Chapter 4

Model definition

The purpose of the chapter is to depart on the first leg of the journey towards obtaining an initial solution to the extended vehicle routing problem, currently referred to as the Vehicle Routing Problem with Multiple Constraints (VRPMC). The model development process used to develop the model is taken from Taniguchi et al. [57] and is presented in figure 4.1.

Problem definition – The conceptual problem is defined in chapter 1.

Objective – As the model is concerned with determining an initial solution to a routing and scheduling problem, the result produced by the algorithm becomes the objective of the model. The choice of solution candidates are influenced by the mathematical objective function of the problem model.

Criteria – To elaborate on the criteria, a comprehensive mathematical model of the VRPMC is presented in section 4.1. The criteria define the solution space through multiple mathematical constraints.

System analysis – The analysis process involves identifying the essential components and interaction within the solution algorithm. Section 4.2.1 describe the interaction of the algorithm’s logical processes at a high level.

System synthesis – Although Taniguchi et al. [57] specify that this involves expressing the model in mathematical terms, it was already formulated in chapter 2, and presented in its entirety in section 4.1. Synthesis, in this dissertation, is the process of constructing and documenting a robust algorithm that will serve as direct input to the coding stage.

Software development – A computer based procedure will be developed in MATLAB. This will allow the mathematical and logical procedures, developed during the synthesis stage, to be used to produce actual
quantitative results. The software development is further discussed in chapter 5.

Verification – Procedures are tested and checked for correct logical structure. This iterative process makes use of manually simulated and calculated instances, and compares the algorithm’s output with its anticipated behavior.

Validation – At this stage the algorithm’s output is compared with published results. The objective of validation is to determine if the initial solution created by the algorithm is comparable with those generated by accepted algorithms, as the result should only be marginally better, or worse, than previously published results.

Application – The algorithm will be tested in parallel with current scheduling applications. Given the nature of the algorithm, and the fact that
the output is only an initial solution, and will act as an input to an optimization algorithm, the quality of the algorithm’s output can not be compared with that of the final algorithm.

4.1 The mathematical model definition

All variables and concepts are defined in chapter 2, and the mathematical model is therefore presented without any declaration of variable or explanation of constraint.

\[
\min z = \sum_{i=0}^{N} \sum_{j=0, j \neq i}^{N} \sum_{k=1}^{K} c_{ij}x_{ijk} + \sum_{k=1}^{K} \sum_{j=1}^{N} f_{k}x_{0jk} + \sum_{i=1}^{N} \alpha_{i} \times \max \{0, L_{i}\} \tag{4.1}
\]

subject to

\[
\sum_{j=1}^{N} x_{0jk} = \sum_{j=1}^{N} x_{j0k} = 1 \quad \forall k = \{1, 2, \ldots, K\} \tag{4.2}
\]

\[
\sum_{j=1}^{N} \sum_{k=1}^{K} x_{0jk} \leq K \tag{4.3}
\]

\[
\sum_{i=1, i \neq j}^{N} \sum_{k=1}^{K} x_{ijk} = 1 \quad \forall j = \{1, 2, \ldots, N\} \tag{4.4}
\]

\[
\sum_{j=1, j \neq i}^{N} \sum_{k=1}^{K} x_{ijk} = 1 \quad \forall i = \{1, 2, \ldots, N\} \tag{4.5}
\]

\[
\sum_{i=1}^{N} q_{i} \sum_{j=0, j \neq i}^{N} x_{ijk} \leq p_{k} \quad \forall k = \{1, 2, \ldots, K\} \tag{4.6}
\]

\[
a_{0} = w_{0} = s_{0} = 0 \tag{4.7}
\]

\[
\sum_{k=1}^{K} \sum_{l=0, l \neq i}^{N} x_{ijk}(a_{i} + w_{i} + s_{i} + t_{ij}) \leq a_{j} \quad \forall j = \{1, 2, \ldots, N\} \tag{4.8}
\]

\[
e_{i} \leq (a_{i} + w_{i}) \leq l_{i} \quad \forall i = \{1, 2, \ldots, N\} \tag{4.9}
\]

\[
x_{ijk} \in \{0, 1\} \tag{4.10}
\]
4.2 System analysis

It is the objective of this dissertation to promote the use of a systematic approach to model development, as opposed to the rapid-prototyping approach often experienced in practice. To ensure that the algorithm acts in a coherent and logical manner, the algorithm is modelled at various levels prior to being coded.

4.2.1 Overview

A graphical overview of the algorithm is presented in figure 4.2. The

![Diagram](image)

Figure 4.2: Overview of initial solution algorithm

number in the lower right-hand corner of a procedure, or decision in the
flowchart, refers to the sequence of discussions in the following subsection with regards to specific algorithm detail that are highlighted.

4.2.2 Algorithm detail

Sections of the overview model is represented using Structured English — a language and syntax, based on the relative strengths of structured programming, and natural English [64]. Structured English is not pseudocode, as it does not concern itself with the declaration and initialization of variables, linking, and other technical issues. The Structured English sections aims to communicate unambiguous logic about the algorithm which is easy to understand, yet not open to misinterpretation [3]. Readability takes preference over programming preferences. It is a strict and logical form of English, and the following constructs reflect structured programming:

- **Sequencing** shows the order of processing a group of instructions — simple, declarative sentences, following one another — without repetition and branching. Compound sentences are avoided, as they create ambiguity. Strong action verbs, such as GET, FIND, CALCULATE, UPDATE, SORT, etc. are used.

- **Selection or decision structure** facilitates the choice of actions under well-specified conditions. Variations of sequencing include:
  - The IF-THEN-ELSE construct specifies the actions that must be taken if a specific condition, or set of conditions, are all true.
  - The CASE construct is an elegant substitute for multiple IF-THEN-ELSE statements. The CASE construct is used where there are more than two sets of actions, based on well-specified conditions, to choose from.

- **Iteration or repetition** facilitates the same action, or set of actions, to be carried out a number of times. Two variations are
  - The REPEAT-UNTIL construct indicates that certain actions are repeated one or more times, based on the value of a stated condition.
  - The DO-WHILE construct indicates that certain actions are to be repeated zero, one, or more times, based on the value of a stated condition. Note that this construct need not be executed, as opposed to the REPEAT-UNTIL construct that will execute the set of actions at least once.

Blocking and indentation are used to indicate the beginning and end of constructs, as opposed to terms such as ENDIF, ENDCASE, ENDDO and ENDREPEAT, as these give the algorithm too much of a programming look.
and feel. Uppercase terms in the algorithm with italicized bold typeface indicate a variable set that is used in the coding of the algorithm. These sets are treated in square brackets in the document text, for example [VEHICLE] indicates the set of vehicles. The row numbers on the left indicate the line number in the complete algorithm. The algorithm presented in the dissertation is aggregated to eliminate unnecessary technical information helpful during the programming of the algorithm, hence the irregular numbering.

Figure 4.3 describes the capturing of input information. A list of all the technical field names appear in Appendix A.

Capture input information
1. Capture vehicle information in VEHICLES
2. Set average speed as 55 km/h
3. Sort available vehicles
4. Clear and set VEHAVAIL as an available vehicle matrix
5. for all available vehicles in VEHICLES
6. Add vehicle to VEHAVAIL
7. Sort VEHAVAIL in ascending order on <volumetric capacity>
8. Capture general CUSTOMER information
9. Capture customer information in CUSTOMER
10. for each entry, i, in CUSTOMER
11. if CUSTOMER has multiple time windows
12. Split customer into customer(i),n time artificial customers
13. Add artificial customer to ARTIF
14. Capture the time window information for each ARTIFicial customer
15. else
16. Add the CUSTOMER as a single ARTIFicial customer
17. Capture the time window information for the single ARTIFicial customer
18. Calculate the DISTance matrix between all the ARTIFicial nodes

Figure 4.3: Capture input information

The depot is captured as the first customer. If a customer specifies more than one time window, the customer is artificially split into n customers, each with a single time window, where n indicate the number of time windows specified. Once the customers are split artificially, reference will only be made to nodes – with each node indicating an artificial customer in the [ARTIF] set. Figure 4.4 describes the initialization process.

Initialise algorithm
1. Set the ROUTED matrix as empty
2. for all the ARTIFicial nodes, except the depot (node 1)
3. Add the ARTIFicial node to the UNROUTED matrix

Figure 4.4: Initialize algorithm

52
If there are vehicles available, a new tour is created. A tour can be made up of one or more routes. The initialization of a tour involves assigning the smallest available vehicle to the tour, and matching the tour capacity to that of the vehicle. This is indicated in figure 4.5.

```plaintext
44 Initialise TOUR
45 Set the TOUR index (t) to 1
46 Establish the starting time for the TOUR
47 Starting time for the current TOUR is \( t_d + s_d \)
48 (It is assumed that vehicles are not loaded at the beginning of the depot's time window)
49 Assign vehicle to TOUR
50 Set the first vehicle in VEHAVAIL as the current vehicle for the TOUR
51 Update vehicle availability
52 Locate the current vehicle in VEHICLE
53 Set vehicle(k).availability = 0
54 Recalculate VEHAVAIL
```

Figure 4.5: Initialize new tour

Once a tour has been created, one or more routes are established to make up the route. The iterative route creation process starts with the initialization of a new route. This entails assigning the route to the current tour, adding the depot as first and last node on the route, and identifying and inserting the seed customer: the first customer, other that the depot, to be added onto the route. The theory behind determining the seed customer has been elaborated upon in section 3.3. The algorithmic procedure for route initialization is indicated in figure 4.6. Nodes in the [UNROUTED] set are evaluated for insertion on the partially constructed route. The iterative route-building procedure is indicated in figure 4.7.

The concept of scheduling a vehicle to complete multiple routes (referred to as double scheduling), is difficult to implement in solution algorithms. The procedure followed in this dissertation to determine multi-route feasibility in a tour, is indicated in figure 4.8. When a vehicle returns to the depot at the end of a route, the multi-route feasibility check procedure determines if the depot’s time window is still open after the vehicle’s capacity has been replenished/renewed. It might be realistic to add some time to the potential route to allow the vehicle to at least service one node. The additional time added, conveniently referred to as minimum route time parameter, is different for each environment, and has been set to one hour in this dissertation. The effect will be that an empty route may be assigned to a number of tours when the initial solution is presented. To overcome the effect of empty routes, the final reporting procedure have been adapted to check for empty routes prior to reporting the initial solution. The procedure in indicated in figure 4.9.
**Figure 4.6: Initialize new route**
Expand partial ROUTE

while UNRouted is not empty and there are customers that fit into the current ROUTE

Clear the node selection matrix C2

for each UNRouted node (u)

Clear the node insertion matrix C1

Select the best position to insert node u on the current ROUTE

for each edge (i,j) on the current ROUTE

Determine feasibility to add node u

Infeasible if either TWC_i or TWC_j is unfeasible

Infeasible if TOUR capacity is exceeded by u

if it is feasible to evaluate node u between i and j

Update the C1 vector for the insertion positions

Calculate c_i(i,u,j)

Add the c_i(i,u,j) value to C1(i).value

else

Check next edge on current ROUTE

Select the best edge (i',j') based on the lowest C1 matrix value

Update the C2 matrix for the insertion position

Calculate c_j(i',u,j')

Add the c_j(i',u,j') value to the C2 matrix

Sort C2 in ascending order

Find first time-feasible node (u*), starting at the beginning of C2

While no u* has been found, and end of C2 has not been reached

Check for time feasibility

if feasible

Identify applicable node as u*

else

Check next element of C2

if a unique u* node has been identified

Insert node u*

Update UNRouted customers

Remove u* from UNRouted

Remove any other artificial nodes related to u* from UNRouted

Update ROUTE

Update ROUTE load

if new vehicle has been indicated

if Q''new > Q''

Find the smallest available vehicle to service Q''new

Update VEHAVAIL

Change the availability status of the current vehicle to available

Change the availability status of the new vehicle to unavailable

Assign new vehicle to current TOUR

Recalculate VEHAVAIL

Recalculate ROUTE schedule for nodes

Actual start-time at origin (s_n) is the start-time indicated for the current route

for each node (v) on the current ROUTE, except the depot at both ends

\[ a_v = \max\{s_1, s_{n1} + s_{12} + t_{12}\} \]

\[ w_v = \max\{0, t_{12}; (a_v + s_1 + t_{12})\} \]

Calculate actual arrival at the depot (n^n node) at the end of the current ROUTE

\[ a_n = a_{n1} + s_{n1} + t_{n1}\]

else

Initialize new ROUTE

Figure 4.7: Expand partially constructed route
**Expand TOUR**

Determine multi-route feasibility

- Check the actual arrival time at the depot of the previous ROUTE of the current TOUR ($a_k$)
  - if $a_k + s_k + 1 \text{ hour} < l_{k+1}^{new}$
    - then feasible
  - else
    - infeasible
- if feasible
  - Initialize new ROUTE
- else
  - if the last ROUTE of the TOUR has no nodes other than the depot
    - Eliminate ROUTE from TOUR
  - Initialize new TOUR

**Figure 4.8: Checking for multi-route feasibility**

**Define ORPHANS**

- if UNROUTED is not empty
  - Assign all elements in UNROUTED to ORPHANs
  - Clear UNROUTED

**Report initial solution**

- Calculate the Objective function value for the initial solution
- Report initial solution
- for each TOUR
  - Report all TOUR and ROUTE information

**Figure 4.9: Report initial solution**
4.3 Conclusion

The chapter introduced the model development process. The objectives and criteria are stipulated in the mathematical definition of the problem. This chapter elaborates on the *system analysis* and *synthesis*. The proposed initial solution algorithm is presented at a high level, with selective detail given in *Structured English*. The complete algorithm is presented in Appendix B. Chapter 5 discusses the implementation, and the results, of the proposed algorithm as coded in *MATLAB*. 