A Dynamic Capacitated Arc Routing Problem with Time-Dependent Service Costs

Mariam Tagmouti
Michel Gendreau
Jean-Yves Potvin

May 2009

CIRRELT-2009-19

A Dynamic Capacitated Arc Routing Problem 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 describes a dynamic capacitated arc routing problem motivated from winter gritting applications. In this problem, the service cost on each arc is a piecewise linear function of the time of beginning of service. This function also exhibits an optimal time interval where the service cost is minimal. Since the timing of an intervention is crucial, the dynamic aspect considered in this work stems from changes to these optimal service time intervals due to weather report updates. A variable neighborhood descent heuristic, initially developed for the static version of the problem, where all service cost functions are known in advance and do not change thereafter, is adapted to this dynamic variant.

Keywords. Capacited arc routing, time-dependent service costs, dynamic, weather reports, variable neighborhood descent.

Acknowledgements. 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: Mariam.Tagmouti@cirrelt.ca

This document is also published as Publication #1339 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, 2009

1 Introduction

In the capacitated arc routing problem (CARP), a set of required arcs, each with an associated demand, is served at minimum cost with a fleet of vehicles of finite capacity based at a given depot. The CARP occurs in real world applications when street segments are served, like winter gritting and street sweeping, or when the demand is aggregated over a number of locations along a street segment, like postman problems. In the classical version of the problem, there is a fixed cost associated with each arc for traveling on this arc or for serving it. In the time-dependent variant of the problem, the service cost on a required arc is also a function of the time of beginning of service.

Winter gritting was first modeled as a CARP by Eglese et Li [8]. The authors discuss routing efficiency issues based on different network characteristics. For example, the presence of T-junctions in rural networks and one-way streets and highways in urban networks are shown to significantly impact routing efficiency measures. In [7], a winter gritting problem is reported where a number of salt depots are located on the network for vehicle replenishment. The problem is solved with simulated annealing, using a savings-based heuristic [6] to generate the starting solution. In [12], the authors propose a two-phase construction heuristic. After selecting an arc to initialize a route, new arcs are inserted in the first phase by going backward from the node at one end of the selected arc back to the depot. In the second phase, arcs are inserted by going forward from the node at the other end of the selected arc to the depot. In [5], the authors first divide the service area into districts that are served from the same depot. Then, routes are constructed in each district with the Augment-Insert heuristic [14]. In [13], a combined depot location-arc routing problem is reported.

Apart from winter gritting, other real-life problems modeled as CARP are:

- Street sweeping: this problem was first studied in [2, 3]. In this work, each arc has to be serviced within a given time window.
- Electric meter reading: the "capacity" constraint here comes from the fact that each employee has a work time limit, see [16].
- Refuse collection: this problem, first considered in [1], is a CARP with two-dimensional capacity constraints due to a true capacity constraint for each vehicle and a work time limit for each driver.
- Snow removal: this problem was addressed in a rural context in [10]. In this work, the authors both consider the minimization of operating costs and the

maximization of road safety while the snow is being removed.

These real-life problems have all been studied in a static context, where it is assumed that all data about the problem are known in advance. However, this is not necessarily the case in real-word applications, where some information might not be readily available when the vehicles start their routes. When new information is unveiled as the routes are executed, the problem becomes dynamic. Although there is fair number of papers on dynamic node routing problems, particularly with regard to the integration of new customer requests into vehicle routes [9, 11, 15], dynamic CARPs have not attracted yet the attention of the research community. In this work, weather report updates lead to modifications to the optimal time of beginning of service on each required arc and thus, to dynamic modifications to the current routes. The service cost function considered on each required arc is a piecewise linear function of the time of beginning of service. This function also exhibits an optimal time interval where the service cost is minimal [17].

The paper is organized as follows. In section 2, the static version of the problem is formally introduced and the variable neighborhood descent (VND) heuristic proposed in [18] to solve it is briefly described. In section 3, the dynamic variant is introduced, followed by a description of the corresponding adaptation of the VND. In section 4, the problem generator to run the tests is detailed and numerical results are reported.

2 The static problem

2.1 Problem definition

Let $G = (V, A)$ be a directed graph where V is the vertex set and A is the arc set. A is partitioned into a subset of required arcs A_1, and a subset of non required arcs A_2. 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. A set of K identical vehicles, each with capacity Q, is available to serve the required arcs.

The objective is to serve all required arcs in the graph with least-cost feasible routes, where the cost is the sum of service costs and travel costs. More precisely, 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 $(e_{l+1}, ..., e_{l+p})$. The route cost is then:

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

where T_0^k is the start time from the depot node of the route served by vehicle k. Given that no waiting time is allowed along a route, the time of beginning of service on any required arc T_{e_i} can be easily derived from the start time of the route T_0^k [17].

Figure 1 shows typical piecewise linear service cost functions. The type illustrated in Figure 1(b), where the flat portion corresponds to the optimal service time interval, is the one used in our computational experiments.

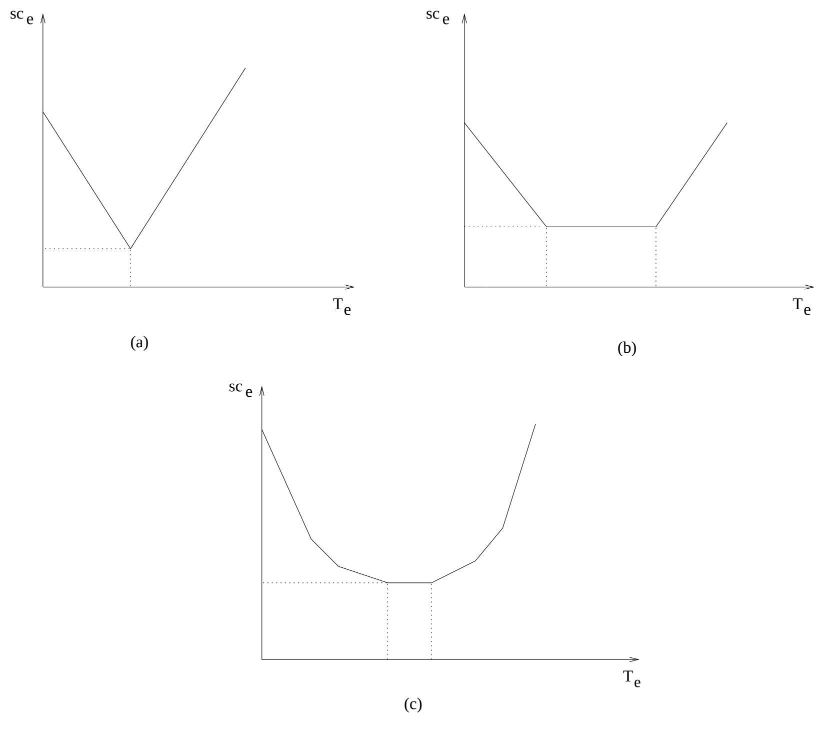

Figure 1: Different types of service cost functions

2.2 Problem-solving methodology

The problem-solving methodology for solving the static version of the problem is reported in [18]. We briefly restate it here for the sake of completeness.

Initial solution

In the first phase, an initial solution is obtained with either an insertion heuristic or an adaptation of the Clarke and Wright's savings heuristic [6].

- *Savings heuristic :*

  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 route merging is feasible.

- *Insertion heuristic*

  First, a route is created for serving the closest required arc from the depot. Then, at each iteration a new unserved required arc is inserted into the route, until no additional insertion is feasible. Additional routes are constructed sequentially in this way until all required arcs are served.

Neighborhood structures

Each neighborhood is explored using a first-improvement local descent. The last procedure presented below, called shorten, does not really generate a neighborhood structure. It is simply aimed at reducing as much as possible the travel cost of the routes by inverting the service and travel on a given arc, when this arc is crossed twice.

- *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.

- *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 set to 5 in our computational experiments.

- *Block exchanges*

  Here, sequences made of consecutive required arcs 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.

- *Shorten*

  When a required 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 2 and 3, using full arcs for service and broken arcs for travel. 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 time-dependent service costs.

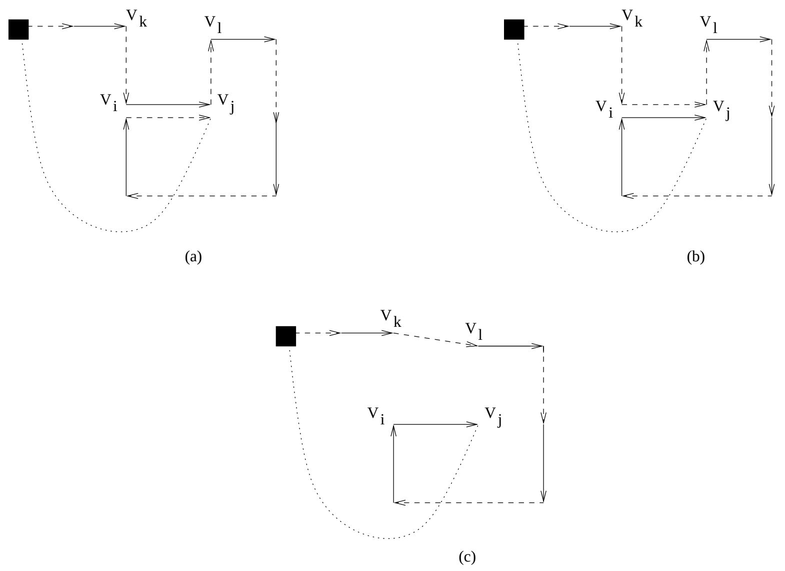

Figure 2: Shorten procedure (service before travel).

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 than the current solution. The shorten procedure then proceeds with the next required arc.

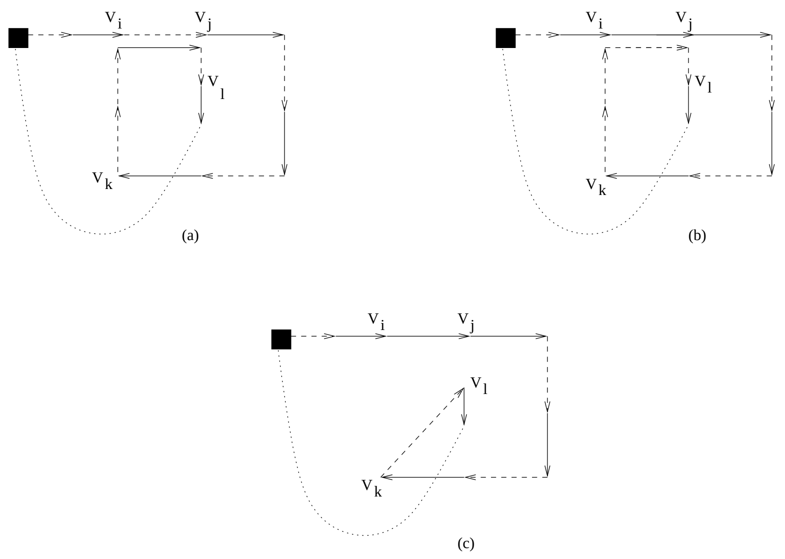

Figure 3: Shorten procedure (travel before service).

Variable neighborhood descent

The complete VND 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 M cross exchange neighborhoods.
- 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.

```
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 $s^{N_{M+2}}$.

This algorithm is executed twice, once with each initialization heuristic, and the best solution is returned at the end.

3 Dynamic variant

In the dynamic variant of the problem, a starting solution is first computed with VND using service time cost functions based on some initial forecast. As vehicles execute their routes, regular weather report updates lead to modifications to the optimal service time interval associated with each required arc. This is explained in the following.

3.1 Context

We assume that a storm (e.g., freezing rain) goes through a city represented as a square in the euclidean plane, where one unit of time is equivalent to one unit of euclidean distance. A first solution is available and computed beforehand based on an initial storm position $(x_0, y_0) = (0,0)$ at time $t = 0$ and some initial storm speed forecast along the x- and y-axis $(sp_0, 0)$. For simplicity purposes, it is thus assumed that the storm moves along the x-axis only. At time h, $2h$, $3h$, ..., where h is a fixed time step, weather reports are received that update the storm speed. This is illustrated in Figure 4, where the storm front is represented as a vertical bar that moves across the euclidean plane (assuming that one unit of time corresponds to one unit of euclidean distance and that the speed of the storm is fixed at one).

In our simulations, the speed varies over time. Namely, if the speed is sp_{t-h} at time $t-h$, then the speed at the current time t is :

$$sp_t = sp_{t-h}(1+\xi_t),$$

where ξ_t is uniformly randomly chosen in $[-\alpha, \alpha]$, with α a parameter.

3.2 Definition of new static problems

Basically, the VND is applied on a new static problem each time an information update (weather report) is received. Each static problem is defined as follows:

- all required arcs that have been serviced are removed from the problem.
- all required arcs that are currently served are assumed to be served by the same vehicle up to their endpoint, using their current service time function.
- all other required arcs have their service time functions updated.

The next section explains how the update is performed on the third category of required arcs.

3.3 Service cost function updates

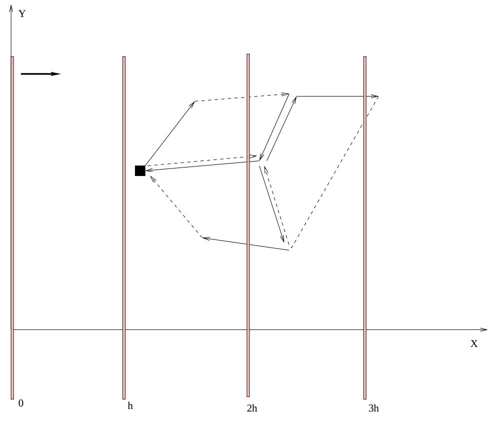

Figure 4: Example of storm movement.

When the speed of the storm varies, the optimal service time interval of each required arc must be shifted along the time line. The time $mt_e(t)$ at which the storm will reach the middle of a required arc $e = (v_i, v_j)$, based on the current time t, is first determined using its new speed:

$$mt_e(t) = t + \frac{mx_e - x_t}{sp_t},$$

where t is the current time, x_t is the location of the storm along the x axis at time t, sp_t is the speed of the storm at time t and $mx_e = \frac{x_{v_i}+x_{v_j}}{2}$ is the middle of arc e.

We chose $mt_e(t)$ to be the middle of the optimal service time interval for arc e. This interval is thus defined as $[mt_e(t)-\gamma, mt_e(t)+\gamma]$, where γ is a parameter set to 3 time units in our computational experiments.

3.4 Time projection

The computation of a new solution with VND takes some time. Accordingly, when a new weather report is received, the current solution is followed for an additional Δt time units using the updated service time functions. During that time, the solution is optimized with VND based on the projected state of the system at time $t+\Delta t$. Due to this time projection, the optimized solution can then be applied as soon as it is available.

3.5 Overall procedure

The evolution of the system over time can be summarized in pseudo-code as follows, using an initial solution S_0 at time $t=0$ based on some initial storm position x_0 and speed sp_0. As previously mentioned, the storm moves only along the x-axis.

```
**Step 0:** Initialization
    $t \leftarrow h$;
**Step 1:** Loop
    While ($t \leq t_{max}$)
        define new static problem with updated service cost functions for
        unserved required arcs, based on new storm location $x_t$ and speed $sp_t$;
        set projected time $t_{proj} \leftarrow t+\Delta t$;
        follow solution $S_{t-h}$ up to $t_{proj}$ while optimizing the solution with VND
        from time $t_{proj}$ onward;
        implement new solution $S_t$ obtained with VND from time $t_{proj}$ onward;
        $t \leftarrow t+h$.
```

3.6 A small example

Figure 5 illustrates a route r with two required arcs e_1 and e_2 with both service time and optimal service cost equal to 1. The travel times and travel costs on non required arcs are also equal to 1. Here, the route cost is recomputed at time $t = 3$, based on new data. As usual, the storm is initially located at $x_0 = 0$ and its speed is 1 euclidean distance unit (du) by time unit (tu), that is sp_0 = 1 du/tu, where the distance and time units are the same.

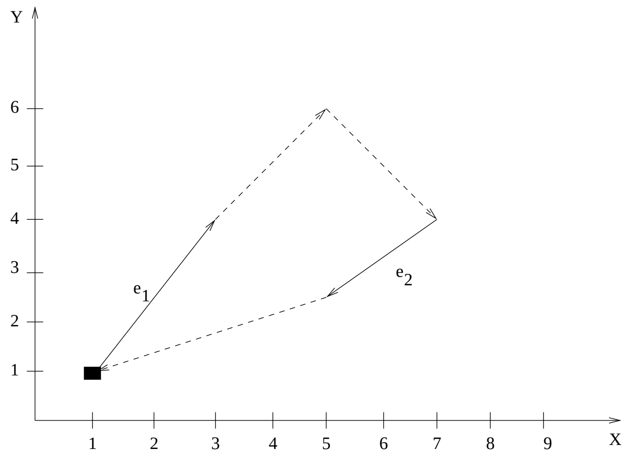

Figure 5: A small example

Since the middle of arcs e_1 and e_2 is located at x = 2 and 6, respectively, their optimal service time interval corresponds to $[1,3]$ and $[5,7]$, respectively, assuming that $\gamma = 1$. If the vehicle starts its route at t = 2, then it serves e_1 within its optimal time interval and the cost is 1. Then, after traveling on the two non required arcs at a cost of 2, it reaches e_2 at time $t = 5$, thus within its optimal time interval, for a cost of 1. Finally, the last arc is traveled at a cost of 1, for a total route cost $C_r = 5$. Now, suppose that a new weather report is received at time $t = 3$, when the storm is located at $x = 3$, and where the speed of the storm is modified to 0.8 du/tu. At this point, a new problem is defined. Since the vehicle has just finished its service on arc e_1, this arc is removed from the problem and a new optimal service time interval is computed for e_2. With its reduced speed, the storm will now reach the middle of arc e_2 at time 6.75 and the optimal time interval becomes $[5.75, 7.75]$. As the new problem contains only required arc e_2, there is nothing to optimize. Thus, the vehicle will arrive at e_2 at time $t = 5$ and an additional cost of 0.75 is incurred (if we assume that the service cost function increases linearly outside of the optimal interval with a slope of one) for a total route cost $C_r = 5.75$. It should be noted that if we knew everything in advance, a better solution could have been obtained. A solution of cost $C_r = 5$ is produced simply by moving the start time of the route from $t = 2$ to $t = 3$, as the vehicle will now reach both required arcs within their optimal time interval. The difference 5.75 - 5 = 0.75 between the two solutions is the value of advanced (or perfect) information.

4 Computational Results

In this section, we first describe the test instances. Then, we report computational results obtained with our VND on these instances.

4.1 Test problems

A simulator was developed to test our algorithm. In this simulator, the vertices are generated within a square in the euclidean plane. The horizontal and vertical sides of the square are divided into n_{max} segments to obtain a grid with n_{max}^2 smaller squares. In each one of them, a vertex is randomly generated, and the depot is located in the bottom left square. Each vertex is connected to the vertices in the North, South, East and West squares with two arcs, one in each direction. Then, 10% of those arcs are eliminated to break the symmetry, while keeping the graph strongly connected. Among the remaining arcs, 50% of them are randomly selected and defined as required arcs. Their demand is randomly set between 1 and 20 while the vehicle capacity is set between 30 and 130.

We generated 3 types of problems with 5×5 = 25, 7×7 = 49 and 10×10 = 100 vertices and with 36, 76, and 162 required arcs, respectively. The first two types of problems are defined in a 10 km $\times$ 10 km squared area which is represented as a 100×100 euclidean square. Accordingly, 1 *du* corresponds to 0.1 km. The vehicle speed is set at 30 km/h or 1 *du*/*tu* (where the distance and time units are the same). Thus, 1 *tu* corresponds to 12 seconds. The storm is initially located at x_0 = 0 with an initial speed sp_0 of 6 km/h or 0.2 *du*/*tu*. New weather reports are received every 5 minutes or 25 *tu*.

In the last type of problems with 100 vertices, a squared area of 25 km $\times$ 25 km is used. Accordingly, 1 *du* is 0.25 km and 1 *tu* corresponds to 30 seconds, assuming again that the vehicle speed is 30 km/h. As in the previous case, the storm is initially located at x_0 = 0 with an initial speed of 6 km/h or 0.2 *du*/*tu*. New weather reports are received every 5 minutes or 10 *tu*.

4.2 Results

Tables 1 to 3 show the average results obtained on each type of problems based on 20 different instances for each type, using three different values for parameter α. We recall that this parameter is used to define the bounds of the storm speed updates, with larger values of α leading to larger speed modifications. In the tables, the first column corresponds to parameter α while the second column contains the number of vehicles in the solutions. The third column contains the cost of the *a priori* solutions, the ratio of their cost to the *a posteriori* solutions (see below) and the computation time in seconds, which includes the time to construct the initial solutions plus the postoptimization time with VND. The *a priori* solutions are the solutions obtained with the initial service cost functions (based on storm speed sp_0), but evaluated in the dynamic setting. The fourth column contains the same values, but for the dynamic solutions. The computation time reported for these solutions corresponds to the largest reoptimization time with VND over all static problems defined during the course of the dynamic process. These times are negligible on the 25 and 50-vertex instances and Δt was set to zero during the simulations. In the case of the 100-vertex instances, Δt was set of $1tu$ or 30 seconds. The last column is the cost of the *a posteriori* solutions, which serve as a basis to quantify the benefits of advanced information, because they are computed with the true service cost functions, namely those obtained at the end of the dynamic process when everything is known. The computation times are not shown for these solutions, because they are very similar to those of the *a priori* solutions.

The cost of the dynamic solutions lies between the cost of the *a priori* and *a posteriori* solutions. The *a priori* solutions are not very good, because they are generated under the initial conditions, but evaluated in the dynamic setting. This explains, in particular, the fast degradation of these solutions with increasing α values. Clearly, when α is large, the initial conditions change a lot and constitute a poor approximation of the true dynamic conditions. On the instances with 25 vertices, the dynamic solutions are quite good for α = 0.05 and 0.1, and lie well within 10% of the *a posteriori* solutions, which are based on perfect information. However, when α = 0.4, the gap jumps to 26.7%. A substantial increase of this gap is also observed on the 50-vertex instances. However, this gap stabilizes on the largest 100-vertex instances, due to a smaller time step between two updates on these instances (i.e., 10 *tu* versus 25 *tu*), which leads to more frequent calls to the reoptimization procedure.

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	7.7	3156.3	1.149	1.03	2850.1	1.042	0.05	2736.7
0.1	7.7	3574.9	1.306	1.04	2919.1	1.070	0.05	2735.0
0.4	7.7	5429.2	1.957	1.03	3522.5	1.267	0.09	2786.3

Table 1: 25-vertex instances

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	16.8	5766.1	1.162	9.39	5409.9	1.089	0.18	4968.6
0.1	16.8	6668.3	1.351	9.17	5669.7	1.146	0.32	4950.3
0.4	16.8	10445.5	2.126	10.38	7404.1	1.502	0.81	4933.6

Table 2: 49-vertex instances

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	22.2	9785.5	1.300	242.18	8188.0	1.086	4.74	7552.3
0.1	22.2	14085.5	1.760	214.48	9240.5	1.164	8.31	7880.0
0.4	22.2	23430.9	2.920	247.75	11951.7	1.485	12.17	8067.3

Table 3: 100-vertex instances

5 Conclusion

In this paper, we have tackled for the first time a dynamic CARP with time-dependent service costs, using a variable neighborhood descent heuristic. We intend to pursue this line of research by improving the proposed problem-solving methodology and by getting closer to real-world applications. With regard to the first issue, it would be interesting to consider an approach that would better utilize the time available between two weather reports. The VND is very fast and allows the system to quickly use the new solution. However, it could be beneficial to use more time for reoptimization, even if it means that the new solution will be made available a little bit later. This trade-off could certainly be the subject of a further study.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

References

[1] **Bodin LD, G Fagin, J Greenberg and R Welebny**, *The design of a computerized sanitation vehicle routing and scheduling system for the town of Oyster Bay, New York*, Computers & Operations Research 16, 45–54, 1989.

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

[3] **Bodin LD and SJ Kursh**, *A detailed description of a computer-assisted system for the routing and scheduling of street sweepers*, Computers & Operations Research 6, 181–198, 1979.

[4] **Campbell JF and A Langevin**, *Roadway Snow and Ice Control*, in: M. Dror (ed.), Arc Routing: Theory, Solutions and Applications, Kluwer, 389–418, 2000.

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

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

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

[8] **Eglese RW and LYO Li**, *Efficient routing for winter gritting*, Journal of the Operational Research Society 43, 1031–1034, 1992.

[9] **Ghiani G, F Guerriero, G Laporte and R Musmanno**, *Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies*, European Journal of Operational Research 141, 1–11, 2003.

[10] **Haslan E and JR Wright**, *Application of routing technologies to rural snow and ice control*, Transportation Research Record 1307, 202–211, 1991.

[11] **Ichoua S, M Gendreau and J-Y Potvin**, *Diversion issues in real-time vehicle dispatching*, Transportation science 34, 426–438, 2000.

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

[13] **Lotan T, D Cattrysse and D Van Oudheusden**, *Winter Gritting in the province of Antwerp: A combined location and routing problem*, Technical Report IS-MG 96/4, Institut de Statistique, Université Libre de Bruxelles, 1996.

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

[15] **Psaraftis HN**, *Dynamic vehicle routing : Status and prospects*, Annals of Operational Research 61, 143–164, 1995.

[16] **Stern HI and M Dror**, *Routing electric letters readers*, Computers & Operations Research 6, 209–223, 1979.

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

[18] **Tagmouti M, M Gendreau and J-Y Potvin**, *A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs*, submitted to Computers & Industrial Engineering, 2009.