Proceedings of the Twentieth Innovative Applications of Artificial Intelligence Conference (2008)

A Vehicle Routing System to Solve a Periodic Vehicle Routing Problem for a Food Chain in Hong Kong

Jianfeng Zhu[1], Wenbin Zhu[2], Chan Hou Che[2], Andrew Lim[1,2,3]

[1]School of Computer Science & Engineering, South China University of Technology, Guangzhou, China
ooniki@gmail.com

[2]Red Jasper Limited, Hong Kong Science Park, Shatin
No. 5 Science Park West Ave, Unit 210, 2/F, New Territories, Hong Kong
{zhuwb@redjasper.com, oscar@redjasper.com}

[3]Dept of Industrial Engineering and Logistics Management, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
iealim@ust.hk

Abstract

In this paper, we describe the system that we have developed to solve a new variant of the periodic vehicle routing problem with time windows (PVRPTW) for one of the largest food and restaurant chains in Hong Kong. The extension is to limit the number of drivers that any store should see during the fixed period. We name this constraint as limited visiting quota (LVQ). We devise a new method to solve this problem. Our experimental results indicate that our method is able to reduce the number of vehicles used by 23% and thus bring substantial savings to our client. The solver has been integrated into an existing vehicle routing product called VROOM for the daily usage of our client.

Introduction

The problem in this paper is derived from a project that our team did for one of the largest food and restaurant chains in Hong Kong. Our client has a few hundreds stores located all over Hong Kong. It does most of its delivery using its fleet of vehicles. The problem that we are addressing is its morning delivery of bread, cakes and buns, drinks to its stores. Due to the high rental costs in Hong Kong and the Hong Kongers' demand for fresh food, deliveries must be made several times a day. The first delivery in the morning is the largest and as it has to meet specific time windows, i.e. before the opening of each store and not later than a specific time. Every night the manager of each store will make the decision on what to order for the next day. The demand may vary substantially over the course of seven days, i.e. the demand during the weekend (Saturday and Sunday) is substantially different from the demands during weekdays. The demand on a Monday may be quite different from the demand on a Friday. In addition, in order to facilitate effective communication and co-ordination between the drivers and store employees, it is also important that the drivers must not be assigned substantially different routings each day and the store employees should not see a new delivery person every other day. Furthermore, as most of the retail stores as located in subway stations (MTR), train stations (KCR), and shopping malls, where finding parking spaces and store location can be a challenge for someone new. Hence, it is important not to assign drivers to new delivery locations frequently.

As a result, the company used a fixed route approach to facilitate their morning delivery. The fixed route approach means that every driver services a fixed number of delivery points. This human based solution is easy to implement but can be extremely inefficient due to the variation of the daily demands and the complexity of the vehicle routing problem with time windows (VRPTW). What it really need is an approach that considers the demand variations across a fixed period together with the limited visiting quota (LVQ) constraint. We named this problem the Periodic Vehicle Routing Problem with Time Windows with Limited Visiting Quota (PVRPTW-LVQ).

Periodic Vehicle Routing Problem with Time Window (PVRPTW) is a variant of VRPTW. Cordeau et al (Cordeau, Laporte, and Mercier 2001) first proposed this problem. Instead of the single period (or single day) problem in VRPTW, the PVRPTW has T days. In addition, each customer has a service frequency and a set of allowable combinations of the visit days. The problem is to select a feasible combination of visit days for each customer to minimize the cost without violating the constraints of the VRPTW.

There are some differences between PVRPTW and the problem what we discuss here:

A. The service frequency of each customer is equal to the number of days of period. It means all the customer has to be visited everyday. So we may not consider the allowable combinations.

B. Every customer has to be visited by no more than R different drivers in the period. (We call it *R-constraint*).

C. For each customer, the location, time window, service time is fixed everyday. But the demand of each customer may be different everyday.

D. The objective is different. Most of the objectives of PVRP and its related variants are to minimize the total traveling cost. In our problem, the objective mainly focused on minimizing the drivers number (crew size) used in the period, and then followed by the number of vehicles used, and finally, the total travel distance.

Formulation of the Problem

In our problem, there is only one depot. The depot is the bakery and the warehouse which the goods are made and stored. As the company has a large number of vehicles now which exceeds what is needed in any good solution, we assume that there is an "unlimited" supply of homogenous vehicles.

Every vehicle has a capacity Q. There is a service period of T days. Every day, the vehicles start from the depot, visit a list of customers and return back to the depot. Every customer i has a time window which has a ready time e_i and due date l_i. Every customer has a service time s_i. The location, time window and service time of the customer i are all fixed over the period. On day t, every customer i has a service demand q_i^t. From customer i to customer j has a travel distance d_{ij} and a travel time t_{ij}, and we assume $d_{ij}=t_{ij}$ in the following. There is a limited visiting quota (LVQ) or the R-Constraints, R. The objectives and constraints are summarized below:

Objectives:

- Minimize the number of drivers needed for the period
- For the same number of drivers, minimize the number of vehicles to serve all the customers of the period
- For the same number of routes, minimize the total travel distance of the period

Subject to:

- For each day of the period, every vehicle starts from depot after e_0 and must return to depot before l_0 at the end
- For each day of the period, every customer should not be serviced before the ready time e, and cannot be serviced after the due date l
- For each day of the period, every customer should be serviced by exactly one vehicle
- The total demand of customers in the same route cannot exceed the capacity Q of the vehicle
- Every customer can be serviced by no more than R different drivers in the period

Our Method

The basic idea of our method is to decompose the drivers into at most R subspaces. Then we allocate the customers to a subspace for each day. The algorithm can be summarized as follows: First, we can compute a demand for each customer based its demands for the entire period then the problem becomes a classic VRPTW. A master solution called *Base Solution* can be generated. Second, we will check the feasibility of every route of *Base Solution* for each day; we may need to kick out some customers into the *kick list* to ensure feasibility. The remaining customers on the routes become the solution of the specified subspace.

We use the same operations to handle the customers in the kick list, more subspaces will be generated until it reaches the terminating condition. The solutions of all subspaces make up the final solution of the problem.

One may think that we can solve the problem by solving the problem for each day as a single day VRPTW problem, and the final solution is made up of these daily solutions. But such an approach cannot ensure *R-constraint* is satisfied.

We use a small example to explain the method we devised in this paper:

- There are 9 customers
- Customer attributes such as time window, location and service time, are ignored. In our example, we only handle the demand constraint, the capacity constraint, and the *R-constraint*
- Q=5 (the vehicle capacity is 5)
- T=3 (the period is 3 days)
- R=2 (all the customers can be serviced by no more than 2 different drivers)
- Assume total number of drivers and total number of vehicles to be infinite
- The daily demand is shown as Table 1 below

Daily Demand			
Customer	Day1	Day2	Day3
1	3	2	2
2	3	1	1
3	3	2	2
4	2	2	2
5	1	1	2
6	1	3	2
7	2	3	2
8	2	2	3.5
9	1	1	3.5

Table 1: *Daily demand of the example*

Suppose the final solution of the example looks like the structure in Figure 1; Five drivers are needed. We consider every driver take charge of one route over the period, so 'R_i' means the route taken charge by driver i. Every route composes of some customer nodes, and the driver will start from the depot and visit his customers according the sequence of the route, and goes back to the depot at last everyday. The empty routes mean the drivers who take charge of these routes needn't to deliver goods for those specified days.

To handle the *R-constraint*, two rules are set up:

Rule1: The problem space can be decomposed into at most *R* subspaces. In this example, we can decompose the 5 drivers into 2 subspaces as shown in Figure 1.

Rule2: For each day, every customer can locate in one of the subspaces, but it can only be located in the same route for the same subspace over the entire period. For example, customer 1 can locate in R4 of *Subspace2* (as Day1), and it can also locate in R1 of *Subspace1* (as Day2), but when determining the route of Day3, customer 1 has only 2 choices: R1 of *Subspace1* or R4 of *Subspace2*.

Under these 2 rules, all the allocations can ensure the final solution satisfy the *R-Constraint*.

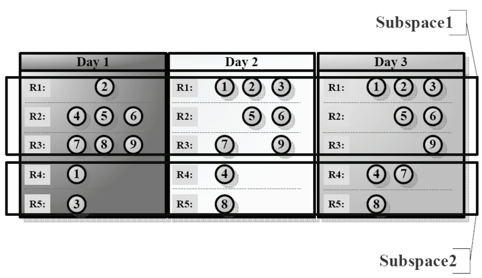

Figure 1: *Final solution structure*

What we need to decide is how to decompose the drivers into subspaces and how to allocate the customers to drivers.

The *R* subspaces are established incrementally through the creation of *base solutions*. To get the first base solution, we select a demand for every customer. The value of this demand is based on the customer's demands over the entire period. With this demand value, this sub-problem becomes the standard VRPTW. Using the solver of VRPTW by (Lim and Zhang 2007), we can generate the first *Base Solution* for the problem.

The important question now is what is a suitable and representative demand for each customer? If we select the largest demand in for the entire period for every customer to generate the *Base Solution*, routes in the based solution will have no constraint violation for each day of the entire period. However, such an approach is excessive. It is similar to the fixed-route approach where each customer will only see one driver for the entire period, and the driver will only visit a fixed set of customer for the entire period. Therefore, such a selection criteria is not ideal.

If we select demand value for each customer to be a value in between the minimum value and the maximum value of that customer during the entire period to generate the *Base Solution,* some of the routes may not be feasible for some days. The strategy of demand selection will be discussed in a later section.

To illustrate the basic idea of our approach, we provide an example. In this example, we choose the median demand for each customer, and let's assume the following *Base Solution* is generated as the table on the left of Figure 2. We call this *Base Solution* as *Base Solution1*.

After checking the total demand of the routes of *Base Solution1*, we find that:

- For Day1, R1 is invalid. (total load = 9> Q)
- For Day2, R2 and R3 are both invalid
- For Day3, R2 and R3 are both invalid

For each day, we can kick out some customers to make each route feasible. However, in the "kick-out" process, we must ensure that every customer must not be totally kicked out from this subspace. As an example, we can continually kick out the customers with large average demand for every infeasible route until the route becomes feasible (see Figure 2). We shall discuss the kick strategy in a later section.

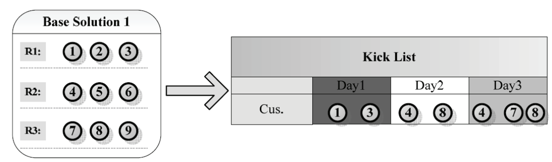

Figure 2: *Kick out from Base Solution1*

Now we can form feasible daily solutions by removing some customers, i.e. those who have been kicked (see Figure 3 for more details). We call the combination of *Day1 Solution1*, *Day2 Solution1* and *Day3 Solution1* as *Solution1*. *Solution1* is the solution of *Subspace1*. This however is not the final solution because not all customers are serviced for all the days in the period.

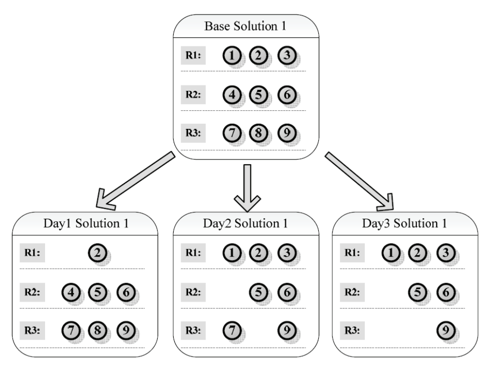

Figure 3: *Solution1*

After earlier step, *Subspace1* is formed. It contains 3 drivers, who take charge of R1, R2 and R3 respectively.

We have to generate another subspace (called *Subspace2*) to service the customers who haven't been visited. We can apply the same method to generate *Subspace2*: first generate the *Base Solution2* and then form the daily solution. *Base Solution2* contains all the customers who appear in the kick list and some of these customers may not appear in all days. In this example, *Base Solution2* should contain customer 1,3,4,7 and 8, but customer 1 just appears in Day1. If R=2, *Subspace2* is the final subspace we can use, therefore no customer in this subspace can be kicked out. (if any customer is kicked out from this subspace, a new subspace has to be created, and this will violate the *R-Constraint*). To satisfy this condition, we may choose the maximum demand of the period for all the customers on the service list to form *Base Solution2*.

Looking into the 'kick list' in Figure 4, we can find that the customers of Day1 (i.e. customer1 and 3) have no demand in Day2 and Day3, thus no overlap with customers 4, 7, and 8. As such, we can use the customers in the kick list of Day1 to form a base solution called *Base Solution2*$_1$, and use customers in the kick list of Day2 and Day3 (i.e. customers 4, 7 and 8) to generate *Base Solution2*$_2$. And then form the daily solution respectively. The process above separates *Subspace2* into two smaller but non intersecting sub-subspaces.

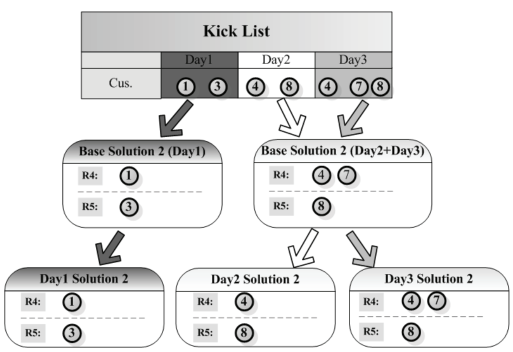

Figure 4: *Flow of generating Subspace2*

After *Subspace2* that comprises of *Subspace2*$_1$ and *Subspace2*$_2$ is generated, the problem is solved. The final solution is make up of *Solution1* (i.e. solution of *Subspace1*) and *Solution2* (i.e. solution of *Subspace2*) (see Figure 5).

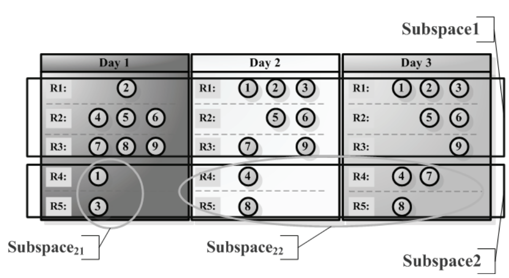

Figure 5: *Final Solution*

R=2 this example, the algorithm can extend to R>2. The flow chart of the process is shown in Figure 6 and the main idea of the algorithm is presented in Algorithm 1 below:

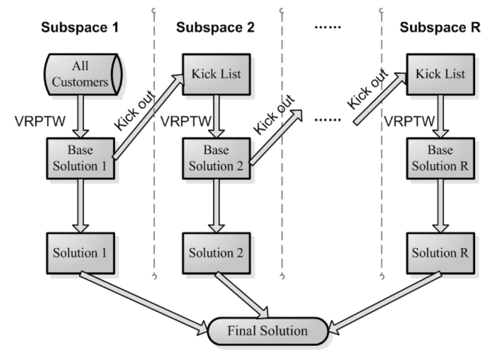

Figure 6: *Flow chart*

Algorithm 1

Put all the customers into kick list of every day;
for i =1 to R
    for j =1 to number of sub-subspaces of *Subspace*$_i$
        if (kick list of *Subspace*$_{ij}$ is not empty)
            Use ***Strategy1*** to generate *Base Solution*$_{ij}$ from the kick list of *Subspace*$_{ij}$;
            Use ***Strategy2*** to kick out the customers from *Base Solution*$_{ij}$ and form new kick list of relative days of *Subspace*$_{i+1}$;
            Form *Solution*$_{ij}$;
        end if
    end for
    Form *Solution*$_i$ and ***Post-Optimize*** (*Solution*$_i$);
end for
Form the *Final Solution*;

Algorithm 1: *Overall Algorithm*

The remaining problem is to design *Strategy1* and *Strategy2*. *Strategy1* is a strategy about selecting the demand for each customer. *Strategy2* is to decide which customer of the invalid route should be ejected to the kick list. We devise some methods for both of the strategies and attempt to obtain the best combination by comparing the results.

Strategy1

Use Threshold to Select the Set of Demand (TH). The method is defined as below:

$$Dem = \{q_i \mid q_i = q_{i\_min} + (q_{i\_max} - q_{i\_min}) \cdot \lambda, \quad i \in C\}$$

- minimal demand in the period of customer i, q_{i_min}
- maximal demand in the period of customer i, q_{i_max}
- parameter λ, $0 < \lambda < 1$
- the chosen demand of customer i, q_i
- the set of chosen demand, Dem
- the set of customers, C

Use the Demand of Single Day to Form the Set of Demand (DS). For each customer, this method selects a representative demand based among all the demands in that period for that particular customer. These demand values are used to generate the *Base Solution*.

Strategy2

Kick Out Greedily (GR). For each day, kick out the customer with maximum average demand from every invalid route until the route becomes valid.

Use Integer Programming to Select the Customers to Be Kicked Out (IP). Firstly, we use Model1 to select which customers should be kicked out during the period, and the results make up the *candidate list*. The objective of Model1 is to minimize the number of kicked out customers in the period.

$$\min \left\{ Q\sum_{i\in K}\sum_{j\in P_i} x_{ij} + \sum_{i\in K}\sum_{j\in P_i} \max_{t\in T_{in}} q_{ij}{}^{t} x_{ij} \right\}$$

$$s.t.\ \sum_{j\in P_i} x_{ij} q_{ij}{}^{t} \geq EQ_i^t \qquad \forall i\in K, \forall t\in T_{in}$$

$$x_{ij}\in\{0,1\} \qquad \forall i\in K, \forall j\in P_i$$

Model 1

- capacity of the vehicle, Q
- the days which contains invalid routes, T_{in}
- overload of route i of day t, EQ_i^t
- route set of the *Base Solution*, K
- customers of route i, P_i
- demand of the customer j in route i of Day t, q_{ij}^t
- x_{ij}=1 if the customer j of route i is ejected into candidate list, otherwise

Secondly, we use Model2 to determine which customer should be kicked out everyday. The kicked out customer are the members of *candidate list*. The objective of Model2 is to minimize the number of kicked out customer everyday.

$$\min \left\{ Q\sum_{i\in K}\sum_{j\in CL_i} x_{ij} + \sum_{i\in K}\sum_{j\in CL_i} \max_{t\in T_{in}} q_{ij}{}^{t} x_{ij} \right\}$$

$$s.t.\ \sum_{j\in CL_i} x_{ij} q_{ij}^{cur} \geq EQ_i^{cur} \qquad \forall i\in K$$

$$x_{ij}\in\{0,1\} \qquad \forall i\in K, \forall j\in CL_i$$

Model 2

- capacity of the vehicle, Q
- the days which contains invalid routes, T_{in}
- current day, *cur*
- overload of route i of current day, EQ_i^{cur}
- route set of the *Base Solution*, K
- customers of the *candidate list* of route i, CL_i
- demand of the customer j in the *candidate list* of route i of current day, q_{ij}^{cur}
- x_{ij}=1 if the customer j in the *candidate list* of route i is kicked out at current day, otherwise.

Post-Optimize

We use two operators to post-optimize the solutions. Firstly, a threshold accepting algorithm (Bräysy et al. 2003) is used to optimize solution of the subspace.

Secondly, re-insert the customers back to the routes. A re-insertion move is performed by removing a customer from every day's kick list and re-inserting it into some routes of the solution. The move is accepted only when the target route is taken charge by the same driver for each customer and all the VRPTW constraints are satisfied. The process of re-insertion uses Solomon I1 operator (Solomon 1987) to re-construct the route for 100 times, if it can use one route to hold all the customers, including the original customer and inserted customer, then save the new route and use Or-Opt operator (Or 1976) to optimize the new route.

Application Description

The algorithm discussed in the earlier sections has been in packaged into a vehicle routing application for our customer. In addition to the optimization engine, the system has many other features. A number of these features include a fully functional GIS subsystem, multi-lingual support, easy to use GUI to view actual routings and other operations including schedule optimization and route adjustments. The system is written in the Java Programming Language and its backend database is the MySQL database.

Scheduling

The vital part of the system is the optimization engine. Choose the service date and click load button will list all jobs to be scheduled on this date, then click schedule button to have the optimization engine perform auto-scheduling. The solution will display on the map and the sequence will be marked (Figure 7). The table on the left hand side of the screen capture lists the job information of the specified day. Two small tables on the lower half shows the route information. And the map support zoom in and out and many layers can be selected optionally (Figure 7).

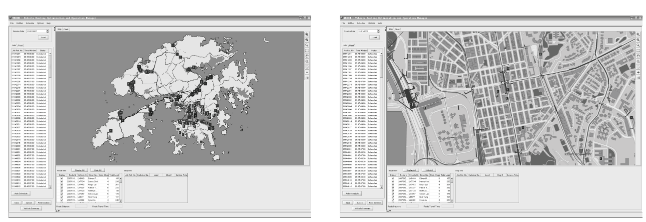

Figure 7: *Perform optimization results and zoom to street level to view routes*

After generating the solution, we can also get the time deviation report of every vehicle and driver if GPS is installed in the trucks. Based on the time deviations, route adjustments can be made accordingly.

Actual Performance

To measure the performance of our system, we choose a group of data provided by the company and use it as the input, and then we use our optimization engine to generate the solution.

Input

A set of data provided by the company is chosen to illustrate the effectiveness of our method. The data is the demand information from 2007-01-31 to 2007-02-06. The total number of customer of the data is 280, and the period is 7 days, and R=2.

Parameter Settings

In the system, our algorithm can adopt two strategies to form the base solutions and two strategies to perform the kick out process. As such we have four different variants. These variants are described below:

- ***M11***: Use *TH* to form Base Solution and use *GR* to kick out.
- ***M12***: Use *TH* to form Base Solution and use *IP* to kick out.
- ***M21***: Use *DS* to form Base Solution and use *GR* to kick out.
- ***M22***: Use *DS* to form Base Solution and use *IP* to kick out.

For *M11* and *M12*, the parameter λ is set to 0.4, 0.5, 0.6, 0.7 and 0.8 to generate the *Base Solution* respectively, the best solution would be saved as the final solution. For *M21* and *M22*, we choose the demand set of Day1, Day2, Day3, Day4 and Day5 to generate the *Base Solution* respectively, the best solution would be saved as the final solution. So each method is a process with 5 loops. We used the algorithm raised by Lim and Zhang (Lim and Zhang 2007) as the VRPTW operator of our algorithm.

Computational Result

In the comparison, we first compare the number of vehicle used in for this period, and then compare the total travel distance.

	M11	M12	M21	M22	Manual
VNUM	57	56	56	54	71
TDIS	12428	12448	12511	12454	

Table 2: *Overall Effect*

Table 2 summarizes results of the 4 algorithms and the one scheduled manually. In Table 2, row *VNUM* means the vehicle number and row *TDIS* means the total travel distance in the 7 days. Column *Manual* indicates the vehicle number used everyday by the company when scheduling was done manually. All the results generated by our variants use less vehicles than the company's manual solution. At least 14 vehicles can be saved. Each vehicle is valued at one million Hong Kong dollars with a 0.3 million Hong Kong dollars of running cost. We also note that *M22* gets the best result.

Conclusion

In this paper, we presented an application that we have developed for one of the largest restaurant and food chain companies in Hong Kong. The problem that we tried to address is a new variant of the periodic vehicle routing problem with time windows that has limited visiting quota. We devised a strategy to solve this new problem by recursively transforming the problem into a series of vehicle routing problems with time windows (VRPTW). We used the current best method for VRPTW (by Lim and Zhang 2007) to solve these sub-problems. We implemented the solver and constructed an application for this new problem by leveraging on an existing VRP product known as VROOM. Our results showed a possible savings of up to 23% in asset costs and annual operating costs for the client.

References

Cordeau, J-F.; Laporte, G.; and Mercier, A. 2001. A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. *Journal of the Operational Research Society* 52: 928-936.

Lim, A., and Zhang, X. 2007. A Two-Stage Heuristic with Ejection Pools and Generalized Ejection Chains for the Vehicle Routing Problem with Time Windows. *Informs Journal on Computing* 19: 443-457.

Solomon, M. M. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operation Research* 35: 254-265.

Or, I. 1976. Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking. Ph.D. thesis, Northwestern Univ., Evanston, Illinois.

Bräysy, O.; Berger, J.; Barkaoui, M.; and Dullaert, W. 2003. A Threshold Accepting Metaheuristic for the Vehicle Routing Problem with Time Windows. *Central European Journal of Operations Research* 11(4): 369-387