Chapter 5: A MTVRSPTW with Meal Break Consideration

5.1. Introduction

In this chapter, we present an insertion-based heuristic to solve Multiple Trips Vehicle Routing and Scheduling Problem with Time Windows and meal break consideration (MTVRSPTW-MB). Insertion heuristics are proven to be popular techniques for solving complicated combinatorial optimization problems with low computational effort. As a result, many commercial routing and scheduling software packages use this heuristic as their solution approach (Palmer et al, 2004).

The insertion algorithm was first implemented in 1977 to solve travelling salesmen problems (Campbell and Savelsbergh, 2004). In the following years, the insertion heuristic was widely used to solve VRP variants, such as multiple vehicle dial-a-ride problems (Jaw et al, 1986), VRPTW (Solomon, 1987), asymmetric capacitated VRP (Vigo, 1996), and a real-life application of transporting handicapped persons (Toth and Vigo, 1997). Furthermore, the insertion technique is also a typical choice in developing an initial solution for metaheuristic approaches, such as Tabu Search and Simulated Annealing (Bräysy and Gendreau, 2005).

Hunsaker and Savelsbergh (2002) and Campbell and Savelsbergh (2004) present an effective implementation of the insertion heuristic in a low complexity algorithm.

In recent works, Dessouky et al (2003) embedded an environmental component into the insertion algorithm to solve pickup and delivery problems with time window. A parallel regret insertion algorithm is proposed by Diana and Dessouky (2004) based on a case study at Los Angeles County. Lu and Dessouky (2006) present a new measurement criterion, the cost of reducing time window slack into classical insertion algorithm to solve a multi-vehicle pickup and delivery problem with time windows. Visual attractiveness is also quantified in the proposed insertion algorithm study, Ren et al (2010), a multi-shift VRP with overtime inspired by a healthcare delivery system. Tabu Search is embedded in the insertion algorithm framework for improvement. The empirical results show the improvement in terms of time and cost by the proposed solution.

To the best of our knowledge, no research has been done on MTVRSPTW-MB using the insertion heuristic. The most related paper, Campbell and Savelsbergh (2004), provides a brief description model of multiple trips per vehicle. This problem is different from our problem because they predefined the number of trips in a route and meal break is not

considered. Furthermore, the trip time limit in our model complicates the classical insertion implementation. Traditionally, the insertion heuristic schedules a set of customers in sequential order. In our problem, we need to break the tradition sequence into multiple trips with additional tasks in between, such as meal break and multiple travelling to the depot. Therefore, basic insertion heuristics will not necessarily produce a feasible solution in our case.

5.2. Problem Definition

We considered the MTVRSPTW-MB as defined in Chapter 2; however, synchronisation of loading teams is not included in this chapter. This constraint will be considered in Chapter 7.

In most of the delivery and pickup application, each customer required only one servicing team. Although the vehicle routing problem with split deliveries allows more than a vehicle to serve a customer, manpower synchronisation does not seem to be critical in most of the case studies. This might due to the holding cost and warehouse capacity at customers' site. Therefore, it is sensible not to include this constraint in this chapter.

5.2.1. Insertion Algorithm

In classical insertion approach, there is only one trip. Therefore, a decision has to be made on which new customer (aircraft, in our case) is to be inserted and where to insert this customer.

However, in our case, since there is more than one trip in a route and a meal break has to be inserted, additional factors need to be considered. The number of trips (a complete travelling cycle from service centre to apron) is not fixed which complicates the implementation further. An extra question the needs to be answered in such scenario is when to define a new trip? To overcome this, there seem to be two options: (a) insert all the customers in a trip, then pack the sequential customers into trips; (b) sequentially construct the trips by insertion. Both options can potentially provide the maximum savings by aggregating the customers in each trip, and achieve minimum trips required. However, option (a) could easily make the previous schedule become infeasible because customers' time windows would be not be met by adding the travelling tasks of newly defined trip at the packing stage. For option (b), we need to consider if the customer to be inserted into a partial trip or to form a new trip. This makes the evaluation process more complex and creates either

too few trips or too many trips in a route. Furthermore, when a customer is proposed to be inserted into any trip of the route, the feasibility check needs to go through every element/task from first trip to the last trip on the route. This is because an insertion will push the precedent trip to earlier time unit and forward the following trip to later time unit. By doing this, the computational effort will be greatly increased. On top of that, when the number of trips is not fixed, travelling tasks (between apron and service centre) can only be allocated along with the first aircraft insertion in a newly added trip. Therefore, potential insertion slots have become another important factor in our case study.

In addition, in our case, there are extra task requirements (the meal break and a few travelling tasks between apron and service centre) that need to be allocated; whereas, previous studies only schedule a sequence of customers. A meal break must be allocated between two consecutive travelling tasks, as they must be conducted at depot, within a given time window. Here, we need to consider when to insert the meal break appropriately. For example, if we only insert the meal break after constructing the trips, it will easily make the previous partial solution impractical due to time window constraints. On the other hand, it will have either too many or too few trips before the meal break if it is allocated before constructing trips for a route.

In this study, the Insertion Algorithm is proposed to construct a shift in three stages to cope with multiple travelling requirements. Initially, all aircraft are sorted in increasing order according to its arrival time, a_i, followed by their laxity, $b_i - p_i$. Then a new pair of decision variables, e_i and l_i, are defined as the earliest time to start service and the latest time to start service, respectively, to ensure the feasibility of solution.

The three stages of the insertion algorithm to form a route are defined as follows:

(a) *Define new shift*

A new shift is initiated by the aircraft on the top of the sorted list. A pair of travelling tasks is allocated together with the aircraft, travel to apron with truck and return to service centre as one complete trip. The first aircraft to be served, the shift time, the meal break, and the timeline for the first trip are defined accordingly. The shift time limit is set by using a dummy aircraft with zero maintenance processing time; meanwhile, the timeline for the first trip is set by the returning travelling task. A Time Window Table (TWT) is created to keep track of each task allocated associated with e_i and l_i. Then, the aircraft chosen in the TWT will be removed from the sorted list. Figure 5.1 shows the components to define a new shift and the implementation of it.

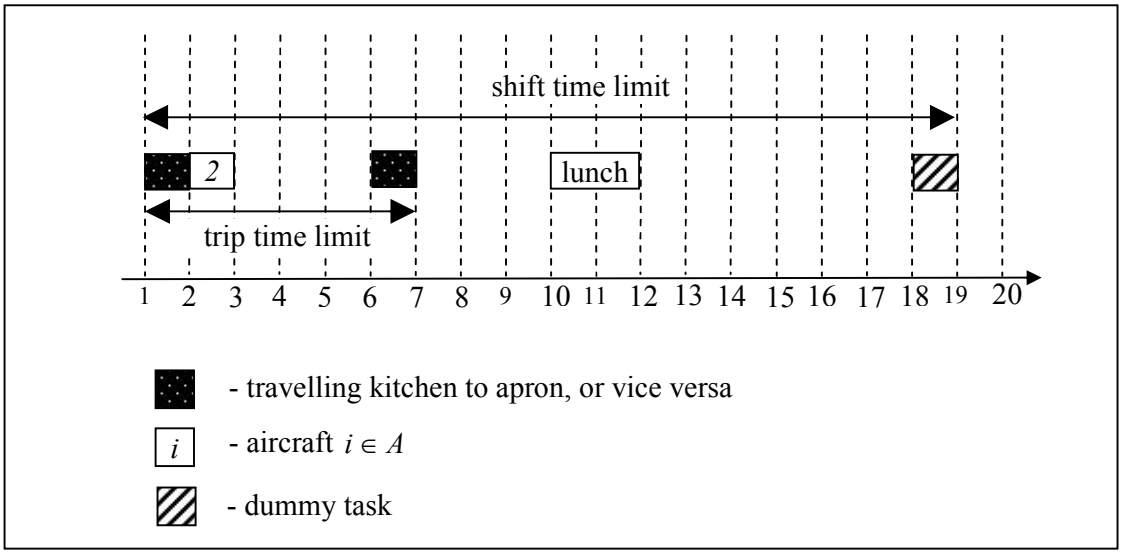


Figure 5.1. Components to define a new shift

(b) *Expand trip*

In this stage, the remaining aircraft in the sorted list will be evaluated for insertion into the newly defined trip. The potential insertion slots are restricted to slots after scheduled aircraft on the particular trip. In other words, the remaining aircrafts are not allowed to be inserted after any travelling tasks, meal breaks or dummy tasks. The best insertion slot, C_1 for each remaining aircraft, u, is selected by using the following equations:

$$C_1(u) = \min[\alpha c_1(u) + \beta c_2(u) + \gamma c_3(u)]; \quad (1)$$

$$\alpha + \beta + \gamma = 1; \quad (2)$$

$$0 \le \alpha \le 1,\ 0 \le \beta \le 1,\ 0 \le \gamma \le 1; \quad (3)$$

Equation (1) evaluates the maximum saving caused by the insertion, which includes distance reduction, c_1, shorter absolute waiting, c_2, and wider width of new time window, c_3. The time window cost, c_3, tends to offer the aircraft a wider time window due to the insertion. By adding this, it allows more aircraft insertions in future iterations. Parameters α, β, and γ are used to represent the weight of each "saving" criteria, based on the different objective functions or scenarios, as managerial parameters for managerial preferences. For example, the greater value of α on the objective function of minimising the total travelled distance, or the smaller value of β in the single travelling trip.

Next, the equation to evaluate the optimum aircraft, C_2 to be inserted is defined as follows:

$$C_2\left(i(u^*), u^*, j(u^*)\right) = \text{optimum}[\mu\, d_{0,u} + 2\theta\, p_0 - C_1(u)]; \quad (4)$$

$$\mu + \theta = 1; \quad (5)$$

$$\mu \geq 0,\ \theta \geq 0; \quad (6)$$

The first and second terms in (4) represent the distance and new pairs of travelling tasks required if the aircraft is served on a new trip. $d_{0,u}$ represents the travelling distance in time between depot to customer u ; meanwhile p_0 refers standard setup processing time at depot plus the travelling time returning to depot. The managerial parameter weight of each criterion is given as μ and θ, respectively. This stage will be repeated until no more feasible aircraft in the sorted list can be inserted into the particular trip.

(c) *Insert new trip*

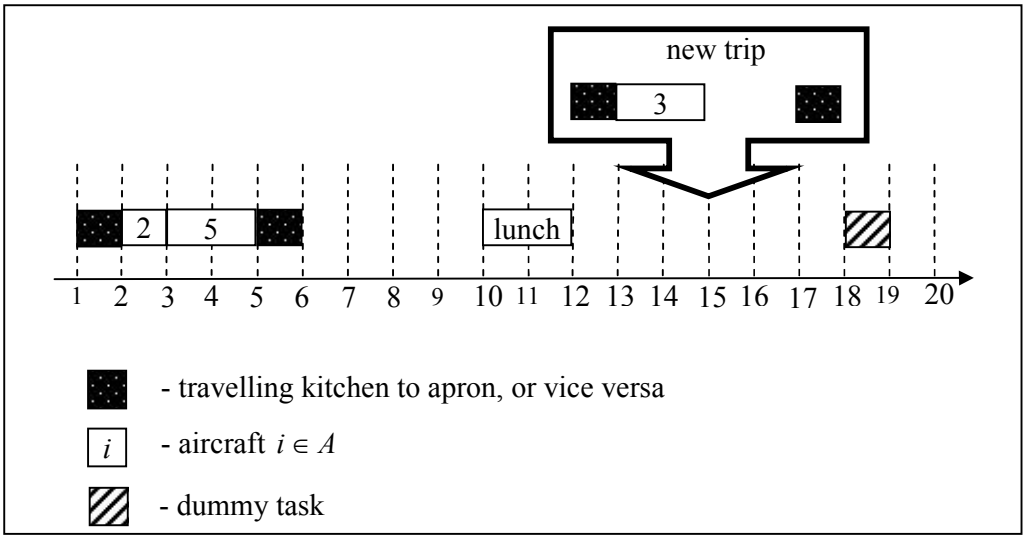


Figure 5.2. Components of new trip to be inserted

A travelling time is defined as the travelling activity between depot and the customer's site plus the standard setup time in the kitchen. The meal break, the dummy task and the travelling task are considered as non-aircraft tasks. A new trip is defined as a "block" of tasks including the best feasible aircraft and a pair of travelling tasks. The potential insertion slots for this new group of tasks are restricted to any slot between two non-aircraft tasks, as shown in Figure 5.2. An evaluation equation similar to Equation (1) is used to obtain

the most feasible aircraft to initiate a new trip. Then, go to stage (b). These stages will be repeated until no more feasible aircraft in the sorted list can be inserted into the route.

The e_i (earliest time to start service) and l_i (the latest time to start service) of each task in the route will be updated after each insertion, except l_i of the returning travelling task of the particular newly defined trip. This is due to its function which consists of setting the time limits of a trip. However, it will be updated at the end of stage (b) by $l_i = l_{i-1} + p_{i-1}$. The time window update of other tasks is as follows:

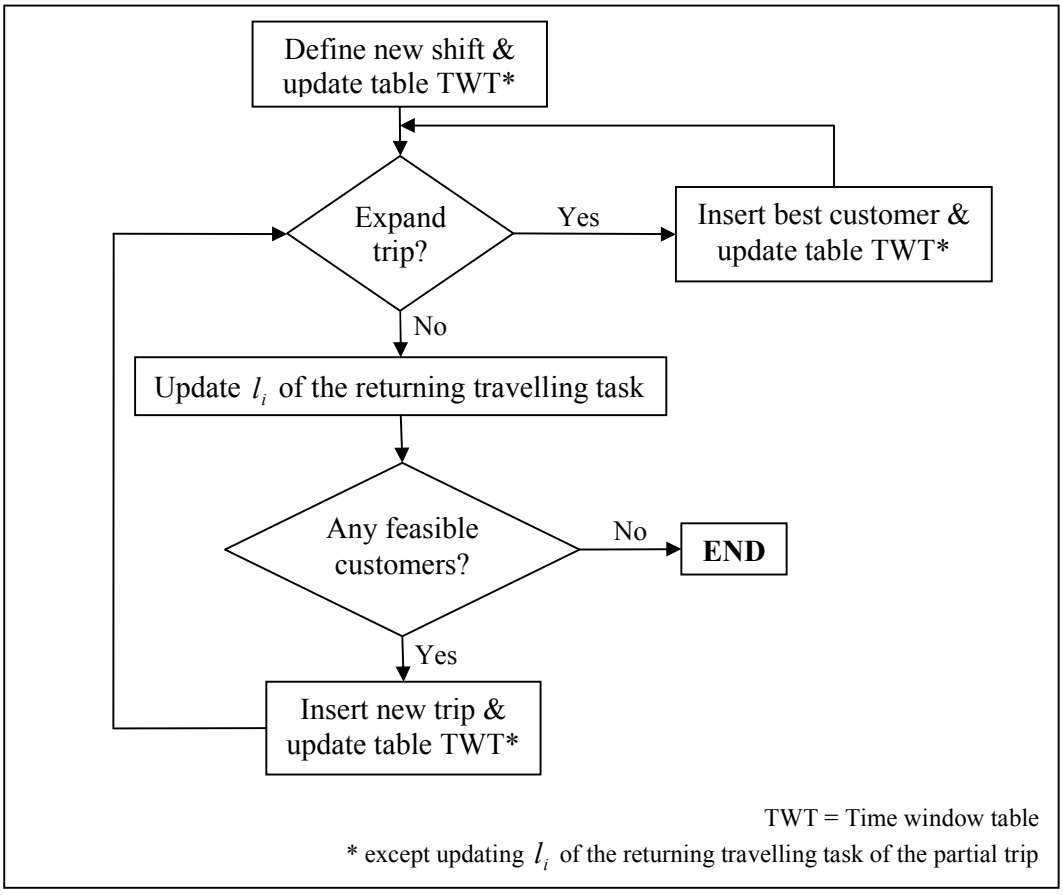
- Update the l values for all prior tasks from the new insertion backwards by,

$$l_i = \min(l_i, l_{i+1} - p_i) \tag{7}$$

- Update the e values for all subsequent tasks from new insertion forwards by,

$$e_i = \max(e_i, e_{i-1} + p_{i-1}) \tag{8}$$

The Insertion Algorithm is presented as flowchart below:



5.3. Performance Analysis

A five-day data set based on an in-flight caterer from Malaysia, was studied to evaluate the performance of the proposed solution. The data can be concluded as follows:

Day 1: 234 aircraft tasks

Day 2: 267 aircraft tasks

Day 3: 252 aircraft tasks

Day 4: 278 aircraft tasks

Day 5: 259 aircraft tasks

The heuristic was coded in C# and all experiments were carried out on a Dell Pentium 4 1.8GHz computer. All experimental tests can be computed in less than five seconds. Computational results with different parameter settings are given to demonstrate the robustness and efficiency of the proposed algorithm.

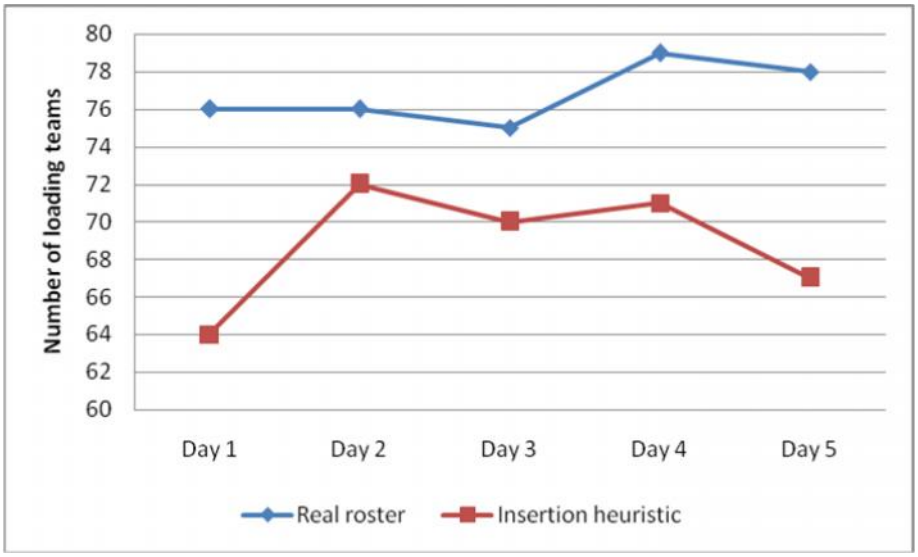


Figure 5.3: Comparison of the number of loading teams required between real roster and Insertion Algorithm solution

The in-flight caterer needs 75-79 loading teams each day within the five-day period with synchronisation of loading teams is applied. Under the same simulation setting,: an eight-hour working shift, a one-hour meal break, a two-hour trip time limit, two aircrafts per trip, and 15 minute kitchen-apron travelling times, Figure 5.3 shows the insertion heuristic is able to reduce the number of loading teams needed when the synchronisation of teams is relaxed. On average, the insertion heuristic reduced the amount of loading teams by eight each day, with a maximum of 12 loading teams on Day 1.

We ran a variation where meal break is not given. However, the working shift remains at eight hours. The number of loading teams required reduces further by 5.55% to 7.46%, as shown in Figure 5.4. This result makes sense as each loading team is given an extra hour to handle the services.

Figure 5.4: Meal break sensitivity analysis

We ran the same test data under different scenarios as follow:

(a) *Limit two travelling trips for each maintenance team*

Each maintenance team returns to the service centre to have a meal break after servicing one or more aircraft in a trip. After that, the teams go back to the apron for more maintenance work before they finish their shift. In this scenario, the meal break is given in the middle of the shift, between the third and sixth hours of working. Furthermore, the limited truck capacity and trip travelling time constraints are also relaxed.

(b) *Two travelling trips where no time restriction on meal break*

It is similar to scenario (a), but teams are given the flexibility to take a one-hour meal break anytime during their shift.

(c) *Single trip travelling*

In this scenario, the problem is reduced to the Vehicle Scheduling Problem with Time Window (VSPTW), where all aircraft must be inspected during their transit times. All

multiple travelling requirement constraints are relaxed, such as limited truck capacity, trip travelling time limits, and meal break allocation. However, the maximum working hours constraint is applied.

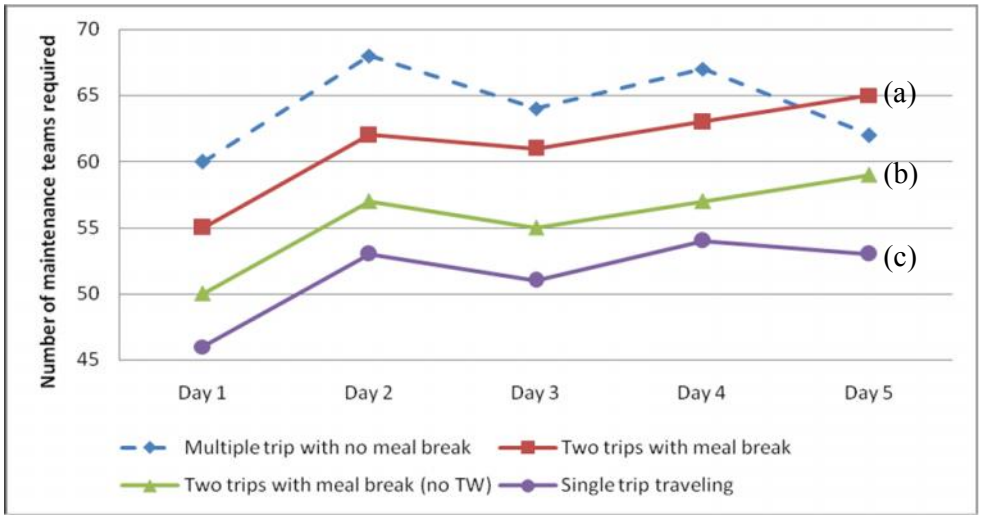


Figure 5.5: Sensivity analysis of Insertion algorithm

Figure 5.5 shows the sensitivity analysis result of the insertion algorithm for scenarios (a), (b), and (c). The dotted lines refer to the insertion solution of the original problem disregarding meal breaks. This sensitivity result demonstrates that the more operations constraints are relaxed, the fewer maintenance teams are required. This result is expected due to the fact that the travelled distance has been reduced accordingly. For example, when the truck limited capacity and the trip travelling time limits are relaxed, the travelling tasks between the apron and the service centre can also be reduced dramatically.

Overall, the manpower reduction increases from scenarios (a), (b), and (c) in a nearly fixed pattern, excluding Day 5 data. This is due to the unavoidable idle time caused by meal break allocation or its time window.

By comparing scenario (a) and (b), the manpower can be reduced by as much as 36.36% by relaxing the time window of meal break allocation. The analysis shows that the insertion algorithm can efficiently accommodate the meal break either at the beginning or at the end of the shift so that more maintenance tasks can be accomplished in a shift. This demonstrates the ability of the insertion algorithm to accommodate complicated operational constraints such as multiple trips travelling and meal break allocation.

More numerical analysis on the effectiveness and efficiency of insertion heuristic is conducted in Chapter 7.

5.4. Conclusion

In this chapter, the insertion heuristic is proposed to solve manpower scheduling for the in-flight catering delivery system, where loading team synchronisation is not considered. After aircraft tasks are sorted according to time window, the insertion heuristic is developed by constructing a trip by trip scenario due to multiple trip travelling requirements, while fulfilling the aircraft's time window.

Real data from an in-flight caterer in Malaysia was collected and tested on insertion heuristic. Computational results illustrates that the insertion heuristic can handle complex manpower scheduling, such as multiple trips travelling and meal break allocation efficiently and effectively. Furthermore, the insertion heuristic can generate large problem instances in seconds.

Chapter 6: A MTVRSPTW with Meal Break Consideration and Workforce Synchronisation

6.1. Introduction

MTVRSPTW with meal break and teams synchronisation is discussed in this chapter, motivated by in-flight catering delivery application in Chapter 2. Workforce synchronisation has never been considered in any MTVRSPTW in the literature. It turns out that solving the MTVRSPTW with meal break and workforce synchronisation consideration by using insertion heuristic is a difficult task, described in Chapter 5. This is because workforce synchronisation is rigid with respect to the service start time. Furthermore, it requires parallel insertion. Therefore, it would complicate the insertion process further on top of number of trips in not predefined and extra tasks requirement. Thus, a Two-Stage Scheduling Heuristic (TSH) is proposed to solve the problem in this chapter.

6.2. Two-Stage Scheduling Heuristic (TSH)

In TSH, Stage I focuses on the vehicle's capacity and trip travelling time limit. It is a greedy algorithm which first tries to assign jobs to vehicles that is currently servicing in apron, followed by vehicles which are idle in the depot. The least priority is given to new vehicles that have not service any customers. A synchronisation procedure is then performed to assign and schedule the vehicle and workforce accordingly. Trips by particular vehicles are formed and each trip is defined as a succession of operations. Then, trips are used as input to Stage II as "block" jobs. Stage II accommodates service teams' unavailability for reasons such as meal break allocation and the maximum working hour limit.

First, all customers are sorted in non-decreasing order of their release times. The secondary sorting is according to their laxity $(d_i - p_i)$. In the case of a tie between two or more customers, we schedule first the customer with the smaller processing time. Each customer will be assigned to service teams based on the resulting customer list. Let U denote the sorted customers in this study.

6.2.1 Stage I

In this stage, we define a set of trips, *H,* to be served by the fleet of vehicles. A new variable, $w_i^{r,v}$, is defined as the service start time of customer $i \in U$ serviced by vehicle

$v \in V$ on trip $r \in H$. Assuming the vehicles are able to work throughout the planning horizon, the main objective is to minimise the number of vehicles used.

A trip is defined as a complete travelling to and from the depot, and delivers and serves one or more customers during the travelling. The trips are in chronological order for each vehicle. Meanwhile, a route consists of one or more travelling trips to form a working shift for a worker. Thus, a vehicle serves trip *f* after trip *e* if, and only if, $e < f$.

A simple vehicle priority rule is used for assigning customers as follows:

(a) Customer site - Set *CS*

A vehicle located at a customer site and available to serve any unscheduled customers is given top priority. This is because of the possibility of accommodating more customers in a single trip.

Let z and z' be the last served customers on a partial trip and the partial trip on a given vehicle, respectively. After a feasibility evaluation, where the time window constraint and the vehicle limited capacity and travel time limitation are fulfilled, each unassigned customer will be assigned to any available vehicle, *k,* with the earliest end time of last customer served, i.e., $\min_k \left\{ w_z^{z',k} + p_z \right\}$. In case of a tie between two or more vehicles, we opt to use the vehicle with the earliest departure time from depot for the partial trip, i.e. $\min_k \left\{ w_{n+1}^{z',k} \right\}$. Following this approach, the idle time between two consecutive customers can be minimised.

(b) Depot – set *D*

Vehicles which returned to the depot after completing a trip will be considered for serving unscheduled customers if there is no feasible vehicle in group (a). A new trip is formed with the goal of serving the unscheduled customer by *k* feasible vehicles with the earliest end time of last returning travelling task, i.e. $\min_k \left\{ w_{n+2}^{s,k} \right\}$ on $s \in H$. This approach serves to minimise the idle time between two subsequent trips.

(c) New vehicle – set *NV*

A new vehicle which has not performed any previous trips is given the least priority. Initially, the fleet of vehicles is categorised in this group.

Let G_j^i be a set of feasible vehicles from $j \in \{CS, D\}$ to serve customer $i \in U$ we define the algorithm for Stage I below.

Stage I Algorithm:

For customer $i = 1, \cdots, n$, $i \in U$ do

Initiate feasible vehicle sets G_{CS}^i and G_D^i for customer *i* from *CS* and *D*

if ($G_{CS}^i \neq 0$)
Assign customer *i* to available vehicles with $\min_k \{w_z^{z',k} + p_z\}$, followed by $\min_k \{w_{n+1}^{z',k}\}$.

else if ($G_D^i \neq 0$)
Assign customer *i* to available vehicles with $\min_k \{w_{n+2}^{s,k}\}$.

else
Assign customer *i* to vehicle in set *NV*.
end if

Update $w_i^{r,k}$
end for-loop

A simple example of scheduling eight aircraft is illustrated in Figure 6.1 to demonstrate the procedures in Stage I. We have applied the same simulation setting as in the real roster: seven-unit trip time limit, two aircrafts per trip, and one-unit kitchen-apron travelling, in this example.

The aircraft are sorted according to their arrival times and laxity, as shown in Figure 6.1. Based on the sorted list, aircraft will be assigned to vehicles accordingly. Initially, all vehicles are in Set *NV*.

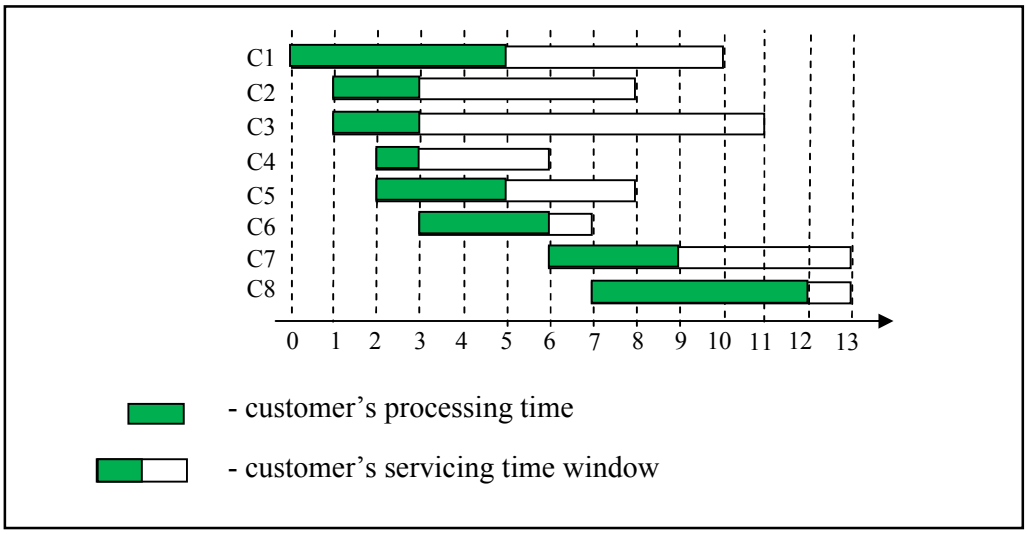


Figure 6.1. An example of eight aircrafts

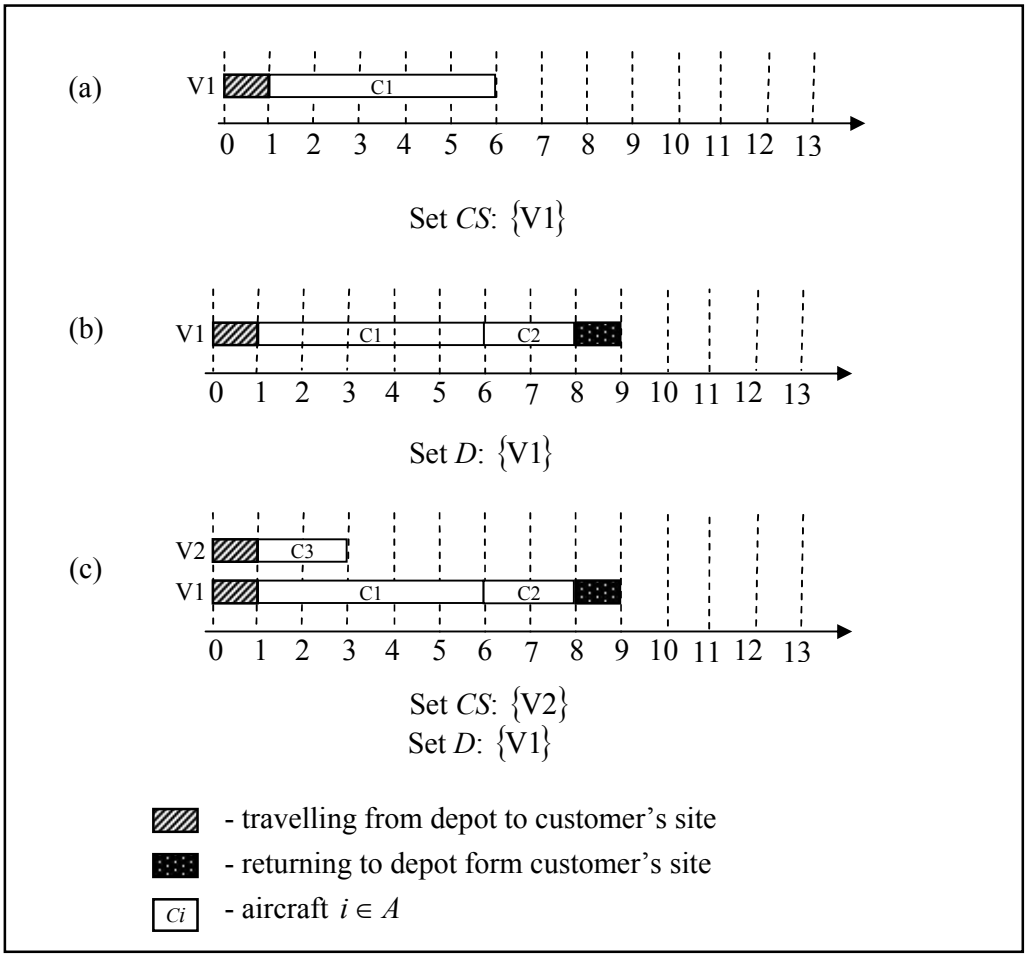


Figure 6.2. C1 assignment

C1 is first assigned to a vehicle based on the sorted list, as shown in Figure 6.2(a). Since set *CS* and set *D* are empty, C1 is assigned to a vehicle in set *NV*. The travelling task from depot to customer's site is also allocated accordingly for every newly defined trip. Then, C2 is assigned to V1 since it has top priority in vehicle groups as in Figure 6.2(b).VI has to return to depot after servicing C2 and categorised in set *D* because no more than two aircraft can be served in a trip. The travelling task of returning depot is allocated after C2.

As Figure 6.2(c), C3 is assigned to V2 from set *NV*, even though V1 has the higher priority than it. This is because assignment to V1 does not fulfil the time window constraint of C3.

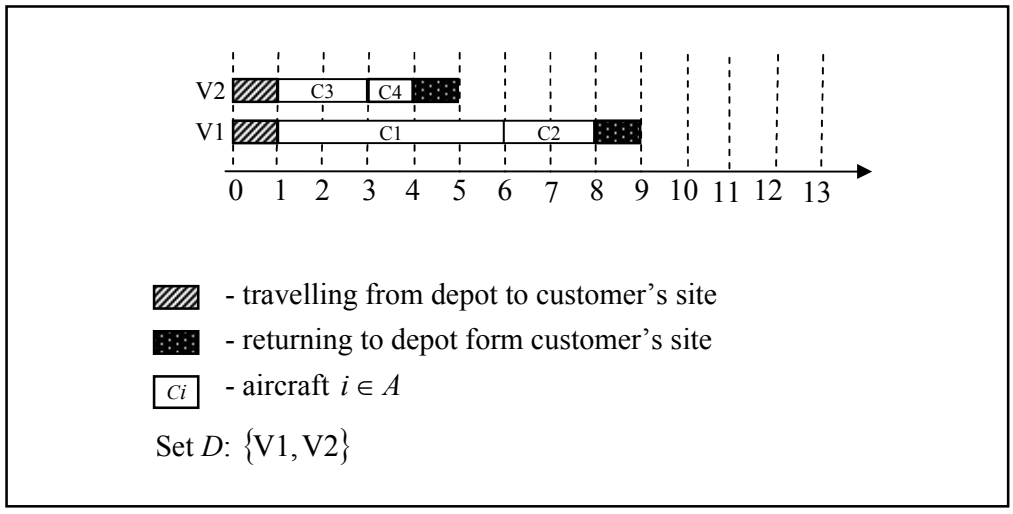


Figure 6.3. C4 assignment

In terms of assigning C4, it could feasibly be assigned to either V2 or a vehicle in set *NV*. Due to the vehicle priority rule, C4 is assigned to V2, which belongs to set *CS,* as shown in Figure 6.3. Again, travelling task is allocated at the end of each trip to return to depot.

A new vehicle is used to assign C5, as shown in Figure 6.4. There is no feasible vehicle in both set *D* and set *CS* to accommodate C5 in its tight time window. Similarly, a new vehicle is assigned for C6.

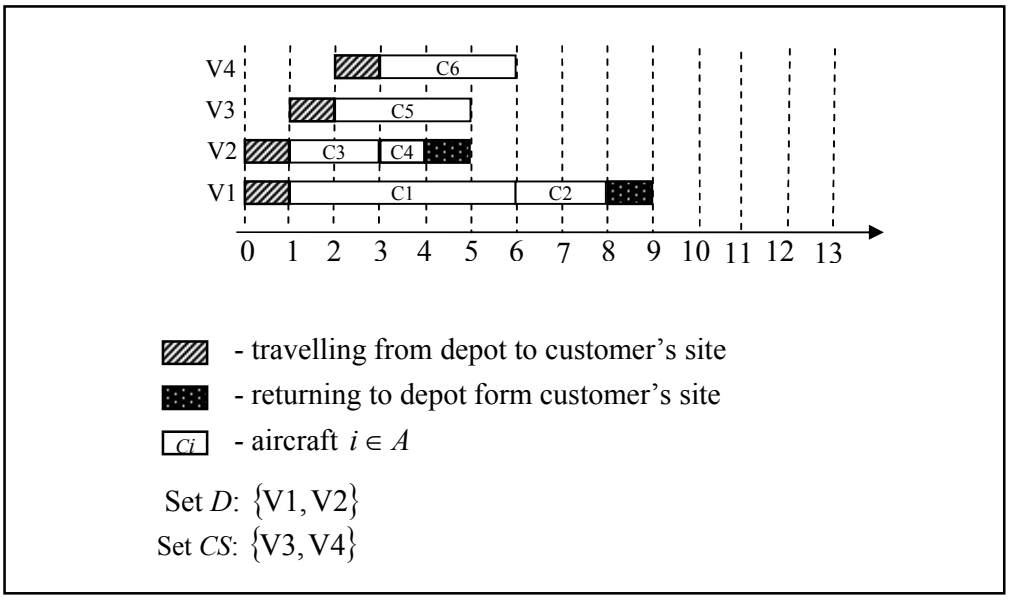


Figure 6.4. C6 assignment

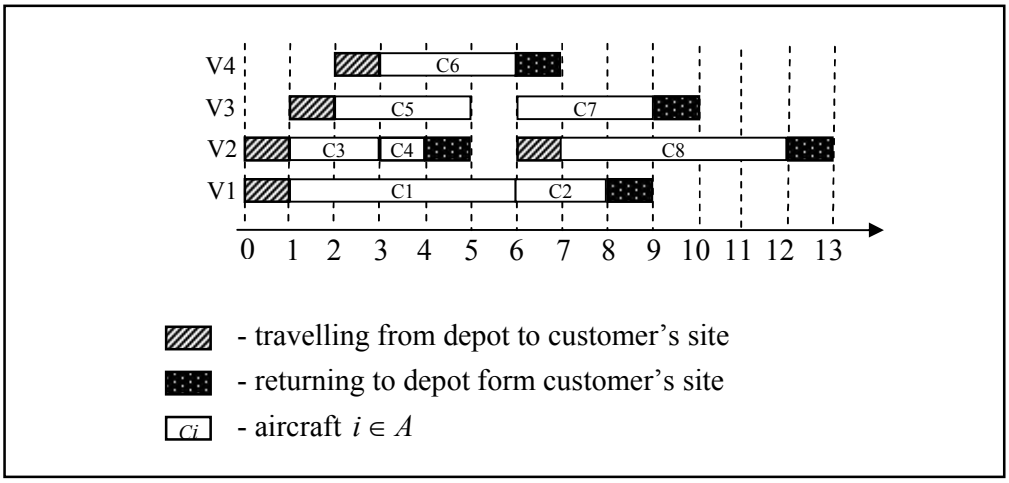


Figure 6.5. Stage 1 scheduling completed

C7 has two options of allocation, either at V3 or V4 as shown in Figure 6.4. Both fulfil the time window and vehicle capacity constraints. In this case of a tie between two vehicles, we opt to use the vehicle with the earliest departure time from depot for the partial trip, i.e. $\min_{k\in(V3,V4)}\left\{w_{n+1}^{z',k}\right\}$, which is V3. For C8, it could only be assign to V2 due to feasibility constraints. Finally, a complete Stage I scheduling for these eight aircraft is shown in Figure 6.5.

6.2.2. Synchronisation of Service Teams

When synchronisation of service teams is considered, a customer must be serviced by the different service teams during the same time interval. Despite the fact that this is a common operational constraint in many real-life systems, past studies have not considered it.

The procedure of teams' synchronisation is to generate a required number of vehicles based on the vehicle priority rule as Stage I. It might produce few different service start times. Then, the service start times will be adjusted to an identical service start time. For example, consider the case where a customer requires two loading teams. Two assigned vehicles with two start times will be generated by using vehicle priority rules. If the start times are different, the earlier start time will be push forward as the later one. However, we need to check the feasibility of the adjusted start time. If it is not feasible, the vehicle with late start time will be removed and reassign the next vehicle. This process will be repeated until the service start times are identical.

r_i is defined as the number of service teams required to serve customer *i*. Let a_i be the service start time for customer *i.* The algorithm used to synchronise the service teams is defined as follows:

For customer $i = 1, \cdots, n$, $i \in U$ do

Step 1 Use vehicle's priority rule in Stage I algorithm for r_i times to generate r_i vehicles without updating $w_i^{r,k}$ of customer *i* and vehicles in *CS* and *D*.

Step 2 Sort all the service start times of all a_i vehicles. Let *v* be vehicle with the latest service start time among all.

Step 3 Push forward service start time of other vehicles as *v*'s and check the feasibility conditions.

If a vehicle does not fulfil feasibility constraints, remove *v* and repeat Stage I algorithm once to generate a new truck, without updating $w_i^{r,k}$ of customer *i* and vehicles in *CS* and *D*.
Then, go to Step 2.

Or else, update each $w_i^{r,k}$ of customer *i* and vehicles in set *CS* and *D*.

end for-loop

6.2.3. Stage II

The main objective of Stage II is to minimise the number of service teams. The working shift times, together with a single break allocation, are assigned to each team.

Prior to this stage, each trip generated from Stage I is considered as a "block" of job, which is defined as a new set of jobs in Stage II. Each "block" job containing one service operation and several customer servicing tasks, is performed in succession. Without altering the service start time assigned in Stage I, we proceed by allocating these "block" jobs to the service teams.

At this stage, a greedy packing approach is used. We assign and schedule all the "block" jobs to service teams according to the earliest start time first policy. Ties are broken according to the smaller processing time first criterion.

In addition, we assign each service team a meal break. We treat the meal break as a distinct job (or as a "block" job) with its own time window. Service teams are assigned as many jobs as possible before l_{early} hours, the minimum number of working hours before a meal break is available. A meal break is given upon completion of a job where the job is completed within $\left[l_{early}, l_{late}\right]$. If a job is completed earlier than l_{early} but the next job completion time is more than l_{late}, then meal break is allocated exactly on l_{early} (creating an unavoidable idle period between the completion time of the last job before the break and the start time of the break). After assigning the meal break, service teams are again assigned as many jobs as possible until the maximum working hours are completed, T.

6.3. Performance Analysis

The TSH procedure was coded in C# and all experiments were carried out on a Dell Pentium IV 1.8GHz computer.

6.3.1. Case Study Analysis

Clearly, one would be interested in comparing the solution quality of TSH and current practice from the company. Again, we evaluated the performance of TSH for larger data sets with over 200 customers by using the same five-day data set as shown in section 4 in Chapter 5. In this case study, the aircraft play the role of customers in the MTVRSPTW-MB with loading teams' synchronisation consideration. The total number of aircraft and the total

number of split tasks when the synchronisation of loading teams is considered are illustrated in Table 6.1.

Table 6.1. Five-day data from case study

	Total aircrafts	No. tasks
Day 1	141	234
Day 2	151	267
Day 3	147	252
Day 4	158	278
Day 5	155	259

In Table 6.2, we provide the solutions obtained by TSH and compare them with the actual roster developed by the company's professional planner. The number of loading teams required by both solutions, the percentage improvement made by TSH, and the computing time (in seconds) of TSH are given. The manpower is reduced by 13.28% on average compared to the actual roster. In addition, the loading teams are synchronised to carry out the servicing operations.

TSH requires less than three seconds of computational time, where most of the computational time is mainly spent on scheduling customers at synchronisation procedures in Stage I. This demonstrates that TSH can solve the MTVRSPTW-MB with workforce synchronisation consideration efficiently and effectively for large data sets.

Table 6.2. Comparison result of TSH and real roster on the number of loading teams

	Real	TSH	Improve (%)	CPU (sec)
Day 1	76	60	21.05	3
Day 2	76	72	5.41	3
Day 3	75	61	18.67	3
Day 4	79	71	10.13	3
Day 5	78	69	11.54	3
Total	384	333	13.28	

Since the raw data from the company consists of varying types of time windows, we propose to pre-process the data with Time Window Reduction (TWR), which is described in

Chapter 4. From Table 6.3, it follows that the use of manpower can be reduced further by 7.81%, on average, when TWR is performed prior to TSH. In total, 26 teams, i.e. 208 man-hours can be saved. The best solution of TWR with TSH is chosen among different values of $W = \{1,2,3,5,10\}$. This result makes sense as the time window reduction is based on the idea of spreading the customers as evenly as possible throughout the planning horizon.

Table 6.3. Computational result when TWR is applied

	TSH	TWR + TSH	Improve (%)	CPU (sec)
Day 1	60	54	10	3
Day 2	72	65	9.72	3
Day 3	61	59	3.28	3
Day 4	71	67	5.63	3
Day 5	69	62	10.14	3
Total	333	307	7.81	

The computational time of the whole process remains less than three seconds. Clearly, the execution time of TWR is negligible compared to that of TSH.

6.3.2. Synchronisation of Loading Teams Relaxed

When synchronisation of loading teams is not considered in the model, different loading teams are allowed to serve the particular aircraft at different time intervals. In some cases, a loading team might serve the same aircraft more than once, if the aircraft has a sufficiently wide time window and $r_i > 1$. This flexibility leads to a further reduction in the number of teams required, as demonstrated in Table 6.4.

Table 6.4. Computational result of TSH when synchronisation of loading teams is relaxed

	TSH	TSH- Team	Reduce (%)	CPU (sec)
Day 1	60	58	3.33	3
Day 2	72	64	11.11	3
Day 3	61	56	8.20	3
Day 4	71	68	4.23	3
Day 5	69	63	8.70	3

Total	333	309	7.21	

#TSH-Team = TSH where synchronisation of loading teams is relaxed

On average, there is a 7.21% reduction in the number of teams required when loading teams' synchronisation is not required compare with the case with synchronisation. These results clearly suggest that TSH can easily be adapted to different requirements and operational constraints.

Table 6.5 provides computational results when TWR is performed prior to TSH for the case where synchronisation of loading teams is relaxed. When TWR is applied, the manpower can be reduced approximately 2.92% in total. The reduction is relatively small compared to the previous experiment. This is due to the flexibility that loading teams possess in serving aircraft anytime within their respective time window.

Table 6.5. Computational result when TWR is applied

	TSH-team	TWR + TSH-team	Reduce (%)	CPU (sec)
Day 1	58	55	5.17	3
Day 2	64	62	3.13	3
Day 3	56	56	0	3
Day 4	68	65	4.41	3
Day 5	63	62	1.59	3
Total	309	300	2.92	

6.3.3. Meal Break Sensitivity

In order to examine the impact of the meal break together with loading teams' synchronisation, we ran the same test data with TSH by disregarding the meal break allocation. The sensitivity results are summarized in Table 6.6.

Table 6.6. Meal break sensitivity analysis

	TSH	TSH-Lunch	Reduce (%)	CPU (sec)
Day 1	60	53	11.67	3
Day 2	72	66	8.33	3
Day 3	61	56	8.20	3
Day 4	71	68	4.23	3

Day 5	69	61	11.59	3
Total	333	304	8.71	

TSH-Lunch = TSH where meal break constraint is relaxed

The increase in the number of teams is due to the fact that the mandatory meal break is considered as an additional job. Therefore, the shift times are now reduced by the break length. Furthermore, due to the time window constraint on the break, it is more likely that unavoidable idle time before and after the break is created. Overall, the results are very satisfactory, since that increase in the number of teams is almost equivalent to the increase in the number of tasks' working hours divided with shift working hours.

6.3.4. Analysis of the Relaxed-MTVRSTW

Since we are the first to study the problem, there is no exact, or even heuristic solution for the problem studied in literature, we could not provide a performance gap with respect to the optimal solution. However, in order to get a sense of how the TSH performs, besides comparing it with the real life data given in previous section, we compare with the solution given by CPLEX for the two trips VRPTW with meal break consideration, as in Chapter 3.

Table 6.7 below provides the number of teams and the computational time for different instances. The CPLEX solution and the TSH heuristic are compared for the case where W is set to 5. The computational (CPU) time of TSH is less than one second for all instances considered.

Table 6.7. Comparison of performance by CPLEX and TSH

No.	TSH	CPLEX	
10	4 teams	2 teams	6 sec
15	5 teams	3 teams	2,608 sec
20	6 teams	6 teams	57,615 sec
25	7 teams	7 teams	100,845 sec
30	9 teams	9 teams	86,400 sec

Since our objective is to develop a heuristic that can efficiently and quickly construct a quality solution, we are not attempting to outperform the CPLEX solution. First, the CPLEX algorithm (branch and bound in Column Generation framework) is significantly

more complex than our approach. Even though TSH did not perform well for 10 and 15 customers, it matches CPLEX for larger problem instances. Furthermore, CPLEX proves to be very sensitive to the size of the data set and requires exponentially increasing computation time as the number of tasks increases (over 15 hours are required to solve instances of 15 to 20 customers), whereas TSH is computationally bounded.

6.4. Conclusion

In this chapter, we developed an original, Two-Stage Scheduling Heuristic (TSH) for solving the MTVRSPTW with meal break and workforce synchronisation consideration. The proposed heuristic is a computationally bounded heuristic and is shown to generate very high quality solutions in a short time. Furthermore, TSH is easily adaptable to different operational environments and can accommodate even the most complicated logistical constraints, such as loading team synchronisation and multiple vehicle trips.

TSH was compared to the actual roster used by a commercial airline operator. The computational results verify that TSH is a superior tool in solving MTVRSTW-MB, both in terms of solution quality and computing time, even when workforce synchronisation is applied. Besides, a comparison between TSH and solutions obtained by CPLEX was undertaken for the two trips VRPTW with meal break consideration. The time window reduction algorithm which is used to pre-process the tasks is shown to further improve the performance of the TSH procedure as it allows smoothing the demand for tasks over the working day.