CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Variable Neighborhood Descent for Arc Routing Problems with Time-Dependent Service Costs

Mariam Tagmouti
Michel Gendreau
Jean-Yves Potvin

July 2008

CIRRELT-2008-27

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · UQÀM · HEC Montréal · École Polytechnique Montréal · Université de Montréal

A Variable Neighborhood Descent for Arc Routing Problems with Time-Dependent Service Costs

Mariam Tagmouti[1], Michel Gendreau[1], Jean-Yves Potvin[1,*]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7

Abstract. This paper proposes a variable neighborhood descent for solving a capacitated arc routing problem with time-dependent service costs. The problem is motivated by winter gritting applications where the timing of each intervention is crucial. The variable neighborhood descent is based on neighborhood structures that manipulate arcs or sequences of arcs. Computational results are reported on problems derived from classical capacitated arc routing problem instances. A comparison is also provided with an alternative approach where the arc routing problem is solved after being transformed into an equivalent node routing problem.

Keywords. Capacitated arc routing problem, time-dependent service costs, variable neighborhood descent.

Acknowledgements. We would like to thank Professor Toshihide Ibaraki and his research team for providing us with the computer code of their algorithm. Also, financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

This document is also published as Publication #1324 by the Department of Computer Science and Operations Research of the Université de Montréal.

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2008

1 Introduction

The origins of arc routing can be traced back to the 1730's when Euler studied the Königsberg bridge problem (from the name of a city in Prussia) which led to the notion of Euler cycles and Euler graphs. In 1962, the Chinese mathematician Kwan Mei-Ko proposed a problem whose name was later derived from his origins: the classical Chinese Postman Problem (CPP) [24]. Later on, various extensions of the CPP were reported in the literature, in particular the capacitated arc routing problem (CARP). In this network problem, a subset of required arcs, with some demand on each arc, must be served at least cost by a fleet of capacity constrained vehicles initially located at a central depot.

As opposed to the CPP, the CARP is NP-hard and was first addressed with relatively simple heuristics [5, 6, 13, 26, 27]. Then, several metaheuristics were adapted to this problem like simulated annealing [10], tabu search [1, 3, 17], genetic algorithms [19, 21], ant colony optimization [8] and variable neighborhood search [16].

The main variants of the CARP are:

- *CARP with multiple depots (M-CARP)*. Here, the vehicle routes can start from and end at several depots. This problem is addressed in [4] by first dividing the service area into smaller areas and by locating a depot within each area. Then, routes are designed with the augment-insert CARP heuristic [27] to serve all required arcs. In [1], the problem is transformed into a multiple center capacitated minimum spanning tree problem with arc constraints, which is then solved with tabu search. Ghiani et al. [12] consider a variant of the problem where intermediate facilities (salt boxes) are available to replenish the vehicles along their routes.
- *CARP with time windows (CARPTW)*. In this variant, each required arc has a time window for the beginning of service. This problem has not been studied much in literature, although some early work can be found in [2, 9]. Wohlk presented a node routing and an arc routing formulation for the CARPTW in her thesis [30]. She also proposed a column generation approach to find good lower bounds. Recently, the authors in [28] proposed a greedy randomized adaptive procedure with path relinking for the CARPTW. It is worth noting that in both [28] and [30] the time windows are hard and vehicles are allowed to wait before beginning the service.
- *Periodic CARP*. Here, the required arcs must be served a number of times over a given time horizon. This problem is found in [20] where a genetic

algorithm is proposed to solve it.

- *Stochastic CARP*. In this variant, the demand on the required arcs is a random variable [11].

In this paper, a new variant is considered where a time-dependent piecewise linear service cost is associated with each required arc. This problem is motivated by winter gritting applications, where the timing of each intervention is crucial. That is, if the intervention is too early or too late, the cost in time and material increases. Also, the vehicle is not allowed to wait along its route. This problem was first studied in [29] where small instances were solved with an exact algorithm. We are not aware of any other work on this problem, although a special case would the CARP with (soft) time windows when the service cost is flat and minimal within a given time interval and linearly increases on both sides of this interval.

In the following section, the problem is first precisely defined. Then, a heuristic approach to this problem is introduced in Section 3, based on a variable neighborhood descent. Computational results are reported in Section 4 and a conclusion follows.

2 Problem definition

Let $G = (V,A)$ be a directed graph where V is the vertex set and A is the arc set. It is assumed that A is partitioned into a subset of required arcs A_1, which must be served, and another subset of arcs A_2 which is used for traveling purposes only. With each required arc $e \in A_1$ is associated a demand d_e, a length l_e, a travel time tt_e, a service time st_e, a travel cost tc_e and a time-dependent piecewise linear service cost function $sc_e(T_e)$, where T_e is the time of beginning of service on arc e. The arcs in A_2 have a length, a travel time and a travel cost only. Note that the service time is typically larger than the travel time because it takes more time to serve an arc than to travel along it.

A set of K identical vehicles with capacity Q is available to serve the required arcs. These vehicles are located at a central depot from which each vehicle serves a single route that starts from and ends at the depot. Also, the vehicles are not allowed to wait along their route. The objective is to serve all required arcs in the graph with least-cost feasible routes, where the cost is the sum of service cost and travel cost (or deadhead cost).

Figure 1 shows typical piecewise linear service cost functions for the required arcs. Note that the function in Figure 1(a) is a degenerate form of the one shown in Fig-

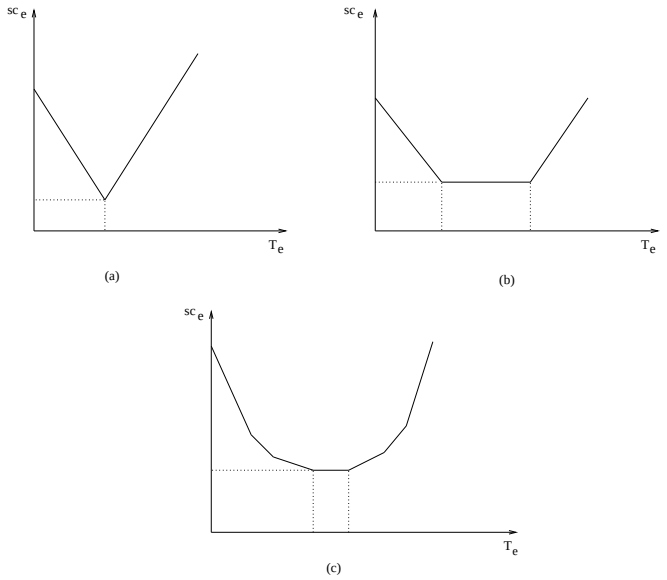

Figure 1: Different arc service cost functions

ure 1(b), where the optimal service time reduces to a single point. The function shown in Figure 1(b) was used in our computational experiments, due to its similarity with soft time windows, but other piecewise linear forms could be used as well, like the one shown in Figure 1(c).

Let R_k be the route traveled by vehicle k, which is made of required arcs $(e_1, ..., e_l)$ and non-required (or deadhead) arcs $(e_{l+1}, ..., e_{l+p})$. These deadhead arcs come from the shortest paths to travel from one required arc to the next. The route cost is then

$$C(R_k) = \sum_{i=1}^{l} sc_{e_i}(T_{e_i}^k) + \sum_{i=l+1}^{l+p} tc_{e_i}, \tag{1}$$

where $T_{e_i}^k$ is the time of beginning of service on each required arc e_i, i = 1, ..., l, served by vehicle k. Since there is no waiting time along the route, the time of beginning of service on each required arc can be derived from the starting time T_0^k of vehicle k at the depot [29]. The route cost can thus be written as:

$$C(R_k) = \sum_{i=1}^{l} sc_{e_i}(T_0^k) + \sum_{i=l+1}^{l+p} tc_{e_i} = sc_k(T_0^k) + tc_k. \tag{2}$$

Figure 2 illustrates such a cost function. In this example, the minimum route cost is obtained for any starting time between 5 and 6 time units.

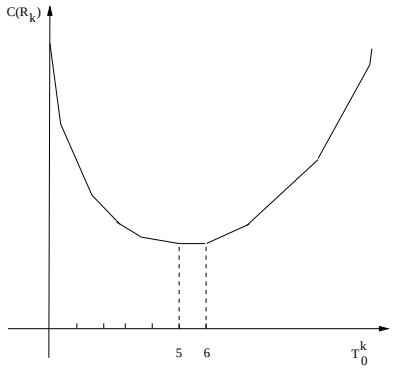

Figure 2: Route cost function

3 Problem-solving methodology

In the first phase, an initial solution is obtained with either an insertion heuristic or an adaptation of the Clarke and Wright heuristic [7]. In the second phase, a variable neighborhood descent (VND), a streamlined variant of variable neighborhood search (VNS) [15, 25], is applied to the initial solution to improve it. The basic idea is to perform a local descent based on a number of different neighborhoods. That is, when a local optimum is reached with the current neighborhood, the search resumes with a different neighborhood to escape from the current local optimum. This is repeated until the current solution cannot be improved anymore, that is, when the solution is a local optimum over every neighborhood considered. This approach was applied, for example, in [16] to solve the CARP using neighborhood structures previously proposed in [17].

3.1 Initial Solution

Savings heuristic

This heuristic in based on the Clarke and Wright savings heuristic [7]. Initially, each required arc is served by a single route that starts and ends at the depot. Then, at each iteration, the pair of routes associated with the largest savings are merged together. This is repeated until no further merging is feasible. Given two routes R_1

and R_2, the savings S_{12} is equal to $C(R_1)+C(R_2)-C(R_{12})$, where R_{12} is the route obtained when R_2 is appended to R_1.

Insertion heuristic

This is a classical sequential insertion heuristic, where the routes are constructed one by one. First, a route is created for serving the closest (unserved) required arc from the depot. Then, at each iteration a new unserved required arc is inserted into the current route, until no additional insertion is feasible. At this point, the procedure is repeated with a newly created route. The next arc to be added to the current route during the insertion procedure is the (feasible) one that incurs the least additional cost to the route.

3.2 Neighborhood structures

Each neighborhood is explored using a first-improvement local descent. These neighborhoods are described in the following in increasing order of complexity. It is worth noting that the last procedure, called shorten, does not really generate a neighborhood structure. It is aimed at reducing as much as possible the travel cost of the current routes.

Arc move

Here, a required arc is removed from one route and inserted between two other required arcs in the same route or in another route. All such moves are systematically considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached. An arc move is illustrated in Figure 3. In this figure, the arcs shown are required arcs while the dotted arcs stand for one or more deadhead arcs on the shortest path from one required arc to the next. Clearly, after moving arc (v_{1j}, v_{1k}), the shortest paths between v_{1j} and v_{1l}, v_{2j} and v_{1j}, v_{1k} and v_{2k} are introduced into the solution to reconnect the two routes.

Cross exchanges

Given a pair of routes in the current solution, two sequences of arcs are exchanged. The two sequences contain exactly m required arcs, $1 \leq m \leq M$, where M is a parameter. This is illustrated in Figure 4 for $m = 2$. Basically, a sequence of arcs in the first route, made of the two required arcs v_{1j} and v_{1k} plus the arcs on the shortest path between them, is exchanged with a sequence in the second route made of the two required arcs v_{2j} and v_{2k}. In the case of the second route, v_{2k} is visited immediately after v_{2j}, so there is no deadhead arc. For a given m, all

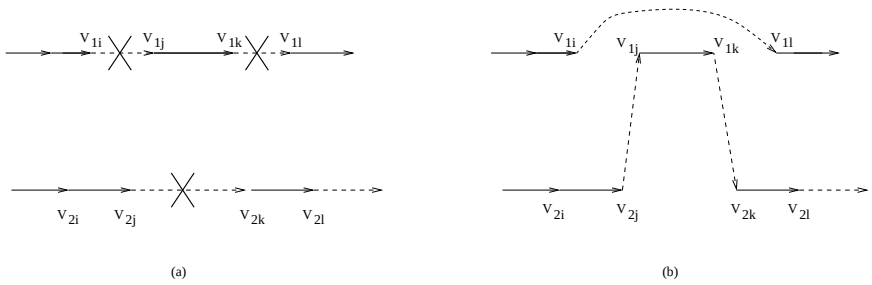

Figure 3: Arc move

possible exchanges on every pair of routes are systematically considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached.

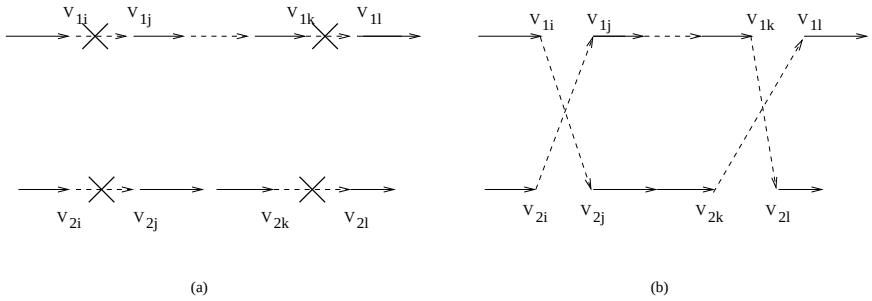

Figure 4: Cross exchange

Block exchanges

Here, sequences made of consecutive required arcs (i.e., sequences with no deadhead arcs in-between), called blocks, are identified and exchanged between two routes. The number of required arcs in a block is not limited and two blocks can be exchanged even if they do not contain the same number of required arcs. Note that exchanges involving two blocks with the same number of required arcs m, for $1 \leq m \leq M$, are discarded because they also belong to the previous cross exchange neighborhoods. For every pair of routes, all possible block exchanges are considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached.

Shorten

In [17], a shorten procedure for the undirected CARP is reported to reduce the

travel cost of a route. This procedure has been adapted for the directed case as follow. When an arc is crossed twice, once for serving it and once for traveling on the shortest path between two other required arcs, it might be possible to improve the route by inverting the order of the two activities on this arc. This is illustrated in Figures 5 and 6. In the first case, the service is postponed after the travel on arc (v_i, v_j) while, in the second case, the travel is postponed after the service. In our context, even if an improvement in travel cost is observed, there is no guarantee that the solution is better overall, due to the service costs. Basically, the procedure is applied sequentially to every required arc which is crossed twice. After processing a required arc, the new solution is kept if it is better overall than the current solution. The shorten procedure then proceeds with the next required arc with this new solution.

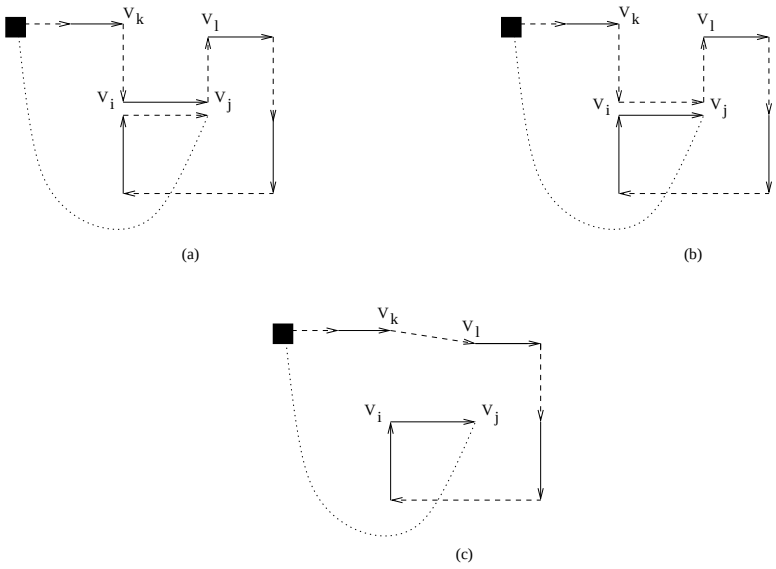

Figure 5: Shorten procedure (service before travel)

3.3 Variable neighborhood descent

The complete algorithm is described in pseudo-code in the following, where:

- N_1 is the arc move neighborhood.
- N_2 to N_{M+1} are the cross exchange neighborhoods with m = 1,..., M.

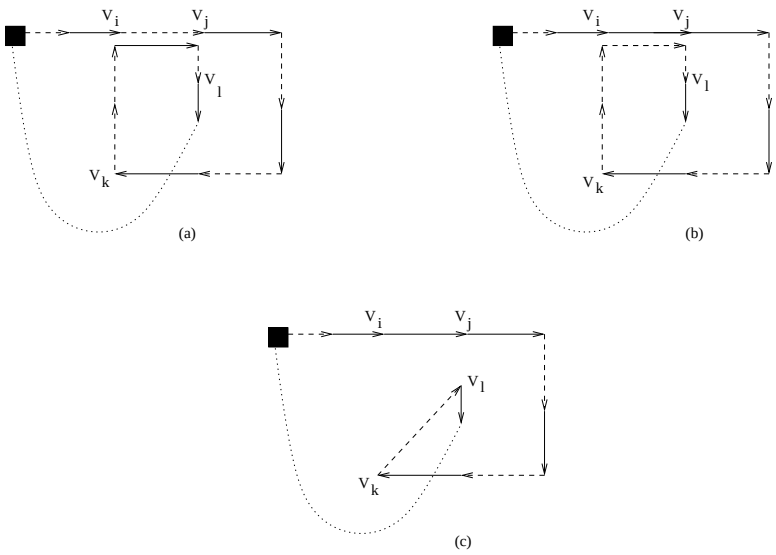

Figure 6: Shorten procedure (travel before service)

- N_{M+2} is the block exchange neighborhood.
- s^{N_j} is a local optimum solution based on neighborhood N_j, j = 1,..., M+2.

VND algorithm

Step 0: Initialization

Create an initial solution s^{N_0}.

Step 1: Loop

For j = 1, ..., M + 2 do:

starting from $s^{N_{j-1}}$, perform a local descent based on neighborhood N_j

and let s^{N_j} be the local optimum obtained.

if s^{N_j} is different from $s^{N_{j-1}}$ then $s^{N_0} \leftarrow s^{N_j}$ and go to Step 1.

Step 2 : Shorten

Apply shorten to $s^{N_{M+2}}$ to obtain s^{short}

If s^{short} is better than $s^{N_{M+2}}$ then $s^{N_0} \leftarrow s^{\text{short}}$ and go to Step 1; else return $s^{N_{M+2}}$.

In this procedure, Step 2 is reached and the shorten procedure is applied only when the M + 2 neighborhoods are explored without any improvement to the starting solution. If the shorten procedure improves the solution, the loop in Step 1 is restarted with the improved solution; otherwise the best solution found is returned. In the computational results, this variable neighborhood descent algorithm is applied twice, using the savings and the insertion heuristics to generate an initial solution. The best of the two runs is then the final result of VND.

4 Computational Results

The undirected CARP instances of Golden et al. [14] and Li and Eglese [22, 23], found at http://www.uv.es/∼belengue/carp.html, have been used to test our VND. In these instances, the length, travel time and travel cost are the same. To use these classical instances, edges were first replaced by arcs, that is, every edge (v_i, v_j) was replaced by two arcs (v_i, v_j) and (v_j, v_i). Then, assuming that (v_i, v_j) is a required edge in the CARP instance, the best known routes for this instance were used to choose an appropriate orientation for the corresponding required arc (i.e., to select either (v_i, v_j) or (v_i, v_j) as the required arc). Then, we applied Dijkstra's algorithm to compute the shortest path between all pairs of required arcs. Finally, a service cost function was defined on each required arc. This function is similar to the one shown in Figure 1(b). It has an optimal time interval where the service cost corresponds to the travel cost in the original CARP instance. Then, on both sides of this optimal interval, the service cost increases linearly based on slope values of -1 and 1. Note that the computational tests reported in the following have all been obtained on a computer with a 3 GHz Intel Pentium IV processor.

4.1 Preliminary study

Our first results were obtained on Golden et al. instances [14] in which the graphs contain up to 27 nodes and 51 required arcs. The results are shown in Table 1, using an objective function with an equal weight on service and travel costs. In these instances, the optimal time interval on each required arc was determined by considering the best known solutions for the corresponding CARP instance. That is, these intervals were set in such a way that the best routes for the CARP also incur the minimum service costs. Thus, every required arc can be visited within its optimal time interval.

For each instance, Table 1 shows the number of vehicles and total route cost (service cost plus travel cost) for VND. The last two columns report the number of vehicles and total route cost for the routes of the best known solutions in the corresponding CARP instances. We can see that our algorithm has always generated the right routes, namely those where each required arc is served within its optimal time interval. This is a strong indication that our algorithm behaves appropriately.

Instance	VND		Best known	
	#Veh.	Route Cost	#Veh.	Route Cost
gdb1	5	316	5	316
gdb2	6	339	6	339
gdb3	5	275	5	275
gdb4	4	287	4	287
gdb5	6	377	6	377
gdb6	5	298	5	298
gdb7	5	325	5	325
gdb8	10	350	10	350
gdb9	10	303	10	303
gdb10	4	275	4	275
gdb11	5	395	5	395
gdb12	7	458	7	458
gdb13	6	536	6	536
gdb14	5	100	5	100
gdb15	4	58	4	58
gdb16	5	127	5	127
gdb17	6	91	6	91
gdb18	5	164	5	164
gdb19	3	55	3	55
gdb20	4	121	4	121
gdb21	6	156	6	156
gdb22	8	200	8	200
gdb23	10	233	10	233

Table 1: Results on Golden et al. instances

4.2 Results with service cost functions of type 1

In this subsection, we report results obtained again on Golden et al. instances, but with another way of setting the optimal time interval for each required arc. Here, the time horizon is divided into three parts (early, intermediate, late) and the optimal time interval for each required arc belongs to exactly one of these three sub-horizons. We also provide a comparison with the recent adaptive multi-start local search algorithm of Ibaraki et al. [18] for solving vehicle (node) routing problems with soft time window constraints that can be used to model our service cost functions. For this purpose, each arc routing problem was first transformed into an equivalent node routing problem, using the technique reported in [29].

As opposed to our arc routing problem, waiting times along the routes are allowed in [18]. The service cost functions, with their optimal time interval, have thus been set to prevent any waiting time to be beneficial (e.g., when the next location can be reached within its optimal service time if a sufficiently long waiting time takes place at the current location). Also, two objective functions are considered in [18] and the second one, which is closer to ours, was selected. It involves a weighted sum of service cost, travel cost and overcapacity. The latter component penalizes capacity constraint violations, as the algorithm is allowed to go into the infeasible domain. We observed that the algorithm failed to find a feasible solution on some instances with the default value of 1 for the weighting coefficients. The weight associated with overcapacity was thus increased to force feasible solutions to emerge.

Each instance was first solved with our VND approach. Then, each instance was solved with the algorithm of Ibaraki et al., based on the number of vehicles obtained with VND (the number of vehicles is an input to the algorithm of Ibaraki et al.). For each instance, Table 2 shows the number of vehicles, total route cost (service cost plus travel cost), and CPU time in seconds to reach the best solution with VND. The last three columns display the same numbers for the code of Ibaraki et al. The CPU time to the best solution was preferred over the total CPU time, because the algorithm of Ibaraki et al. often finds its best solution early, even when it runs for a long time. In the case of our algorithm, the two values are similar because the VND is basically a descent approach where the best solution is generated toward the end of a run (only one additional pass through the neighborhoods is needed to confirm that no further improvement is possible).

The results in Table 2 show that the algorithm of Ibaraki et al. ended up with a solution with fewer vehicles than the number provided in input on instances gdb8 and gdb13. Although our algorithm is not specifically designed to minimize the

number of vehicles, we introduced this consideration into the objective by adding a penalty whenever an arc that leaves the depot is used. As expected, the new solutions obtained often use fewer vehicles, but at the expense of additional route costs, as shown in Table 3. The number of vehicles, total route cost and CPU time in seconds to reach the best solution with the algorithm of Ibaraki et al., using the reduced number of vehicles in input, are also shown in the last three columns of Table 3. Now, the algorithm of Ibaraki et al. could not save any additional route.

The results in Table 3 indicate that, for the same number of vehicles, an average reduction of 1.5% on the total route cost is obtained. Furthermore, the time to get to the best solution with VND is six times faster on average when compared with the algorithm of Ibaraki et al. (although this figure mostly comes from the large gaps observed for gdb22 and gdb23).

4.3 Results with service cost functions of type 2

In these new experiments, the optimal time interval for service is set to $[0, x]$, where x is interpreted as a soft deadline. This parameter was set to three different values, depending on the required arc. This approach is directly motivated by winter gritting operations where different priorities are associated with highways, roads close to schools, etc. The results are shown in Tables 4 and 5, using the format of Tables 2 and 3. Once again, when the VND algorithm was applied without explicitly considering the number of vehicles, the algorithm of Ibaraki et al. ended up with fewer vehicles on 5 different instances (see Table 4). The new results in Table 5 have been obtained by introducing a penalty for the use of any arc that leaves the depot, as it was done for Table 3. In this case, the algorithm of Ibaraki et al. could not save any additional route over the solutions produced by VND.

The results in Table 5 indicate that, for the same number of vehicles, an average reduction of 3.8% on the total route cost is obtained. Furthermore, the time to obtain the best solutions with VND is about four times faster on average when compared with the algorithm of Ibaraki et al.

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
gdb1	5	322	0.03	5	326	0.02
gdb2	6	343	0.14	6	346	0.12
gdb3	6	303	0.02	6	310	0.02
gdb4	5	321	0.02	5	323	0.02
gdb5	6	423	0.06	6	437	0.11
gdb6	5	350	0.16	5	351	0.02
gdb7	5	348	0.18	5	351	0.02
gdb8	13	368	0.63	11	375	1.43
gdb9	12	398	1.35	12	398	11.15
gdb10	5	309	0.06	5	312	0.05
gdb11	7	435	0.87	7	436	0.45
gdb12	7	464	0.03	7	480	0.04
gdb13	9	591	0.07	8	593	0.06
gdb14	5	105	0.07	5	105	0.02
gdb15	4	60	0.05	4	61	0.08
gdb16	6	131	0.05	6	133	0.10
gdb17	8	93	0.04	8	101	0.44
gdb18	5	165	0.56	5	169	0.15
gdb19	4	63	0.01	4	63	0.01
gdb20	6	125	0.02	6	125	0.02
gdb21	7	164	0.12	7	164	0.21
gdb22	15	219	0.31	15	220	0.22
gdb23	12	241	0.57	12	241	13.20
avg.	7.1	275.7	0.23	7.0	279.1	1.22

Table 2: Results on Golden et al. instances

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU time (s)	#Veh.	Route Cost	CPU time (s)
gdb1	5	322	0.03	5	326	0.02
gdb2	6	343	0.14	6	358	0.12
gdb3	5	324	0.03	5	326	0.03
gdb4	5	321	0.02	5	323	0.02
gdb5	6	423	0.07	6	437	0.11
gdb6	5	350	0.13	5	351	0.02
gdb7	5	348	0.18	5	351	0.02
gdb8	11	369	0.66	11	375	1.24
gdb9	11	384	1.45	11	398	3.32
gdb10	5	309	0.06	5	312	0.05
gdb11	6	424	0.87	6	438	0.22
gdb12	7	464	0.03	7	480	0.04
gdb13	7	600	0.26	7	605	0.15
gdb14	5	105	0.08	5	105	0.02
gdb15	4	60	0.06	4	60	0.12
gdb16	5	134	0.06	5	137	2.20
gdb17	5	115	0.20	5	106	0.29
gdb18	5	165	0.60	5	165	0.15
gdb19	3	69	0.01	3	71	0.01
gdb20	5	131	0.05	5	133	0.23
gdb21	7	164	0.13	7	164	0.21
gdb22	9	255	0.41	9	258	14.08
gdb23	12	241	0.60	12	241	13.20
avg.	6.3	279.1	0.26	6.3	283.5	1.55

Table 3: Results on Golden et al. instances with penalty on number of vehicles

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
gdb1	6	370	0.08	6	387	0.15
gdb2	6	415	0.20	6	415	0.26
gdb3	5	331	0.10	5	352	0.17
gdb4	5	352	0.03	5	356	0.02
gdb5	6	430	0.20	6	437	0.06
gdb6	7	375	0.07	6	396	0.23
gdb7	5	336	0.04	5	357	0.13
gdb8	12	399	0.82	11	421	1.71
gdb9	10	379	1.73	10	396	6.78
gdb10	7	321	0.09	6	329	0.17
gdb11	9	467	1.14	8	478	13.49
gdb12	7	515	0.13	7	530	0.13
gdb13	9	584	0.11	9	601	0.24
gdb14	6	105	0.03	6	105	0.17
gdb15	4	60	0.07	4	60	0.18
gdb16	8	135	0.06	8	135	0.10
gdb17	9	95	0.09	7	97	0.59
gdb18	5	194	0.71	5	204	2.82
gdb19	4	63	0.01	4	66	0.02
gdb20	5	127	0.03	5	127	0.07
gdb21	6	225	0.45	6	225	6.03
gdb22	8	297	0.75	8	321	6.04
gdb23	10	327	1.22	10	341	17.87
avg.	6.9	300.1	0.35	6.7	310.3	2.48

Table 4: Results on Golden et al. instances

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	Time to best (s)	#Veh.	Route Cost	Time to best (s)
gdb1	5	379	0.12	5	406	0.07
gdb2	7	385	0.14	7	407	0.13
gdb3	5	331	0.14	5	352	0.17
gdb4	4	360	0.02	4	372	0.03
gdb5	6	430	0.13	6	437	0.06
gdb6	6	373	0.07	6	396	0.18
gdb7	5	336	0.04	5	357	0.13
gdb8	11	404	0.81	11	421	0.86
gdb9	12	357	2.01	12	385	9.60
gdb10	5	311	0.24	5	331	0.09
gdb11	8	455	1.20	8	478	2.09
gdb12	7	515	0.13	7	530	0.13
gdb13	8	595	0.08	8	601	0.46
gdb14	6	105	0.04	6	105	0.17
gdb15	4	60	0.09	4	60	0.18
gdb16	8	135	0.04	8	135	0.10
gdb17	5	109	0.15	5	107	2.88
gdb18	5	194	0.69	5	204	2.82
gdb19	3	64	0.01	3	68	0.02
gdb20	6	125	0.08	6	126	0.07
gdb21	6	225	0.45	6	225	6.03
gdb22	10	256	0.52	10	262	1.64
gdb23	12	277	1.92	12	280	11.50
avg.	6.7	294.8	0.39	6.7	306.3	1.71

Table 5: Results on Golden et al. instances with penalty on number of vehicles

The service cost functions of type 2, which are closer to real applications, were also integrated into the instances of Li and Eglese [22, 23]. Given that manual transformations are required to transform undirected instances into directed ones (e.g., for choosing an adequate orientation of the required arcs, based on the best known solution for the undirected CARP instance) our tests are restricted to a subset of 12 instances. In these instances, the graphs all have 77 nodes and up to 98 required edges.

The results are summarized in Table 6 in the usual format, with the introduction of a penalty on any arc that leaves the depot. We note that the algorithm of Ibaraki et al. has generated fewer vehicle routes on the last seven instances, while the number of vehicles in the VND solutions could not be reduced further, even through an increase in the penalty (but note, once again, that our algorithm was not really designed to minimize the number of vehicles). Thus, neither algorithm really dominates the other here, since the VND solutions have a smaller total route cost, while the solutions of Ibaraki et al. use fewer vehicles, on average.

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
egl-e1-A	5	3705	6.55	5	3775	29.92
egl-e1-B	7	4618	4.88	7	4674	3.13
egl-e1-C	10	5917	3.75	10	5990	85.26
egl-e2-A	7	5270	20.80	7	5272	86.17
egl-e2-B	10	6567	15.57	10	6582	70.31
egl-e2-C	15	8742	9.02	14	8744	120.77
egl-e3-A	9	6662	27.73	8	6687	80.11
egl-e3-B	13	8460	23.57	12	8511	31.19
egl-e3-C	18	10937	22.79	17	10978	119.20
egl-e4-A	10	7064	34.70	9	7082	45.71
egl-e4-B	15	9625	41.53	14	9686	244.08
egl-e4-C	22	12890	27.06	20	12938	170.26
avg.	11.8	7540.3	19.82	11.1	7576.6	90.50

Table 6: Results on Li and Eglese instances with penalty on number of vehicles

5 Conclusion

In this paper, we have considered a CARP with time-dependent service costs. To solve this problem, a heuristic method based on a variable neighborhood descent was proposed. Using instances derived from two benchmark data sets, the method was shown to be fast and competitive when compared with a recently reported algorithm. Future work will now be aimed at introducing perturbation mechanisms into the heuristic search to get a more global exploration of the solution space (at a computational cost, though). We also want to consider a dynamic variant of this problem where the service cost functions are regularly updated due to the arrival of new information, like meteorological forecasts in the case of winter gritting applications.

Acknowledgements. We would like to thank Prof. Toshihide Ibaraki and his research team for providing us with the computer code of their algorithm. Also, financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

References

[1] **A Amberg, W Domschke and S Voss**, *Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees*, European Journal of Operational Research, 360–376, 2000.

[2] **LD Bodin and SJ Kursh**, *A computer-assisted system for the routing and scheduling of street sweepers*, Operations Research 26, 525–537, 1987.

[3] **J Brandão and RW Eglese**, *A deterministic tabu search algorithm for the capacitated arc routing problem*, Computers & Operations Research 35, 1112–1126, 2008.

[4] **D Cattrysse, T Lotan, L Muyldermans and DV Oudheusden**, *Districting for salt spreading operations*, European Journal of Operational Research 139, 521–532, 2002.

[5] **L Chapleau, JA Ferland, G Lapalme and J-M Rousseau**, *A parallel insert method for the capacitated arc routing problem*, Operations Research Letters 3, 95–99, 1984.

[6] **N Christofides**, *The optimum traversal of a graph*, OMEGA, The International Journal of Management Science 1, 719–732, 1973.

[7] **G Clarke and J Wright**, *Scheduling of vehicles from a central depot to a number of delivery points*, Operations Research 12, 568–581, 1964.

[8] **KF Doerner, RF Hartl, V Maniezzo and M Reimann**, *An ant system metaheuristic for the capacitated arc routing problem*, in Proceedings of the 5th Metaheuristics International Conference, Tokyo, Japan, 2003.

[9] **RW Eglese**, *Routeing winter gritting vehicles*, Discrete Applied Mathematics 48, 231–244, 1994.

[10] **RW Eglese**, *Simulated annealing: A tool for operational research*, European Journal of Operational Research 46, 271–281, 1990.

[11] **G Fleury, P Lacomme and C Prins**, *Evolutionary algorithms for stochastic arc routing problems*, Evo Workshop 2004, Lecture Notes in Computer Science 3005, 501–512, 2004.

[12] **D Ghiani, G Improta and G Laporte**, *The capacitated arc routing problem with intermediate facilities*, Networks 37, 134143, 2001.

[13] **BL Golden and RT Wong**, *Capacitated arc routing problems*, Networks 11, 305–315, 1981.

[14] **BL Golden, JS DeArmon and EK Baker**, *Computational experiments with algorithms for a class of routing problems*, Computer & Operations Research, 10, 47–59, 1983.

[15] **P Hansen and N Mladenović**, *Variable neighborhood search methods*, Les Cahiers du GERAD G-2007-52, Montreal, Canada, 2007 (to appear in Encyclopedia of Optimization).

[16] **A Hertz and M Mittaz**, *A variable neighborhood descent algorithm for the undirected capacitated arc routing problem*, Transportation Science 35, 425–434, 2001.

[17] **A Hertz, G Laporte and M Mittaz**, *A tabu search heuristic for the capacitated arc routing problem*, Operations research 48, 129–135, 2000.

[18] **T Ibaraki, S Imahori, M Kudo, T Masuda, T Uno and M Yagiura**, *Effective local search algorithms for routing and scheduling problems with general time-window constraints*, Transportation science 39, 206–232, 2005.

[19] **P Lacomme, C Prins and WR Cherif**, *A genetic algorithm for the capacitated arc routing problem and its extensions*, Evo Workshop 2001, Lecture Notes in Computer Science 2037, 473–483, 2001.

[20] **P Lacomme, C Prins and WR Cherif**, *Evolutionary algorithms for multi-period arc routing problems*, In Proceedings of the Ninth Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems, Annecy, France, 845–852, 2002.

[21] **P Lacomme, C Prins and WR Cherif**, *Competitive memetic algorithms for arc routing problems*, Annals of Operations Research 131, 159–185, 2004.

[22] **LYO Li**, *Vehicle routeing for winter gritting*, Ph.D. Thesis, Dept. of Management Science, Lancaster University, UK, 1992.

[23] **LYO Li and RW Eglese**, *An iterative algorithm for vehicle routeing for winter-gritting*, Journal of the Operational Research Society 47, 217–228, 1996.

[24] **K Mei-Ko**, *Graphic programming using odd and even points*, Chinese Mathematics, 1, 273–277, 1962.

[25] **N Mladenović and P Hansen**, *Variable Neighborhood Search*, Computers & Operations Research 34, 1097–1100, 1997.

[26] **WL Pearn**, *Approximate solutions for the capacitated arc routing problem*, Computers & Operations Research 16, 589–600, 1989.

[27] **WL Pearn**, *Augment-insert algorithms for the capacitated arc routing problem*, Computers & Operations Research 18, 189–198, 1991.

[28] **M Reguioui, C Prins and N Labadi**, *GRASP with path relinking for the capacitated arc routing problem with time windows*, Lecture Notes in Computer Science 4448, 722–731, 2007.

[29] **M Tagmouti, M Gendreau and J-Y Potvin**, *Arc routing problem with time-dependent service costs*, European Journal of Operational Research 181, 30–39, 2007.

[30] **S Wohlk**, *Contributions for arc routing*, PhD thesis, Faculty of Social Sciences, University of Southern Denmark, 2005.