Effects of update frequencies in a dynamic capacitated arc routing problem

Wasin Padungwech*
Jonathan Thompson
Rhyd Lewis

Abstract The Capacitated Arc Routing Problem (CARP) concerns a minimum-cost set of routes for vehicles that provide service on edges in a given graph while ensuring that the total demand in each route does not exceed the vehicle's capacity. This paper concerns a dynamic variant of the CARP. In particular, it focuses on a problem in which new tasks appear over time. We find that simply increasing the number of iterations of a tabu search algorithm does not always lead to a better solution for a dynamic CARP. This paper investigates how the solution quality can be affected by changing the frequency of updating solutions. Furthermore, we investigate whether or not such effect varies with a method of integrating new tasks into the solution at each update.

Keywords capacitated arc routing · dynamic · new tasks · update schedule · metaheuristic · tabu search

1 Introduction

In the Capacitated Arc Routing Problem (CARP), introduced by Golden and Wong [6], the goal is to find a minimum-cost set of routes for vehicles that provide service on edges in a given graph while ensuring that the total demand in each route does not exceed the vehicle's capacity. A dynamic CARP is an extension of the CARP in which some information in the problem changes while vehicles are travelling and servicing tasks. Those changes may cause a set of routes that are planned before the changes to have lower quality with respect to total distance (or some other quantity that is being optimised) or even become infeasible, hence the need to update the routes accordingly.

---
*Corresponding author. E-mail: padwasin@gmail.com

School of Mathematics, Cardiff University, Senghennydd Road, Cardiff, CF24 4AG, United Kingdom.

The type of change that we focus on in this paper is the appearance of new demands. It is, however, interesting to note that there are many types of changes that can be considered in a dynamic CARP such as dynamic graphs [13, 16], changes of edge costs [15], and broken-down vehicles [11]. A dynamic CARP with multiple types of changes has also been studied by Liu et al. [8, 9]. Interested readers are referred to relevant papers as referenced.

In most existing studies on dynamic CARPs, solutions are updated as soon as a change occurs. However, for some types of changes such as the appearance of new tasks, a solution does not necessarily need to be updated immediately. In fact, it might be possible that updating a solution at different times would lead to different solution qualities. In this paper, we are interested in how the solution quality might be affected by an update schedule, i.e. a set of times at which a solution is updated.

In practice, route planning and amending needs to be performed in an indefinite period of time as long as new tasks appear [14]. For the purposes of this paper, however, we restrict our attention to a finite period of time, which will also be referred to as a *planning horizon*. It is assumed that all demands that appear within the planning horizon cannot be rejected and must be serviced. In practice, demands that appear after the end of the planning horizon would be dealt with in the next planning horizon, e.g. the next working day. In other words, new demands must be included in the solution by the end of the planning horizon. Nevertheless, they do not need to be added to the solution immediately when they appear. This means that a route planner can decide when to update the solution.

In an extreme case, a route planner may decide to update the solution just once at the end of the planning horizon (while keeping all vehicles idle at the depot until the end of the planning horizon), which would allow all tasks to be added to the solution at the same time. This would be an ideal approach if the total distance travelled by the vehicles was the only measure of concern. However, this would also greatly delay the completion of the services since those new tasks could be started only after the solution was updated. Therefore, in this paper, we consider performing a number of solution updates over the planning horizon as new tasks appear while vehicles are travelling and servicing tasks. We are interested in finding a way to amend a solution as new tasks appear while ensuring that both total distance and service completion time do not increase excessively.

Notice that updating a solution infrequently would allow the route planner to deal with many tasks at the same update. However, a large amount of time between updates means that a large proportion, if not the whole, of a route would be traversed (assuming that the corresponding vehicle travels without stopping from leaving the depot until returning to the depot) and thus could no longer be amended. This reduces the number of possible ways in which a solution can be changed. In contrast, updating the solution more frequently would give more flexibility in making changes to the solution, although a route planner would have less information to exploit about new tasks. This illustrates

the need to identify a solution update frequency that facilitates effective route planning for the dynamic CARP (and dynamic routing problems in general).

This paper describes how the dynamic CARP will be tackled and investigates how the frequency of updating the solution can affect solution quality. The main objective is to minimise total distance, although the solution quality will also be analysed based on service completion time, i.e. the time at which all vehicles return to the depot after servicing all tasks. Apart from the frequency of updating the solution, a comparison will also be made between different ways of integrating new tasks into an existing set of routes.

Section 2 describes the CARP and its dynamic variants. Section 3 then describes components of a proposed solver. In Section 4, we explain a way of generating instances for the dynamic CARP with new tasks. Computational results given by variants of our solver are shown and discussed in Section 5. Section 6 then compares different ways of integrating new tasks into a solution. Conclusions of this paper are given in Section 7.

2 Problem Definition and Solution Representation

2.1 The CARP

In the (static) CARP, we are given a graph $G = (V, E)$ with a set of vertices V and a set of edges E, a positive cost or distance c_{ij} and a non-negative demand d_{ij} for each edge $\{i, j\} \in E$, a capacity Q (assumed to be no less than any demand), and a depot $v_0 \in V$. The objective of the CARP is to find a minimum-cost set of routes that satisfy the following conditions:

- each edge with non-zero demand, also called a *task* (or a *required edge*), is serviced in one of the routes;
- the total demand of edges serviced in each route does not exceed the capacity;
- each route starts and ends at the depot.

A route in a solution can be represented by a sequence $(v_0, a_1, \ldots a_n, v_0)$, where v_0 is the depot, and $a_1, \ldots, a_n$ are tasks that are serviced in the route in the order they appear in the sequence [1]. Since the objective is to minimise the total cost, a path between consecutive tasks can be easily deduced: it is a shortest path between them, which can be found by an algorithm such as that of Dijkstra [4]. Note that even if the graph given in the problem is undirected, it is necessary to state in which direction each task is serviced in each route since such directions can affect the total distance. In other words, each a_i in the sequence $(v_0, a_1, \ldots a_n, v_0)$ is viewed as an arc (i.e. a directed edge).

2.2 Dynamic CARP

In a dynamic CARP, a solution needs to be updated in order to maintain the quality or feasibility of the solution in the face of changes to the problem. Some

tasks may be known before vehicles start travelling (i.e. at time 0), and so an initial set of routes may be constructed based on those tasks. With new tasks appearing over time, a solution needs to be updated to ensure that all tasks are serviced, regardless of when they appear (as long as they appear within the time frame being considered).

Once it has been specified when a solution is to be updated, the dynamic CARP can be viewed as a sequence of static CARPs, where each static CARP occurs at one of the specified points in time. One possible approach to the dynamic CARP is to use an existing algorithm for the CARP to solve each static CARP in the sequence. However, it should be noted that obtaining an optimal solution to each static CARP in the sequence is not guaranteed to give the best possible solution to the underlying dynamic CARP. This is illustrated by the sample dynamic CARP instance in Fig. 1. At time 0, the solution in Fig. 1(a) is optimal, whereas that in Fig. 1(b) is sub-optimal. Then, when task AC appears at time 3, the vehicle in Fig. 1(a) is at Vertex A, and the best way to service the remaining tasks is to travel from A to C, A, and then D, resulting in the total cost of 6. In contrast, for Fig. 1(b), at time 3 the vehicle is at Vertex C, and the best way to service the remaining tasks is to travel from C to A and then D, resulting in the total cost of 5. Notice that the solution in Fig. 1(a) has higher total cost than the final solution in Fig. 1(b) even though it is obtained from an optimal solution at both time 0 and time 3. This is possible because features of the static CARP (e.g. the current position of the vehicle, the remaining tasks) at each update are not only affected by changes that occur in the problem, but also by solutions from previous updates. This highlights the importance of investigating and devising an algorithm specifically for the dynamic CARP, rather than solely relying on algorithms that are specifically designed for the static CARP.

3 Components of a Dynamic CARP Solver

Our process of finding a solution for the dynamic CARP can be divided into three main components: deciding when to update the solution, determining the current state of the problem at each update, and deciding how to re-calibrate the solution to fit with the new problem state. We now consider these in turn.

3.1 Solution Update Schedules

Here we shall focus on a *regular update schedule*. Let T be the length of the planning horizon and N be the number of updates, which is to be specified by the user. Without loss of generality, let the planning horizon start at time 0 and end at time T. Solution updates then take place at time $\frac{kT}{N}$ for $k = 1, \ldots, N$. Intuitively, fewer updates mean more time to collect information about new tasks that appear in the interval prior to each update, while more updates means new tasks can be dealt with more promptly. A regular update schedule
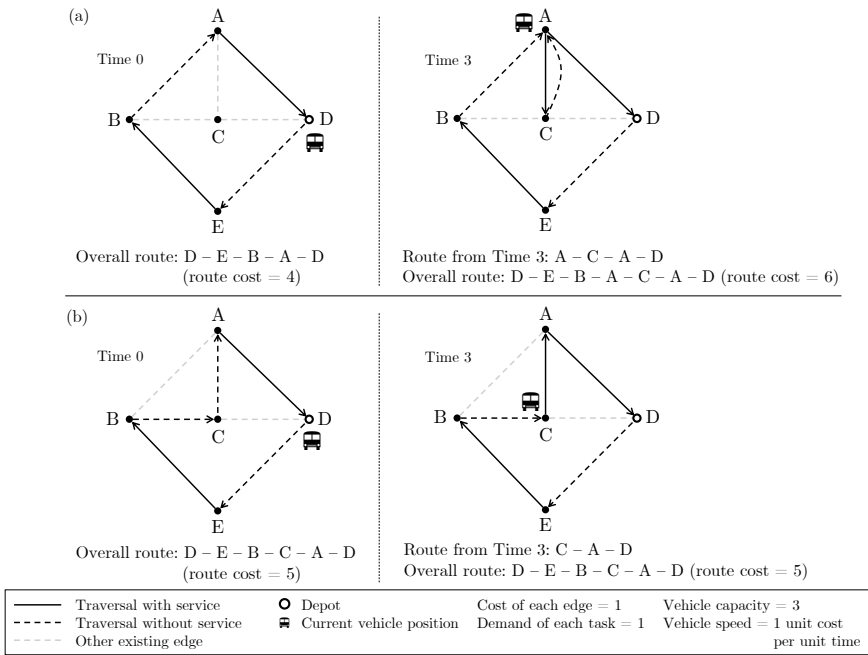
Fig. 1: Sample dynamic CARP instance

has previously been used in dynamic vehicle routing [3, 12]. In particular, computational results given by Montemanni et al. [12] showed that the best solution (with respect to total distance) could be achieved when the number of updates was neither too high nor too low, although it was not explicitly investigated how (or whether) the rate of appearance of new tasks affects the best number of updates. In Section 5.2, different numbers of updates will be tested on various dynamic CARP instances.

3.2 Determining the Current State of the Problem

Before each solution update, the current state of the problem needs to be determined. This involves updating the set of tasks that still need to be serviced, the vehicles' positions, and their remaining capacities. The current state of the problem depends on both the changes in the problem itself and the solutions from previous updates (which define the tasks that have been serviced, and the vehicles' current locations). Also, notice that parts of the routes that have been traversed cannot be amended, so those parts should be clearly identified to ensure that changes made to the solution are feasible. For clarity, this section describes how the current state of the problem and the (non-)amendable parts of the routes are determined.
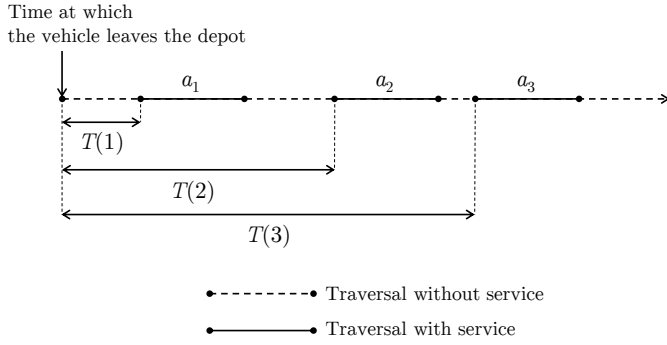
Time at which
the vehicle leaves the depot

Fig. 2: The time $T(i)$ at which a vehicle reaches the ith task in its route.

Given the time at which the solution is updated, each route $R = (v_s, a_1, \ldots,$ $a_n, v_0)$ (where v_s is the starting vertex of the route) is divided into two parts, for some index $\tilde{i}$ and some vertex $\tilde{v}_s$:

$$(v_s, a_1, \ldots, a_{\tilde{i}}, \tilde{v}_s); (\tilde{v}_s, a_{\tilde{i}+1}, \ldots, a_n, v_0) \tag{1}$$

such that the first part cannot be amended (i.e. it is fixed), whereas the second part can still be amended. The index $\tilde{i}$ is thus the largest index i such that a_i has been reached by the vehicle corresponding to the route R. In other words, the index $\tilde{i}$ can be determined by finding the largest index i such that

$$T(i) = \frac{1}{\nu} \sum_{k=1}^{i} [c(a_{k-1}) + D(a_{k-1}, a_k)] \leq t_u - t_s(R), \tag{2}$$

where $T(i)$ is the time at which task a_i is reached (with $T(0) = 0$ by convention), t_u is the time of the update, $t_s(R)$ is the start time of the route R (not all routes need to leave the depot at time 0; see Section 3.3), $c(a)$ is the distance of an arc a (for ease of notation, let $a_0 = (v_s, v_s)$ and $c(a_0) = 0$), $D(a, b)$ the shortest distance from the head of an arc a to the tail of another arc b, and ν is the vehicle speed (distance per unit time). Here it is assumed that ν is a constant and that all vehicles travel at the same constant speed. Inequality (2) ensures that the arc $a_{\tilde{i}}$ is included in the fixed part not only when it has been serviced but also when it is being serviced at the time of the update. This agrees with the assumption that partial service of an arc is not allowed, from which it follows that if a vehicle is currently servicing $a_{\tilde{i}}$, it must continue the service until completion.

The *starting vertex* $\tilde{v}_s$ in (1) is the first vertex to be visited in the route after the given update according to the following rules:

– If the vehicle is precisely at some vertex at the time of the update, then $\tilde{v}_s$ is simply that vertex.

– If the vehicle is in the middle of an edge, then $\tilde{v}_s$ is the endpoint of that edge which the vehicle is heading towards. That edge may be either a task that is being serviced by the vehicle or an edge in the middle of a deadheading path between two consecutive tasks a_i and a_{i+1} for some i.

Once the fixed and the amendable parts of each route are determined, its remaining capacity can be updated accordingly by simply subtracting the total demand of tasks in the fixed part from the remaining capacity at the previous update. After that, its fixed part is archived (so at the end of the planning horizon, a complete journey of each vehicle is a concatenation of fixed parts of the corresponding route that are archived in all updates).

The next step is to combine the amendable parts of the routes and the set of new tasks that have appeared since the previous update.

It should be noted that even though initially (at time 0) all vehicles have the same capacity and their routes start at the same vertex (the depot), they can service different amounts of demands and be at different locations at later time $t > 0$. This means that when updating the solution within the planning horizon, it is possible that a route planner encounters a more general version of the CARP, namely an *open* CARP with *heterogeneous* vehicles. Here, "open" refers to an open route, i.e. a vehicle's starting and ending vertices are not necessarily the same (it could be away from the depot at the time of the update), and "heterogeneous" means that different vehicles can have different (remaining) capacities. These features must be taken into consideration when integrating new tasks into a solution, as we will now consider.

3.3 Integrating New Tasks into the Solution

Once the current state of the problem has been determined, new tasks need to be integrated into the solution to form a feasible solution subject to the current state of the problem. One way to do so is to reconstruct the solution from scratch: all tasks that have not been serviced are removed from the solution and then added back to the solution according to a certain rule. Here we try adding the tasks to the solution using a greedy algorithm, namely the Path Scanning algorithm [7]. Although the Path Scanning algorithm was originally designed for the standard CARP (where all vehicles have the same capacity and start their routes from the same vertex), it can be easily adapted for an open CARP with heterogeneous vehicles.

Obviously, reconstructing the solution from scratch is not the only way to integrate new tasks into the solution. In Section 6, an alternative way of integrating new tasks will be considered.

After new tasks are integrated into the solution, an attempt is then made to improve the solution by means of a tabu search algorithm.

3.4 A Tabu Search Algorithm

3.4.1 Neighbourhood Moves

Our heuristic algorithm for improving a solution is based on a metaheuristic called tabu search [5] and involves four types of neighbourhood moves: single insertion, double insertion, swap, and 2-opt. Details of how these moves work can be found in the appendix. For single insertion, double insertion, and swap, when inserting a task into a route, both directions of traversal on that task are tested.

   A neighbourhood move is said to be *admissible* if either it is non-tabu or it is tabu but leads to the solution that is better that the current best solution. In each iteration of the tabu search algorithm, all neighbourhood moves that are both admissible and feasible with respect to the capacity constraint are considered. Among those moves, the best move (i.e. it leads to the lowest total distance) is selected and applied to the current solution; if there is more than one best move, one of them is selected randomly. The next section describes how the tabu status of neighbourhood moves are determined.

3.4.2 Determining Tabu Moves

After the best neighbourhood move in each iteration is selected, some solution attributes corresponding to the move are recorded in a so-called tabu list (which is initially empty). Each solution attribute that is recorded in the tabu list remains in the list for a certain number of iterations; this number (a *tabu tenure*) is to be specified. To determine whether each neighbourhood move is tabu, solution attributes that would arise as a result of the move are checked, and the move is regarded as tabu if all of those attributes are currently in the tabu list.

Recording solution attributes: Let $R = (v_s, a_1, \ldots, a_n, v_0)$ be a route. When task a_i is removed from route R by either a Single Insertion, Double Insertion, or Swap move, two pairs (a_{i-1}, a_i) and (a_i, a_{i+1}) are recorded in the tabu list (that is, each pair of consecutive tasks involving the move task is recorded). For ease of notation, let $a_0 = v_s$ (the starting vertex) and $a_{n+1} = v_0$ (the depot at the end of the route).

   For 2-opt, the pair of tasks next to the cutting position in each route is recorded in the tabu list. More precisely, if a 2-opt move cuts a route $R = (v_s, a_1, \ldots, a_n, v_0)$ into two parts $(v_s, a_1, \ldots, a_{i-1})$ and $(a_i, \ldots, a_n, v_0)$, then a pair (a_{i-1}, a_i) is recorded.

   Let $\tilde{a}$ denote the opposite direction of traversal on task a. Due to symmetry of routes in an undirected graph, for any tasks a and b, the pair $(\tilde{a}, \tilde{b})$ is regarded as identical to the pair (b, a).

Determining tabu status of moves: A Single Insertion move that inserts task t into a route $(v_s, b_1, \ldots, b_n, v_0)$ between tasks b_{j-1} and b_j is tabu if both pairs

(b_{j-1}, t) and (t, b_j) are currently in the tabu list. For a Double Insertion or Swap move, two pairs corresponding to each inserted task are checked, and the move is tabu if all the pairs are currently in the tabu list.

A 2-opt move is tabu if the pairs of tasks next to the joining positions in both routes are currently in the tabu list. More precisely, consider a 2-opt move that modifies two routes $(v_{s_1}, a_1, \ldots, a_{n_1}, v_0)$ and $(v_{s_2}, b_1, \ldots, b_{n_2}, v_0)$ and gives two new routes, namely $(v_{s_1}, \ldots, a_{i-1}, b_j, \ldots, v_0)$ and $(v_{s_2}, \ldots, b_{j-1}, a_i, \ldots, v_0)$. This 2-opt move is tabu if both (a_{i-1}, b_j) and (b_{j-1}, a_i) are currently in the tabu list.

A pseudocode for the whole dynamic CARP solver is given in Algorithm 1.

Algorithm 1 Configuration of the dynamic CARP solver

1: construct an initial solution S by the Path Scanning algorithm
2: apply the tabu search algorithm to S
3: **for** each update time (see Section 3.1) **do**
4:     let $\mathcal{T}$ be the set of new tasks that appear since last update (or since the beginning if this is the first update)
5:     **if** let $\mathcal{T}$ is not empty **then**
6:         identify the fixed part and the starting vertex of each route for the current update (see Section 3.2)
7:         remove tasks from the non-fixed part of each route and add them to the set of tasks $\mathcal{T}$
8:         reconstruct the solution S with the set of tasks $\mathcal{T}$ by the Path Scanning algorithm (see Section 3.3)
9:         apply the tabu search algorithm to (the amendable part of) S (see Section 3.4)

---

4 Generation of Dynamic CARP Instances

To test variants of the dynamic CARP solver, a set of dynamic CARP instances is required. There exists a benchmark instance generator for dynamic CARPs with various types of changes in the literature [8]. However, this generator explicitly specifies the number of updates, the time of each update, and a set of changes that need to be considered at each update. In contrast, for the dynamic CARP being considered here, the appearance time of each new task is given as part of the problem, whereas the number of updates and the time of each update are variables that are specified by the user. The set of new tasks to be considered in each update then depends on these decisions. To the best of our knowledge, there are no instances for this type of dynamic CARP at the time of writing. Consequently, this section describes how new instances for this type of dynamic CARP were generated for the experiments in this paper.

Before introducing a way of generating dynamic CARP instances, it is useful to know how to measure "dynamism," which will help to classify those instances according to how changes occur. A task that appears after time 0

will be called a *dynamic task*. The *degree of dynamism* (DoD) of a dynamic CARP instance is then defined as the ratio of the number of dynamic tasks to the number of all tasks, including those that appear at time 0. This follows the definition of the degree of dynamism for dynamic vehicle routing given Lund et al. [10].

Our dynamic CARP instances were generated based on existing static CARP instances. Given a static CARP, a dynamic CARP instance can be obtained by first splitting tasks into two groups, static tasks (appearance time = 0) and dynamic tasks, according to the degree of dynamism, and then assigning a random appearance time $t \in \{1, 2, \ldots, T\}$ to each dynamic task (recall that T is the length of the planning horizon). For an instance with the degree of dynamism δ, $\lceil n_t \times \delta \rceil$ tasks are randomly chosen to be dynamic tasks, where n_t is the number of all tasks in that instance.

To allow meaningful comparison of total distances on dynamic CARP instances with different degrees of dynamism, our instances were generated with the following property: a task that is dynamic on an instance with smaller δ is also dynamic and has the same appearance time on an instance with larger δ. By doing so, a feasible solution at the end of the planning horizon on an instance with larger δ is also feasible on an instance with smaller δ. To see this, let $\mathcal{I}_1, \mathcal{I}_2$ be dynamic CARP instances that are generated in the same execution with the DoD on $\mathcal{I}_1$ smaller than that on $\mathcal{I}_2$. Let S_1, S_2 be feasible solutions at the end of the planning horizon on $\mathcal{I}_1, \mathcal{I}_2$, respectively. Because the appearance time of any task, say t, in $\mathcal{I}_2$ is no less than the appearance time of the same task t in $\mathcal{I}_1$, the service on t in S_2 starts no earlier than the appearance time of t in S_1 (notice that the time at which the service on a given task starts must be greater than or equal to the appearance time of the same task). Thus, the solution S_2 would also be feasible in the instance $\mathcal{I}_1$. It follows that the optimal total distance on an instance with larger DoD is no less than the optimal total distance on an instance with smaller DoD that is based on the same static CARP instance and the same appearance times.

For experiments in this paper, a set of dynamic CARP instances were generated based on 20 existing static CARP instances from the BMCV dataset [1]. Details of these static CARP instances are shown in Table 1. The best known lower and upper bounds on these instances are taken from `http://logistik.bwl.uni-mainz.de/benchmarks.php` (last accessed 23 March 2018). Here, the length of the planning horizon is set to 500, which is roughly equal to the average route cost in optimal solutions on the static CARP instances (the last column in Table 1); this helps prevent the planning horizon from being so long that a new task appears after all vehicles return to the depot (with the vehicle speed being 1 unit distance per unit time). In total, 360 dynamic CARP instances[1] were generated: 20 static CARP instances $\times$ 9 degrees of dynamism ($\delta = 0.1, 0.2, \ldots, 0.9$) $\times$ 2 rounds of generating dynamic CARP instances (with different sets of dynamic tasks and their appearance times in different rounds).

---

[1] These instances may be accessed from `https://sites.google.com/view/wasinpad/home`.

Table 1: Characteristics of static CARP instances on which a generation of dynamic CARP instances is based; LB and UB are the best known lower and upper bounds, respectively (LB is omitted when UB is optimal), and n_{veh} is the least number of vehicles needed (total demand divided by capacity, rounded up to the nearest integer); the capacity is 300 for all instances

Instance	Number of vertices	Number of edges	Number of tasks	LB	UB	Total demand	n_{veh}	$\frac{\mathrm{UB}}{n_{\mathrm{veh}}}$
C01	69	98	79		4150	2490	9	461.1
C04	60	84	72		3510	2170	8	438.8
C09	76	117	97	5245	5260	3440	12	438.3
C11	83	118	94	4615	4630	2825	10	463.0
C12	62	88	72		4240	2630	9	471.1
C15	97	140	107	4920	4940	3080	11	449.1
C21	60	84	76		3970	2245	8	496.3
C23	78	109	92	4075	4085	2395	8	510.6
C24	77	115	84		3400	2040	7	485.7
E01	73	105	85	4900	4910	2975	10	491.0
E04	70	99	77		4155	2545	9	461.7
E09	93	141	103	5805	5820	3585	12	485.0
E11	80	113	94		4650	2820	10	465.0
E12	74	103	67		4180	2485	9	464.4
E15	85	126	107		4205	2615	9	467.2
E18	78	110	88		3835	2225	8	479.4
E19	77	103	66		3235	1800	6	539.2
E21	57	82	72		3730	2025	7	532.9
E23	93	130	89		3710	2280	8	463.8
E24	97	142	86		4020	2235	8	502.5

5 Comparison of Variants of the Dynamic CARP Solver

This section presents an analysis of how the dynamic CARP solutions can be affected by adjusting two key components of the dynamic CARP solver: the number of iterations of tabu search in each update, and the frequency of solution updates. Several variants of the dynamic CARP solver are tested on the instances generated in Section 4. Due to the stochastic nature of the algorithm, each variant is run 20 times on each dynamic CARP instance and its performance on that instance is assessed based on average results over 20 runs.

The algorithm performance on a given dynamic CARP instance will be measured in relation to *a posteriori lower bound*, which is the best known lower bound on the corresponding static CARP instance (in other words, when all tasks are treated as if they are all known in advance). More precisely, the algorithm performance will be reported in the form of percentage deviations from *a posteriori* lower bounds:

$$\text{percentage deviation} = \left(\frac{\text{solution cost} - a\ posteriori\ \text{lower bound}}{a\ posteriori\ \text{lower bound}} \right) \times 100.$$

(3)

5.1 The Number of Iterations of Tabu Search in Each Update

In the static CARP, executing tabu search for more iterations will give a better (or at least equally good) solution. In the dynamic CARP, however, it is not guaranteed that a better solution at one update leads to a better solution at a subsequent update (an example is given in Section 2.2). Thus, it remains unclear whether executing tabu search for more iterations in each update would give a better solution (with respect to total distance) at the end of the planning horizon. This section presents a comparison between variants of the dynamic CARP solver that differ in how long tabu search is executed in each update.

Notice that the number of tasks that remain to be serviced can vary over time as new tasks appear and existing tasks are serviced. It is anticipated that the number of iterations that tabu search would need to improve a solution varies with the number of tasks. For this reason, the maximum iteration limit for the tabu search algorithm in each update is given in the form of $k \times n_t$, where n_t is the number of tasks in the given update (that are not in fixed parts of the routes; see Section 3.2 for the description of a fixed part of a route) and k is a constant to be specified. In this section, four different maximum iteration limits (corresponding to $k = 10, 25, 50$, and 100) are tested with three regular update schedules (5, 10, and 20 updates), giving 12 variants of the dynamic CARP solver in total. Each variant is tested on three scenarios: "low," "moderate," and "high" dynamism, represented by dynamic CARP instances with degrees of dynamism $\delta = 0.2, 0.5$, and 0.8, respectively. The tabu tenure is set to half the number of tasks, following the choice of the tabu tenure chosen by Brandão and Eglese [2].

Fig. 3 shows distributions of percentage deviations over 40 instances given by each variant of the dynamic CARP solver; the percentage deviations are computed from solution costs (total distances) at the end of the planning horizon. The experimental results in Fig. 3 show little improvement due to increasing the maximum iteration limit across different degrees of dynamism. Furthermore, a two-tailed Wilcoxon signed-rank test was conducted to make a comparison between each pair of maximum iteration limits (with a Bonferroni correction, resulting in a significance level of $0.05/6 \approx 0.0083$). Table 2 shows that a higher maximum iteration limit does not consistently lead to statistically significant improvement. This suggests that increasing the maximum iteration limit is not a reliable way to improve the performance of the dynamic CARP solver.

Fig. 4 shows averages of elapsed time for executing each variant of the dynamic CARP solver in the whole planning horizon (i.e. accumulating computation time from all updates). It can be seen that a higher iteration limit indeed leads to a greater amount of elapsed time on average, suggesting that the lack of significant improvement from the use of a higher iteration limit observed in Fig. 3 is unlikely to be caused by premature termination (i.e. tabu search terminating before reaching the maximum iteration limit due to the absence of admissible solutions).
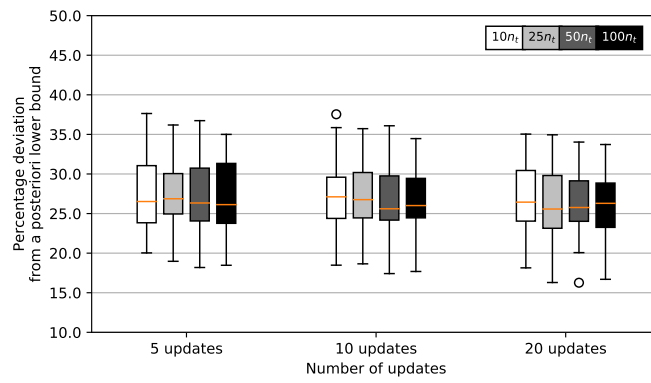
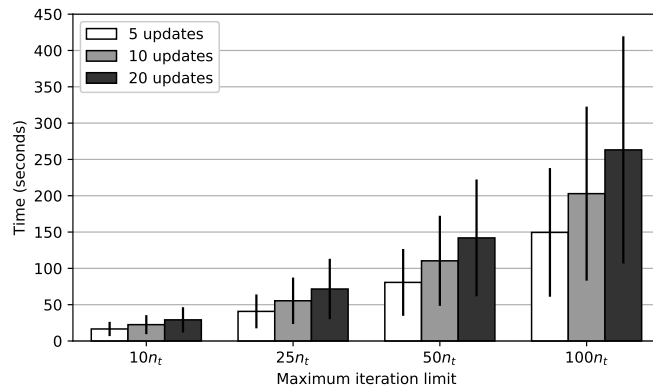(a) Degree of dynamism = 0.2
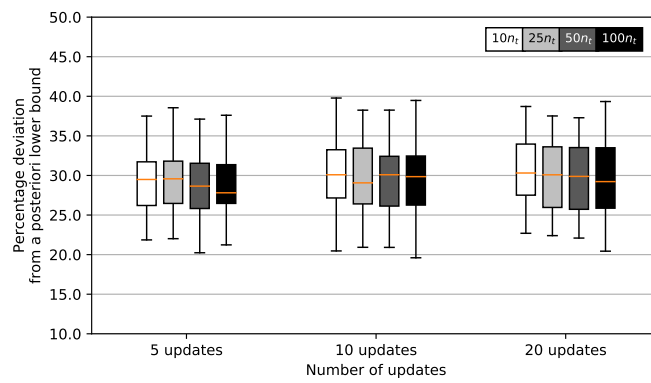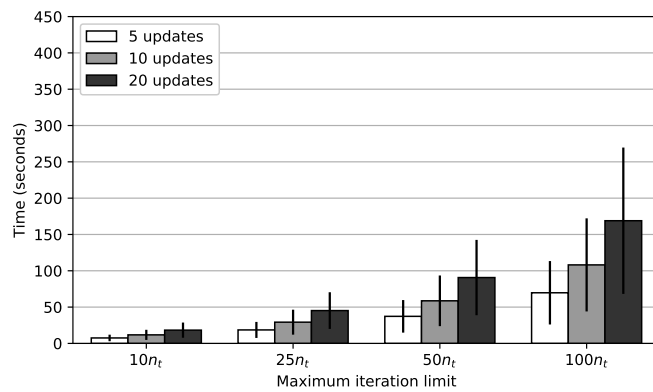

(b) Degree of dynamism = 0.5
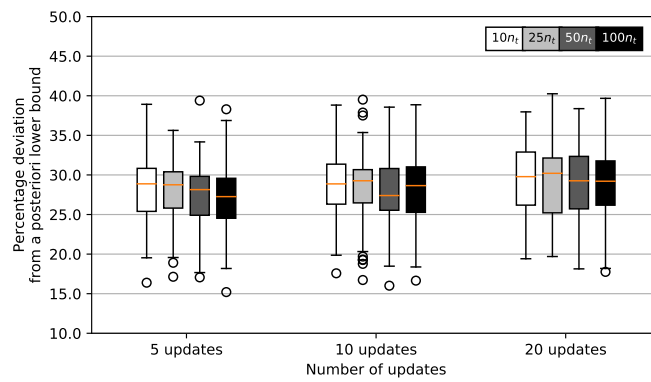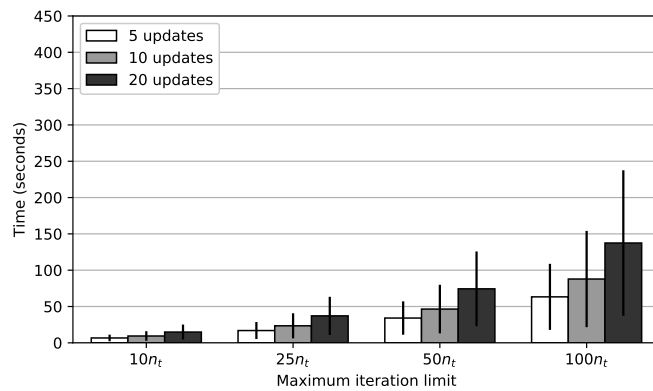

(c) Degree of dynamism = 0.8

Fig. 3: Distributions of average percentage deviations from *a posteriori* lower bounds with respect to total distances given by the dynamic CARP solver with different maximum iteration limits ($10n_t$, $25n_t$, $50n_t$, and $100n_t$, where n_t is the number of tasks) for various degrees of dynamism

(a) Degree of dynamism = 0.2

(b) Degree of dynamism = 0.5

(c) Degree of dynamism = 0.8

Fig. 4: Average elapsed time taken by each variant of the dynamic CARP solver in the whole planning horizon; black vertical lines show one standard deviation from each side of the averages

Table 2: Medians of percentage deviations from *a posteriori* ("static CARP") lower bounds given by the dynamic CARP solver with different maximum iteration limits

Degree of dynamism	Update schedule	Maximum iteration limit			
		$10n_t$	$25n_t$	$50n_t$	$100n_t$
0.2	5 updates	26.5	26.9	26.3	26.1^b
	10 updates	27.1	26.8	25.6	26.0
	20 updates	26.4	25.6	25.8	26.3^a
0.5	5 updates	29.5	29.6	28.7^a	27.8^a
	10 updates	30.1	29.1	30.1	29.9^a
	20 updates	30.3	30.1	29.9^a	29.2^a
0.8	5 updates	28.9	28.8	28.1^a	$27.3^{a,b}$
	10 updates	28.9	29.3	27.4^a	$28.7^{a,b}$
	20 updates	29.8	30.2	29.3	29.2

[a] significantly better than the maximum iteration limit $10n_t$
[b] significantly better than the maximum iteration limit $25n_t$
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction
(for 6 pairwise comparisons), resulting in a significance level of $0.05/6 \approx 0.0083$

It was also observed in Table 2 that the median of percentage deviations increases when using a higher maximum iteration limit in some cases. This illustrates a striking difference between the static CARP and the dynamic CARP: running an algorithm for the static CARP for more iterations would never lead to a worse solution. In contrast, this can occur in the dynamic CARP due to the existence of changes in the problem. Moreover, as executing tabu search with different numbers of iterations generally leads to different solutions, this leads to different problem states (i.e. sets of tasks to be serviced, vehicles positions and capacities). This means that, even though they are based on the same tabu search algorithm, dynamic CARP solvers with different maximum iteration limits generally encounter different sequences of static CARPs over the planning horizon. This allows the relative performance of difference variants of the dynamic CARP solvers to vary and, in some cases, allows the variant with a higher iteration limit to return a worse solution. This further emphasises the need to devise a way to reliably improve the dynamic CARP solver rather than simply increasing the maximum iteration limit.

5.2 Update Schedules

We now turn our attention to the effect of changing the update schedule. For more comprehensive results, three regular update schedules (with 5, 10, and 20 updates) were tested on a wider range of degrees of dynamism $\delta = 0.1, 0.2, \ldots, 0.9$. The maximum number of iterations for the tabu search algorithm at each update was set to $50n_t$, where n_t is the number of tasks. The experimental results with respect to total distances given by different

Table 3: Medians of percentage deviations from *a posteriori* lower bounds given by the dynamic CARP solver with different update schedules

Degree of dynamism	Update schedule		
	5 updates	10 updates	20 updates
0.1	23.4	23.0^a	22.2^a
0.2	26.3	25.6	25.8
0.3	28.5	29.1	28.6
0.4	29.1	28.6	29.5
0.5	28.7^c	30.1	29.9
0.6	$28.1^{b,c}$	29.1	29.8
0.7	$28.2^{b,c}$	29.6	29.0
0.8	28.1	27.4^c	29.3
0.9	$24.9^{b,c}$	26.1	28.0

[a] significantly better than the 5-update schedule
[b] significantly better than the 10-update schedule
[c] significantly better than the 20-update schedule
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction
(for 3 pairwise comparisons), resulting in a significance level of $0.05/3 \approx 0.017$

update schedules are shown in Fig. 5(a). Also, a two-tailed Wilcoxon signed-rank test was conducted to compare each pair of the update schedules and the test results are shown in Table 3. It was found that the best update schedule with respect to total distance varies with the degree of dynamism. For relatively low degrees of dynamism, a more frequent update schedule tends to perform better. In particular, the 10-update and the 20-update schedules are significantly better than the 5-update schedule on the instances with the degree of dynamism = 0.1. As the degree of dynamism increases, the performance of different update schedules becomes more and more similar to each other. Then, for relatively high degrees of dynamism, a less frequent update schedule generally performs better; in some cases, the 5-update schedule is significantly better than the 10-update and the 20-update schedules.

The experimental results in terms of service completion times are shown in Fig. 5(b). A similar Wilcoxon signed-rank test was also conducted to compare each pair of the update schedules in terms of service completion times, and the test results are shown in Table 4. The results suggest that a less frequent update schedule tends to result in a later service completion time, especially when the degree of dynamism is relatively low. In fact, the service completion time given by the 5-update schedule is significantly worse than the 10-update and the 20-update schedules on the instances with degrees of dynamism in the range 0.1 to 0.4. For relatively high degrees of dynamism (≥ 0.5), however, the service completion times for different update schedules are generally similar to each other, and no significant difference between the results of different update schedules were found in most cases.

As we have seen in Fig. 5(a), updating the solution many times generally leads to relatively poor results for relatively high degrees of dynamism. A possible cause of this is the current way of integrating new tasks - that
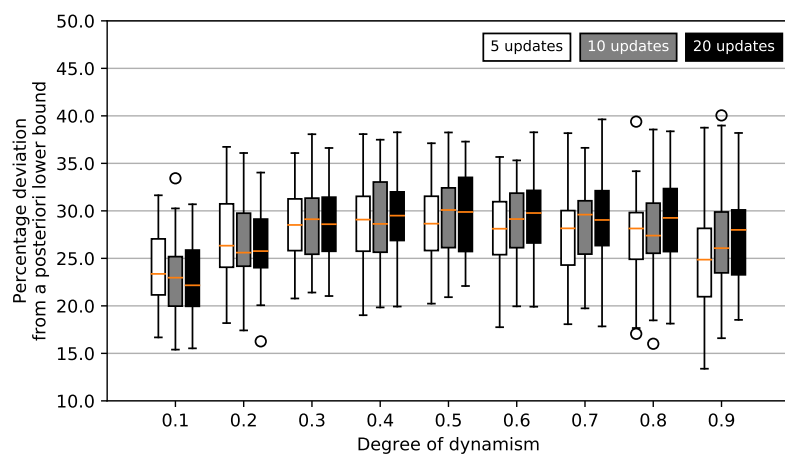
(a)
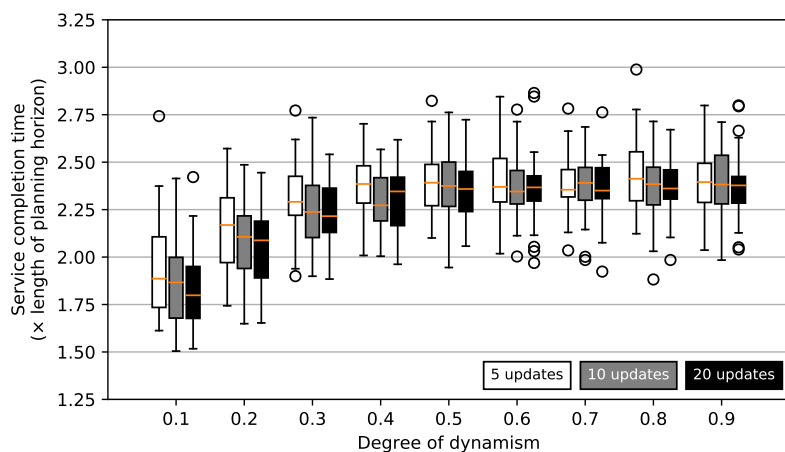
(b)

Fig. 5: Distributions of percentage deviations from *a posteriori* lower bounds with respect to total distances (a) and distributions of service completion times (b) given by different update schedules over 40 instances for each degree of dynamism (0.1, 0.2, ..., 0.9)

is, the solution is reconstructed from scratch before tabu search is applied in each update. As a result, any knowledge that tabu search has gained in previous stages about the problem (e.g. promising or unpromising sequences of tasks that should be serviced in the same route) is not transferred to future updates. In this case, a less frequent update would allow the solver to collect more information about new tasks for each update, which could then help the algorithm find better solutions. On the other hand, increasing the update

Table 4: Medians of service completion times (as multiples of the planning horizon length) given by the dynamic CARP solver with different update schedules

Degree of dynamism	Update schedule		
	5 updates	10 updates	20 updates
0.1	1.89	1.87^a	1.80^a
0.2	2.17	2.11^a	2.09^a
0.3	2.29	2.24^a	2.22^a
0.4	2.38	2.27^a	2.35^a
0.5	2.39	2.37	2.36
0.6	2.37	2.35	2.37
0.7	2.35	2.39	2.35
0.8	2.41	2.38^a	2.36^a
0.9	2.40	2.38	2.38

[a] significantly better than the 5-update schedule
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction
(for 3 pairwise comparisons), resulting in a significance level of $0.05/3 \approx 0.017$

frequency results in less information about new tasks in each update, and consequently the solver is more susceptible to making poor decisions when amending the solution in each update due to limited information.

To understand the effect of update schedules observed in the case of relatively low degrees of dynamism, it should be noted that for some of the dynamic CARP instances considered here, some updates are omitted due to the absence of new tasks, especially when the degree of dynamism is low. This is illustrated by Fig. 6, which shows the number of actual updates, i.e. those in which there exist new tasks, under different update schedules on the dynamic CARP instances considered here across different degrees of dynamism. Notice that the difference between the numbers of actual updates for different update schedules is relatively small for low degrees of dynamism. Also notice that fewer updates are omitted as the degree of dynamism increases. In fact, no updates under the 5-update and the 10-update schedules are omitted for sufficiently large degrees of dynamism.

When the degree of dynamism is low, the solver does not suffer much from updating the solution too frequently even if a frequent update schedule is used; this is due to the absence of new tasks in some updates in the dynamic CARP instances considered here. In fact, a more frequent update schedule means that there are more updates arranged throughout the planning horizon, which allows new tasks to be added to the solution more promptly. The effect of this is particularly evident in Fig. 5(b): the service tends to be completed at an earlier time under a more frequent update schedule. Furthermore, since more and more parts of the routes would be fixed as vehicles travel along their routes, adding new tasks to the solution at an earlier time would allow more possibilities to amend the solution, hence a greater chance of finding better solutions.
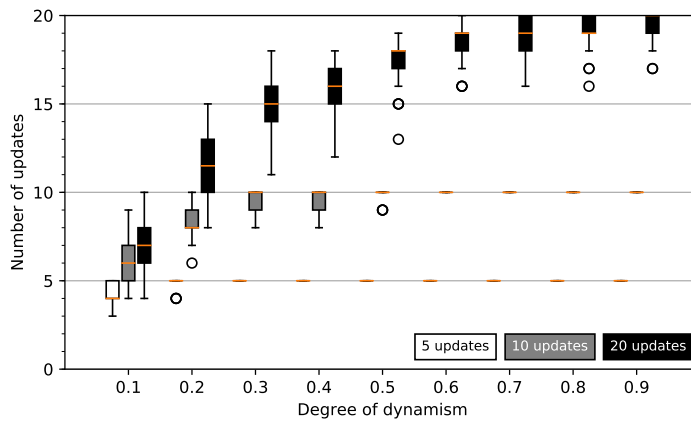
Fig. 6: The number of updates in which new tasks exist on 40 dynamic CARP instances generated in Section 4 for each degree of dynamism (0.1, 0.2, ..., 0.9); note that some box plots are reduced to single lines due to zero variance

It is worth reminding ourselves at this stage that the above comparison of different update schedules is based on the dynamic CARP solver in which an initial solution in each update is reconstructed from scratch. The next section proposes another way of integrating new tasks to the dynamic CARP solution in each update and investigates whether it is beneficial to retain solutions from previous updates as opposed to solving the problem in each update from scratch.

6 An Alternative Method of Integrating New Tasks

In previous sections, as well as in existing work on the dynamic CARP in the literature [9], the problem at each update is solved from scratch. This section proposes a novel concept that aims to retain vehicle routes that have been improved by the dynamic CARP solver throughout the planning horizon: in each update, instead of reconstructing a solution from scratch (the "Reconstruction method"), new tasks are inserted into existing routes one by one in a random order. This alternative method will be called the *Random Insertion* method. The motivation behind this idea is that the solution has been improved by tabu search in previous updates and so would be likely to contain some favourable characteristics (such as certain sequences of tasks), which would be lost if the solution was reconstructed from scratch. The Random Insertion method attempts to integrate new tasks into a solution in such a way that retains most characteristics of the solution given by the previous update.

The Random Insertion method works as follows. First, new tasks are arranged in a random order; they are then added to a given solution in that

order by means of a greedy method: that is, each task is added to a given solution in a way that results in the least possible increase in the total distance. To find such a cheapest way to insert a task, let $D(a, b)$ denote the shortest distance from the head of task a to the tail of task b for any tasks a, b. For a route $R = (v_s, a_1, \ldots, a_n, v_0)$ and a task a', the cost (or more precisely, change in the total distance of the solution) incurred by inserting task a' into route R between tasks a_j and a_{j+1} for some $j \in \{0, 1, \ldots, n\}$ is equal to

$$D(a_j, a') + D(a', a_{j+1}) - D(a_j, a_{j+1}), \tag{4}$$

where, for ease of notation, $a_0 = (v_s, v_s)$ and $a_{n+1} = (v_0, v_0)$. For each new task, all routes with sufficient capacities are considered, and the task is inserted into the route that results in the cheapest insertion cost according to the expression $(4)^2$. If there are many routes into which the task can be inserted with the cheapest cost, then one of them is chosen randomly. If there are no routes with sufficient capacities, then a new route is created for the task being considered. A description for the Random Insertion method is given in Algorithm 2. The Random Insertion method is implemented in place of the Reconstruction method in the dynamic CARP solver (see Lines 7 and 8 in Algorithm 1).

Algorithm 2 The Random Insertion method

1: **given** a set of new tasks $\mathcal{T}$, and a set of routes S
2: **for** each task $t \in \mathcal{T}$ (in a random order) **do**
3:     **if** there exists a route with sufficient remaining capacity to service t **then**
4:         insert t into one of the routes in S that incurs the cheapest cost (if there is more than one such route, choose one of them randomly)
5:     **else**
6:   add t to a new route

---

6.1 Computational Results

We now look at the performance of the dynamic CARP solver with different update schedules when the Random Insertion method is used in place of the Reconstruction method. Figures 7(a) and 7(b) show the performance of the dynamic CARP solver using the Random Insertion method with different update schedules based on two measures, namely total distances and service completion times, respectively. Statistical significance of the results with respect to total distances and service completion times are shown in Table 5 and Table 6, respectively.
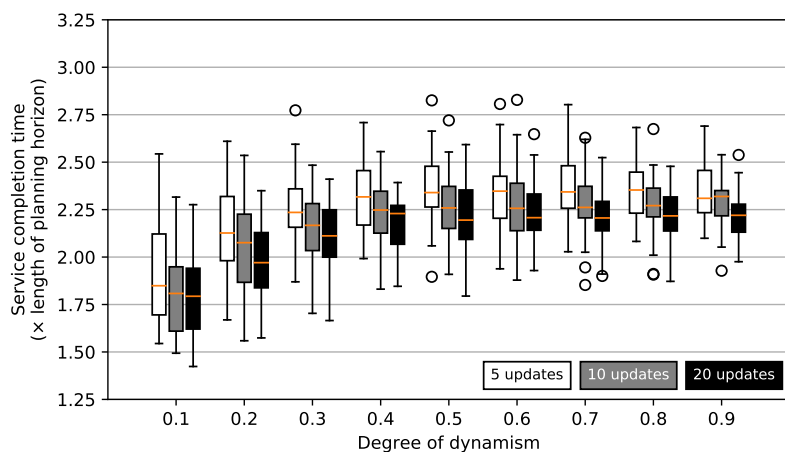
The experimental results show that when the Random Insertion is implemented in place of the Reconstruction method, a more frequent update

---

^{2} Note that the cost of servicing task a' is omitted in the expression (4) because it is independent of the position of insertion and thus has no effect on the best insertion position.
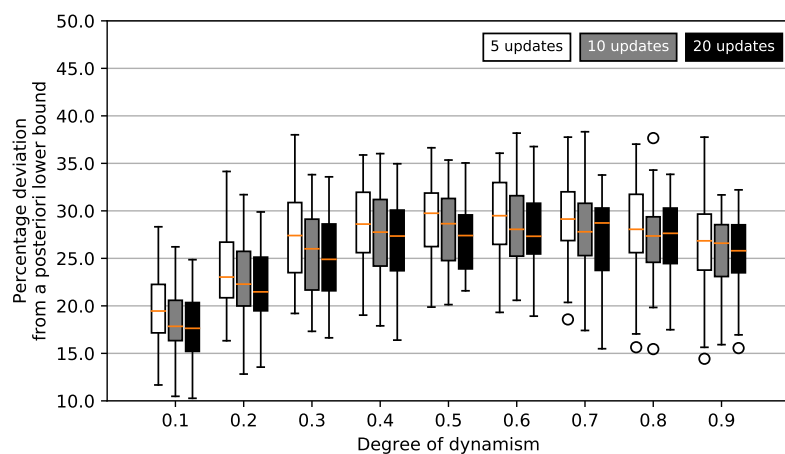
(a)

(b)

Fig. 7: Distributions of percentage deviations from *a posteriori* lower bounds with respect to total distances (a) and distributions of service completion times (b) given by different update schedules with the Random Insertion method over 40 instances for each degree of dynamism (0.1, 0.2, . . . , 0.9)

schedule generally leads to better solutions across different degrees of dynamism. This is different from the results with the Reconstruction method seen previously in Section 5.2, where the best update schedule varies with the degree of dynamism. A possible reason for this is that the Random Insertion method allows the solver to retain a solution that has been improved from previous updates. This means that an initial solution in each update has a higher chance of already containing promising features (such as certain

Table 5: Medians of percentage deviations from *a posteriori* lower bounds given by the dynamic CARP solver with the Random Insertion method and different update schedules

Degree of dynamism	Update schedule		
	5 updates	10 updates	20 updates
0.1	19.5	17.8^a	$17.6^{a,b}$
0.2	23.0	22.3^a	$21.5^{a,b}$
0.3	27.4	26.0^a	$24.9^{a,b}$
0.4	28.6	27.8	$27.3^{a,b}$
0.5	29.7	28.7^a	27.4^a
0.6	29.5	28.1	27.3^a
0.7	29.1	27.8^a	28.7^a
0.8	28.1	27.4^a	27.6^a
0.9	26.8	26.6	25.8

[a] significantly better than the 5-update schedule
[b] significantly better than the 10-update schedule
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction
(for 3 pairwise comparisons), resulting in a significance level of $0.05/3 \approx 0.017$

Table 6: Medians of service completion times (as multiples of the planning horizon length) given by the dynamic CARP solver with the Random Insertion method and different update schedules

Degree of dynamism	Update schedule		
	5 updates	10 updates	20 updates
0.1	1.85	1.81^a	1.79^a
0.2	2.13	2.08^a	$1.97^{a,b}$
0.3	2.24	2.17^a	$2.11^{a,b}$
0.4	2.32	2.25^a	$2.23^{a,b}$
0.5	2.34	2.26^a	$2.19^{a,b}$
0.6	2.35	2.26^a	$2.21^{a,b}$
0.7	2.34	2.26^a	$2.21^{a,b}$
0.8	2.35	2.27^a	$2.22^{a,b}$
0.9	2.31	2.32^a	$2.22^{a,b}$

[a] significantly better than the 5-update schedule
[b] significantly better than the 10-update schedule
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction
(for 3 pairwise comparisons), resulting in a significance level of $0.05/3 \approx 0.017$

sequences of tasks to be serviced in the same route). Consequently, there is not as much need to collect a lot of information about new tasks before each update in order to obtain high quality solutions. In this case, the benefit of accumulating information about new tasks in each update appears to be less prominent than the benefit of adding new tasks to the solution promptly. This is particularly evident when comparing the service completion times for the Reconstruction method and the Random Insertion method; see Fig. 5(b) and Fig. 7(b).

So far the update schedules have been compared based on the same method of integrating new tasks. It is also interesting to compare 6 variants of the dynamic CARP solver (3 update schedules × 2 methods of integrating new tasks) all together; in particular, this would allow us to clearly see how the performance of the dynamic CARP solver is affected by the method of integrating new tasks. By comparing Fig. 5(a) and Fig. 7(a), we found that the Random Insertion generally gave better solutions than the Reconstruction method. In fact, using the Random Insertion with 20 updates is the most promising variant of the dynamic CARP solver; it is significantly better than the other variants in most cases (see Tables 7 and 8). This further highlights the benefits of retaining the solution from previous updates compared with solving the problem at each update from scratch.

7 Conclusion

This paper concerns a heuristic algorithm for solving the dynamic CARP. It begins by describing the main components of a dynamic CARP solver, including an update schedule, how the current state of the problem can be determined, a method of integrating new tasks into the solution, and a heuristic algorithm for improving the solution, which in this paper is based on tabu search. The purpose of this paper is to investigate how the performance of a heuristic algorithm for solving the dynamic CARP can be affected by adjusting its configuration. In particular, here we have considered adjusting 3 components of the solver: a maximum iteration limit for tabu search in each update, the frequency of solution updates, and a method of integrating new tasks to an existing solution. An analysis has been conducted to compare several options of these components and to investigate how each of the components could affect the solution quality with respect to total distance and service completion time.

Regarding the maximum iteration limit, experimental results show that increasing the maximum iteration limit for tabu search in each update yields little improvement. Moreover, a larger maximum iteration limit can sometimes even give worse results. This suggests that to consistently achieve a better solution in the dynamic CARP, it is not sufficient to rely solely on running the tabu search algorithm at each update for more iterations. This suggests the need to improve the dynamic CARP solver by other means.

Two ways of amending the dynamic CARP have been investigated: adjusting the frequency of solution updates and the way of integrating new tasks to the solution in each update. Here we consider 3 regular update schedules (with 5, 10, and 20 update) and 2 methods of integrating new tasks (the Reconstruction method and the Random Insertion method). Computational results show that the effect of adjusting the frequency of solution updates and the way of integrating new tasks to the solution in each update varies with the degree of dynamism, i.e. the ratio of the number of

dynamic tasks (known after vehicles leave the depot at the beginning of the planning horizon) to the number of all tasks in the whole the planning horizon.

More precisely, for relatively low degrees of dynamism (up to 0.4 for the instances considered here), a more frequent update schedule tends to give better results, regardless of the method of integrating new tasks. Nevertheless, the Random Insertion method yields more promising results than the Reconstruction method. In contrast, for relatively high degrees of dynamism (at least 0.5 for the instances considered here), the performance of different update schedules depends on the method of integrating new tasks. With the Reconstruction method, a less frequent update schedule tends to give better results. In contrast, with the Random Insertion method, a more frequent update schedule tends to give better results.

Among all variants of the dynamic CARP solver considered, the Random Insertion method with 20 updates give the best results; it is significantly better than the other variants in many cases (see Tables 7 and 8). This highlights the benefit of retaining solutions from previous updates as opposed to solving the problem at each update from scratch.

Appendix: Neighbourhood moves in the tabu search algorithm

Recall that a CARP solution is a set of routes, and a route can be represented by a sequence of tasks with specified directions (preceded by its starting vertex and followed by the depot v_0). Let $R_1 = (v_{s_1}, a_1, \ldots, a_{n_1}, v_0)$ and $R_2 = (v_{s_2}, b_1, \ldots, b_{n_2}, v_0)$ be two routes, for some tasks $a_1, \ldots, a_{n_1}, b_1, \ldots, b_{n_2}$, some vertices v_{s_1}, v_{s_2}, and some positive integers n_1, n_2. For ease of notation, a_{n_1+1} and b_{n_2+1} are identified with the depot v_0 at the end of the routes.

- Single Insertion: remove a task from one route and insert it in another route. Given a removal index i ($1 \leq i \leq n_1$) and an insertion index j ($1 \leq j \leq n_2 + 1$), remove a_i from R_1 and insert it in front of b_j in R_2.
- Double Insertion: remove two tasks from one route and insert them in another route. Given two removal indices i_1, i_2 ($1 \leq i_1 < i_2 \leq n + 1$) and two insertion indices j_1, j_2 ($1 \leq j_1, j_2 \leq n_2 + 1$), remove a_{i_1} and a_{i_2} from R_1 and insert them in R_2 in front of b_{j_1} and b_{j_2}, respectively. If $j_1 = j_2$, i.e. a_{i_1}, a_{i_2} are inserted at the same position, both possible orders (a_{i_1} followed by a_{i_2} or vice versa) are considered.
- Swap: swap tasks between two routes (one task from each route). Given two removal indices i, j ($1 \leq i \leq n_1, 1 \leq j \leq n_2$) and two insertion indices i', j' ($1 \leq i' \leq n_1 + 1, 1 \leq j' \leq n_2 + 1$) such that $i \neq i'$ and $j \neq j'$, remove a_i from R_1 and place it in front of $b_{j'}$ in R_2; remove b_j from R_2 and place it in front of $a_{i'}$ in R_1.

Table 7: Medians of percentage deviations from *a posteriori* lower bounds given by the dynamic CARP solver with different update schedules and different methods of integrating new tasks

Degree of dynamism	Reconstruction			Random Insertion		
	5 updates	10 updates	20 updates	5 updates	10 updates	20 updates
0.1	23.4	23.0^a	22.2^a	19.5^{abc}	17.8^{abcd}	17.6^{abcde}
0.2	26.3	25.6	25.8	23.0^{abc}	22.3^{abcd}	21.5^{abcde}
0.3	28.5	29.1	28.6	27.4^{ab}	26.0^{abcd}	24.9^{abcde}
0.4	29.1	28.6	29.5	28.6	27.8^{bc}	27.3^{abcde}
0.5	28.7	30.1	29.9	29.7	28.7^{bc}	27.4^{abcd}
0.6	28.1^c	29.1	29.8	29.5	28.1	27.3^c
0.7	28.2^{bcd}	29.6	29.0	29.1	27.8^d	28.7^{bcd}
0.8	28.1	27.4	29.3	28.1	27.4^{cd}	27.6^{cd}
0.9	24.9^c	26.1	28.0	26.8	26.6	25.8^c

[a] significantly better than the Reconstruction method with 5 updates
[b] significantly better than the Reconstruction method with 10 updates
[c] significantly better than the Reconstruction method with 20 updates
[d] significantly better than the Random Insertion method with 5 updates
[e] significantly better than the Random Insertion method with 10 updates
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction (for 15 pairwise comparisons), resulting in a significance level of $0.05/15 \approx 0.0033$

- 2-Opt: cut two routes each into two subroutes and re-connect subroutes to obtain two new routes. Given two cutting indices i, j ($1 \le i \le n_1 + 1, 1 \le j \le n_2 + 1$), cut R_1 into two parts $(v_s, a_1, \ldots, a_{i-1})$ and $(a_i, \ldots, a_{n_1}, v_0)^3$ and cut R_2 in a similar fashion. Then, join one part of R_1 with one part of R_2 (and join the remaining parts together). Note that there are two possible ways to join the parts, as illustrated in Fig. 8.

These four types of neighbourhood moves have been used in the literature (see, for example, Beullens et al. [1], Brandão and Eglese [2]), although some moves are performed in a slightly different way in this paper. For a double insertion move, removed tasks do not need to be consecutive (i.e. i_2 is not necessarily equal to $i_1 + 1$) and they do not need to be inserted at the same position (i.e. j_2 is not necessarily equal to j_1). For 2-opt, it is allowed that all tasks in one route (e.g. when $i = 1$) is moved to another route.

To avoid unnecessary computation, 2-opt moves that have essentially no effect to the solution are omitted (e.g. $i = j = 1$ with a particular way of joining routes simply results in renumbering routes). The 2-opt moves that resemble other neighbourhood moves are also omitted (e.g. $i = n_1$ and $j = n_2$ with a particular way of joining routes resembles a swap move).

References

1. P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643, 2003.

---

[3] If $i = 1$, then the route R_1 is divided into (v_s) and $(a_1, \ldots, v_0)$. If $i = n_1 + 1$, then the route is divided into $(v_s, \ldots, a_{n_1})$ and (v_0).
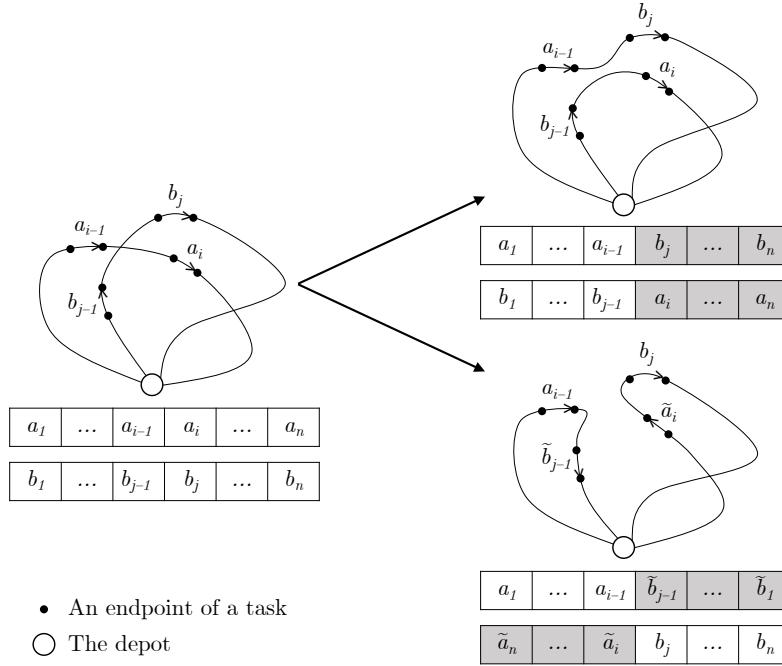
Fig. 8: Two possible ways of joining parts of routes as a result of a 2-opt move; tasks that are removed from their original routes are highlighted; $\tilde{a}$ denotes the opposite direction of traversal on task a

Table 8: Medians of service completion times (as multiples of the planning horizon length) the dynamic CARP solver with different update schedules and different methods of integrating new tasks

Degree of dynamism	Reconstruction			Random Insertion		
	5 updates	10 updates	20 updates	5 updates	10 updates	20 updates
0.1	1.89	1.87^a	1.80^{ad}	1.85	1.81^{ad}	1.79^{abd}
0.2	2.17	2.11^{ad}	2.09^{ad}	2.13	2.08^{ad}	1.97^{abcde}
0.3	2.29	2.24^a	2.22^a	2.24^a	2.17^{abcd}	2.11^{abcde}
0.4	2.38	2.27^a	2.35^a	2.32^a	2.25^{abcd}	2.23^{abcd}
0.5	2.39	2.37	2.36	2.34	2.26^{abcd}	2.19^{abcd}
0.6	2.37	2.35	2.37	2.35	2.26^{abcd}	2.21^{abcd}
0.7	2.35	2.39	2.35	2.34	2.26^{abcd}	2.21^{abcde}
0.8	2.41	2.38	2.36^a	2.35^a	2.27^{abcd}	2.22^{abcde}
0.9	2.40	2.38	2.38	2.31	2.32^{abcd}	2.22^{abcde}

[a] significantly better than the Reconstruction method with 5 updates
[b] significantly better than the Reconstruction method with 10 updates
[c] significantly better than the Reconstruction method with 20 updates
[d] significantly better than the Random Insertion method with 5 updates
[e] significantly better than the Random Insertion method with 10 updates
based on a two-tailed Wilcoxon signed-rank test with a Bonferroni correction (for 15 pairwise comparisons), resulting in a significance level of $0.05/15 \approx 0.0033$

2. J. Brandão and R. Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35 (4):1112–1126, 2008.
3. Z.-L. Chen and H. Xu. Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science*, 40(1):74–88, 2006.
4. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
5. F. Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.
6. B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.
7. B. L. Golden, J. S. DeArmon, and E. K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.
8. M. Liu, H. K. Singh, and T. Ray. A benchmark generator for dynamic capacitated arc routing problems. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 579–586. IEEE, 2014.
9. M. Liu, H. K. Singh, and T. Ray. A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 595–602. IEEE, 2014.
10. K. Lund, O. B. Madsen, and J. M. Rygaard. Vehicle routing with varying degree of dynamism. 1996.
11. M. Monroy-Licht, C. A. Amaya, A. Langevin, and L.-M. Rousseau. The rescheduling arc routing problem. *International Transactions in Operational Research*, 2016.
12. R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.
13. L. M. Moreira, J. F. Oliveira, A. M. Gomes, and J. S. Ferreira. Heuristics for a dynamic rural postman problem. *Computers & Operations Research*, 34(11):3281–3294, 2007.
14. H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.
15. M. Tagmouti, M. Gendreau, and J.-Y. Potvin. A dynamic capacitated arc routing problem with time-dependent service costs. *Transportation Research Part C: Emerging Technologies*, 19(1):20–28, 2011.
16. A. Yazici, G. Kirlik, O. Parlaktuna, and A. Sipahioglu. A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints. *IEEE Transactions on Cybernetics*, 44(3):305–314, 2014.