criteria, and therefore considers every node for insertion on every edge of a partially constructed route.
- As no *time window compatibility* is calculated for any node, the initialization criteria is changed to identify the seed customer as the unrouted customer with the earliest deadline.

The control algorithm is executed for two extreme problem classes, namely clustered customers with a short scheduling horizon ($C1$), and uniformly distributed customers with a long scheduling horizon ($R2$). Five runs for each class were executed in a similar fashion to the proposed algorithm, with regards to the computers used, and the distribution of runs on the computers.

Figure 5.7 illustrates the significant improvement that the *time window compatibility* has on the computational burden for the $C1$ class of problems. The average CPU time is down from 10950 seconds to 2170, an improvement in excess of 80%. Not only did the *time window compatibility* reduce the computational burden, but it also improved the quality of the initial solution by almost 13%. By the *quality of the solution* is implied the total scheduling distance. The comparative results summary is given in Table 5.4.

Figure 5.8 indicates a computational saving of more than 24% for the $R2$ class problems, while the quality of the initial solution itself is improved by almost 13%. The $R2$ results correspond with the expectation that the

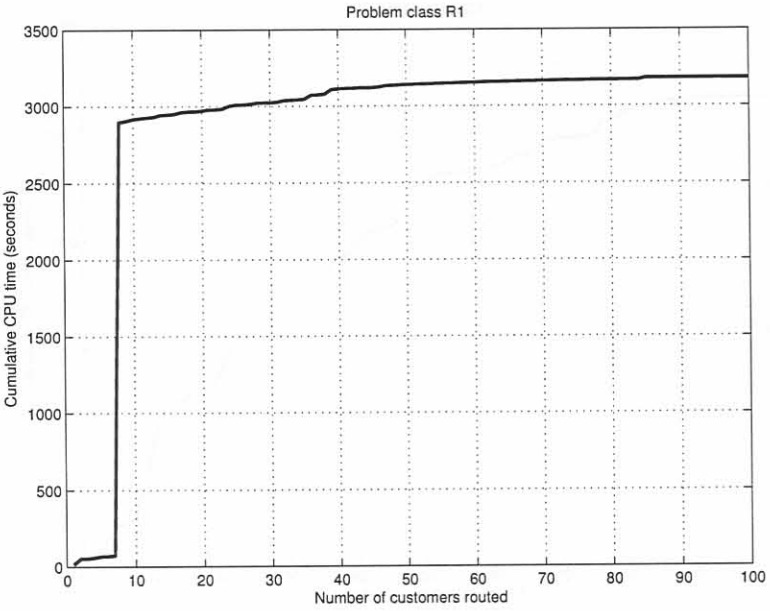

Figure 5.1: Cumulative progress for the $R1$ class problem

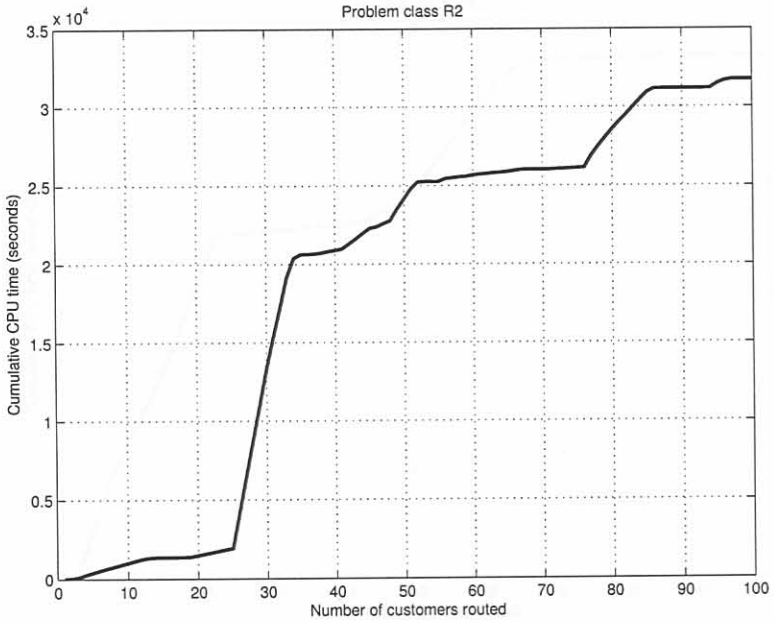

Figure 5.2: Cumulative progress for the $R2$ class problem

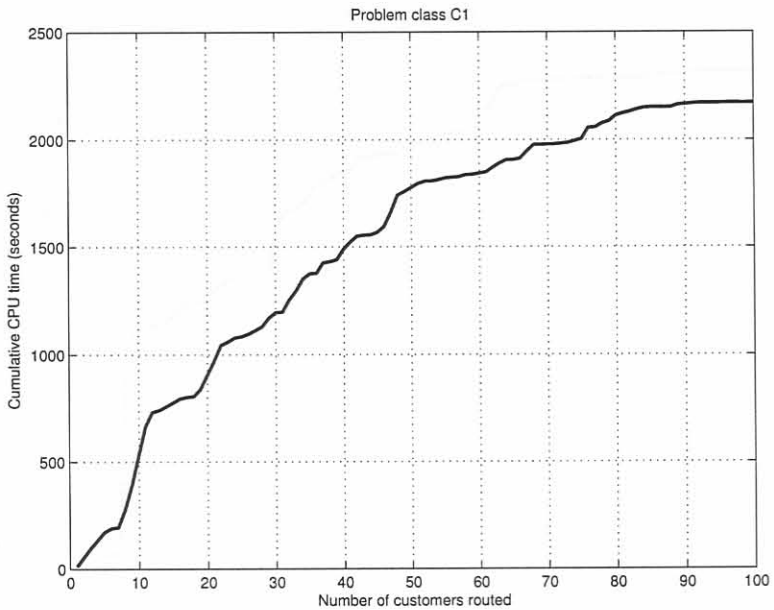

Figure 5.3: Cumulative progress for the $C1$ class problem

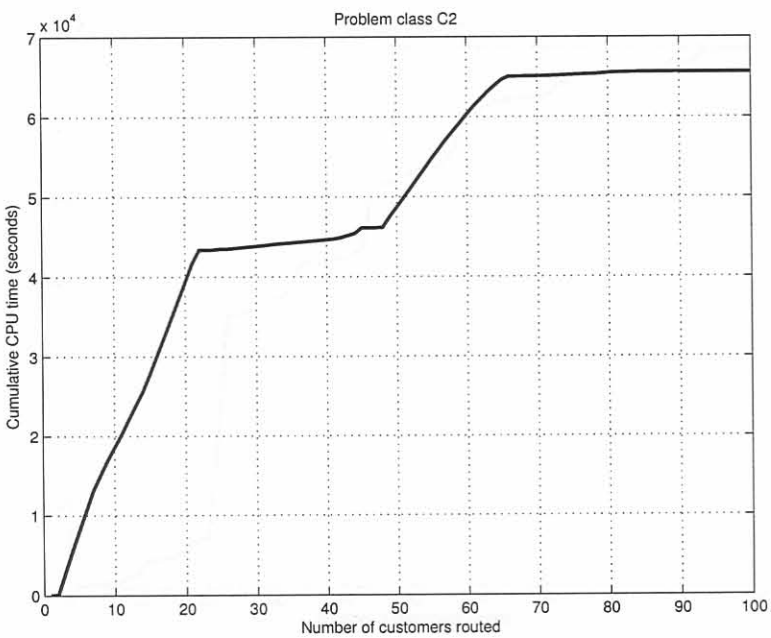

Figure 5.4: Cumulative progress for the $C2$ class problem

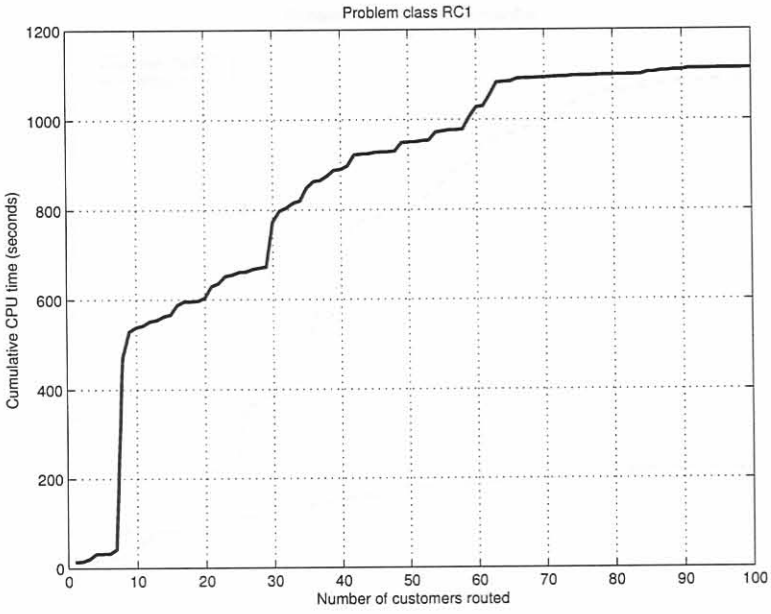

Figure 5.5: Cumulative progress for the $RC1$ class problem

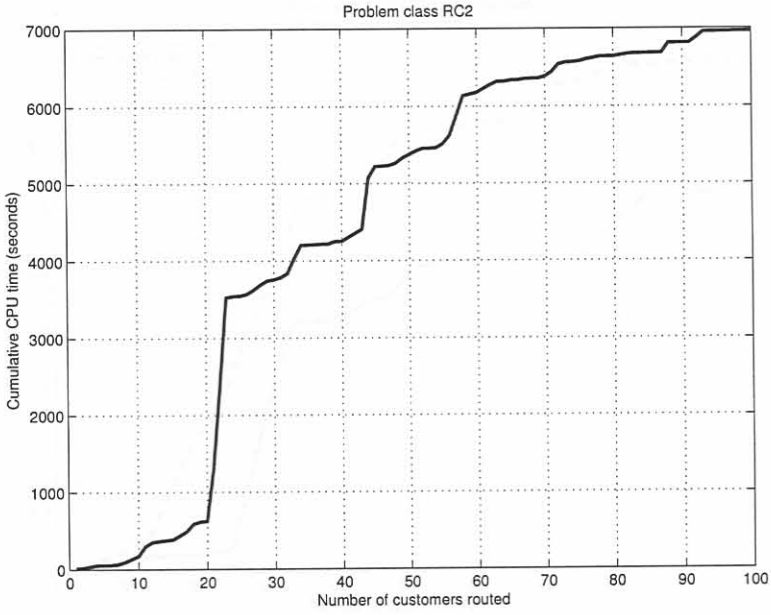

Figure 5.6: Cumulative progress for the $RC2$ class problem

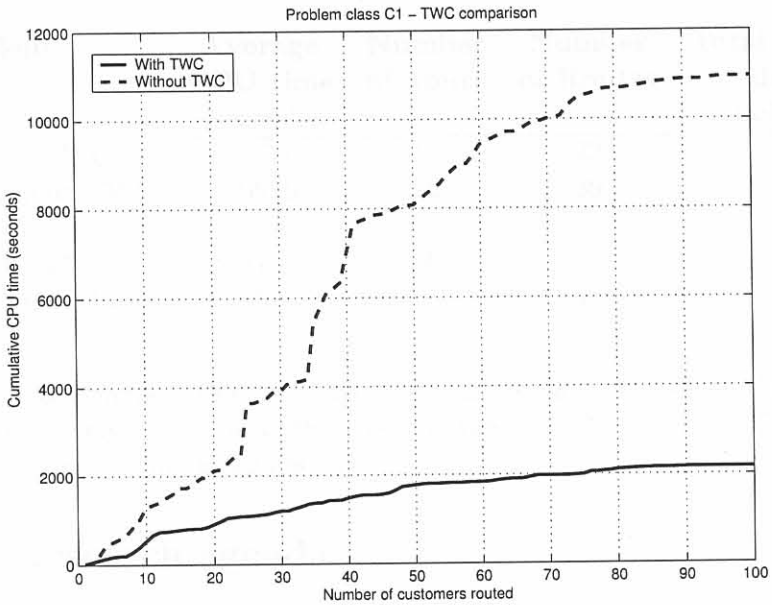

Figure 5.7: The effect of time window compatibility on the $C1$ class

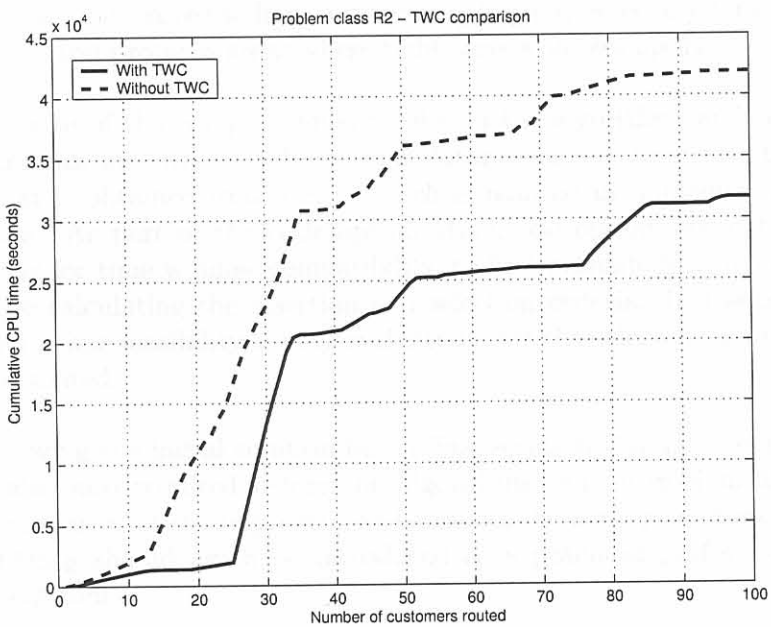

Figure 5.8: The effect of time window compatibility on the $R2$ class

Table 5.4: Summary of comparative results

Problem class	Average CPU time (seconds)	Number of Tours	Number of Routes	Total scheduling distance (kilometers)
$C1$ with TWC	2170	11	22	7330
$C1$ without TWC	10950	12	39	8876
$R2$ with TWC	31730	4	7	8722
$R2$ without TWC	41920	2	13	9990

time window compatibility will have less of an impact if the scheduling horizon is relatively large, as is the case for the type 2 problems. Still, the computational saving is significant.

5.5 Research agenda

Chapter 1 states the research question. The objective is to attempt to create an initial solution that caters for multiple time windows, a heterogeneous fleet, as well as double scheduling. Although the results generated during this dissertation is computationally expensive, the aim has been achieved, and with unexpectedly high returns. The concept of *time window compatibility* has proved to have a staggering impact, especially for clustered customers, and problem areas where tight time windows apply.

The issue of the computational burden of the algorithm, and technical code optimization, will be addressed in future research to ensure that the value that is obtained from this research is realized in implementable applications. As part of the code optimization, one option would be, after evaluating for time window compatibility, to first evaluate for time feasibility before calculating the insertion and selection criteria. In the proposed algorithm, time feasibility is only evaluated after the selection criteria have been calculated.

Improving the initial solution into a final solution through meta heuristics is also also required before the algorithms can be implemented. In the development of the improvement heuristic, the concept of *time window compatibility* should again be introduced as a potential performance improvement tool.

A number of minor parameters in the calculation of the insertion and selection criteria have been taken directly from literature, based on their

relative performance. These parameters could be challenged to ensure that they contribute to the particularity of the proposed algorithm.

Initial solutions are only the first step in an optimization process such as the vehicle routing problems. The next step would be to create a number of unique (different) initial solutions. The value of having varying initial solutions is that the *Tabu Search* improvement meta heuristic, which is often used for vehicle routing problems, requires a set of unique initial solutions before being invoked. It is envisaged that a single customer will be removed from the original customer data list before invoking the initial solution algorithm. A simple insertion heuristic will then be used to insert the omitted customer into the generated solution after it is created. It is anticipated that if the process is repeated for different customers, different solutions will be generated.

5.6 Conclusion

City Logistics is concerned with the mobility of cities, and often aims to establish best practices and initiatives to improve the state of transport planning. The algorithm proposed in this dissertation contributes to the process of optimizing urban logistics as it proves that initiatives such as time windows and load factors can be planned for by shippers and carriers. The results prove that multiple variants of the vehicle routing problem can be integrated into the initial solution algorithm. The increased complexity is addressed by the time saving concept of *time window compatibility*, which proved to have a significant impact on both the computational burden, and the quality of the initial solution.