PROSIDING SEMINAR NASIONAL
TEKNIK DAN MANAJEMEN INDUSTRI 2011

ISBN : 978-979-796-189-3

Aplikasi Multi-Level Heuristik dan Fixed Threshold pada Variable Neighborhood Search untuk Heterogeneous Fleet Vehicle Routing Problem

Arif Imran
Staf Pengajar, Jurusan Teknik Industri Institut Teknologi Nasional, Bandung
Kontak Person:
Arif Imran
Institut Teknologi Nasional Jalan PHH Mustopha 23
Bandung, 40124
Telp: 022-7272215, Fax: 022-7202892, E-mail: arifimr@yahoo.com

Abstrak

Heterogeneous Fleet Vehicle Routing Problem (HFVRP) diselesaikan dengan mengaplikasikan multi-level heuristic, fixed threshold dan beberapa adaptasi pada variable neighborhood search (VNS). Solusi inisial diperoleh dengan menggunakan algoritma Dijkstra berdasarkan cost network yang dibentuk oleh algoritma Sweep dan 2-opt. Solusi yang dibangkitkan secara random dipilih menggunakan fixed threshold. Algoritma yang diusulkan pada penelitian ini menggunakan beberapa neighborhood. Sebagai tambahan pada algoritma juga diaplikasikan beberapa local search, dan prosedur diversifikasi. Multi-level heuristik digunakan dalam pemakaian local search. Algoritma usulan diuji dengan data set yang terdapat pada literatur. Solusi yang diperoleh kompetitif jika dibandingkan dengan solusi-solusi yang telah dipublikasikan.

***Kata kunci**: metaheuristik, routing, threshold, multi-level, variable neighborhood.*

Abstract

The heterogeneous fleet vehicle routing problem is investigated using multi-level heuristic, fixed threshold and some adaptations of the variable neighborhood search (VNS). The initial solution is obtained by Dijkstra's algorithm based on a cost network constructed by the sweep algorithm and the 2-opt. Random generated solutions are selected using a fixed threshold. Our VNS algorithm uses several neighborhoods which are adapted for this problem. In addition, a number of local search methods together with a diversification procedure are used. Multi-level heuristic is used when applying local search. Our algorithm is tested using data set from literature and the results appear to be competitive when compared to the published results in the literature.

***Keywords:** metaheuristic, routing, threshold, multi-level, variable neighborhood.*

1. PENDAHULUAN

Terdapat banyak perusahaan distribusi yang dalam melakukan pendistribusian menggunakan beberapa jenis kendaraan. Permasalahan ini dikenal juga dengan sebutan *Heterogeneous Fleet Vehicle Routing Problem* (HFVRP). Oleh karena itu HFVRP mempunyai peranan penting dalam manajemen distribusi dan merupakan salah satu problem yang dipelajari secara intensif di bidang optimisasi kombinatorik. HFVRP dapat didefinisikan sebagai suatu masalah dimana sejumlah *customer* harus dilayani oleh sejumlah kendaraan yang terdiri lebih dari satu jenis yang berada pada satu depot. Setiap *customer* hanya boleh dikunjungi satu kali; kapasitas maksimum dan panjang rute maksimum tidak boleh dilanggar. Tujuan dari penyelesaian HFVRP adalah untuk menemukan rute yang memenuhi persyaratan yang disebutkan di atas dengan biaya yang minimum.

Berikut ini beberapa penelitian terhadap HFVRP. Golden *et al.* [7] merupakan yang pertama membahas HFVRP. Mereka mengembangkan algoritma berdasarkan algoritma *saving* Clarke and Wright [3] untuk VRP. Salhi and Rand [15] mengembangkan *interactive route perturbation procedure* (RPERT). Osman and Salhi [12] mengajukan dua algoritma; yang pertama berdasarkan *tabu search* dan yang kedua merupakan modifikasi dari algoritma RPERT. Taillard

[17] mengembangkan metoda heuristik dengan metode *column generation*. Renaud and Boctor [13] mengambangkan algoritma yang menggunakan algoritma *sweep* dan *set partitioning*. Wassan and Osman [18] mengembangkan algoritma *reactive tabu search* Choi and Tcha [2] mengaplikasikan teknik *column generation* yang dikombinasikan dengan *dynamic programming*. Lee et al. [9] menggunakan *tabu search* dan *set partitioning*. Terakhir, Brandao [1] mengembangkan dua varian algoritma *tabu search* yang deterministik.

2. METODE PENELITIAN

VNS dikembangkan oleh Mladenovic dan Hansen [11] untuk menyelesaikan masalah optimisasi kombinatorik. Sedangkan *Multi-Level* heuristik diusulkan oleh Salhi et al. [16]. Ide dasar dari VNS adalah perubahan *neighborhood* yang sistematis dalam metode pencarian lokal (*local search*). Pada penelitian ini *Multi-Level* heuristik dan pemilihan solusi (*fixed threshold*) diaplikasikan di dalam algoritma VNS. Algoritma juga dilengkapi dengan beberapa *local search* termasuk algoritma Dijkstra dan skema diversifikasi. Algoritma yang digunakan pada penelitian ini dapat dilihat pada Algoritma VNS Usulan berikut.

1. Inisialisasi. Definisikan set struktur neighbourhood $N_k, k = 1,...,k_{\max}$ dan set local search R_l, $l = 1,...,l_{\max}$. Bangkitkan solusi awal x $x_{best} = x$, dan iterMax=100;
2. Set $k \leftarrow 1$
3. Repeat langkah-langkah berikut until $k = k_{\max}$:
   a. Shaking. Bangkitkan solusi fisibel x' secara random dari k^{th} neighbourhood $x(x' \in N_k(x))$,dimana x' didapat dari Algoritma Pemilihan Solusi (*Fixed Threshold*)
   b. Local search: Gunakan multi-level untuk mendapatkan neighbor terbaik x''. Jika $l > l_{\max}$ buat cost network menggunakan incumbent x dan gunakan algoritma Dijkstra untuk mendapatkan $\tilde{x}$. If $\tilde{x}$ lebih baik dari x, $x \leftarrow \tilde{x}$ dan lanjutkan ke (2).
   c. Move or not. If optimum lokal x'' lebih baik dari incumbent x, $x \leftarrow x''$ dan lanjutkan ke (2); Sebaliknya $k \leftarrow k+1$. Jika $k \le k_{\max}$ lanjutkan, else if $k > k_{\max}$ lanjutkan ke (4).
   d. Gunakan prosedur diversifikasi dan lanjutkan ke (3b).
4. If x lebih baik dari x_{best}, $x_{best} \leftarrow x$;
   If $iter > iterMax$ stop, else $iter \leftarrow iter+1$, lanjutkan ke (2).

2.1 Prosedur *Shaking* dengan *Fixed Threshold*

Pada langkah 4a. ditambahkan prosedur untuk memilih solusi yang dibangkitkan secara random. Kita hanya memilih solusi random yang nilainya α % lebih dari solusi awal. Proses pencarian solusi tersebut dapat diulangi 100 kali sampai didapat solusi yang memenuhi syarat tapi. Jika setelah 100 percobaan tidak didapat solusi *yang memenuhi persyaratan*, solusi terbaik sebelumnya digunakan kembali. Algoritma dari strategi ini dapat dilihat pada Algoritma Pemilihan Solusi (*Fixed Threshold*) berikut.

Set $iter = 100$ dan set f_{best} (the best cost corresponding to the best solution).

Repeat:

a. Bangkitkan x' secara random dari k^{th} neighbourhood $x(x' \in N_k(x))$ dan hitung cost $f(x')$
b. *If* $f(x') - f_{best} < \alpha . f_{best}$, terima x' dan $x = x'$ dan lanjutkan ke (3b) di Algoritma VNS Usulan. *Else* record solusi fisibel terbaik yang didapat, $\hat{x}$.
c. *If* $iter = 100$ set $x = \hat{x}$ dan lanjutkan ke (3b) di Algoritma VNS Usulan.

2.2 Solusi Inisial

Solusi inisial diperoleh dari tiga langkah; (a) Membuat satu *giant tour* menggunakan algoritma *sweep* [6], (b) perbaiki hasil langkah (a) menggunakan 2-opt [10], dan (c) bentuk *cost network* dan kemudian gunakan algoritma Dijkstra [4] untuk menemukan ukuran *optimal fleet*. Untuk menghindari menggunakan jarak terbesar antara dua *customer* berurutan di satu rute, titik awal dalam pembentukan *cost network* adalah titik titik yang mempunyai jarak terjauh antara dua *customer* (gap) di dalam *giant tour*. Jumlah gap (*NG*) yang di gunakan didefinisikan sebagai berikut:

$$NG = Min\{\max(8, \tfrac{NR}{2}), \left|((i, i+1) : g_i > \min(\bar{g}, \frac{g^+}{2}))\right|\} \quad (1)$$

Alasan penggunaan (1) didasarkan pada ide untuk menyambung nilai *NG* dengan jumlah rute dan jumlah gap yang berhubungan dengan rata-rata dan gap terbesar. Untuk setiap *NG* gap yang terpilih misalkan (i_1,i_1+1), dua *cost network* dibangkitkan mulai dari i_1 berlawanan arah putaran jarum jam dan mulai dari i_1+1 searah putaran jarum jam Algoritma Dijkstra's kemudian dipakai pada setiap $2 \times NG$*cost network.*

2.3 Pembentukan *Cost Network*

Untuk mengaplikasikan algoritma Dijkstra, pertama dibuat *cost network* dengan memperhatikan data *customer*, pembatas kapasitas, pembatas jarak dan biaya variabel (*variable cost*) dan biaya tetap (*fixed cost*) kendaraan. Algoritma Dijkstra digunakan untuk mendapatkan biaya terkecil dari jaringan yang berawal dari depot dan berakhir di simpul terakhir.

2.4 Struktur *Neighborhood*

Enam *neighborhood*, digunakan pada penelitian ini (k_{max} = 6). *Neighborhood-neighborhood* tersebut adalah 1-1 *interchange* (*swap*), dua jenis of the 2-0 *shift*, 2-1 *interchange*, dan dua jenis *perturbation*. Urutan penggunaan *neighborhood* adalah sebagai berikut: 1-1 *interchange* (N_1) , 2-0 shift jenis 1 (N_2), 2-1 *interchange* (N_3), *perturbation* jenis 1 (N_4), *perturbation* jenis 2 (N_5), dan 2-0 *shift* jenis 2 (N_6).

1-1 interchange (prosedur swap)

Neigborhood ini bertujuan untuk membangkitkan solusi fisibel dangan jalan menukar dua *customer* yang berasal dari dua rute. Prosedur dimulai dengan mengambil *customer* secara random dari rute yang juga dipilih secara random dan kemudian menukarkannya secara sistematis dengan *customer* lain dari semua rute yang berbeda. Prosedur berulang sampai satu pergerakan yang fisibel didapatkan.

2-0 shift

Pada 2-0 *shift*, dua *customer* berurutan dipilih secara random dari suatu rute yang juga dipilih secara random. Dua *customer* ini di *insert* ke rute yang lain secara sistematis. Prosedur berulang sampai satu pergerakan yang fisibel didapatkan. Prosedur ini dinamakan 2-0 *shift* tipe 1. Prosedur 2-0 *shift* yang lain yang dinamakan , 2-0 *shift* type 2, seperti prosedur tipe 1 kecuali pada prosedur ini kita dapat meng*insert* dua *customer* kedua rute yang berbeda.

2-1 interchange

Prosedur ini memindahkan dua *customer* yang dipilih secara random ke rute yang lain secara sistematis dan menerima satu *customer* dari rute yang menerima dua *customer* tersebut.

Prosedur perturbation

Prosedur ini dikembangkan oleh Salhi and Rand [14] untuk menyelesaikan VRP. Dalam proses tiga rute dilibatkan. Prosedur dimulai dengan mengambil satu *customer* secara random dari satu rute (pertama) yang juga dipilih secara random dan kemudian di relokasi ke rute yang lain tanpa mempertimbangkan kapasitas di rute yang dituju (kedua). Satu *customer* dari rute ini

kemudian dipindahkan ke rute yang lain dengan mempertimbangkan kapasitas dari rute ke dua dan rute yang ketiga. Prosedur ini kita namakan *perturbation* tipe 1. *Perturbation* tipe 2 memindahkan dua *customer* yang berurutan dari rute pertama.

2.5 Local Search

Enam prosedur perbaikan digunakan sebagai *local search*. Urutan dari prosedur-prosedur tersebut adalah sebagai berikut: 1-*insertion inter-route* (R_1), 2-opt *inter-route* (R_2), 2-opt *intra-route* (R_3), *swap intra-route* (R_4), 1- *insertion intra-route* (R_5) dan 2-*insertion intra-route* (R_6.) Proses dimulai dengan membangkitkan solusi fisibel secara random x' dari N_1, yang digunakan sebagai solusi sementara (*temporary*). Kemudian *multi-level* heuristik (Salhi & Sari 1997) diaplikasikan untuk mendapatkan solusi terbaik x'' menggunakan R_1. Jika x'' mempunyai nilai yang lebih baik dari pada x', maka $x' = x''$ dan proses pencarian (*search*) kembali ke R_1, sebaliknya prosedur perbaikan berikutnya diaplikasikan. Proses ini diulangi sampai R_6 tidak dapat menghasilkan solusi yang lebih baik.

Prosedur *1-insertion (inter-route and intra-route)*

Dua jenis prosedur 1-*insertion* diterapkan. Yang pertama adalah 1-*insertion intra-route* dan yang kedua adalah 1-*insertion inter-route*. Pada 1-*insertion intra-route* satu *customer* dipindahkan dari tempatnya di suatu rute dan di cobu disisipkan ditempat dalam suatu rute untuk memperoleh rute yang lebih baik. Sedangkan pada 1-*insertion inter-route*, setiap *customer* dari suatu rute dipindahkan ke rute yang lain. Jika perpindahan ini tidak melanggar pembatas yang ada maka *customer* yang terpilih akan dipindahkan.

2-insertion (intra-route)

2-*insertion intra-route* memungkinkan kita untuk memindahkan dua *customer* yang berurutan dan menyisipkan mereka ke suatu tempat dalam rute yang sama untuk mendapatkan rute yang lebih baik.

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

2-opt *intra-route*, mempunyai nama lain 2-opt (lihat Lin (1963)), merupakan prosedur perbaikan yang efektif. Prosedur ini bekerja dengan menghilang dua *arc* yang tidak berdekatan pada satu rute dan menambahkan dua arch yang baru. Proses pertukaran dilakukan sampai perbaikan dari *total cost* tidak ditemukan lagi. 2-opt *inter-route* sama dengan 2-opt *intra-route* kecuali pada 2-opt *intra-route* dua rute diperhatikan dua rute dalam pertukaran.

Prosedur swap (intra-route)

Swap intra-route bertujuan untuk mengurangi *total cost* suatu rute dengan cara menukar tempat dua *customer* dalam suatu rute.

2.6 Penggunaan Algorithm Dijkstra sebagai Prosedur Perbaikan

Disamping digunakan untuk membangkitkan solusi inisial, algoritma Dijkstra juga digunakan sebagai prosedur perbaikan. *Cost network* dibentuk dari solusi terbaik *incumbent*. Tujuannya adalah untuk melihat apakah solusi yang didapat sudah merupakan solusi optimum dari *cost network* yang dibuat.

2.7 Prosedur Diversifikasi

Prosedur ini digunakan jika setelah semua *local search* dilakukan tidak ditemukan lagi solusi yang lebih baik. Prosedur ini bertujuan untuk mengeksplorasi daerah ruang pencarian (*search space*) yang lain yang mungkin belum dikunjungi. Solusi terbaik *incumbent* digunakan sebagai untuk mendapatkan solusi inisial yang baru. Pada penelitian ini jumlah diversifikasi yang dilakukan (*ND*) ditentukan sebagai berikut:

$$ND = Min\,(100, 2N) \tag{2}$$

Langkah-langkah prosedur diversifikasi adalah sebagai berikut:

a. Hubungkan semua simpul. Simpul terakhir dari rute awal disambungkan dengan simpul pertama dari simpul pertama rute berikutnya.
b. Hitung semua jarak antara dua simpul berurutan.
c. Pilih jarak terbesar antara dua simpul berurutan yang bukan merupakan ujung simpul dari rute yang berbeda, sebut (e_1, e_2) sebagai *starting point*.
d. Buat *cost network* mulai dari e_2 searah putaran jam dan gunakan algoritma Dijkstra.
e. Seperti langkah 4, tapi mulai dari e_1 berlawanan arah putaran jarum jam.

Untuk proses pengujian, algoritma usulan diprogram ke dalam bahasa C++ dan digunakan untuk menyelesaikan data set [7]. Program dieksekusi menggunakan komputer dengan prosesor Intel M 1.7 GHz PC dan 1GB RAM.

Tabel 1. Kualitas Solusi VNS dengan *Threshold* Berbeda

No	Size	Best Solution	Renaud & Boctor (2002)	Wassan & Osman	Choi &Tcha (2007)	Lee et al. (2008)	Brandao (2009)	VNS9 (10%)	VNS11 (15%)	VNS12 (5%)	VNS13 (1%)	VNS14 (0.5%)
3	20	**961.03**	963.61	**961.03**	**961.03**	**961.03**	**961.03**	**961.03**	**961.03**	**961.03**	**961.03**	**961.03**
4	20	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**	**6437.33**
5	20	**1007.05**	1007.96	**1007.05**	**1007.05**	**1007.05**	**1007.05**	**1007.05**	**1007.05**	1008.59	**1007.05**	**1007.05**
6	20	**6516.47**	6537.74	**6516.47**	**6516.47**	**6516.47**	**6516.47**	**6516.47**	**6516.47**	**6516.47**	**6516.47**	**6516.47**
13	50	**2406.36**	2406.43	2422.10	**2406.36**	2408.41	**2406.36**	**2406.36**	**2406.36**	**2406.36**	**2406.36**	2413.78
14	50	**9119.03**	9122.01	9119.86	**9119.03**	9160.42	**9119.03**	**9119.03**	**9119.03**	**9119.03**	**9119.03**	**9119.03**
15	50	**2586.37**	2618.03	**2586.37**	**2586.37**	**2586.37**	**2586.37**	**2586.37**	2587.38	2586.84	2586.84	2586.84
16	50	**2720.43**	2761.96	2730.08	**2720.43**	2724.33	2728.14	**2720.43**	2743.03	2745.87	2744.98	2736.16
17	75	**1734.53**	1757.21	1755.10	1744.83	1745.45	**1734.53**	1744.23	1749.67	1745.32	1745.32	1755.21
18	75	**2369.65**	2413.39	2385.52	2371.49	2373.63	**2369.65**	2377.56	2379.07	2378.16	**2369.65**	2387.79
19	100	**8659.74**	8687.31	**8659.74**	8664.29	8699.98	8661.81	8665.74	8675.58	8674.97	8665.74	8676.79
20	100	**4039.49**	4094.54	4061.64	**4039.49**	4043.47	4042.59	4055.49	4059.86	4044.84	4063.01	4065.61
# Best Solutions			1	6	**9**	5	**9**	8	6	5	7	5
Average Deviation (%)			0.692	0.285	0.060	0.170	**0.032**	0.113	0.236	0.200	0.183	0.309

Dari **Tabel 1** terlihat algoritma usulan (VNS1) menghasilkan 8 solusi yang sama dengan solusi terbaik. Deviasi rata-rata yang dihasilkan lebih baik dari [13], [18] dan [9] tetapi masih di bawah [1] dan [2]. Pada Tabel 2 dapat dilihat bahwa algoritma usulan menggunakan waktu komputasi (*CPU Time*) yang *acceptable*. Pada penelitian ini juga di investigasi penggunaan nilai *threshold* yang lain yaitu 15%, 5%, 2%, and 1% untuk melihat perubahan yang terjadi jika nilai *threshold* yang digunakan berbeda. Solusi yang didapat dan waktu komputasi diberikan pada **Tabel 1** dan **Tabel 2**. Dari **Tabel 1** dapat dilihat bahwa VNS dengan threshold 10% menghasilkan deviasi rata-rata terbaik dibandingkan dengan VNS dengan threshold 15%, 5%, 2%, and 1% . Secara keseluruhan kualitas solusi yang dihasilkan dari tiap *threshold* dapat dikatakn kompetitif jika dibandingkan dengan hasil-hasil yang ada pada literatur. Jika dilihat dari waktu komputasi (**Tabel 2**) VNS dengan *threshold* 1% menggunakan waktu komputasi terkecil.

Table 2. Perbandingan CPU *Time* (dalam detik)

No	Size	Wassan & Osman	Wassan & Osman	Choi & Tcha +	Lee et al.	Bran-dao **	VNS1 (10%)	VNS2 (15%)	VNS3 (5%)	VNS4 (2%)	VNS5 (1%)
3	20	88	88	0	59	21	22	23	20	17	23
4	20	80	80	1	79	22	22	25	25	24	20
5	20	52	52	1	41	20	23	22	20	17	18
6	20	88	88	0	89	25	27	28	35	28	27
13	50	2084	2084	10	258	145	308	301	181	148	146
14	50	1660	1660	51	544	220	293	311	235	228	228
15	50	2349	2349	10	908	110	363	349	244	221	210
16	50	689	689	11	859	111	281	283	173	166	165
17	75	1874	1874	207	1488	322	797	771	611	501	739
18	75	2261	2261	70	2058	267	957	963	826	601	516
19	100	8570	8570	1179	2503	438	2012	1562	1137	1114	1049
20	100	2692	2692	264	2261	601	1895	1935	1476	1041	1012

+ CPU time for the best run only, * The average CPU time, ** CPU time of version 2 algorithm.

3. KESIMPULAN

Adaptasi dari algoritma dasar VNS telah dikembangkan untuk menyelesaikan HFVRP. Algoritma VNS dasar dilengkapi dengan beberapa fitur-fitur tambahan seperti, mengadopsi beberapa *local search*, algoritma Dijkstra, *multi-level* heuristik, *fixed threshold* dan prosedur diversifikasi. Dari uji coba menggunakan data set yang ada pada literatur, algoritma usulan dapat menghasilkan solusi yang lebih baik dari beberapa solusi yang telah dipublikasikan

Untuk penelitian lebih lanjut, algoritma VNS usulan dengan beberapa modifikasi akan diaplikasikan untuk menyelesaikan masalah-masalah VRP lainnya seperti *heterogeneous fixed fleet* VRP, *multi-depot* VRP dan *heterogeneous fleet multi-depot* VRP.

Keterangan :

NR : jumlah rute yang didapat oleh algoritma Dijkstra's
$(i,i+1)$: merupakan urutan konsumen
g_i : gap ke i (jarak antara konsumen i and i+1)
$\overline{g}$: rata-rata gap
g^+ : gap terbesar
N : jumlah *customer*

5. REFERENSI

[1] Brandao, J. (2009). A Deterministic Tabu Search Algorithm for the Fleet Size and Mix Vehicle Routing Problem. *European Journal of Operational Research* **195,** 716-728.

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

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

[4] Dijkstra, E.W. (1959). A Note on Two Problems in Connection with Graphs. *Numerische Mathematik* **1**, 269-271.

[5] Dueck, G. (1993). New Optimization Heuristics: The Great Deluge Algorithm and Record to Record Travel. *Journal of Computation Physics* **104**, 86-92.

[6] Gillett, B.E. and Miller, L.R. (1974). A Heuristic Algorithm for the Vehicle Dispatch Problem. *Operations Research* **22**, 340-344.

[7] Golden, B., Assad, A., Levy, L. and Gheysens (1984). The Fleet Size and Mix Vehicle Routing. *Computers & Operations Research* **11**, 49-66.

[8] Hansen, P. and Mladenovic, N. (2003). Variable Neighbourhood Search. In: Glover, F. and Kochenberger G.A. (Eds.), *Handbook of Metaheuristic,* pages 145-184. Kluwer Academic Publisher, London.

[9] Lee, Y.H., Kim, J.I., Kang, K.H, and 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.

[10] Lin, S. (1965). Computers Solutions of the Traveling Salesman Problem. *Bell System Technical Journal* **44**, 2245-2269.

[11] Mladenovic, N. and Hansen, P. (1997). Variable Neighbourhood Search. *Computers & Operations Research* **24**, 1097-1100.

[12] Osman, I.H. and Salhi, S. (1996). Local Search Strategies for the Mix Fleet Routing Problem. In: Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. and Smith, G.D. (Eds.), *Modern Heuristic Search Methods,* pages 132-153. John Wiley & Sons.

[13] Renaud, J. and 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.

[14] Salhi, S. and Rand, G.K. (1987). Improvements to Vehicle Routing Heuristics. *Journal of the Operational Research Society* **38**, 293-295.

[15] Salhi, S. and Rand, G.K. (1993). Incorporating Vehicle Routing into the Vehicle Fleet Composition Problem. *European Journal of Operational Research* **66**, 313-360.

[16] 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.

[17] Taillard, E.D. (1999). A Heuristic Column Generation Method for the Heterogeneous Fleet VRP. *Recherche Operationnelle* **33**, 1-14.

[18] Wassan, N.A. and Osman, I.H. (2002). Tabu Search Variants for the Mix Fleet Vehicle Routing Problem. *Journal of the Operational Research Society* **53**, 768-782.