A Variable Neighbourhood Search Algorithm for the Open Vehicle Routing Problem

Krzysztof Fleszar*
School of Business

Ibrahim H. Osman
School of Business

Khalil S. Hindi
Faculty of Engineering and Architecture and School of Business
American University of Beirut (AUB)
P.O. BOX 11-0236
Riad El Solh, Beirut 1107 2020
Lebanon

Abstract

In the open vehicle routing problem (OVRP), the objective is to minimise the number of vehicles and then minimise the total distance (or time) travelled. Each route starts at the depot and ends at a customer, visiting a number of customers, each once, en route, without returning to the depot. The demand of each customer must be completely fulfilled by a single vehicle. The total demand serviced by each vehicle must not exceed vehicle capacity. Additionally, in one variant of the problem, the travel time of each vehicle should not exceed an upper limit.

An effective variable neighbourhood search (VNS) heuristic for this problem is proposed. The neighbourhoods are based on reversing segments of routes (sub-routes) and exchanging segments between routes. Computational results on sixteen standard benchmark problem instances show that the proposed VNS is comparable in terms of solution quality to the best performing published heuristics.

*corresponding author, email: kf09@aub.edu.lb

Keywords: variable neighbourhood search, vehicle routing, meta-heuristics

1 Introduction

Distribution of goods is an important operational task faced continually by many enterprises. One problem that arises in this context is the open vehicle routing problem (OVRP), which is one of defining the best routes for a fleet of vehicles that must service a set of customers; the demand and geographical location of each of whom are known. Each route starts at the depot and ends at a customer, visiting a number of customers, each once, en route, without returning to the depot. All vehicles have the same capacity. Each customer must be visited by one, and only one, of the vehicles and its demand must be completely fulfilled. The total demand of all the customers on a route must not exceed vehicle capacity. In one variant of the problem, there is the additional constraint that the travel time of each vehicle should not exceed a given limit, which is defined by driver legal travel time. The objective is to minimise the number of vehicles and for that number of vehicles, to minimise the total distance (or time) travelled by the vehicles. This implies that the cost of an additional vehicle is larger than any savings that may be achieved by reduction of total travel time.

In practice, the problem arises when, as is increasingly the case, the vehicle fleet is hired, or when distribution is entrusted to a third party; in which case the vehicles are not required to return to the depot and the service is charged for by both the number of vehicles and the total distance covered. Clearly, the practical importance of the problem stems from the fact that the associated operational costs are likely to constitute a significant share of overall operational expenses.

From the point of view of graph theory, the difference between the OVRP and the classical vehicle routing problem (VRP) is that a solution of the former consists of a set of Hamiltonian paths, rather than Hamiltonian cycles. The problem of finding the best Hamiltonian path for each set of customers once they are assigned to a vehicle is NP-hard [21]. Hence OVRP is also NP-hard [1].

The VRP has been studied extensively, but few works address the OVRP. Brandão [1] observes that Schrage was the first to raise the problem in an

article dedicated to the description of realistic routing problems [20]. However, the earliest work that addressed solving the OVRP seems to be that of Sariklis and Powell [19], who do not impose a maximum route length. Their proposed algorithm has two phases. In the first, customers are assigned to clusters, taking into account the capacity constraint, while attempting to create the minimum number of clusters, i.e., use the minimum number of vehicles. Total travel cost is then reduced by applying some given rules for moving customers among clusters. In the second phase, a minimum spanning tree is created for each cluster, which is then transformed, through a set of operations, into a, hopefully, short open route.

Employing several features from previous tabu search implementations for the classical VRP [11, 12], Brandão [1] proposes a tabu search, in which the neighbourhood structure is defined by insertions and swaps between different routes. Infeasibilities in intermediate solutions are managed through penalising the objective function by two penalty terms, one for capacity violation and the other for route length violation.

Fu, Eglese and Li [9, 10] also offer a tabu search algorithm, in which the initial solution is provided by a 'farthest first heuristic' and moves are based on the 2-interchange generation mechanism, but with a combination of vertex reassignment, vertex swap, 2-opt and tails swap, within the same route or between two routes.

Tarantilis et. al. [22] address a variant of the problem in which the objective is to minimise the total distance covered, without attempting directly to minimise the number of vehicles and without imposing an upper limit on route length. Their solution algorithm is a single-parameter metaheuristic method that exploits a list of threshold values to guide intelligently a local search, which is based on a variety of edge and node exchanges.

In a more recent work, Li et al. [15] develop a variant of record-to-record travel algorithm for the standard vehicle routing problem to handle open routes. Also, Pisinger and Ropke [17] offer an adaptive large neighbourhood search algorithm, in which solutions are generated by a destruct-and-repair procedure. For example, customers can be removed at random from the solution and then reinserted in the cheapest possible route. Various removal and insertion heuristics can be used to diversify and intensify the search. Moving from one solution to the next is carried out within a simulated annealing framework.

In this work, we present an effective variable neighbourhood scheme (VNS) for solving both variants of OVRP; with and without an upper limit

on route length. It is worth noting that VNS has been applied to various other vehicle routing problems [2, 4, 14, 18].

2 Problem definition and notation

Let n be the number of customers, $N = \{1, 2, \ldots, n\}$ the set of customers, and $N_0 = \{0, 1, \ldots, n\}$ the set of all customers and the depot, which is identified by 0. Let m be the minimum necessary number of vehicles, each travelling exactly one route and $M = \{1, 2, \ldots, m\}$ be a set of vehicles (routes). Let L_i denote the demand of customer $i \in N$, $L(j)$ denote the load of vehicle j, and $L^{\max}$ denote the capacity of each vehicle $j \in M$. Furthermore, let t_{ih} be the travel time between locations $i, h \in N_0$, u_i be the unloading time for customer i, and $T^{\max}$ be the maximum vehicle travel time.

The OVRP is to find the minimum number of vehicles, m, and then find the route of each vehicle, such that the total travel time of all vehicles is minimised and all customer demands are satisfied, while each customer is visited by exactly one vehicle, and for each vehicle neither the capacity, $L^{\max}$, nor the maximum travel time, $T^{\max}$, is exceeded.

3 VNS Solution Scheme

The overall solution scheme is presented in Figure 1. At various stages of the algorithm, we denote by a, a_0, and a' complete solutions. We also denote by $T(j)$ the travel time of vehicle j and by T the total travel time of all vehicles. Throughout, intermediate solutions are generated such that the vehicle capacity constraint is satisfied, but not necessarily the maximum travel time constraint. Let $O(j) = \max\{0, T(j) - T^{\max}\}$ be the overtime of vehicle j, and, therefore, the total overtime of a solution is $O = \sum_{j \in M} O(j)$. For a given number of vehicles, m, we denote by (O, T) the criterion that in comparing two solutions prefers the one with a smaller total overtime, O, or if both solutions have the same total overtime, including zero, prefers the one with the smaller total travel time, T. Clearly, (O, T) aims first at eliminating infeasibility, and then at minimising the total travel time.

The initial number of vehicles is calculated as the maximum of two lower bounds, described later. The first solution, a_0, is calculated using a Best-Fit-Decreasing-Demand (BFDD) heuristic. This is followed by a Variable

```
procedure VNSSolutionScheme();
    m^min = max{LB_1, LB_2};
    for m = m^min to n do
        a_0 = BFDD(m);
        a = VNS(a_0);
        if a is travel-time-feasible return a;
    end;
end;

procedure VNS(a_0)
    a = LocalOptimisation(a_0);
    for k = 1 to k^max do
        if a is not travel-time-feasible and k > k^feasible then return a;
        repeat r times
            a' = Shake(k, a);
            a' = LocalOptimisation(a');
            if a' better than a with respect to (O, T) then
                a = a';
                Restart for loop;
            end;
        end;
    end;
    return a;
end;
```

Figure 1: VNS Solution Scheme

Neighbourhood Search (VNS). If VNS returns a travel-time-feasible solution, the procedure is terminated; otherwise, the number of vehicles is incremented and the whole procedure is repeated.

The VNS follows closely that used in a previous work of the authors [8], which, in turn, is a variant of the basic VNS scheme suggested by Mladenović and Hansen [16]. The first incumbent solution, a, is generated by calling the local optimisation algorithm for the initial solution, a_0. Then, the procedure repeatedly generates a random solution from a neighbourhood of a of size k by shaking, finds a new local minimum starting from that random solution, and adopts it if it is better than the current incumbent with respect to (O, T). In an attempt to intensify the search at each neighbourhood of size k, it is repeated r times. This is a departure from the usual VNS scheme, but one that has proved beneficial. If the incumbent is not improved in any of the

r attempts, the neighbourhood size k is increased. When the incumbent is improved, the size of neighbourhood is reset to 1. The algorithm is stopped when k exceeds the maximum neighbourhood size, $k^{\max}$.

For a particular number of vehicles, m, VNS may fail to find a travel-time-feasible solution. To avert unnecessary search, if a feasible solution is not found within a neighbourhood of size k^{feasible}, that is smaller than $k^{\max}$, the VNS is terminated.

3.1 Estimating minimum number of vehicles

A simple lower bound on the number of vehicles, referred to here as the vehicle capacity lower bound, is $\text{LB}_1 = \lceil \sum_i^n L_i / L^{\max} \rceil$. In addition, in travel-time constrained problems, another lower bound can be calculated as $\text{LB}_2 = \lceil (\text{MST} + \sum_i^n u_i)/T^{\max} \rceil$, where MST is the length of the minimum spanning tree on the nodes representing customers and the depot, with arc lengths equal to travel times. Naturally, the initial estimate of the necessary number of vehicles, $m^{\min} = \max\{\text{LB}_1, \text{LB}_2\}$. It is worth noting, however, that in problem instances where some demands are large relative to vehicle capacity, a better estimate can be determined by bin-packing bounding arguments other than LB_1, or, indeed, by a bin-packing heuristic [7].

3.2 Best-Fit-Decreasing-Demand (BFDD) heuristic

BFDD, which is presented in Figure 2, constructs a solution for a given number of vehicles, m. First, an empty route is created for each vehicle $j \in M$. Then, customers are considered in the non-increasing order of demand. For each customer i, all vehicles $j \in M$ that can accommodate the demand d_i are considered, and the position in which customer i should be inserted into route j in order to minimise the increase of travel time is determined. Once all vehicles have been considered, the best insertion with respect to (O, T) is performed.

When insertion of a customer in a position, p, in a route is tested, the better of two ways of reconnecting the route, shown in Figure 3, is selected. The newly inserted customer is always connected upstream to the customer at position $p-1$, but downstream either to customer at position p or to the last customer in the route. In fact, the second case corresponds to reversing the order in which downstream customers (tail) are visited. This scheme bears some similarity to the insertion schemes in [1, 11].

procedure BFDD(m)
    Create empty route for each vehicle $j \in M$;
    foreach customer $i \in N$ by non-increasing load d_i **do**
        foreach route $j \in M : L(j) + d_i \leq L^{\max}$ **do**
            Determine insertion point of i into j that minimises travel time;
        end;
        Insert i into the best route j with respect to (O, T) criterion;
    end;
end;

Figure 2: Best-Fit-Decreasing-Demand heuristic

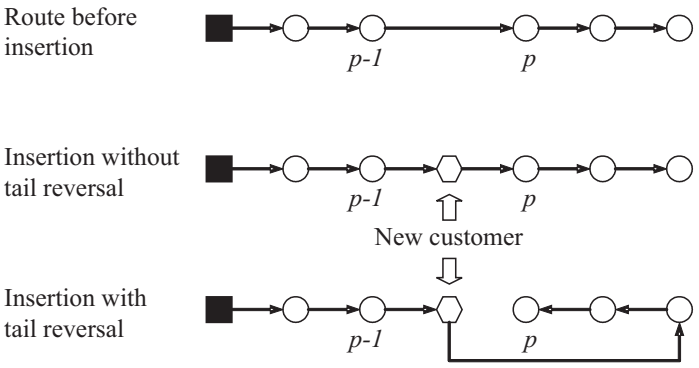

Figure 3: Customer insertion in BFDD

It is possible that at some stage of the heuristic, no vehicle is capable of accommodating the demand of a customer; in which case the heuristic returns no solution. Should this happen, then it would indicate that an attempt should be made to arrive at an assignment of customers to vehicles by bin packing [7].

3.3 BFDD sampling heuristic

In order to generate different initial solutions for multiple runs, a BFDD sampling heuristic has been introduced. Solutions are generated as in BFDD. For each customer, the best insertion in each route is determined. Then, instead of selecting the best route with respect to (O, T), a random route is selected with preference for better routes.

The selection of route in which the customer is inserted is based on the

sampling approach used within Parameterised Regret-Based Biased Random Sampling heuristic for resource-constrained project scheduling [13]. First the set of routes is limited to only those that minimise the overtime increase, denoted by D. Then, the regret of not selecting route $i \in D$ is calculated:

$$\rho_i = \max_{j \in D} T_{\text{inc}}(j) - T_{\text{inc}}(i), \tag{1}$$

where $T_{\text{inc}}(i)$ denotes the increase of travel time if the customer is inserted in route i. Then the route is selected at random with probability of selecting route i equal:

$$\psi_i = \frac{(\rho_i + \epsilon)^\alpha}{\sum_{j \in D} (\rho_j + \epsilon)^\alpha}, \tag{2}$$

where $\epsilon = 1$ and $\alpha = 5$ have been selected on the basis of some experimentation.

3.4 Local optimisation

A segment of route $j \in M$ is a set of consecutive customers assigned to route j. The local optimisation is based on moves involving reversing segments of a route and exchanging segments between two routes.

First, each route is optimised separately using the single-route local optimisation algorithm, described in subsection 3.4.1. Then, all possible exchanges of segments between pairs of routes are tested. The pairs are considered in a random order and the first exchange improving the solution with respect to (O, T) is adopted. This process is described in detail in subsection 3.4.2. Every time an improving exchange is performed, both of the routes involved are immediately re-optimised using the single-route local optimisation algorithm. The local optimisation is completed when no improving exchange exists.

3.4.1 Single-route local optimisation

A move consists of reversing a segment. The procedure tests reversing all segments consisting of at least two customers and performs the reversal that decreases total travel time most (steepest descent). The procedure is completed when no improving move exists. An example of segment reversal is shown in Figure 4.

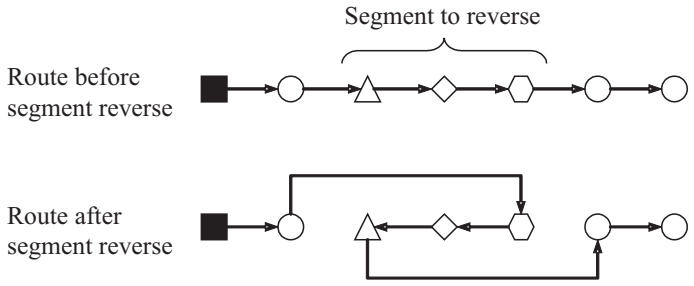

Figure 4: Single-route segment reversal

Note that here minimising vehicle travel time is equivalent to optimising with respect to (O, T). Note also that segment reversal is equivalent to node exchange only for segments of a cardinality (number of customers) less than 4.

3.4.2 Two route segment exchange

For a given pair of routes $i, j \in M, i \neq j$, the procedure considers exchanging pairs of segments, with the cardinality of both segments varying from 0 to all customers on the route; other than, naturally, exchanging two segments of 0 cardinality or two complete routes. This type of exchange obviously includes also moving segments from one route to the other.

It is worth recalling that exchanges are considered only if they do not lead to violation of the vehicle capacity constraint, which limits the number of exchanges considered considerably. To see this, assume that the demand of a segment is the total demand of customers belonging to the segment. Once a segment in the first route is selected, the demand of the segment in the second route must be large enough to create enough space for the first segment in the second vehicle and small enough to fit in the space left after removing the first segment from the first vehicle.

The segment removed from one route is always inserted in the position within the other route from where the other segment was removed. Inserting segments anywhere on the route has been experimented with, but proved much more time consuming without being significantly beneficial.

When a segment is inserted into a route, four possible ways of reconnection are considered and the one minimising the travel time is selected. These cases result from the fact that the segment may be inserted in the original order or in the reversed order and that the part of the route from the breakpoint to the end (the tail) may also be reversed or kept in the original order.

Figure 5 presents the four cases.

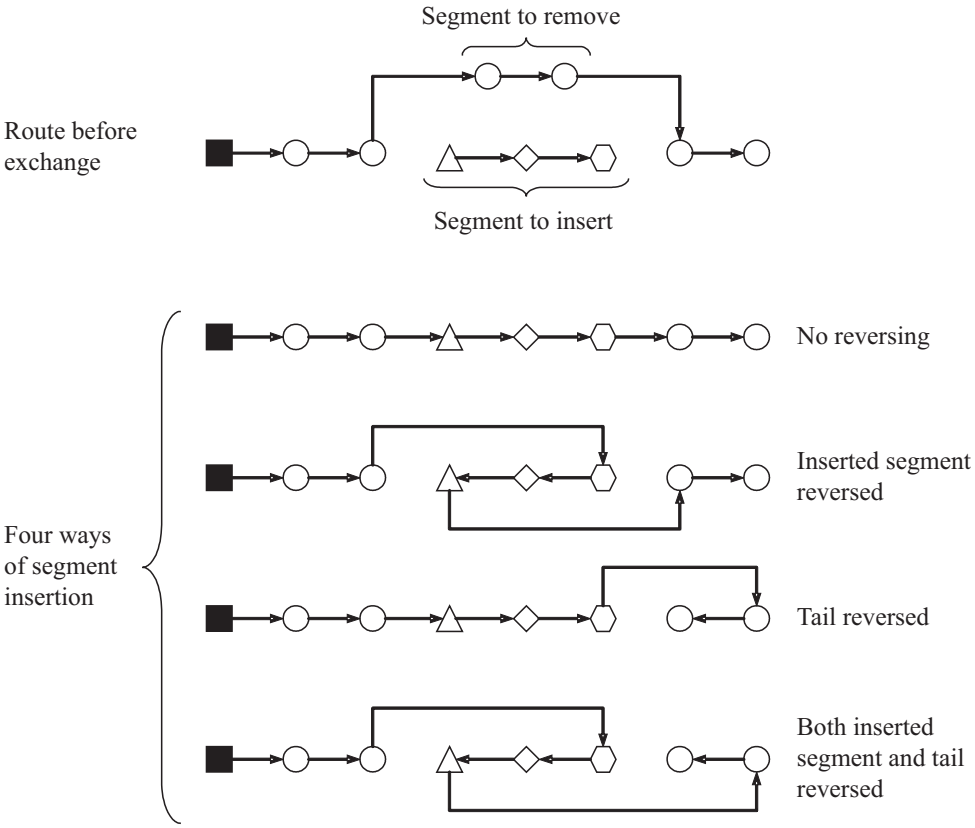

Figure 5: Segment insertion alternatives

3.5 Shaking

The 'Shake' procedure generates a random solution from a neighbourhood of a of size k. This is done by performing k random segment exchanges; each generated as follows. First, two routes are chosen at random and a segment of random cardinality in the first route is randomly selected. Then, the procedure enumerates in a random order segments in the second route until one is found that can be successfully exchanged without violating the vehicle capacity constraint. If no successful exchange is found, the process is repeated as many times as is necessary to succeed.

4 Computational results

The algorithm has been implemented in Visual C# and run on a Dell Latitude D610 with a Pentium M, 2 GHz processor. In the sequel, all comparisons of computing time on various computers are made taking into consideration a relative speed, as established by [5]. However, it must be noted that such comparisons are only indicative rather then definitive.

The test problem instances used in our computational experiments are the benchmark ones used in the literature. They are taken from Christofides et al. [3] and from Fisher [6]. Following Brandão [1], the problem instances are identified as C1–C14 and F11–F12. The number of customers ranges from 50 to 199. C1–C5, C11, C12, F11 and F12 have no travel time constraint. C6–C10, C13 and C14 are the same instances as C1–C5, C11 and C12, but with a travel time constraint.

Tables 1 and 2 present a comparison of the solutions of the various heuristics available in literature with a single-start version of our VNS solution scheme, with $k^{\max} = 4$ and $k^{\max} = 10$, in addition to a multi-start version, where the algorithm is repeated 20 times and the best solution is kept. In all variants, $r = 10$ and $k^{\text{feasible}} = 2$. For the single-start version, the initial solution is generated by the BFDD heuristic (section 3.2), while in the multi-start, the first initial solution is generated by BFDD and the subsequent ones are generated by BFDD sampling (section 3.3).

For each instance, the number of customers, n, the travel time limit, $T^{\max}$, the lower bound on the number of vehicles, $m^{\min}$, and the best number of vehicles achieved by any known algorithm, m^{best}, is presented. Recall that $m^{\min}$ is calculated as the higher value of the capacity and the MST lower bounds. In fact, the MST procedure improved the value of the lower bound in two instances; in C13 from 7 to 10 and in C14 from 10 to 11.

For each algorithm, the columns are the number of vehicles, m, the travel time (without unloading time), and the CPU time. The number of vehicles is reported only if it is higher than m^{best}. For each instance, if an algorithm achieves the best solution out of all algorithms, the corresponding value is boldfaced. At the bottom of the tables the number of times each algorithm achieved the best known solution, #best, is reported.

Sariklis and Powell [19] solve only instances without the travel time constraint. Their results are inferior to those of other heuristics, but their algorithm was given very little CPU time on a relatively much slower computer.

Taking into account the relative performance of different processors, the

Table 1: Comparison of VNS with other heuristics

inst.	n	$T^{\max}$	$m^{\min}$	m^{best}	Sariklis & Powell		Fu et al. (R)		Fu et al. (F)	
					(m)/time	CPU	(m)/time	CPU	(m)/time	CPU
C1	50		5	5	488.2	0.22	**416.1**	0.8	**416.1**	4.1
C2	75		10	10	795.3	0.16	**567.1**	7.8	569.8	8.2
C3	100		8	8	815.0	0.94	643.1	3.0	641.9	23.2
C4	150		12	12	1034.1	0.88	738.9	6.8	742.4	47.3
C5	199		16	16	1349.7	2.20	(17)/879.0	61.9	(17)/879.9	78.0
C6	50	180	5	6			**413.0**	0.6	**413.0**	5.9
C7	75	144	10	10			(11)/568.5	6.3	(11)/568.5	6.0
C8	100	207	8	9			647.3	10.7	648.0	36.3
C9	150	180	12	13			(14)/761.3	46.6	(14)/767.1	79.3
C10	199	180	16	17			903.1	51.9	904.1	133.6
C11	120		7	7	828.3	1.54	724.5	27.5	717.2	23.1
C12	100		10	10	882.3	0.76	534.7	4.2	537.8	10.9
C13	120	648	10	11			(12)/922.3	9.6	(12)/917.9	82.1
C14	100	936	11	11			600.7	2.5	660.2	13.0
F11	71		4	4			**177.0**	0.4	178.2	6.3
F12	134		7	7			778.6	2.8	777.1	28.4
				avg.		0.96		15.2		36.6
				#best	0		4		2	
					Pentium 133 MHz		Pentium IV 3 GHz			

inst.	Brandão (C)		Brandão (D)		VNS $k^{\max}=4$		VNS $k^{\max}=10$	
	(m)/time	CPU	(m)/time	CPU	(m)/time	CPU	(m)/time	CPU
C1	438.2	1.7	**416.1**	88.8	**416.06**	0.5	**416.06**	1.0
C2	584.7	4.9	574.5	167.5	570.62	0.8	**567.14**	2.3
C3	643.4	12.3	641.6	325.3	641.40	4.4	641.40	9.5
C4	767.4	33.2	740.8	870.2	745.41	12.1	737.82	45.4
C5	1010.9	116.9	953.4	1415.0	905.96	13.2	905.96	17.1
C6	416.0	1.4	**413.0**	55.8	**412.96**	1.2	**412.96**	2.8
C7	(11)/581.0	3.4	634.5	123.7	596.47	1.0	596.47	1.8
C8	652.1	10.4	**644.6**	351.7	646.78	13.4	646.78	25.6
C9	(14)/827.6	25.2	785.2	622.2	778.12	25.4	776.12	61.3
C10	946.8	100.1	884.6	2060.3	887.00	26.4	887.00	50.7
C11	713.3	15.7	683.4	696.0	**682.12**	5.7	**682.12**	10.7
C12	543.2	7.8	535.1	233.6	534.40	2.7	534.40	6.7
C13	994.3	25.8	943.7	401.9	920.44	69.7	920.44	108.5
C14	(12)/651.9	8.1	597.3	419.8	592.15	10.6	592.15	19.6
F11	179.5	5.7	177.4	398.1	178.16	3.0	178.16	6.2
F12	825.9	32.7	781.2	1000.2	774.18	24.3	**769.66**	75.4
avg.		25.3		576.9		13.4		27.8
#best	0		3		3		5	
	HP Vectra VEi8, Pentium III 500 MHz				Dell D610, Pentium M 2 GHz			

Table 2: Comparison of VNS with other heuristics

inst.	Pisinger & Ropke 25K		Pisinger & Ropke 50K		Li et al.		20 repetitions of VNS $k^{\max} = 10$	
	(m)/time	CPU	(m)/time	CPU	(m)/time	CPU	(m)/time	CPU
C1	**416.06**	120	**416.06**	230	**416.06**	6.2	**416.06**	17.6
C2	**567.14**	360	**567.14**	530	**567.14**	31.3	**567.14**	29.0
C3	641.76	850	641.76	1280	**639.74**	39.5	**639.74**	239.6
C4	**733.13**	1790	**733.13**	2790	**733.13**	128.6	**733.13**	585.0
C5	897.93	1240	**896.08**	2370	924.96	380.6	905.96	292.1
C6	**412.96**	200	**412.96**	310	**412.96**	10.3	**412.96**	75.8
C7	584.15	180	**583.19**	330	(11)/568.49	32.2	596.47	22.3
C8	645.31	730	645.16	1140	**644.63**	53.2	**644.63**	587.6
C9	759.35	1080	**757.84**	1850	(14)/756.38	195.1	760.06	1094.1
C10	**875.67**	1200	**875.67**	2240	876.02	363.5	**875.67**	1252.4
C11	**682.12**	730	**682.12**	1410	682.54	121.6	**682.12**	231.6
C12	**534.24**	800	**534.24**	1180	**534.24**	32.9	**534.24**	163.7
C13	909.80	610	909.80	1160	(12)/896.50	120.3	**904.04**	1820.1
C14	**591.87**	400	**591.87**	750	**591.87**	62.9	**591.87**	389.0
F11	**177.00**	690	**177.00**	1040	**177.00**	19.5	178.09	140.2
F12	770.17	2370	770.17	3590	**769.66**	158.2	**769.66**	1237.5
avg.		834.4		1387.5		109.7		511.1
#best	9		12		10		12	
	Pentium IV 3 GHz				Athlon 1 GHz		Pentium M 2 GHz	

algorithms of Fu et al. [10] have a CPU time similar to our single-start VNS algorithms, but perform well only on the small instances (C1, C2, C6, F11). Moreover, for 4 instances (C5, C7, C9, C13), they do not find solutions with the smallest known number of vehicles, m^{best}.

Brandão's (C) algorithm [1] achieves solutions inferior to those found by our single-start VNS algorithms, but, taking into account the relative performance of different processors, its computation times are much shorter. Moreover, the solutions found for instances C7, C9, C14 are not with the smallest known number of vehicles.

For all instances, Brandão's (D) algorithm [1] and both variants of our single-start VNS always return solutions with the smallest known number of vehicles. The computation effort of Brandão's (D) algorithm is similar to our single-start VNS with $k^{\max} = 4$; again taking the relative performance of different processors into account. In comparison, our VNS with $k^{\max} = 4$ finds better solutions for 10 instances, the same for 2, and worse for only 4. When $k^{\max}$ is increased from 4 to 10, our VNS improves the solutions of 4 instances, but the CPU time is roughly doubled.

Table 2 compares our multi-start VNS with two high-performance algorithms that were published recently. Pisinger & Ropke run their randomised algorithm ten times and report the best result achieved overall [17]. The results of their second algorithm (50K) are comparable to the results of out multi-start VNS, with each achieving 12 best known solutions (not the same 12). However, taking into consideration that the processors used in both works are comparable in speed, it is clear that our algorithm is about twice faster.

The algorithm of Li et al. [15] has the clear advantage of being effective while requiring relatively short computing times. However, it achieves only 10 of the best-known solutions in terms of travel time with m^{best} number of vehicles. Moreover, for three instances they report results with the number of vehicles being larger than m^{best}. In fact, it is not clear from their presentation of their results what composite objective Li et al. solve to.

5 Conclusions

An effective variable neighbourhood search heuristic for the open vehicle routing problem has been proposed. Computational results on sixteen benchmark problem instances attest to its effectiveness.

Much of the power of the proposed VNS is due to the definition of the neighbourhood structure, which is based on reversing segments of routes (sub-routes) and exchanging segments between routes. The size of neighbourhoods is limited by allowing only exchanges that are feasible in terms of vehicle capacity. However, it would be interesting to find out whether allowing this constraint to be violated in intermediate solution, at the expense of increasing computation time considerably, can lead to the discovery of better solutions, for then the search would be allowed greater freedom.

The work presented demonstrates the power of VNS, being, as shown, capable of finding excellent solutions to an eminently difficult combinatorial optimisation problem. This power should make it a useful metaheuristic for solving many problems in transportation and logistics.

Acknowledgement

We would like to thank Professor José Brandão for providing us with the benchmark problem instances. We are also grateful to Hala Dairy for help in coding.

The comments of four referees helped us improve content and presentation; for that we are immensely grateful.

References

[1] J. Brandão. A tabu search algorithm for the open vehicle routing problem. *European Journal of Operational Research*, 157:552–564, 2004.

[2] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[3] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 313–338. Wiley, Chichester, 1979.

[4] J. Crispim and J. Brandão. Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. *Journal of the Operational Research Society*, 56(11):1296–1302, 2005.

[5] J. Dongarra. Performance of various computers using standard linear equations software. Available at `http://www.netlib.org/benchmark/performance.ps`, 2006.

[6] M. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, 42(4):626–642, 1994.

[7] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Computers and Operations Research*, 29(7):821–839, 2002.

[8] K. Fleszar and K. S. Hindi. An effective VNS for the capacitated p-median problem. *European Journal of Operational Research*, due to appear.

[9] Z. Fu, R. Eglese, and L. Y. O. Li. A new tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society*, 56:267–274, 2005.

[10] Z. Fu, R. Eglese, and L. Y. O. Li. Corrigendum: A new tabu search heuristic for the open vehicle routing problem. *Journal of the Operational Research Society*, 57:1018, 2006.

[11] M. Gendreau, A. Hertz, and G. Laporte. New insertion and post-optimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[12] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[13] R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43:23–40, 1996.

[14] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers and Operations Research*, 34(9):2743–2757, 2007.

[15] F. Li, B. Golden, and E. Wasil. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. *Computers & Operations Research*, 34:2918–2930, 2007.

[16] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[17] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

[18] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10(6):613–627, 2004.

[19] D. Sariklis and S. Powell. A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, 51:564–573, 2000.

[20] L. Schrage. Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, 11:229–232, 1981.

[21] M. Syslo, N. Deo, and J. Kowalik. *Discrete Optimization Algorithms with Pascal Programs*. Prentice Hall, 1983.

[22] C. D. Tarantilis, G. Ioannou, C. T. Kiranoudis, and G. P. Prastacos. Solving the open vehicle routeing problem via a single parameter metaheuristic algorithm. *Journal of the Operational Research Society*, 56:588–596, 2005.