22.

ISSN: 2086-8561

TOPIC THIS VOLUME

- Distribution
- Transport
- Cost Issues
- Inventory Planning

ISSUES:

ISSN 2086-8561

Jurnal Logistik Bisnis

VOLUME 1 NO 2 NOVEMBER 2010

Distributions Aspects of Logistic

Politeknik Pos Indonesia

J. Logistik Bisnis	Vol. 1	No. 2	Hal. 1-107	Bandung, November 2010	ISSN: 2086-8561

ISSN: 2086-8561

Jurnal Logistik Bisnis

Volume 1, Nomor 2 –November 2010

CONTENTS

A THRESHOLD ACCEPTING HEURISTIC FOR THE HETEROGENEOUS FLEET VEHICLE ROUTING PROBLEM

ARIF IMRAN[1], LIANE OKDINAWATI[2]

[1]Industrial Engineering Department Institut Teknologi Nasional, Bandung 40125.
E-mail: arifimr@yahoo.com
[2]Business Logistics Department Politeknik Pos Indonesia, Bandung 40191
Email: aneu88@yahoo.com

Abstract

The heterogeneous fleet vehicle routing problem (HFVRP) is investigated using the Threshold Accepting (TA) heuristic. Here we apply a non-monotonic TA heuristics as it produces better solution than the monotonic one. The initial solution is obtained by Dijkstra's algorithm based on a cost network constructed by the sweep algorithm and the 2-opt. Our TA algorithm uses a number of local search methods. The algorithm is then tested on the data sets from the literature and produces good results.

Key words: threshold accepting, metaheuristic, routing, heterogeneous fleet.

1. Introduction

The heterogeneous fleet vehicle routing problem is a variant of the vehicle routing problem (VRP) where the vehicles do not necessary have the same capacity, vehicle fixed cost and unit variable cost. We are also given a set of customers, N, a certain number of vehicle types, M, each of which has a vehicle capacity Qm, a fixed cost Fm and a unit variable cost Rm (m = 1,...,M). As in the classical VRP, each customer must be served by one vehicle only, each vehicle must start and finish its journey at a central depot and the capacity of a vehicle and the maximum length of a route must not be exceeded. The objective of the HFVRP is to minimize the total cost which includes both the vehicle variable and fixed costs. The idea is not only to consider the routing of the vehicles, but also the composition of the vehicle fleet. According to Liu and Shen [16] the HFVRP can be regarded as a short-term or mid-term (or long-term) issue, depending on the planning purpose.

There are a number of published papers addressing the HFVRP. Golden et al. [12] were among the first authors to tackle this problem using a constant unit variable cost. They developed algorithms based on the Clarke and Wright [4] saving technique for the VRP as well as two implementations of the giant tour based algorithm. Desrochers and Verhoog [5] proposed a savings based algorithm using the idea of matching. Salhi and Rand [23] put forward an interactive route perturbation procedure (RPERT) which contains seven refinement phases, each aimed at constructing a newly constructed fleet with a lower total cost. Osman and Salhi [18] proposed two algorithms; the first one based on a tabu search and the second is a modification of RPERT. Ochi et al. [17] presented an evolutionary hybrid metaheuristic which combines a parallel genetic

algorithm with scatter search. Gendreau et al. [10] implemented a tabu search approach using GENIUS, initially developed by Gendreau et al. [8] for the TSP, and some search strategies from Gendreau et al. [9] as well as the adaptive memory procedure originally developed for the VRP by Rochat and Taillard [20]. Taillard [26] presented a heuristic using a column generation method. Renaud and Boctor [19] proposed a sweep-based algorithm to generate a large set of good routes, which are then used in a set partitioning algorithm. Wassan and Osman [31] developed tabu search variants including reactive tabu search that uses special data memory structures and hashing functions. Yaman [32] put forward six interesting formulations for the HFVRP which are enhanced by valid inequalities and lifting. Tights lower bounds and comparable upper bounds are found when tested on the Golden et al. [12] instances. The first four formulations are based on Miller-Tucker-Zemlin constraints whereas the last two, which proved to be more successful, use flow variables. Choi and Tcha [3] used an efficient application of column generation technique which is enhanced by dynamic programming schemes. New tight lower bounds as well as very competitive upper bounds for the Golden et al. [12] instances are obtained. Lee et al. [14] put forward an algorithm that uses tabu search and set partitioning. More recently, Brandao [2] developed two variants of the deterministic tabu search algorithm and Imran et al. [13] put forward an adaptation of the VNS algorithm, which produce excellent results when tested to three data sets from the literature.

The remaining parts of the paper are organized as follows. The proposed TA algorithm is presented in Section 2. The explanation of its main steps is provided in Section 3. The computational results are given in Section 4. The last section summarizes our findings.

2. The Threshold Accepting

The Threshold Accepting (TA) metaheuristic was introduced by Dueck and Scheuer in [7]. This method constitutes a modification of Simulated Annealing. Instead of using a probabilistic function in accepting nonimproving solutions, TA uses deterministic thresholds. In this method, all better solutions are accepted and those inferior ones that have a cost smaller or equal to a threshold say T_s are also accepted. The value of T_s is determined by the user or can be adaptively produced. Defining the way the threshold is defined is one of the challenges in the TA.

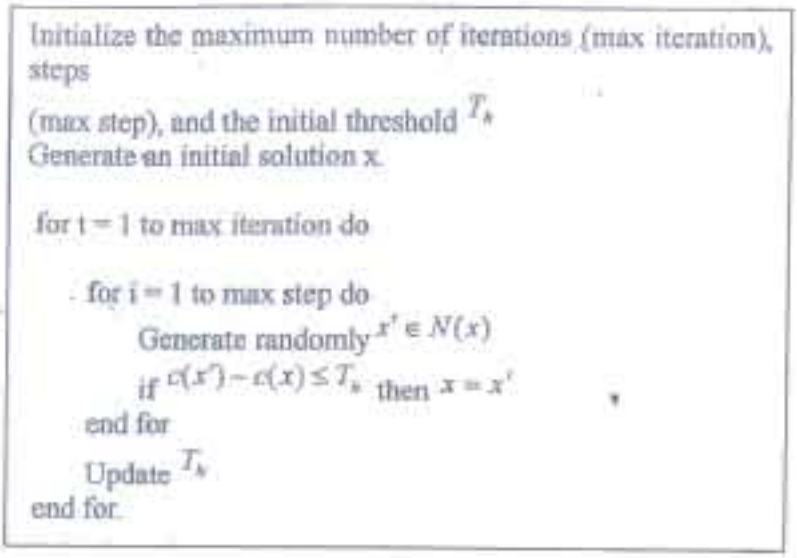

Figure 1: A Basic Threshold Accepting Algorithm

The search begins by initializing the number of iterations, the number of steps needed within each iteration to explore neighbourhoods, and a threshold value. Threshold values during the search process can be determined by a function $R(t)$, where t is the iteration number. It is followed by generating randomly a solution x' from the neighbourhood N(x) of the current solution x. x' is accepted if $c(x') - c(x) \le T_0$ where $T_0 > 0$. The threshold value is gradually decreased during the search process and approaches to zero at the end of the search, which means only the improving moves are accepted. A flexible update where T_0 is also allowed to increase or to reset does also exist. A basic algorithm of the TA is displayed in Figure 1.

Non-Monotonic TAs for the HFVRP

In research, we propose a non-monotonic threshold heuristics for the HFVRP as the non-monotonic threshold algorithm produced better results than the monotonic one (see see Tarantilis et al. [27] and Tarantilis et al. [28]). In non-monotonic TA, instead of decreasing the threshold in a monotonic fashion as in the Classical TA, we allow the threshold value to be increased if a certain condition which we will be defined later is fulfilled.

Initial Investigation

We modify the current TA heuristic by applying a non-monotonic schedule. The following notations are used here:

max iter	: Maximum number of iterations,
Nmax	: Minimum number of acceptance to reduce the threshold value,
I_{max}	: Maximum number of local searches,
T0	: A certain percentage of the initial solution,
mg	: Maximum number of generations within each local search,
m	: The number of feasible solutions within each local search,
NAI	: Number of acceptances within local search I,
R_I	: Ith local search.

Step (0) Initialization. Define a set of local searches R_I, for $I = 1,\ldots,I_{max}$, generate an initial solution x, set $x_{best} = x$, and define maxiter =5000, T0 , mg , m=100 and Nmax = 5.

Step (1) Repeat the following steps until $iter = \max iter$:

a) For each local search $I = 1,\ldots,I_{max}$, set NAI=0 and generate m feasible solutions x' ($x' \in R_I(x)$)at random.

If $c(x') - c(x) \le T_h$, then

Set NAI=NAI+1 and $x = x'$; check if $c(x) < c(x_{best})$ then set $x_{best} = x$.

Compute NAmax = $Max\{NA_I, I = 1,\ldots,I_{max}\}$, where NAmax denotes the largest number of acceptances from all local searches.

b) Compare the solutions obtained from each local search and select the best one.

Compute $\overline{NA} = (\sum_{I=1}^{I_{max}} NA_I)/I_{max}$

c) Update Th using $\overline{NA}$:

If ($\overline{NA}$ >Nmax), reduce the threshold as follows:

$T_{hnew} = T_h\ (1 - ((iter / \max iter)\ \rho)$, where $\rho = \overline{NA}/NA_{max}$

Else if ($0 < \overline{NA} \le N_{max}$) keep the last threshold, T_h

Else if ($NA = 0$) raise threshold

$T_{hnew} = T_h + 0.5\,(T_{h0} - T_h)$, where Th0 denotes threshold value before T_h.

Step (2) Report the best solution obtained, x_{best}.

Figure 2: TA Algorithm

3. Explanation of Main Steps

The algorithm of the TA is displayed in Figure 2. In the initialization phase (Step 0), the initial solution is generated using the procedure as follows:

Initial solution (Step 0)

The initial solution is obtained in three steps; (a) construct a giant tour using the sweep algorithm of Gillett and Miller [11], (b) improve this tour using the 2-opt of Lin [15], and (c) construct the cost network and then apply Dijkstra's algorithm [6] to find the corresponding optimal fleet size. Dijkstra's algorithm systematically provides an initial solution that contains routes with their appropriate types of vehicles. This partitioning procedure based on solving the shortest path problem was presented by Beasley [1] for solving the VRP and by Golden et al. [12] for the HFVRP. Since then, this partitioning approach has been used by several authors, including Ulusoy [30] for the fleet size and mix problem for the capacitated arc routing problem, Ryan et al. [21] and Salhi and Sari [24] for the multi-depot HFVRP. However, it is worth noting that this partitioning procedure cannot be used in its original form when the number of vehicles of each type is required. To avoid using the largest distance between two successive customers in a given route, the starting points, in the construction of the cost network, are used as those that generate the highest largest distances between two successive customers (i.e. gaps) in the giant tour. The number of gaps (NG) generated is defined as follows: $NG = Min(\max(8, \frac{2R}{3}), |(i, i+1) : g_i > \min(\bar{g}, \frac{g^+}{2}))|$

where, NR is the number of routes found by Dijkstra's algorithm, (i,i+1) the ordered sequence of customers, g_i the ith gap (i.e. the distance between customer i and i+1), $\bar{g}$ the average gap, and g^+ the largest gap. The reasoning of using (1) is based on the idea of linking the value of NG to the number of routes and also to the number of gaps that relate to the average as well as the largest gap. For each of the NG selected gaps, say (i1,i1+1), two cost networks are then generated starting from i1 anticlockwise and from i1+1 clockwise. Dijkstra's algorithm is then applied to each of these $2 \times NG$ cost networks. Note that since the network on which the shortest path to be found is acyclic, one could obviously use the standard dynamic programming algorithm instead. In this paper we implement the former as it is available to us.

Construction of a cost network

In order to apply Dijkstra's algorithm, we first construct the cost network considering customer data, capacity constraint, distance constraint, and vehicle variable and fixed costs. For illustration, consider 11 customers in a giant tour made up in the following sequence $i = (1,2,\ldots,11)$ with corresponding demand $q_i(2,1,2,3,4,1,2,1,2,2,3)$. In this example we have two types of vehicles with maximum capacity of 6 units (type 1) and 9 units (type 2).

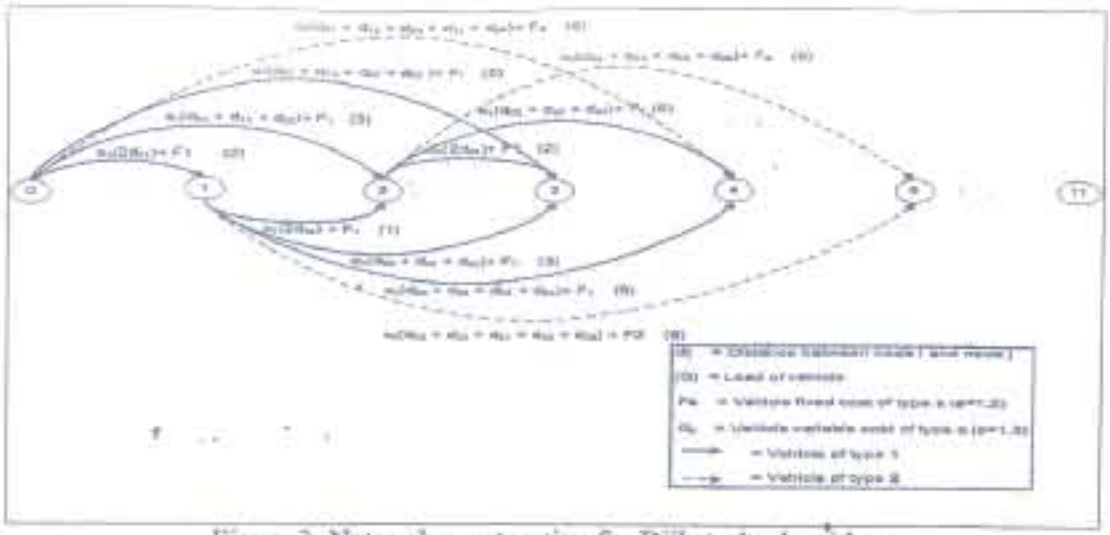

Figure 3: Network construction for Dijkstra's algorithm

We start to construct a network by calculating the cost from the depot to customer 1 and from customer 1 to the depot (return journey) as the cost of the arc 0-1. This is expressed as $C_{01} = F_1 + \alpha_1(2d_{01})$ If having customer 2 and customer 1 together do not violate the capacity constraint of the vehicle of type 1, we calculate the cost of the arc 0-2 as $C_{02} = F_1 + \alpha_1(d_{01} + d_{12} + d_{20})$ We continue with this cost construction until the vehicle of type 1 is full, and then we start using the vehicle of type 2. Figure 3 shows that we can only put customers 1, 2, and 3 together in the vehicle of type 1. Customers 1, 2, 3, and 4 are served by the vehicle of type 2. It is represented by the arc 0-4 which has a cost $C_{04} = F_2 + \alpha_2(d_{01} + d_{12} + d_{23} + d_{34} + d_{40})$ If the vehicle of type 2 is full we start again by using the vehicle of type 1 and calculate the distance from the depot to customer 2 and from customer 2 to the depot (arc 1-2). Customer 3 (arc 1-3) will be added to the vehicle of type 1 if the maximum capacity of this vehicle is not violated, otherwise the vehicle of type 2 is used. The process is continued until all customers are connected. The cost of arc ij is defined as follows:

$$C_{ij} = F_s + \alpha_s\left(d_{0,i+1} + \sum_{k=i+1}^{j-1} d_{k,k+1} + d_{j,0}\right)$$

where, s denotes the smallest vehicle type that accommodate $\sum_{k=i}^{j} Q_k$ After creating this cost network, whose origin is depot '0' and the destination is the last node in the giant tour, we apply Dijkstra's algorithm to produce the least cost path from the origin to the destination.

We also select a positive value of the threshold T0, the maximum number iterations (maxiter), a minimal number of acceptance in order to reduce the threshold (Nmax), and a set of local searches to be employed in Step (1). Local searches we use in this work are the 2-opt intra-route, 2-opt inter-route, 1-1 interchange (swap) inter-

route, 1-1 interchange intra-route, 1-insertion intra-route, 2-insertion intra-route, and the 1-insertion inter-route (1-0 interchange).

The 1-1 interchange (inter-route and intra-route)

The swap intra-route is aimed at reducing the total cost of a route by swapping positions of a pair of customers within the route. Meanwhile the intra-route one reducing the total cost by by swapping positions of a pair of customers from different route.

The 1-insertion procedures (inter-route and intra-route)

Two types of the 1-insertion procedures are used. The first is the 1-insertion intra-route and the second is the 1-insertion inter-route. In the 1-insertion intra-route we remove a customer from its position in a route and try to insert it elsewhere within that route in order to have a better solution. Meanwhile, in the 1-insertion inter-route, each customer from a route is shifted from its position and tried to be inserted elsewhere into another route. If this shifting does not violate any constraints and improves the solution, the selected customer is then permanently removed.

The 2-insertion (intra-route)

The 2-insertion intra-route allows us to remove two consecutive customers and insert them elsewhere within a route to produce a cheaper route.

The 2-opt (inter-route and intra-route)

The 2-opt intra-route, usually refer to as the 2-opt (see Lin [15]), is an old but a simple and an effective improvement procedure that works by removing two non adjacent arcs and adding two new arcs while maintaining the tour structure. A given exchange is accepted if the resulting total cost is lower than the previous total cost. The exchange process is continued until no further improvement can be found. The 2-opt inter-route is similar to the 2-opt intra-route except that it considers two routes where each of the two arcs belong to a different route and reverse directions of the corresponding affected path of each route.

In Step (1a), for each local search we generate 500 solutions or 100 feasible solutions whichever is satisfied first. Also here each local search starts from the same initial solution. The number of acceptances for each local search, the best solution, the local search which produces the best solution and the largest number of acceptances from all local searches are recorded. In Step (1b), we compare the best solutions obtained from all local searches and the best one is selected to be used as the next initial solution in Step (1a). We also compute the average number of acceptance, $\overline{NA}$ which is obtained by using the formula in Step (1b). If $\overline{NA}$ is larger than a certain value Nmax, the threshold value is reduced, but if $\overline{NA}$ is between zero and Nmax we keep the threshold value. Meanwhile, if there is no acceptance (i.e. $\overline{NA}$=0 or all NAI =0), the threshold value is increased by using the bisection method (i.e. select a value at the middle of the range (T_0, T_{40})). The formulas for decreasing and increasing the threshold can be found in Figure 5.6. For

clarity, the formula for decreasing the threshold is given again here by the following equation.

$$T_{new} = T_h + (0.5\,(T_{h0} - T_h))$$

The equation ensures that the new threshold value T_{new} will be larger than T_h but always smaller than the threshold value found before T_h namely T_{h0}. This is important as we want to achieve the accepted moves by increasing the threshold as a small amount as possible only. The search process then reverts to Step (1a) until the maximum number of iterations is reached. As the search process evolves and the threshold value decreases, the accepted moves will be more difficult to find. To overcome this drawback, Nmax is reduced by 1/5 each time the number of iterations is increased by a maxiter/5 iterations. In the last step (Step (2)), the best solution obtained is reported.

4. Computational Experience

The algorithm is programmed in C++ and tested to solve a data set of HFVRP from Golden et al. [12]. The lower bounds we record in this study are found by Choi and Tcha [3] except when noted otherwise (i.e. tighter bounds by Yaman [32]). In the subsequent tables the best solutions are recorded in bold and the new best solutions are underlined. For each instance, say k, we compute the relative percentage deviation as $((cost_k - best_k)/best_k) \times 100$, where costk and bestk denote, for the kth instance, the cost found by our heuristic and the best known solution respectively. The average deviation is then computed over all instances in the data set.

For our experiment, we apply several initial threshold values. We repeat the search process 10 times as randomly generated solutions are implemented within the TA algorithm. The threshold values we use are 0.5%, 1%, 2% and 5% of the initial solution as this generates a large enough initial threshold value. The CPU time and the solutions obtained are displayed in Table 1 and Table 2 respectively.

In Table 1 we report the total CPU time over ten runs for VNS1 and the corresponding number of runs for VNS2 (usually 5 to 7 runs). Our algorithm is executed on a Pentium M 1.7 GHz PC with 1GB RAM. The total CPU time of Osman and Salhi [18] was obtained from a Vax 4500 machine (5.5 Mflop/s). Renaud and Boctor [19] use Pentium II 233 MHz computer and record the average CPU time over several runs. The CPU time of Wassan and Osman [31] is the total time from five runs on a Sun Sparc 1000 with 50 MHz (10 Mflop/s). Choi and Tcha [3] found their results from five runs on a Pentium IV 2.6 GHz with 526 MB RAM. Lee et al. [14] use a Pentium IV 1.8 PC. Brandao [2] uses a Pentium M 1.4 GHz with 256 MB RAM and the CPU time is the CPU time of one variant. The faster heuristic is that by [18], but this method produced much lower solution quality due to its simple though well structured tabu search. Based on these results, though it is difficult to compare the CPU time under different machines, it is clear that our heuristic methods require a reasonable amount of CPU time.

Table 2 shows that the best result is given by the TA algorithm with 1% threshold. The results are better than the ones of Osman and Salhi [18] and Renaud and Boctor [19]. Meanwhile, TA with other threshold values produced results that are better than the one of Osman and Salhi [18] but using more CPU time.

The drawback of this algorithm is that it spends a relatively large CPU time when compared with other CPU time of the previous researchers. Note that our CPU times displayed in Table 1 are the average CPU times of 10 runs.

Table 1: CPU Time of TA (in Seconds)

No	Size	Osman & Salhi *	Renaud & Boctor *	Wassan & Osman	Choi &Tcha +	Lee et al.	Brandao**	TA (0.5%) *	TA (1%) *	TA (2%) *	TA (5%) *
3	20	5	4	88	0	59	21	96	92	92	81
4	20	6	6	80	1	79	22	77	77	77	76
5	20	5	5	52	1	41	20	84	83	84	76
6	20	4	9	88	0	89	25	79	77	81	74
13	50	62	50	2084	10	258	145	203	182	180	140
14	50	71	160	1660	51	544	220	155	157	156	131
15	50	46	45	2349	10	908	110	165	183	159	138
16	50	35	28	689	11	859	111	154	157	158	133
17	75	85	652	1874	207	1488	322	234	232	217	179
18	75	116	1037	2261	70	2058	267	259	259	327	263
19	100	289	1110	8570	1179	2503	438	277	278	282	207
20	100	306	307	2692	264	2261	601	292	246	298	217

+ CPU time for the best run only.

* The average CPU time.

** CPU time of version 1.

Table 2:

No	Size	Lower Bounds	Best Solution	Osman & Salhi	Renaud & Boctor	Wassan & Osman	Choi &Tcha	Lee et al.	Brandao	TA1 (1.5%)	TA(1%)	TA(2%)	TA(3%)
3	20	951.61	961.03	961.03	963.03	961.03	961.03	961.03	961.03	961.03	961.03	961	961.03
4	20	6369.15	6437.33	6445.10	6437.33	6437.33	6437.3	6437.3	6437.33	6437.3	6437.3	6437	6437.3
5	20	988.01	1007.05	1009.15	1007.35	1007.05	1007.1	1007.1	1007.05	1009.15	1007.1	1007	1007.1
6	20	6468.85	6516.47	6516.56	6516.47	6516.47	6516.5	6516.5	6516.47	6516.5	6516.5	6516	6516.5
13	50	2392.77	2406.36	2471.07	2446.43	2422.10	2406.4	2408.41	2406.36	2420.97	2411.96	2424.91	2408.41
14	50	8943.94	9119.03	9125.65	9121.41	9119.03	9119	9160.42	9119.03	9119	9120.92	9119.28	9126.10
15	50	2546.86	2586.37	2606.72	2618.03	2586.37	2586.4	2586.4	2586.37	2612.60	2590.88	2621.93	2606.58
16	50	2655.92	2720.43	2745.01	2761.50	2730.01	2720.4	2724.23	2725.36	2782.80	2725.30	2778.44	2776.46
17	75	1700.11	1734.08	1763.08	1857.21	1755.32	1744.84	1762.45	1734.53	1773.04	1750.56	1758.54	1770.87
18	75	2342.84	2369.65	2432.56	2432.36	2385.52	2371.48	2373.03	2369.65	2415.41	2406.10	2420.06	2434.91
19	100	8578.23	8659.74	8681.71	8687.21	8659.74	8664.29	8659.29	8661.81	8683.87	8683.04	8684.56	8691.43
20	100	3993.36	4029.49	4056.73	4094.34	4061.64	4039.5	4045.47	4042.36	4081.13	4087.06	4073.58	4073.39
# Best Solutions				1	1	6	8	3	8	4	6	4	4
Average Deviation (%)				2.015	3.0??	1.249	0.028	0.270	0.033	0.771	0.338	0.764	0.834

6. Conclusions

We have put forward a non monotonic TA algorithm to tackle the HFVRP. This algorithm adopting a set of local search procedures. It was found that our proposed TA heuristics (TA 1%) yield competitive results when compared to the best known results found in the literature. Finally, this study shows that a suitable implementation of TA can be applied successfully to solve the HFVRP. For further investigation the algorithm can be improved by adopting other local search such as GENI and it can be used to tackle other related distribution problems.

References

[1] Beasley, J., 1983. Route First-Cluster Second Methods for Vehicle Routing. Omega 11, 403-408.

[2] Brandao, J., 2008. A Deterministic Tabu Search Algorithm for the Fleet Size and Mix Vehicle Routing Problem. European Journal of Operational Research, doi:10.1016/j.ejor.2007.05.059.

[3] Choi, E., Tcha D.-W., 2007. A Column Generation Approach to the Heterogeneous Fleet Vehicle Routing Problem. European Journal of Operational Research 34, 2080-2095.

[4] Clarke, G., Wright, J.W., 1964. Scheduling of Vehicle from Central Depot to a Number of Delivery Points. Operations Research 12, 568-581.

[5] Desrochers, M., Verhoog, T.W., 1991. A New Heuristic for the Fleet Size and Mix Vehicle Routing Problem. Computers & Operations Research 18, 263-274.
[6] Dijkstra, E.W., 1959. A Note on Two Problems in Connection with Graphs. Numerische Mathematik 1, 269-271.
[7] Dueck, G. and Scheuer, T. (1990). Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computation Physics* **90**, 161-175.
[8] Gendreau, M., Hertz, A., Laporte, G., 1992. New Insertion and Post Optimization Procedures for the Traveling Salesman Problem. Operations Research 40, 1086-1094.
[9] Gendreau, M., Hertz, A., Laporte, G., 1994. A Tabu Search Algorithm for the Vehicle Routing Problem. Management Science 40, 1276-1290.
[10] Gendreau, M., Laporte, G., Musaraganyi, C., Taillard, E.D., 1999. A Tabu Search Heuristic for the Heterogeneous Fleet Vehicle Routing Problem. Computers & Operations Research 26, 1153-1173.
[11] Gillett, B.E., Miller, L.R., 1974. A Heuristic Algorithm for the Vehicle Dispatch Problem. Operations Research 22, 340-344.
[12] Golden, B., Assad, A., Levy, L., Gheysens, F., 1984. The Fleet Size and Mix Vehicle Routing. Computers & Operations Research 11, 49-66.
[13] Imran, A., Salhi, S. and Wassan N.A. (2009). A Variable Neighborhood-Based Heuristic for the Heterogeneous Fleet Vehicle Routing Problem. *European Journal of Operational Research* **197**, 509-518.
[14] Lee, Y.H., Kim, J.I., Kang, K.H., Kim, K.H., 2008. A Heuristic for Vehicle Fleet Mix Problem Using Tabu Search and Set Partitioning. Journal of the Operational Research Society 59, 833-841.
[15] Lin, S., 1965. Computers Solutions of the Traveling Salesman Problem. Bell System Technical Journal 44, 2245-2269.
[16] Liu, F.H., Shen, S.H., 1999. The Fleet Size and Mix Vehicle Routing Problem with Time Windows. Journal of the Operational Research Society 50, 721-732.
[17] Ochi, L.S., Vianna, D.S., Drummond L.M.A., Victor A.O., 1998. A parallel Evolutionary Algorithm for the Vehicle Routing Problem with Heterogeneous Fleet. Future Generation Computation System 14, 285-292.
[18] Osman, I.H., Salhi, S., 1996. Local Search Strategies for the Mix Fleet Routing Problem. In: Rayward-Smith, V.J., Osman, I.H., Reeves, C.R., Smith, G.D. (Eds.), Modern Heuristic Search Methods, pages 132-153. John Wiley & Sons.
[19] Renaud, J., Boctor, F.F., 2002. A Sweep-Based Algorithm for the Fleet Size and Mix Vehicle Routing Problem. European Journal of Operational Research 140, 618-628.
[20] Rochat, Y., Taillard, E.D., 1995. Probabilistic Diversification and Intensification in Local Search Vehicle Routing. Journal of Heuristic 1, 147-167.
[21] Ryan, D.M., Hjorring, C., Glover, F., 1993. Extensions of the Petal Method for the Vehicle Routing. Journal of the Operational Research Society 44, 289-296.
[22] Salhi, S., Rand, G.K., 1987. Improvements to Vehicle Routing Heuristics. Journal of the Operational Research Society 38, 293-295.
[23] Salhi, S., Rand, G.K., 1993. Incorporating Vehicle Routing into the Vehicle Fleet Composition Problem. European Journal of Operational Research 66, 313-360.

[24] Salhi, S., Sari, M., 1997. A Multi-Level Composite Heuristic for the Multi-Depot Vehicle Fleet Mix Problem. European Journal of Operational Research 103, 95-112.
[25] Salhi, S., Sari, M., Sadi, D., Touati, N., 1992. Adaptation of Some Vehicle Fleet Mix Heuristic. OMEGA 20, 653-660.
[26] Taillard, E.D., 1999. A Heuristic Column Generation Method for the Heterogeneous Fleet VRP. Recherche Operationnelle 33, 1-14.
[27] Tarantilis, C.D., Kiranoudis, C.T., 2007. A Flexible Adaptive Memory-Based Algorithm for Real-Life Transportation Operations: Two Case Studies from Diary and Construction Company. European Journal of Operational Research 179, 806-822.
[28] Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S., 2003. A List Based Threshold Accepting Metaheuristic for the Heterogeneous Fixed Fleet Vehicle Routing Problem. Journal of the Operational Research Society 54, 65-71.
[29] Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S., 2004. A Threshold Accepting Metaheuristic for the Heterogeneous Fixed Fleet Vehicle Routing Problem. European Journal of Operational Research 152, 148-158.
[30] Ulusoy, G., 1985. The Fleet Size and Mix Problem for Capacitated Arc Routing. European Journal of Operational Research 22, 329-337.
[31] Wassan, N.A., Osman, I.H., 2002. Tabu Search Variants for the Mix Fleet Vehicle Routing Problem. Journal of the Operational Research Society 53, 768-782.
[32] Yaman, H., 2006. Formulations and Valid Inequalities for the Heterogeneous Vehicle Routing Problem. Mathematical Programming 106, 365-390.