Evaluating Meta-heuristic Algorithms for Dynamic Capacitated Arc Routing Problems Based on a Novel Lower Bound Method

Hao Tong, Leandro L. Minku, University of Birmingham, UK.
Stefan Menzel, Bernhard Sendhoff, Honda Research Institute Europe GmbH, Germany.
Xin Yao, School of Data Science, Lingnan University, Hong Kong, China, and School of Computer Science, University of Birmingham, Birmingham, UK.

***Abstract*—Meta-heuristic algorithms, especially evolutionary algorithms, have been frequently used to find near optimal solutions to combinatorial optimization problems. The evaluation of such algorithms is often conducted through comparisons with other algorithms on a set of benchmark problems. However, even if one algorithm is the best among all those compared, it still has difficulties in determining the true quality of the solutions found because the true optima are unknown, especially in dynamic environments. It would be desirable to evaluate algorithms not only relatively through comparisons with others, but also in absolute terms by estimating their quality compared to the true global optima. Unfortunately, true global optima are normally unknown or hard to find since the problems being addressed are NP-hard. In this paper, instead of using true global optima, lower bounds are derived to carry out an objective evaluation of the solution quality. In particular, the first approach capable of deriving a lower bound for dynamic capacitated arc routing problem (DCARP) instances is proposed, and two optimization algorithms for DCARP are evaluated based on such a lower bound approach. An effective graph pruning strategy is introduced to reduce the time complexity of our proposed approach. Our experiments demonstrate that our approach provides tight lower bounds for small DCARP instances. Two optimization algorithms are evaluated on a set of DCARP instances through the derived lower bounds in our experimental studies, and the results reveal that the algorithms still have room for improvement for large complex instances.**

***Index Terms*—Performance evaluation, Meta-heuristic algorithms, Lower bounds, Dynamic capacitated arc routing problem.**

I. Introduction

Combinatorial optimization problems (COPs) are ubiquitous across various industrial sectors [1]. They are hard to solve due to their large search spaces and NP-hard nature. Meta-heuristic algorithms, such as evolutionary algorithms and memetic algorithms, are one of the most popular approaches for tackling various COPs [2], especially in dynamic environments [3], [4]. They are effective at finding near optimal solutions to these problems by balancing the exploration and exploitation through powerful search operators.

Performance measurement for a meta-heuristic algorithm is crucially important in understanding its optimization capabilities [5]. Currently, in the literature, newly proposed algorithms are commonly evaluated using a suite of benchmark problems, and the majority of studies focus on comparing optimization results with a baseline algorithm using statistical tests in terms of either the solution quality or the computational time required [6]. However, such comparative evaluations only provide insights into the relative performance of algorithms compared to others, without providing an estimation of the solution quality with respect to the global optimum. The performance measurement in relation to the global optimum is crucial for assessing the remaining potential for improvement in the optimization algorithms.

In continuous optimization, the global optimum of the artificially designed benchmark problems is typically known, allowing for a straightforward assessment of the gap between the obtained solution and the global optimum [7]. Unfortunately, the theoretical optimum is usually either unknown or exceedingly hard to find for most NP-hard COPs. Utilizing lower bounds as an alternative to the theoretical optimum offers a viable approach for measuring the performance of meta-heuristic algorithm [8], and also for a better understanding of the problem and its solution method when the theoretical optimum is unknown in COPs.

The use of lower bounds for evaluating the performance of heuristic algorithms is a well-established practice in the literature, and several studies have employed this performance metric to evaluate algorithms. For example, Chao et al. [9] defined the *approximate error rates* to assess solutions in bin-packing problem. Similarly, Jarboui et al. [10] defined a "$\%gap$" metric representing the average deviation of solution value from the lower bound to measuring the algorithm for the location routing problem. Lower bounds are particularly useful for dynamic optimization problems, as changes in the environment may make it even more challenging to know the exact global optimum. In this context, as problem instances change, the lower bound of the problem also shifts over time at each time period [11]. Consequently, the lower bound is required to be derived for the problem instance in each environment to assess the performance of algorithms in the new environment.

The Capacitated Arc Routing Problem (CARP) is a typical challenging NP-hard [12] combinatorial optimization problem, abstracted from many real-world problems [13], such as salt gritting [14] and waste collection [15]. The static CARP aims to assign a fleet of vehicles with limited capacities

Corresponding Author: Xin Yao (xinyao@LN.edu.hk) to serve a set of tasks in a graph [16]. All vehicles must start and end their services at the depot. However, some dynamic events, such as road congestion and newly added tasks/demands [17], may occur during the vehicles' services, leading to a degradation in the original schedule or even rendering it infeasible. Dynamic CARP (DCARP) instances are generated in response to these dynamic events [18], [19], [20], and the optimization for these DCARP instances is highly challenging. A few effective algorithms for solving DCARP have been proposed in the literature [18], [20], [21], [22]. To the best of our knowledge, although performance evaluation based on the lower bound has been widely used in the literature, only a percentage deviation based on a *posterior* lower bound has been proposed to estimate the performance of optimization algorithms [22]. However, this *posterior* lower bound corresponds to the static CARP instance instead of the lower bound of each actual DCARP instance in a dynamic environment.

Therefore, in this paper, a tight lower bound for DCARP instances is derived. Then, the performance of meta-heuristic algorithms for DCARP is evaluated through assessing the quality of DCARP solutions based on the derived lower bound. Our contributions are as follows:

- The first lower bound approach for DCARP, named node matching lower bounds (NMLB), was proposed, which considers the outside vehicles with different remaining capacities and locating at different vertices. The lower bound problem is converted into a minimum cost matching problem (MCMP). A lower bound for a DCARP instance is obtained by applying the Blossom algorithm [23] to solve the MCMP. The tightness of lower bound is analyzed on a set of DCARP instances whose theoretical optima are known and the analysis concludes that the obtained NMLB is very close to the theoretical optimum on the testing DCARP instances.
- A graph pruning strategy was proposed to remove a subset of redundant vertices in the MCMP, which increases the computational efficiency of the proposed NMLB approach. Its efficiency is demonstrated and analyzed in our experimental studies.
- The derived lower bound is applied to evaluate the performance of two meta-heuristic algorithms for DCARP on a benchmark set of DCARP instances. The experimental analysis reveals excellent performance on relatively small instances and potential improvements on large complex instances, which demonstrates how our lower bound approach could enhance our understanding of meta-heuristic algorithms in solving hard problems.

The remainder of this paper is organized as follows. Section II introduces the definition of the DCARP, the notations used in this paper, and the lower bound approaches for static CARP instances. Section III introduces our proposed node matching lower bound approach and the graph pruning strategy. Section IV presents the experimental results on the tightness of the lower bound, the performance measurement of optimization algorithm, and the efficacy of the pruning strategy. Conclusions and future work are provided in Section V.

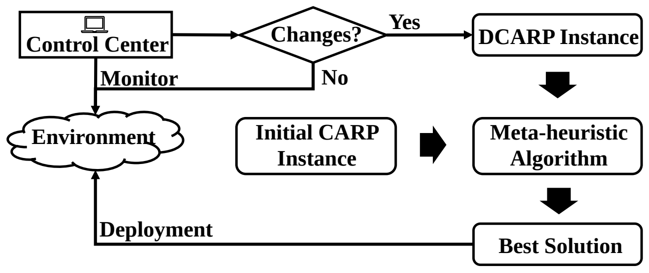


Figure 1: The illustration of a DCARP scenario.

II. Background

A. Characteristics of Dynamic Behavior in Studied DCARP

When dynamic events occur and deteriorate the current deployed solution during vehicles' services, it becomes necessary to re-schedule the vehicles for the new environment. This can be seen as a new DCARP instance in the new environment, and optimization for the new DCARP instance is required to update vehicle routing. Therefore, a complete DCARP scenario consists of a series of discrete DCARP instances, each of which corresponds to an environmental change, and optimizing these DCARP instances is necessary to update vehicle routing.

The process of a DCARP scenario studied in this paper is presented in Figure 1, starting with an initial static CARP instance. A meta-heuristic algorithm can be employed to obtain the best solution for the static CARP instance, which will be used to schedule vehicles to serve the pre-defined tasks. There is a control center which can receive all traffic and environment information, and it will detect changes in the environment. Whenever any dynamic events occur while vehicles are in service, the control center will report the type of dynamic events and generate a new DCARP instance according to the occurred dynamic events. The optimization algorithm will then dynamically optimize the newly generated DCARP instance. After the dynamic optimization process for the new DCARP instance, an updated solution will be obtained and deployed to the environment to update the service plan of all vehicles. The DCARP scenario will terminate only when all tasks have been served. This paper focuses on deriving the lower bound for each DCARP instance within a DCARP scenario.

In the literature, Uncertain CARP (UCARP) also focused on addressing environmental uncertainties, but they primarily focused on finding robust solutions prior to deployment that are expected to perform well under various dynamic events [17]. However, in some dynamic optimization problems, finding solutions that are robust over time may not be sufficient. In cases where dynamic changes have led to significant solution deterioration, a "robust" solution with poor quality may not be useful for many applications. In such cases, our studied DCARP optimization is needed to ensure that the vehicle routing remains efficient throughout the service period.

The DCARP studied in this work is also different from typical dynamic optimization problems in the literature, where uncertain factors only affect objective functions, causing a change of global optima. In our DCARP, the decision variable space of the new DCARP instance changes after dynamic events. In particular, the solution is first partially executed, meaning that some tasks in the original problem may have already been served, and this service is not required to be repeated. Moreover, newly emerging tasks and potential road closures are likely to render previous solutions infeasible. As a result, the decision variable space for the new DCARP instance differs from the previous DCARP instances, causing solutions obtained for the previous DCARP instances unsuitable for the new DCARP instances.

B. Problem Definition and Notations[1]

In this paper, an un-directed graph $G = (V, E)$ for a CARP instance was considered, where the vertex (node) v_0 is the depot. Each edge $e \in E$ has a demand of $dm(e)$ and a traverse cost. Edges with $dm(e) > 0$ are tasks with a serving cost, which are required to be served by vehicles, and all tasks form a set $E_R = \{e \in E | dm(e) > 0\}$. Each vehicle initially starts from the depot v_0 with a full capacity Q. During the *deployment* of a solution, unforeseen dynamic events [20] may occur at a random point in time t and generate a new DCARP instance required to be optimized. Specifically, various types of dynamic changes may occur, generating the DCARP instance. However, most common dynamic events only affect the values of the problem instance (i.e., arcs' costs, tasks' demands, and the number of tasks) without altering the objective or constraint formulas. For example, cost-change events only modify the arcs' costs, adding demands events only adjusts the demands of existing tasks, and new task events only impact the total number of tasks in the problem instance. Moreover, the degree of dynamic events also primarily affects the magnitude of the values in the problem instance, rather than the objective or constraint formulas of the DCARP. Therefore, in this paper, such general DCARP instances are considered, in which the dynamic events and their degree solely affect values of the problem instance.

Suppose the DCARP instance generated at a time point t is denoted by I_t and assume that there are N_{ov} outside vehicles in the DCARP instance I_t. These vehicles were either serving tasks or had served tasks when the dynamic event occurred. Therefore, they are located at different vertices $V_{ov} = \{sp_1, sp_2, ..., sp_{N_{ov}}\}$ (nodes of outside vehicles) and have different remaining capacities $\{q_1, q_2, ..., q_{N_{ov}}\}$ [20]. The optimization problem for the DCARP instance I_t targets an optimal assignment of tasks for different vehicles, including outside vehicles and those starting from the depot, with the lowest total costs, which is the sum of total service costs and traversing costs in a solution.

A solution to the DCARP instance I_t, is denoted as $S_t = \{R_1, R_2, ..., R_K\}$, where K is the number of routes in the solution. Among these routes, N_{ov} routes correspond to outside vehicles, while the remaining $K - N_{ov}$ routes start from the depot. A route R_k traversed by a given vehicle k can be represented as $R_k = (v_k, t_{k,1}, t_{k,2}, ..., t_{k,l_k}, v_0)$, where l_k denotes the number of tasks in the k^{th} route, v_k represents the starting point of the route R_k, and $t_{k,i}$ is the i^{th} task in route R_k. Assuming that the demand of a task $t_{k,i}$ is $dm(t_{k,i})$, the objective function for the DCARP instance I_t can be formulated by the following equation [20], [24]:

$$\textbf{Min} \quad TC(S_t) = \sum_{k=1}^{K} RC_{R_k} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{k=1}^{K} l_k = N_t \tag{2a}$$

$$t_{k_1,i_1} \neq t_{k_2,i_2}, \text{for all } (k_1, i_1) \neq (k_2, i_2) \tag{2b}$$

$$\sum_{i=1}^{l_k} dm(t_{k,i}) \leq q_k, \forall k \in \{1, 2, ...N_{ov}\} \tag{2c}$$

$$\sum_{i=1}^{l_k} dm(t_{k,i}) \leq Q, \forall k \in \{N_{ov} + 1, ..., K\} \tag{2d}$$

where RC_{R_k} is the total cost of route R_k which is calculated by Eq. (3):

$$\begin{aligned} RC_{R_k} = mc(v_k, head_{t_{k,1}}) + mc(tail_{t_{k,l_k}}, v_0) + \\ \sum_{i=1}^{l_k - 1} mc(head_{t_{k,i}}, tail_{t_{k,i+1}}) + \sum_{i=1}^{l_k} sc(t_{k,i}) \end{aligned} \tag{3}$$

where the $head_{t_{k,i}}$, $tail_{t_{k,i}}$ denote the task's head and tail vertices. The direction of an arc is assumed to be from its tail to its head. The minimal total traversing cost from node v_i to node v_j is denoted as $mc(v_i, v_j)$, and $sc(t_{k,i})$ denotes the serving cost of the task $t_{k,i}$. The constraint (2a) guarantees that the number of tasks in the solution equals to the number of tasks of the instance. The constraints (2b) guarantee that any two tasks in the solution are different, i.e., the solution does not attempt to serve the same task more than once. Therefore, these two constraints guarantee that all tasks are served exactly once. Then, constraints (2c) and (2d) are formulated to satisfy the vehicles' capacity constraint.

For simplicity, the depot and all tasks are extracted in the graph and a *required graph* $G_R = (V_R, E_R)$ is constructed, which only includes the required edges E_R for deriving the lower bound. V_R is a set of nodes, including the depot and nodes incident with at least one task in V. Let $d(v_i)$ be the degree of node v_i in G_R, which is equal to the number of tasks that node v_i is incident with in G_R. The cost between two nodes v_i and v_j in G_R is set as equal to its cost $mc(v_i, v_j)$ in the original graph G. As the (D)CARP solution is only related to tasks, all deductions made for the lower bound are based on G_R. C_T and W_T are the total serving costs and the total demands for all tasks in a (D)CARP instance, respectively.

Suppose $S^* = \{R_1^*, R_2^*, ..., R_K^*\}$ is the optimal solution for a DCARP instance, in which route R_k^* is denoted as $(v_k, t_{k,1}^*, t_{k,2}^*, ..., t_{k,l_k}^*, v_0)$. As formulated in Eq. (1) and Eq. (3), the total costs of the optimal solution are the summation of all tasks' service costs and the total deadheading costs traversed between any two nodes in R_k^*. Thus, the traversed path between two separate nodes which do not belong to the same task in a solution are considered as the *artificial link* in

[1]A full list of mathematical notations used in this paper are summarized in the Appendix.

G_R. Since the service costs of all tasks are fixed for a given DCARP instance, the sum of the total costs of these *artificial links* determines the lower bound of a (D)CARP instance, which will be shown how to compute in Section III.

C. Lower Bounds for Static CARP and DCARP

During the past decades, several lower-bound methods have been introduced for static CARP instances. Two fundamental approaches, including the matching lower bound (MLB) [12] and the nodes scanning lower bound (NSLB) [25], have stood out. MLB deduces the lower bound by adding artificial edges to transform the graph into an Euler graph [12]. NSLB considers that vehicles traverse a path from the depot to a given node before serving any tasks and that the number of paths linking to a node would not exceed the number of connected tasks. Thus, NSLB adds artificial edges between the depot and nodes to obtain a lower bound [25]. Further approaches have been proposed based on these two basic methods. For example, [26] proposed two lower bounds (LB1 and LB2) based on NSLB, and [27] proposed the node duplication lower bound (NDLB) method based on MLB. [28] combined MLB and NSLB and proposed a tighter lower bound, called MCNDLB. Besides, [29] used cut-and-column generation to get a new improved lower bound for static CARP instances.

Although the current lower bounds for static CARP instances have been reasonably tight, these existing lower bound approaches are unsuitable for deriving the lower bound for DCARP instances, as they assume all vehicles start at the depot. This is because the lower bound approaches for static CARP instances only consider routes starting from the depot. However, since DCARP instances are generated due to the dynamic events while vehicles are in their services, the DCARP solution includes routes belonging to outside vehicles which start from different locations instead of the depot. These outside vehicles can continue serving some tasks and then return to the depot, or they can also return to the depot directly. These special cases have to be taken into account when deriving the lower bound, but the static lower bound approach is not capable of handling them.

In [20], a “virtual-task” strategy was proposed for converting a DCARP instance into a “virtual” static CARP instance, enabling optimization algorithms designed for static CARP to be applied to DCARP. The virtual-task strategy creates a virtual task between the depot and the location where each outside vehicle stops. Each virtual task is assigned the same demand as that already served by a given outside vehicle. Serving a virtual task once using a depot vehicle results in the location and remaining capacity of this vehicle being the same as that of the corresponding outside vehicle when the change happened. Therefore, during optimization, each outside vehicle can be treated as if it was a vehicle located at the depot, but serving its corresponding virtual task once. However, even though all vehicles are located at the depot with the same capacity in the “virtual” static CARP instance, it is still not suitable to apply the existing lower bound approaches for static CARP instances to such “virtual” static CARP instances. This is because the virtual task is constructed for optimization with static CARP optimization algorithms, and these virtual tasks have to be the first task in a route. Furthermore, two virtual tasks cannot exist in the same route in the final solution. Therefore, the outside vehicles are still required to be considered specifically when deriving the lower bound using the converted “virtual” static CARP instances with static lower bound approaches.

Therefore, to fill the gap of deriving the lower bound for DCARP instances, a node matching lower bound approach is proposed, which was motivated by the NSLB [25] and NDLB [27] approaches for static CARP instances. It is capable of dealing with the outside vehicles by transforming the origin lower bound problem into a matching problem. More details of the proposed approaches are presented in Section III.

III. Performance evaluation with node matching lower bound for DCARP

A. The Lower Bound Algorithm

For a given DCARP instance, a solution S is assumed to be composed of several routes $\{R_1, R_2, ..., R_K\}$. In each route $R_k : (v_k, t_{k,1}, t_{k,2}, ..., t_{k,l_k}, v_0)$, v_k is the starting point of the route, and $t_{k,i}$ is the i^{th} task of the route. Suppose $head_{t_{k,i}}$, $tail_{t_{k,i}}$ denote the head and tail vertices of task $t_{k,i}$, the route R_k can be represented by a node sequence as follows:

$$R_k = (v_k, tail_{t_{k,1}}, head_{t_{k,1}}, tail_{t_{k,2}}, head_{t_{k,2}}, ..., head_{t_{k,l_k}}, v_0)$$

Therefore, a DCARP solution can be formed by adding the following three types of artificial links:

1) Artificial links between the starting vertex of the route and the head vertex of the first task.
2) Artificial links between the tail vertex of the first task and the returning vertex of the route (the depot).
3) Artificial links between the tail vertex of one task and the head vertex of its next task.

The total costs of a solution are determined by the costs of these artificial links according to Eq. (1) and Eq. (3), because the total service costs of all tasks are fixed. Thus, if the case of adding artificial links between tasks and starting and returning vertices with minimum total costs can be identified, a valid lower bound for the DCARP instance can be obtained.

At first, the minimum number of routes required for the given DCARP instance must be determined to calculate the minimum number of starting and returning vertices required in a DCARP solution. Due to the outside vehicles in the DCARP instance, the solution of the DCARP instance consists of routes starting not only from the depot but also from these outside vehicles. These outside vehicles can also continue to serve some tasks, thereby affecting the minimum number of vehicles required to start from the depot, which can be calculated as:

$$k_0 = \left\lceil \frac{W_T - \sum_{i=1}^{N_{ov}} q_i}{Q} \right\rceil \tag{4}$$

The number of routes belonging to the outside vehicles equals the number of outside vehicles, i.e., N_{ov}.

In a DCARP solution, the starting vertices of routes can be either the depot or nodes associated with outside vehicles (outside vehicles' nodes). The returning vertices are always the depot. Thus, for adding artificial links related to the starting and returning vertices, five constraints have to be satisfied in a DCARP solution:

1) The depot in k_0 routes associated with vehicles starting from the depot must link to a task's node $v_i \in V_R$.
2) The depot in N_{ov} routes associated with outside vehicles can link to a task's node or directly link to a starting vertex $sp_i \in V_{ov}$.
3) The starting vertex $sp_i \in V_{ov}$ in N_{ov} outside vehicles' routes can link to a task's node or directly link to a depot.
4) If a starting vertex $sp_i \in V_{ov}$ in N_{ov} outside vehicles' routes links to a task's node, the returning vertices (depot) in these routes must also need to link to a task's node.
5) If a starting vertex $sp_i \in V_{ov}$ in N_{ov} outside vehicles' routes link to the depot, the returning vertex (depot) in these routes cannot link to other tasks' nodes anymore.

Algorithm 1: Pseudo-code of constructing the auxiliary graph

```
   Input: Graph $G_R = (V_R, E_R)$
1  Initialise four empty sets $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$.
2  Add $2k_0$ nodes into $\mathbf{A}$ representing $2k_0$ copies of the
       depot $v_0$. // Depots needed for new
       vehicles to start at, and return
       to.
3  Add $N_{ov}$ nodes into $\mathbf{C}$ representing $N_{ov}$ copies of the
       depot $v_0$. // Depots needed for outside
       vehicles to return to.
4  Add all nodes in $V_{ov}$ into $\mathbf{D}$, i.e., $\mathbf{D} = V_{ov}$.
       // Nodes where outside vehicles
       are located.
5  for each node $v_i \in V_R$ do
6      Add $d(v_i)$ copies of node $v_i$ into $\mathbf{B}$. // Nodes
         which vehicles can pass through
         when serving tasks
7  Set $V_x = \mathbf{A} \bigcup \mathbf{B} \bigcup \mathbf{C} \bigcup \mathbf{D}$, $E_x = \emptyset$.
8  Add links between $\mathbf{A}$ and $\mathbf{B}$,
     $E_x = E_x \bigcup \{(v_i, v_j) | v_i \in \mathbf{A}, v_j \in \mathbf{B}\}$.
9  Add links between $\mathbf{B}$ and $\mathbf{C}$,
     $E_x = E_x \bigcup \{(v_i, v_j) | v_i \in \mathbf{B}, v_j \in \mathbf{C}\}$.
10 Add links between $\mathbf{B}$ and $\mathbf{D}$,
     $E_x = E_x \bigcup \{(v_i, v_j) | v_i \in \mathbf{B}, v_j \in \mathbf{D}\}$.
11 Add links between $\mathbf{C}$ and $\mathbf{D}$,
     $E_x = E_x \bigcup \{(v_i, v_j) | v_i \in \mathbf{C}, v_j \in \mathbf{D}\}$.
12 Add links inside set $\mathbf{B}$,
     $E_x = E_x \bigcup \{(v_i, v_j) | v_i \in \mathbf{B}, v_j \in \mathbf{B}, v_i \neq v_j\}$.
13 for each link $(v_i, v_j) \in E_x$ do
14     Set its cost equal to the cost between the two
         corresponding nodes in $G_R$.
   Output: Auxiliary graph $G_x = (V_x, E_x)$
```

Therefore, an auxiliary graph G_x can be constructed based on the above constraints, and the pseudo-code for constructing the auxiliary graph is presented in Algorithm 1. Four different empty subsets, named as **A**, **B**, **C**, and **D**, are initialized to include nodes with different roles in a DCARP solution, as presented in Lines 1-6. Set **A** includes $2k_0$ nodes representing the starting and returning vertices of k_0 routes starting from the depot, which are all depots (Line 2). Set **C** includes N_{ov} nodes representing the returning vertices of N_{ov} routes associated with the outside vehicles (Line 3). Set **D** includes the starting vertices of N_{ov} routes belonging to the outside vehicles, i.e., the locations where outside vehicles stop (Line 4). Set **B** mainly consists of the vertices incident with the tasks in G_R. Since each task can only be served once in a DCARP solution, any vertex v_i cannot be linked over $d(v_i)$ times (i.e., the degree in G_R) when adding artificial links to form the DCARP solution. Thus, $d(v_i)$ copies of each vertex $v_i \in V_R$ were added into set **B** (Lines 5-6). These four sets are combined to form the set V_x (Line 7). Then, according to the above five constraints, links between sets **A** and **B**, **B** and **C**, **B** and **D**, and **C** and **D**, were added as presented in Lines 8-11. Links between nodes in set **B**, representing the links between tasks in a solution were also added (Line 12). All these links form a set E_x. For each link in E_x, its cost equals to the corresponding minimum costs between two vertices in G_R (Lines 13-14). Finally, the auxiliary graph $G_x = (V_x, E_x)$ is constructed and used to calculate the instance's lower bound.

Theorem 1. *Let C_{MCMP} be the cost of the optimal solution of the minimum cost matching problem on G_x, and C_T be the total serving costs of all tasks. Then, $NMLB = C_T + C_{MCMP}$ provides a valid lower bound for a DCARP instance.*

Proof. The optimal solution of the DCARP instance comprises tasks, artificial links between tasks' head and tail vertices, and artificial links between tasks' endpoints and the starting and returning vertices of the routes. The total service costs C_T are fixed for a given DCARP instance. The case of all artificial links in the optimal solution must be a matching of the auxiliary graph G_x: each vertex in G_x must be linked with exactly one other vertex, and each vertex can only be linked once in G_x. Let C^*_{links} denote the total costs of all artificial links in the optimal solution of the DCARP instance. Then, C^*_{links} must be greater than or equal to the cost of the solution to the minimum cost matching problem on G_x, which is denoted as C_{MCMP}. Thus, the following inequality must be satisfied:

$$C^* = C_T + C^*_{links} \geq C_T + C_{MCMP} = NMLB \quad (5)$$

which completes the proof. □

B. An Example of NMLB Approach

Example 1. *An example of a DCARP instance is shown in Figure 2, in which the node '0' denotes the depot, and the cost and demand of each edge are provided in brackets. In the example, the traversing cost is equal to the serving cost for all tasks. The vehicle capacity Q is 4. Two outside vehicles, i.e., ov_1 and ov_2, stop at nodes '2' and '3', respectively, with remaining capacities $q_1 = q_2 = 2$.*

The optimal solution this instance has the following routes: $\{1 : v_2 \Rightarrow v_1 - v_0, 2 : v_3 \Rightarrow v_1 - v_0, 3 : v_0 \Rightarrow v_1 - v_4 \Rightarrow v_2 - v_1 - v_0, 4 : v_0 - v_1 \Rightarrow v_4 \Rightarrow v_3 - v_1 - v_0\}$, where "$\Rightarrow$" denotes that the edge between two nodes is a task, and the

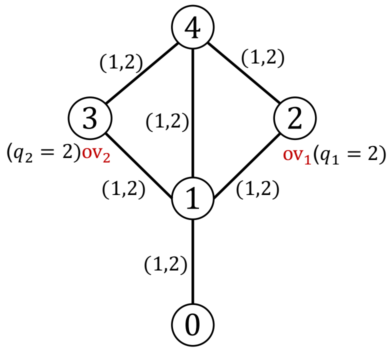
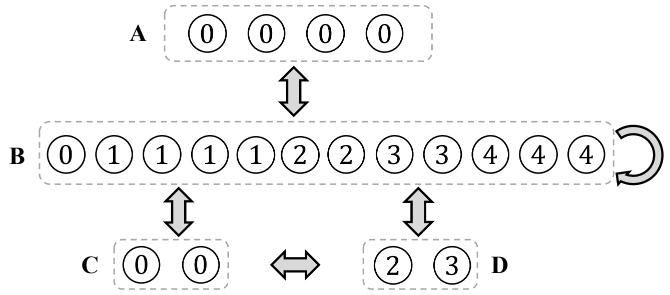
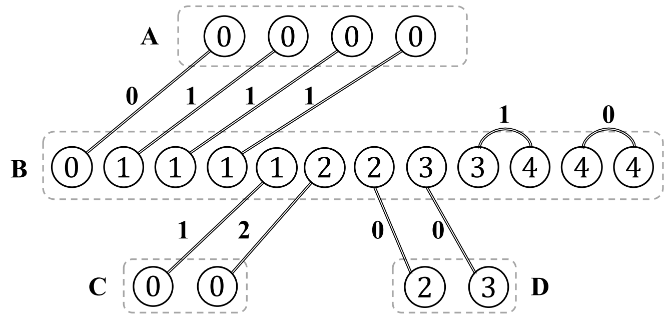


Figure 2: An example of a DCARP instance. Node ‘0’ represents the depot. Values in brackets denote the cost and demand of the corresponding edge.



(a) Graph G_x for example 1 as shown in Figure 2. It has links between all pairs of nodes in sets **A** and **B**, sets **B** and **C**, sets **B** and **D**, sets **C** and **D**, and between nodes in set **B**, as shown with big grey arrows.



(b) The solution of MCMP for the graph G_x of example 1.

Figure 3: Example of applying NMLB approach in a DCARP instance.

“−” denotes the traversed edge between two nodes. Thus, the optimal solution’s cost for this example is 14*.*

The total demand for all tasks is $W_T = 12$*. Thus, the minimum number of new vehicles dispatched from the depot is* $k_0 = 2$*. Therefore, the auxiliary graph* G_x *for the given instance can be constructed as shown in Figure 3a including four different sets. Set **A** includes* $2k_0 = 4$ *copies of the depot. Set **B** includes all nodes in* G_R*, and each node is duplicated by its degree copies. Set **C** includes* $N_{ov} = 2$ *copies of the depot. Set **D** includes the nodes where outside vehicles stop. The big grey arrows indicate the links between all pairs of nodes in the two sets.*

The solution to the MCMP on G_x *is shown in Figure 3b with solid lines. Thus, the total costs for the solution of MCMP, i.e.,* C_{MCMP}*, is equal to* 7*. Therefore, the lower bound* $NALB$ *for this example is* 13*, since the total service costs are* $C_T = 6$*.*

C. Graph Pruning Strategy

Even though the above approach can obtain a lower bound for the DCARP instance, the process of solving the minimum cost matching problem on G_x is computationally expensive when the number of tasks in the instance is large. This is because of the large number of nodes in the set **B** of G_x for instances with many tasks and the time complexity of “Blossom algorithm[2]” [23]. The Blossom algorithm focused on dealing with odd-length cycles in the graph to progressively improve the matching until it achieves optimality. While it is effective in finding the optimal solution for the MCMP, its time complexity of $O(|V_x|^3)$ makes solving our converted MCMP computationally expensive. However, since the degree of many nodes in G_R is greater than 1, resulting in multiple copies of these nodes being in set **B**, many nodes in set **B** probably match their own copies in the final solution of MCMP. Thus, it is possible to exclude nodes which definitely will match their copies in the final solution of MCMP to reduce the number of nodes in the set **B**. Therefore, a pruning strategy is proposed in this subsection to mitigate the computational burden of solving the MCMP.

To remove nodes which will definitely match their copies in set **B** of G_x in the final MCMP solution, it is necessary to identify those nodes in set **B**, which are possible to match nodes in sets **A**, **C**, and **D** first. Then, the remaining nodes in set **B** can be pruned to remove duplicates and reduce the size of set **B**. The pruning algorithm for set **B** is presented in Algorithm 2 for obtaining a new set **B’** before solving the MCMP.

First, to identify which nodes in set **B** could potentially match nodes in sets **A**, **C**, and **D**, we consider the $2k_0 + N_{ov}$ depots in sets **A** and **C** as well as the nodes of the outside vehicles in set **D** are required to be considered. Therefore, in Lines 2-3 of Algorithm 2, all nodes in **B** are sorted according to their minimum costs from the depot and add the nearest $2(k_0+N_{ov})$ nodes to **B’** because the outside vehicles’ nodes at most match to N_{ov} nodes from the nearest $2(k_0 + N_{ov})$ nodes in **B**, and the remaining $2k_0+N_{ov}$ nearest nodes are enough for matching the $2k_0 + N_{ov}$ depots in **A** and **C**. Then, for each outside vehicle node, the whole set **B** are sorted according to their costs to the outside vehicle node (Line 5). For each node in the nearest N_{ov} nodes of the sorted **B**, if it has not been included in set **B’**, then add it into the set **B’**, which are presented in Line 6-8 of Algorithm 2. This is because the remaining $N_{ov}-1$ outside vehicles might match the first $N_{ov}-1$ nearest nodes, and each outside vehicle node at least will match its N_{ov}^{th} nearest node. Thus, the first N_{ov} nearest nodes were added to set **B’**, which could guarantee the remaining nodes will not match the outside vehicles’ nodes in set **D**.

Then, a set **X** consisting of nodes is defined that definitely match nodes in set **B** in the solution of MCMP. It is obtained

[2]The python library (Networkx) was used to solve the MCMP.

Algorithm 2: Pseudo-code of pruning the set **B**.

```
   Input: Graph $G_R$, set B
1  Set B' = $\emptyset$.
2  Sort nodes in B according to their minimum costs
     from the depot.
3  Select the first $2(k_0 + N_{ov})$ nodes in B and add them
     to B'.
4  for each $sp_i \in V_{ov}$ do
5      Set $\mathbf{B}_{sort}$ by sorting nodes in B according to their
         minimum costs to $sp_i$.
6      for each $v_i \in \mathbf{B}_{sort}[1 : N_{ov}]$ do
7          if $v_i \notin$ B' then
8              Add $v_i$ to B'
9  Set $X = \mathbf{B} \setminus$ B'.
10 for each $v_i \in X$ do
11     if $v_i$ has even number of copies then
12         Remove all $v_i$ in $X$.
13     if $v_i$ has odd number of copies then
14         Just keep one $v_i$ in $X$.
15 B' = B' $\cup X$.
   Output: B'
```

by removing set **B'** from the set **B**, as presented in Line 9. Each node will first match the node, which is one copy of this node, in the solution of MCMP. This is proved in Theorem 2. Thus, for each node in set **X**, if it has an even number of copies, all copies of this node are removed, while if it has an odd number of copies, only one copy of this node is kept in X, as shown in Lines 10-14.

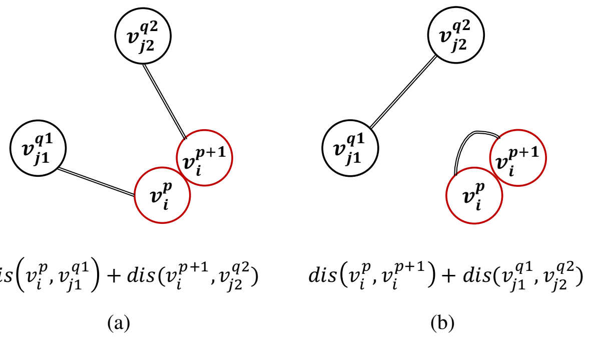
Finally, the sets **X** and **B'** are combined to be the new set **B** for constructing the auxiliary graph G_x and solving the MCMP to obtain the lower bound.

Theorem 2. *Let $X = \{v_1^1, v_1^2, ..., v_1^{d_1}, ..., v_n^1, v_n^2, ..., v_n^{d_n}\}$ be a node set, in which the distance between each two nodes in the subset $\{v_i^1, v_i^2, ..., v_i^{d_i}\}$ is 0. Then, in the solution of MCMP on the complete graph constructed based on X:*

- *If d_i is an even number, the pairs $\{(v_i^1, v_i^2), ..., (v_i^{d_i-1}, v_i^{d_i})\}$ must be in the MCMP's solution.*
- *If d_i is an odd number, the pairs $\{(v_i^1, v_i^2), ..., (v_i^{d_i-2}, v_i^{d_i-1})\}$ must be in the MCMP's solution.*

Proof. For the case that d_i is an even number, if one pair (v_i^p, v_{j1}^{q1}) exists in the MCMP's solution, it must have another pair (v_i^{p+1}, v_{j2}^{q2}) among the MCMP's solution (suppose p is the odd number). However, $dis(v_i^p, v_{j1}^{q1}) + dis(v_i^{p+1}, v_{j2}^{q2})$ is greater than $dis(v_i^p, v_i^{p+1}) + dis(v_{j1}^{q1}, v_{j2}^{q2})$ due to the triangle inequality because $dis(v_i^p, v_i^{p+1}) = 0$ so that they can be regarded as one point. Therefore, the pairs $\{(v_i^p, v_i^{p+1}), (v_{j1}^{q1}, v_{j2}^{q2})\}$ must be better than $\{(v_i^p, v_{j1}^{q1}), (v_i^{p+1}, v_{j2}^{q2})\}$. This is shown with a demonstration in Figure 4. Therefore, the nodes in $\{v_i^1, v_i^2, ..., v_i^{d_i}\}$ must have the pairs $\{(v_i^1, v_i^2), ..., (v_i^{d_i-1}, v_i^{d_i})\}$ in the MCMP's solution.

For the case that d_i is an odd number, if one node is excluded, such as $v_i^{d_i}$, the number of nodes in set $\{v_i^1, v_i^2, ..., v_i^{d_i-1}\}$ becomes even, and it is the same as the above case, which completes the proof. □



v_{j2}^{q2}
v_{j1}^{q1}
v_i^{p+1}
v_i^p
$dis(v_i^p, v_{j1}^{q1}) + dis(v_i^{p+1}, v_{j2}^{q2})$
(a)
v_{j2}^{q2}
v_{j1}^{q1}
v_i^{p+1}
v_i^p
$dis(v_i^p, v_i^{p+1}) + dis(v_{j1}^{q1}, v_{j2}^{q2})$
(b)

Figure 4: A demonstration of proof for Theorem 2.

IV. Computational Results

In this section, the tightness of the lower bound will be evaluated first on a set of DCARP instances with known global optima. Then, the performance of two optimization algorithms will be evaluated based on NMLB on another set of more complex DCARP instances, for which the global optima are unknown. In addition, the efficiency of the proposed NMLB approach with the graph pruning strategy is also evaluated on these test instances.

A. Experimental Settings

The DCARP instances used in our experiments are all generated based on a static CARP benchmark instance using a DCARP simulation system [20]. Firstly, to evaluate the tightness of the NMLB, a set of DCARP instances with known global optima is required. However, there currently exists no exact solver for DCARP in the literature. Moreover, the current formulation of DCARP, which defines its decision space as the sequence of tasks, is incompatible with the use of solvers such as CPLEX [32] or Gurobi [33]. Consequently, it is challenging to calculate the exact global optimum for most DCARP instances. Therefore, the *gdb* [34] static CARP instances are employed whose problem dimensionalities are relatively low, and their global optima have been established in the literature, generating a set of special DCARP instances whose global optima can be easily derived. For a static CARP instance, its global optimal solution is executed in the simulation system. Then, vehicles were stopped in random time points to generate a new DCARP instance and keep all traversing costs of edges in the map unchanged. The demands of all tasks also remain the same. Thus, the service routes of the global optimal solution for the new DCARP instance remain consistent with those originally optimized from the static CARP instance.

However, the above generated DCARP instances are considered relatively straightforward for the state-of-the-art optimization algorithm [30], [20]. Therefore, another two sets

Table I: The gap between the obtained lower bound and the global optimum, i.e., Gap calculated by Eq. 6, on DCARP instances created based on the *gdb* static dataset. N_v, N_n, and N_t denote the number of required vehicles, the number of nodes, and the number of tasks, respectively. LB and OPT represent the NMLB and the global optimal solution's total costs for the DCARP instance.

Map	N_v	N_n	N_t	LB	OPT	$Gap(\%)$
gdb1	5	12	18	239	245	2.4
gdb2	6	12	22	299	299	0
gdb3	5	12	18	222	222	0
gdb4	4	11	12	183	196	6.6
gdb5	6	13	20	294	294	0
gdb6	5	12	16	206	209	1.4
gdb7	5	12	15	228	241	5.4
gdb8	10	27	38	235	287	18.1
gdb9	10	26	43	215	241	10.8
gdb10	4	12	18	203	203	0
gdb11	5	22	35	303	303	0
gdb12	7	13	18	329	359	8.4
gdb13	6	10	20	421	421	0
gdb14	5	7	17	78	78	0
gdb15	4	7	15	42	42	0
gdb16	5	8	18	88	88	0
gdb17	5	8	20	64	64	0
gdb18	5	9	31	147	147	0
gdb19	3	8	8	47	47	0
gdb20	4	11	16	94	94	0
gdb21	6	11	28	137	137	0
gdb22	8	11	30	143	143	0
gdb23	10	11	52	230	230	0

Table II: The approximate percentage deviation performance, i.e., APD, for the MAENS [30], [20] and the RTS [31], [20] on the *val* DCARP instances based on the proposed NMLB, calculated by Equation (7). N_v denotes the number of required vehicles. N_n represents the number of nodes in the corresponding required graph. N_t represents the number of tasks on the instance. LB represents the lower bound obtained by NMLB approach. $Best^*$ represents the best solution's cost obtained by the corresponding algorithm. T_O and T_P denote the computational time (unit: sec) for calculating the lower bound with and without pruning strategy, respectively, where the bold one denotes the shorter time used for calculating the NMLB.

Map	N_v	N_n	N_t	T_O	T_P	LB	MAENS		RTS	
							$Best^*$	$APD(\%)$	$Best^*$	$APD(\%)$
val1A	4	24	34	0.219	**0.015**	171	175	2.3	176	2.9
val1B	3	23	28	0.177	**0.022**	142	148	4.2	148	4.2
val1C	7	19	22	0.148	**0.072**	123	143	16.3	143	16.3
val2A	2	21	20	0.061	**0.013**	137	137	0	137	0.0
val2B	2	21	17	0.051	**0.017**	148	157	6.1	157	6.1
val2C	6	19	18	0.053	**0.03**	178	267	50	267	50.0
val3A	2	16	15	0.033	**0.007**	40	46	15	46	15.0
val3B	4	23	30	0.227	**0.014**	76	89	17.1	91	19.7
val3C	5	19	17	0.058	**0.016**	60	90	50	90	50.0
val4A	4	32	35	0.339	**0.059**	247	258	4.5	259	4.9
val4B	3	32	36	0.386	**0.052**	201	220	9.5	221	10.0
val4C	5	35	38	0.524	**0.071**	217	264	21.7	264	21.7
val4D	11	37	42	0.834	**0.176**	355	455	28.2	474	33.5
val5A	6	33	48	0.729	**0.078**	339	361	6.5	385	13.6
val5B	3	31	38	0.313	**0.026**	254	256	0.8	256	0.8
val5C	4	32	36	0.402	**0.054**	254	262	3.1	262	3.1
val5D	7	31	39	0.531	**0.085**	352	382	8.5	382	8.5
val6A	3	26	33	0.272	**0.031**	162	166	2.5	166	2.5
val6B	3	27	28	0.196	**0.02**	139	152	9.4	152	9.4
val6C	9	29	33	0.439	**0.106**	199	246	23.6	246	23.6
val7A	3	33	31	0.298	**0.07**	151	170	12.6	171	13.2
val7B	3	32	36	0.396	**0.046**	179	190	6.1	190	6.1
val7C	8	32	42	0.803	**0.136**	225	227	0.9	227	0.9
val8A	8	30	57	1.495	**0.082**	438	475	8.4	486	11.0
val8B	4	29	38	0.525	**0.034**	264	268	1.5	268	1.5
val8C	10	29	40	0.725	**0.139**	349	416	19.2	429	22.9
val9A	2	46	51	0.876	**0.042**	193	193	0	193	0.0
val9B	3	45	55	1.016	**0.051**	185	187	1.1	187	1.1
val9C	4	42	44	0.648	**0.063**	167	175	4.8	175	4.8
val9D	9	48	60	1.24	**0.168**	252	266	5.6	266	5.6
val10A	5	48	64	1.507	**0.152**	362	368	1.7	375	3.6
val10B	3	45	54	1.018	**0.069**	243	245	0.8	245	0.8
val10C	6	44	55	1.201	**0.108**	284	301	6	304	7.0
val10D	12	49	68	2.366	**0.257**	352	402	14.2	414	17.6

of more complex DCARP instances are generated, based on *egl* [35] and *val* [36] static CARP instances with more tasks. For each static CARP instance, one DCARP instance is generated by executing the solution on the instance's map using the DCARP simulation system [20]. The cost-change events, adding demand events, and new task events happen

Table III: The approximate percentage deviation performance, i.e., APD, for the MAENS [30], [20] and the RTS [31], [20] on the *egl* DCARP instances based on the proposed NMLB, calculated by Equation (7). N_v denotes the number of required vehicles. N_n represents the number of nodes in the corresponding required graph. N_t represents the number of tasks on the instance. LB represents the lower bound obtained by NMLB approach. $Best^*$ represents the best solution's cost obtained by the corresponding algorithm. T_O and T_P denote the computational time (unit: sec) for calculating the lower bound with and without pruning strategy, respectively, where the bold one denotes the shorter time used for calculating the NMLB.

Map	N_v	N_n	N_t	T_O	T_P	LB	MAENS		RTS	
							$Best^*$	$APD(\%)$	$Best^*$	$APD(\%)$
egl-e1-A	6	59	47	0.861	**0.237**	2247	3466	54.3	3564	58.6
egl-e1-B	8	56	42	0.937	**0.476**	4158	5106	22.8	5672	36.4
egl-e1-C	14	64	54	1.833	**0.862**	6045	7985	32.1	9386	55.3
egl-e2-A	6	59	49	0.969	**0.247**	2199	3639	65.5	3729	69.6
egl-e2-B	11	61	53	1.835	**0.666**	7695	12312	60	14728	91.4
egl-e2-C	15	63	56	2.149	**1.019**	11846	15900	34.2	18256	54.1
egl-e3-A	9	69	64	2.52	**0.648**	3997	5181	29.6	5742	43.7
egl-e3-B	13	64	60	2.267	**0.918**	5286	7113	34.6	9497	79.7
egl-e3-C	22	69	66	3.723	**1.758**	7878	12468	58.3	15095	91.6
egl-e4-A	10	67	66	2.699	**0.547**	4652	6208	33.4	6312	35.7
egl-e4-B	16	66	66	3.363	**1.108**	5882	9889	68.1	12139	106.4
egl-e4-C	22	62	62	3.249	**1.744**	10152	16386	61.4	20402	101.0
egl-s1-A	10	98	77	4.594	**1.263**	3949	5412	37	6628	67.8
egl-s1-B	15	100	79	5.466	**1.757**	5027	7837	55.9	10217	103.2
egl-s1-C	20	88	59	3.264	**2.372**	5697	10052	76.4	12621	121.5
egl-s2-A	22	118	110	14.729	**3.677**	9178	17022	85.5	21488	134.1
egl-s2-B	35	118	118	21.728	**8.162**	13789	23062	67.2	30672	122.4
egl-s2-C	40	120	112	19.756	**9.259**	19898	29896	50.2	38065	91.3
egl-s3-A	22	116	116	17.712	**3.713**	7124	13524	89.8	18144	154.7
egl-s3-B	30	122	115	17.746	**6.014**	13872	22219	60.2	31095	124.2
egl-s3-C	33	114	100	13.538	**7.103**	12479	19922	59.6	23303	86.7
egl-s4-A	34	116	122	22.258	**6.193**	14401	21939	52.3	30219	109.8
egl-s4-B	37	119	123	22.778	**8.866**	14873	23080	55.2	33886	127.8
egl-s4-C	40	119	113	18.204	**10.38**	16298	25979	59.4	34462	111.4

concurrently to generate the testing DCARP instances. The details of dynamic changes in each DCARP instances have been thoroughly documented in our repository of generated DCARP instances, which are all available on Github[3].

The proposed NMLB approach and the graph pruning strategy were both implemented in Python using **NetworkX** library [37] in our experiments since the algorithm for solving MCMP is available in the **NetworkX** library. The generated instances and the source code of the proposed algorithms are available on Github[3]. The experimental results were obtained using an Intel Core i7 3.5 GHz with 16G RAM.

B. The Tightness of NMLB

The tightness of the proposed NMLB is evaluated on a set of DCARP instances with known global optima based on the static CARP instances (*gdb*). To measure the tightness of this lower bound, the "Gap" metric is used to represent the disparity between the NMLB and the global optimum, and is calculated using the following equation:

$$Gap = \frac{|LB - OPT|}{OPT} \times 100\%. \tag{6}$$

The computational results are summarized in Table I. It is clear that on most created DCARP instances, the gap is zero. Only two instances, i.e., *gdb8* and *gdb9*, have a gap greater than 10%, and the gap of remaining instances are all smaller than 10%. Thus, we can conclude that our proposed NMLB can obtain a tight lower bound in comparison to the global optimal value on these small DCARP instances. To investigate the relation between the tightness of the lower bound and various characteristics of DCARP instances, the number of nodes in the corresponding *required graph* (G_r), the number of tasks on the instance, the least number of required vehicles, and the Gap are presented in the Figure 5. From the figure, instances featuring more nodes in the *required graph*, more number of tasks, and more number of vehicles generally have larger gaps between the NMLB and the global optimum. For the tightness of the proposed lower bound on more complex DCARP instances, especially those on larger instances, identifying the theoretical global optimum is essential. However, as mentioned before, the current formulation of DCARP, which defines its decision space as a sequence of tasks, is incompatible with the use of exact solvers such as CPLEX [32] or Gurobi [33]. Thus, in the future, it is valuable to adapt existing exact solver to precisely solve DCARP instances, enabling the evaluation of tightness of proposed lower bound on large instances.

C. Performance Measurement for the Optimization Algorithm

In our study, two meta-heuristic optimization algorithms for DCARP instances, known as the memetic algorithm with extended neighborhood search (MAENS) [30] and the repair operator based tabu search (RTS) [31] combined with a generalized DCARP optimization framework [20], were evaluated. The objective here is to demonstrate how lower bound approaches proposed in this paper can be used to evaluate meta-heuristic algorithms and provide insight. MAENS is a memetic algorithm equipped with a specialized merge-split operator enabling big exploration steps within the search space of the CARP [30]. RTS is a tabu search algorithm with a specific global repair operator capable of repairing infeasible solutions

[3]https://github.com/HawkTom/NMLB

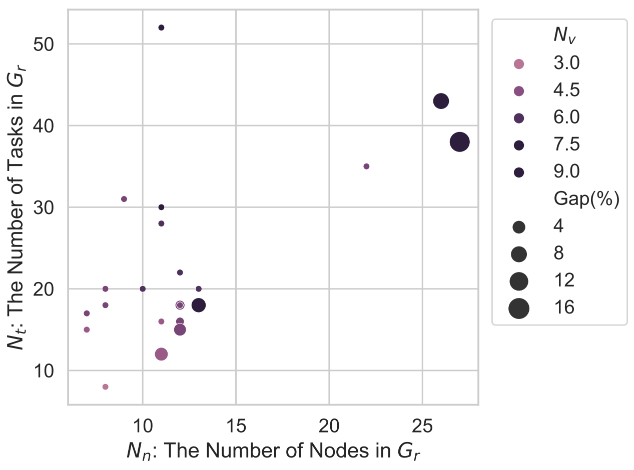


Figure 5: The demonstration of the relation between the tightness of the lower bound and the characteristics of the DCARP instance. Points with bigger size indicate instances with larger value of Gap.

to facilitate the optimization [31]. These algorithms are applied to a set of more complex DCARP instances generated from *egl* and *val* static datasets. Enough computational resources was provided for both meta-heuristic algorithms to obtain the best solution.

On the basis of the proposed NMLB, the following performance metric, termed as *Approximate Percentage Deviation*[4] was used, as presented in the Equation (7), where the $Best^*$ is the best objective value obtained by the optimization algorithm.

$$APD = \frac{|Best^* - LB|}{LB} \times 100\%. \tag{7}$$

The computational results on the generated DCARP instances are presented in Tables II and III. The APD of MAENS for *val* (Table II) is relatively small with around 5% for most instances. Notably, there are two instances with APD equal to 0 and three instances with APD less than 1%. This suggests that MAENS is effective for these DCARP instances. RTS obtained the same results on 12 instances and had a slightly larger APD on the remaining instances. It indicates that RTS performs comparably to MEANS and proves to be effective for these *val* instances as well. Concurrently, it provides additional evidence indicating that our lower bound is tight in these DCARP instances. This is because, even though the optimality on these instances is unknown, MAENS and RTS would not have been possible to find solutions close to the lower bound if the lower bound was not tight.

In contrast, the APD of MAENS and RTS is consistently larger for *egl* DCARP instances (Table III) compared to those for *val* DCARP instances. Specifically, most *egl* DCARP instances have an APD greater than 50% for both MAENS and RTS, with the smallest APD being 29.6% for MEANS and 35.7% for RTS. This suggests that there exists room for further optimization enhancement for both MAENS and RTS on *egl* DCARP instances, although such results could also be due to a potentially large gap between the lower bound and the (unknown) exact global optima on these instances. The correlation between the APD and the characteristics of the DCARP instances, including the number of nodes in G_r, the number of tasks, and the number of required vehicles are illustrated in Figure 6. Each point denotes an instance and points with a bigger size indicate instances with a larger value of APD. From Figure 6, instances with more tasks, more nodes in G_r, and more required vehicles have larger APD. This suggests the performance of both MAENS and RTS on these complex DCARP instances can still be improved.

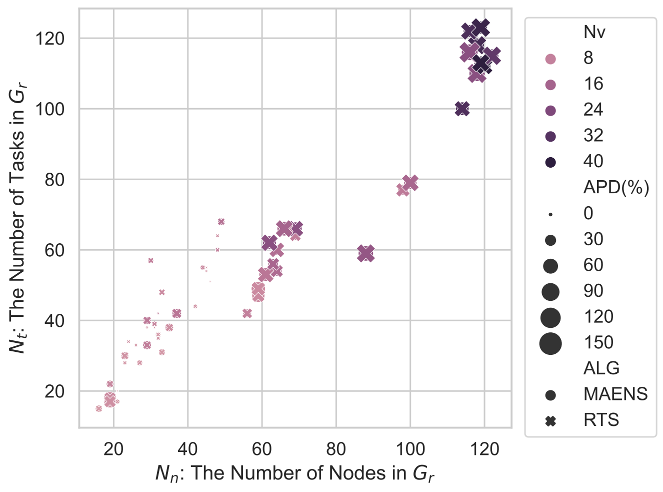


Figure 6: Relationship between the algorithm's performance (APD) and the characteristics of DCARP instances. Points with bigger sizes indicate instances with larger values of APD[5].

D. Efficiency of the Graph Pruning Strategy

The computational time for the original NMLB approach (T_O) and the NMLB approach with the graph pruning strategy (T_P) to calculate the lower bound are also presented in Tables II and III. It is clear that the graph pruning strategy significantly reduces the computational time for calculating the lower bound on almost all tested DCARP instances, especially for instances in the *egl* dataset.

In our proposed graph pruning strategy, a subset of nodes in the constructed auxiliary graph are removed, which will definitely match its copies in the solution of MCMP. Thus, if a vertex has more copies in the auxiliary graph, more vertices will be removed by the pruning strategy. For each vertex in G_R, the number of its copies in the auxiliary graph is equal to its degree in G_R. Since one vertex is likely to be pruned only if it has at least two copies in the auxiliary graph, the number of vertices whose degrees are greater than 2 in G_R are calculated. The results are presented in Figure 7, in which the time difference is more or less linearly correlated with the number of vertices whose degrees are greater than 2.

[4]In the literature, there is no a standardized term for using lower bounds to evaluate meta-heuristic algorithms. Therefore, for convenience, we termed it as *Approximate Percentage Deviation* here.

[5]The data points for MAENS and RTS overlap because each point corresponds to a specific instance.

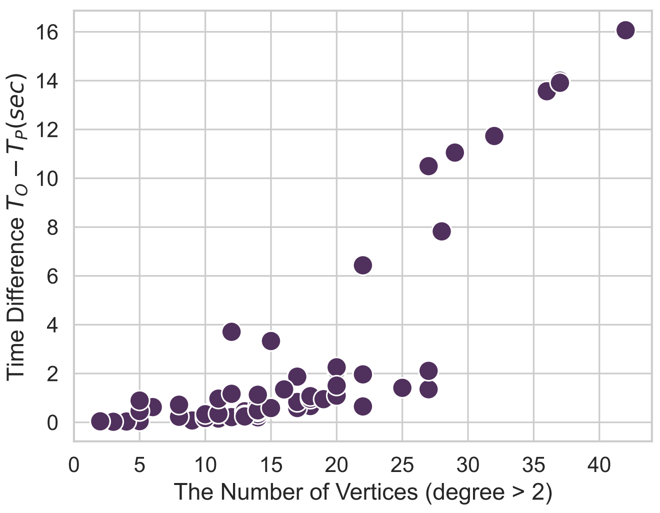


Figure 7: Relationship between the difference of computational time and the number of vertices in G_R whose degree are greater than 2.

V. Conclusion

This paper mainly focuses on measuring the absolute performance of meta-heuristic algorithms for DCARP in terms of closeness to the global optimum. Since the theoretical global optimum of most DCARP instances is not known, a node matching lower bound (NMLB) was derived as the alternative to the unknown global optimum to assess the quality of solutions. Then, the metric *Approximate Percentage Deviation* (APD) based on the derived lower bound was used to evaluate the performance of meta-heuristic algorithms for DCARP.

To derive a tight lower bound for DCARP instances, the *required graph*, which is a complete graph constructed by the depot and all nodes linking at least one task in the original DCARP instance was considered. Then, NMLB was proposed based on the idea that the number of times a node can be assigned to link different tasks, starting point, or ending point of a route cannot exceed its degree in the *required graph*. To handle routes belonging to the outside vehicles, which either directly return to the depot without serving any tasks or continue serving remaining tasks, the lower bound derivation was modeled as a minimum cost matching problem (MCMP). The solution to the constructed MCMP is the case with minimum costs for adding *artificial links* in the *required graph*. In addition, to enhance the computational efficiency of our proposed NMLB, a pruning strategy was further proposed.

Our experimental studies evaluated the tightness of NMLB, i.e., the gap between obtained lower bound and the global optimum, on some specifically designed DCARP instances. The results demonstrate that the proposed NMLB provides tight lower bounds on small DCARP instances. Then, the performance of two optimization algorithms was evaluated through APD on large complex DCARP instances. The results show that there are still large gaps between the optimized solutions found by these two algorithms and the lower bounds, indicating much room for further improvement to the algorithms. In addition, our experimental studies also demonstrated that the NMLB approach benefits a lot from the pruning strategy for DCARP instances with more vertices whose degrees are greater than 2 in the *required graph*.

In the future, more DCARP meta-heuristic algorithms can be evaluated using our lower bound approach to analyze their optimization abilities relative to the global optimum, thereby enhancing the understanding of these methods. In addition, since the gap between NMLB and the theoretical global optimum is still large for some instances, it is valuable to investigate more about the specific influence of instance characteristics on the NMLB and develop tighter lower bounds for DCARP instances.

Acknowledgments

Hao Tong gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU).

References

[1] M. A. Rahman, R. Sokkalingam, M. Othman, K. Biswas, L. Abdullah, and E. Abdul Kadir, “Nature-inspired metaheuristic techniques for combinatorial optimization problems: overview and recent advances,” *Mathematics*, vol. 9, no. 20, p. 2633, 2021.

[2] A. Radhakrishnan and G. Jeyakumar, “Evolutionary algorithm for solving combinatorial optimization—a review,” in *Innovations in Computer Science and Engineering*, pp. 539–545, Springer Singapore, 2021.

[3] M. Mavrovouniotis, C. Li, and S. Yang, “A survey of swarm intelligence for dynamic optimization: Algorithms and applications,” *Swarm and Evolutionary Computation*, vol. 33, pp. 1–17, 2017.

[4] P. K. Lehre and X. Qin, “Self-adaptation can help evolutionary algorithms track dynamic optima,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 1619–1627, 2023.

[5] A. H. Halim, I. Ismail, and S. Das, “Performance assessment of the metaheuristic optimization algorithms: an exhaustive review,” *Artificial Intelligence Review*, vol. 54, pp. 2323–2409, 2021.

[6] C. Munien, S. Mahabeer, E. Dzitiro, S. Singh, S. Zungu, and A. E.-S. Ezugwu, “Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study,” *IEEE Access*, vol. 8, pp. 227438–227465, 2020.

[7] A. W. Mohamed, K. M. Sallam, P. Agrawal, A. A. Hadi, and A. K. Mohamed, “Evaluating the performance of meta-heuristic algorithms on CEC 2021 benchmark problems,” *Neural Computing and Applications*, vol. 35, pp. 1493–1517, sep 2022.

[8] D. S. Hochba, “Approximation algorithms for np-hard problems,” *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[9] H.-Y. Chao, M. P. Harper, and R. W. Quong, “A tight lower bound for optimal bin packing,” *Operations Research Letters*, vol. 18, no. 3, pp. 133–138, 1995.

[10] B. Jarboui, H. Derbel, S. Hanafi, and N. Mladenović, “Variable neighborhood search for location routing,” *Computers & Operations Research*, vol. 40, no. 1, pp. 47–57, 2013.

[11] A. Haghani and S. Jung, “A dynamic vehicle routing problem with time-dependent travel times,” *Computers & Operations Research*, vol. 32, no. 11, pp. 2959–2986, 2005.

[12] B. L. Golden and R. T. Wong, “Capacitated arc routing problems,” *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[13] A. Mor and M. G. Speranza, “Vehicle routing problems over time: a survey,” *Annals of Operations Research*, pp. 1–21, 2022.

[14] H. Handa, L. Chapman, and X. Yao, “Robust route optimization for gritting/salting trucks: A cercia experience,” *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 6–9, 2006.

[15] E. B. Tirkolaee, I. Mahdavi, and M. M. S. Esfahani, “A robust periodic capacitated arc routing problem for urban waste collection considering drivers and crew’s working time,” *Waste Management*, vol. 76, pp. 138–146, 2018.

[16] Y. Chen, J. Hao, and F. W. Glover, “A hybrid metaheuristic approach for the capacitated arc routing problem,” *European Journal of Operational Research*, vol. 253, no. 1, pp. 25–39, 2016.

[17] J. Liu, K. Tang, and X. Yao, “Robust optimization in uncertain capacitated arc routing problems: Progresses and perspectives,” *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 63–82, 2021.

[18] M. Liu, H. K. Singh, and T. Ray, "A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 595–602, IEEE, 2014.

[19] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A hybrid local search framework for the dynamic capacitated arc routing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 139–140, 2021.

[20] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "A novel generalized meta-heuristic framework for dynamic capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 26, pp. 1486–1500, dec 2022.

[21] Z. Nagy, Á. Werner-Stark, and T. Dulai, "An artificial bee colony algorithm for static and dynamic capacitated arc routing problems," *Mathematics*, vol. 10, no. 13, p. 2205, 2022.

[22] W. Padungwech, J. Thompson, and R. Lewis, "Effects of update frequencies in a dynamic capacitated arc routing problem," *Networks*, vol. 76, no. 4, pp. 522–538, 2020.

[23] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys (CSUR)*, vol. 18, no. 1, pp. 23–38, 1986.

[24] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff, and X. Yao, "What makes the dynamic capacitated arc routing problem hard to solve: insights from fitness landscape analysis," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 305–313, 2022.

[25] A. A. Assad, W.-L. Pearn, and B. L. Golden, "The capacitated chinese postman problem: Lower bounds and solvable cases," *American Journal of Mathematical and Management Sciences*, vol. 7, pp. 63–88, Jan. 1987.

[26] E. Benavent, V. Campos, A. Corberán, and E. Mota, "The capacitated arc routing problem: lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, 1992.

[27] Y. Saruwatari, R. Hirabayashi, and N. Nishida, "Node duplication lower bounds for the capacitated arc routing problem," *Journal of the Operations Research Society of Japan*, vol. 35, no. 2, pp. 119–133, 1992.

[28] S. Wøhlk, "New lower bound for the capacitated arc routing problem," *Computers & Operations Research*, vol. 33, pp. 3458–3472, Dec. 2006.

[29] E. Bartolini, J.-F. Cordeau, and G. Laporte, "Improved lower bounds and exact algorithm for the capacitated arc routing problem," *Mathematical Programming*, vol. 137, no. 1, pp. 409–452, 2013.

[30] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.

[31] Y. Mei, K. Tang, and X. Yao, "A global repair operator for capacitated arc routing problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 3, pp. 723–734, 2009.

[32] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[33] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.

[34] B. L. Golden, J. S. DeArmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Computers & Operations Research*, vol. 10, no. 1, pp. 47–59, 1983.

[35] L. Y. Li and R. W. Eglese, "An interactive algorithm for vehicle routeing for winter—gritting," *Journal of the Operational Research Society*, vol. 47, no. 2, pp. 217–228, 1996.

[36] E. Benavent, V. Campos, A. Corberán, and E. Mota, "The capacitated arc routing problem: lower bounds," *Networks*, vol. 22, no. 7, pp. 669–690, 1992.

[37] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

APPENDIX

Table IV: The notations related to the problem formulation.

Notation	Meaning
G	Graph $G = (V, E)$.
V	Set of nodes.
E	Set of edges.
G_R	Required graph $G_R = (V_R, E_R)$. (A complete graph)
V_R	Set of nodes including the depot and nodes which are incident with at least one task.
E_R	Set of tasks.
I_t	The DCARP instance at the time point t.
v_0	The depot.
$dm(e)$	The demand of an edge $e \in E$.
$sc(t)$	The serving cost of a task $t \in E_R$.
$mc(v_i, v_j)$	The minimal total deadheading cost from vertex v_i to v_j.
$head_t$	The head node of task t.
$tail_t$	The tail node of task t.
N_t	The number of tasks, $N_t = \|E_R\|$.
Q	The capacity of empty vehicles.
V_{ov}	The set of outside vehicles' stopping nodes.
N_{ov}	The number of outside vehicles, $N_{ov} = \|V_{ov}\|$.
q_k	The remaining capacity of the k^{th} outside vehicle.
S	A DCARP solution, i.e., a set of routes.
R_k	The k^{th} route in the S.
l_k	The number of tasks in the R_k.
$t_{k,i}$	The i^{th} task in the R_k.
RC_{R_k}	The total cost of the route R_k.
$TC(S)$	The total cost of solution S.

Table V: The notations related to the problem's lower bound.

Notation	Meaning
C_T	The total serving costs of all tasks.
W_T	The total demands of all tasks.
k_0	The minimum number of vehicles required for a (D)CARP instance.
C^*	The cost of the optimal (D)CARP instance.
S^*	The optimal solution for a DCARP instance.
R^*_k	The k^{th} route in the S^*.
$t^*_{k,i}$	The i^{th} task in the R^*_k.
$d(v_i)$	The degree of the node $v_i \in V_R$ in G_R.
C_1	The total costs of artificial edges directly linking to the depot in a (D)CARP solution.
C_2	The total costs of artificial edges linking tasks in a (D)CARP solution.
G_x	The auxiliary graph $G_x = (V_x, E_x)$.
V_x	Set of nodes for building G_x.
E_x	Set of edges for building G_x.
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$	Subsets of nodes in V_x and $V_x = \mathbf{A} \cup \mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$
C_{MCMP}	The cost of optimal solution of the minimum cost matching problem on G_x.
C^*_{links}	The total costs of all artificial links in S^* of a DCARP instance.