General heuristics algorithms for solving capacitated arc routing problem

Mohammad Fadzli Ramli
Universiti Malaysia Perlis
46 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Nurul Najwa
Universiti Malaysia Perlis
2 PUBLICATIONS **1** CITATION
SEE PROFILE

Muhamad Hafiz Masran
Universiti Malaysia Perlis
25 PUBLICATIONS **25** CITATIONS
SEE PROFILE

General heuristics algorithms for solving capacitated arc routing problem

Mohammad Fadzli, Nurul Najwa, and Hafiz Masran

Citation: AIP Conference Proceedings **1660**, 050002 (2015); doi: 10.1063/1.4915635
View online: http://dx.doi.org/10.1063/1.4915635
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1660?ver=pdfcov
Published by the AIP Publishing

General Heuristics Algorithms for Solving Capacitated Arc Routing Problem

Mohammad Fadzli, Nurul Najwa and Hafiz Masran

Institut Matematik Kejuruteraan, Universiti Malaysia Perlis, Kampus Pauh Putra, 02600 Arau, Perlis, Malaysia

Abstract. In this paper, we try to determine the near-optimum solution for the capacitated arc routing problem (CARP). In general, NP-hard CARP is a special graph theory specifically arises from street services such as residential waste collection and road maintenance. By purpose, the design of the CARP model and its solution techniques is to find optimum (or near-optimum) routing cost for a fleet of vehicles involved in operation. In other words, finding minimum-cost routing is compulsory in order to reduce overall operation cost that related with vehicles. In this article, we provide a combination of various heuristics algorithm to solve a real case of CARP in waste collection and benchmark instances. These heuristics work as a central engine in finding initial solutions or near-optimum in search space without violating the pre-setting constraints. The results clearly show that these heuristics algorithms could provide good initial solutions in both real-life and benchmark instances.

Keywords: Capacitated arc routing problem; heuristics; waste collection; graph theory.
PACS: 02.10.Ox

INTRODUCTION

Capacitated arc routing problem (CARP) is a class of routing problem that specially design for street services. CARP is NP-hard and is considered as a special connected graph network specifically arises from street operations such as household refuse collection, road maintenance, meter reading and mail delivery. Classical CARP deals only with one problem-type; where the vehicle starts from a depot, services a set of arcs exactly once without exceeding its capacity before returns to the depot. By purpose, CARP focuses on servicing a set of arcs with minimum cost within a set of pre-specified constraints. This model was proposed by Golden and Wong [1], in contras to its counterpart node routing or vehicle routing problem (VRP). In VRP, the vehicle needs to service a set of required nodes, meanwhile vehicle in CARP must service a set of required arcs. This differentiation separates both models in theory and methodology but CARP receives little attention in research when compared to VRP. The design of the CARP model and its solution techniques purposefully is to find optimum (or near-optimum) routing cost for a fleet of vehicles involved in operation. In other words, finding minimum-cost routing is compulsory in order to reduce overall operation cost that related with vehicles.

CARP is defined on the connected graph $G = (V, E)$; where V is a set of vertices (or nodes) and E is a set of edges. A fleet of vehicles with fixed capacity W is stationed at a depot V_0 and they are required to service all edges $E_{ij} \in E$ exactly once without exceeding its capacity. In deadheading circumstance, the vehicle is allowed to traverse each edge more than once without servicing it. Note that in this entire paper we consider our CARP as two-way street direction, which can be modelled as a set of edges in the network. On the other hand, one-way street is modelled as a set of directed arcs. .

Unknown and non-negative demands, $q_{ij} > 0$ and non-negative traversal cost, $c_{ij} > 0$ are assigned to each edge, so that $q_{ij} \subseteq E$ and $c_{ij} \subseteq E$. In CARP, $\forall E_{ij} \in E$ is a set of customers that need the service whereby $\forall V \setminus V_o \in V$ in contrary is a set of customers that need service in VRP. The goal of CARP model is to determine a minimum routing cost and reduce the vehicle trips, such that each trip starts and ends at the depot. In this paper, we consider CARP with single depot and no split demand is permitted. These CARP constraints have the following properties based on symmetric and undirected network:

a) Each trip for one identical vehicle starts and ends at a single depot, in such that $S_i = \{V_0, V_1, \ldots, V_n, V_0\}$.
b) No split demand is allowed.
c) Demands on edges are unknown but followed certain distribution.
d) Traversal cost and demands are symmetric, such that $c_{ij} = c_{ji}$ and $q_{ij} = q_{ji}$ respectively.

The CARP is NP-hard and no method could be claimed as the best so far. Usually, one method only fits for a specific designed problem and could only solve a certain set of instances. Exact methods are limited to 20 edges [2], explaining the importance of heuristics. Path-scanning [3], augment-merge [1] and Ulusoy's heuristics [4] are good examples of fast heuristics to compute acceptable solutions. In general, metaheuristics such like tabu search [5], genetic algorithm [6], memetic algorithm [7] and guided local search [8] could produced better results, but in longer computation time when compared to simple heuristics [9].

HEURISTICS: ALGORITHMS AND SOLUTIONS

In this section, we start by introduce a framework of heuristic algorithm for initial search that works as a guideline which will be used in the computation process. The framework will be extended in programming phase to produce initial solution of local search thus estimates the result. Initial or local search is the most frequently used heuristic technique for solving combinatorial optimization problems and also the basis for modern metaheuristics like tabu search. Most of the effort spent within a local search algorithm is used for scanning the neighbourhood, i.e., the set of solutions which close to the current solution [10].

This step in this study will also delineate how mathematical formulation and algorithms interrelated and hence work in conjunction with heuristics as well. This type of modelling-algorithm interface is a staple characteristic but is not fully examined in operations research. Metaheuristics such like genetic algorithm, tabu search and ant system often produced better cost solutions when compared to common heuristics, but in higher computation time. In contrary, heuristics algorithms do not seek global optimal solutions, but can quickly provide near-optimum solutions [11], which is more demanding in practice. Classical CARP heuristics are devoted to augment-merge [1], path-scanning [3] and Ulusoy's algorithm [4]. Belenguer et al. [12] applied path-scanning, augment-merge and Ulusoy's heuristic to determine lower and upper bounds for the mixed CARP (MCARP). They conclude that these simple heuristics are very fast in term of computation time, but the average deviation to the optimal objective function is considerably high (23 percent), explaining the inconsistency of the algorithms to produce a better and more stable result. As a matter of fact, heuristics provide wider approaches to be explored, to make it competence with other metaheuristics such like tabu search and genetic algorithms. General heuristic framework for the routing problem consists of the following steps:

.

<u>Step 1</u>: Input basic data, such as demand, supply parameters and network, $G = (V, A)$.

<u>Step 2</u>: Denote the depot as the beginning of a route.

<u>Step 3</u>: Determine the customer closest to the last customer added to the route. Assume customer i is the first customer added to the route and customer $i + 1$ represents the next customer added to the route.

<u>Step 4</u>: Repeat Step 3 until the vehicle is filled to capacity.

<u>Step 5</u>: Assign another vehicle and repeat Step 2 until all customers have been served.

Notations and Heuristics Algorithms

$k_{init} := 0$, *initiate the number of trips before first trip starts* (1)

$y_{init} := 0$, *initiate the number of unserviced arcs before first trip starts* (2)

$q_{init} := 0$, *initiate the initial quantity for one vehicle before first trip starts* (3)

$c_{init} := 0$, *initiate the initial cost for one vehicle before first trip starts* (4)

$c_{ij} := c_{ji}$, *cost of arc* E_{ij} *which randomly distributed* (5)

E_{i+1}: *next successor edge* (6)

$q_{new} := q_{init} + q_{ij}$, *capacity at arc* ij (7)

$q_{balnew} := W - q_{new}$, *spare capacity after serviced the arc* ij (8)

$c_{new} := c_{init} + c_{ij}$, *sum of serviced arc cost from depot to point i to point j* (9)

$c_{glocost}$:= *global cost after each trip terminates* (10)

$q_{new} := \sum_{(i,j)\in R} q_{ijk}$, *sum of capacity from point i to point n, assigned to capacity variable* (11)

$q_{balnew} := W - \sum_{(i,j)\in R} q_{ijk}$ *, spare capacity from point i to point n,*
assigned to spare capacity variable (12)
$c_{new} := \sum c_{ij}$ *, sum of all arc cost,*
assigned to cost variable (13)
$q_{init} := q_{new}$*, new capacity reassign to initial capacity*
after each cycle (14)
$c_{init} := c_{new}$*, new cost reassign to initial cost*
after each cycle (15)
$q_{balnew} \geq q$ *, decision operator for capacity* (16)

Global design of the heuristics algorithm that control the searching process is drawn below in Step 1 to Step 4. In general, the designed algorithm builds a feasible route by inserting at every iteration an unrouted customer into a previous connected serviced routes. This process is performed one route a time and is gradually improves in each iteration.

Step 1: Generate network graph $G(V, E)$, demand, q and cost, c according to pre-spesified distribution. Input vehicle capacity, W. Set depot, $O = initial$; $y_{init} = 0$; quantity, $q_{init} = 0$; spare capacity, $q_{balnew} = W$, cost, $c_{init} = 0$; trip $k = 0$.

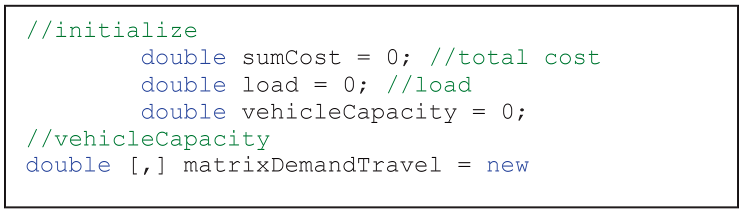
Step 2: Compare and choose the best $E_{(i,j)} \in V_n$ using nearest procedure of highest demand/cost (NPHDC).
Find the first node, $V_n = \{V_i\}$ and edge, $E_{(i,j)}$. Mark $q_{ij} \in E_{(i,j)}$ and $c_{ij} \in E_{(i,j)}$.
From O, $k := k + 1$. Set $y = y_{init} + 1$. Count new quantity, $q_{new} = q_{init} + q_{ij}$ and $q_{balnew} = W - q_{new}$. Count new cost, $c_{new} = c_{init} + c_{ij}$.
Mark the covered $E'_{(i,j)} = E'_{(j,i)} = 1$.

Step 3: REPEAT mark the next successor arc, $E_{(i+1, j+1)}$ using NPHDC UNTIL $q_{new} \geq W$ or $q_{balnew} \leq 0$. Assigned $q_{new} := \sum_{(i,j)\in R'} q_{ijk}$, $q_{balnew} := W - \sum_{(i,j)\in R'} q_{ijk}$ and $c_{new} := \sum_{(i,j)\in R'} c_{ijk}$.

```
if $q_{balnew} \leq 0$ OR $q_{new} \geq W$ then
        Stop, cancel previous repeat and check all serviced $E'_{(i,j)} \in R'$.
if $(E(i,j) \in R) - (E'(i,j) \in R') = 0$ OR $(E(i,j) \in R) - (E'(j,i) \in R') = 0$ then
        Stops Step 3. Go to Step 4.
if $(E(i,j) \in R) - (E'(i,j) \in R') > 0$ OR $(E(i,j) \in R) - (E'(j,i) \in R') > 0$ then
        Count $c_{glocost} := c_{new}$. Mark the covered $E'_{(i,j)} \in R'$ , terminate $\forall q_{ijk} \in E'_{(i,j)}$ and repeat Step 2.
```

Step 4: Store $c_{glocost} := c_{new}$ and k. Terminate $\forall q_{ijk} \in E'$ and $\forall c_{ijk} \in E'$. Count all variables.

Step 1 is described as follows. Before the first iterations start, all vertices and edges, customer demands and arc cost are randomly generated by a specified distribution. Before the first cycle begins the searching process, all variables such as decision variables y, quantity q, cost c and trip k initially sets to zero. FIGURE 1 depicts the Step 1 algorithm in C# language.



```
//initialize
        double sumCost = 0; //total cost
        double load = 0; //load
        double vehicleCapacity = 0;
//vehicleCapacity
double [,] matrixDemandTravel = new
```

FIGURE 1. Initialize start-up parameters in C#.

Step 2 is a start-up moves that finding the first customer node and service arc according to heuristic rule. This first move generated the first trip $k = 1$. When an arc is chosen to move, that arc is covered. Then its demand and cost will accumulate in quantity and cost variables, they are q_{new} and c_{new} respectively. q_{new} and c_{new} variables hold this values until the next iteration. The spare capacity, q_{balnew} is calculated by subtracting q_{new} from total capacity, W. Before proceeds to the next neighbourhood moves, that arc y is marked for next search cycles. Note that the algorithm is designed to cover symmetrical CARP where all edges are bidirectional $E(i, j) = E(j, i)$. Similarly, the demand and cost for every edge in the network graph are also assumed symmetric, and denoted as $q_{ij} = q_{ji}$ and $c_{ij} = c_{ji}$ respectively. Next, Step 3 is explained as follows. This step continuously adds the selected unserviced demand q_{ij} to q_{new} thus increased the total capacity W. In contrary, the spare capacity, q_{balnew} is decreased simultaneously until no empty is available. This cycle stops when the vehicle is nearly or fully loaded, $q_{balnew} \leq 0$ or the collected demands exceeds the maximum capacity, $q_{new} \geq W$. This process works in conjunction with Step 1, Step 2 and Step 4 and recalls their functions until all arcs in the network graph are serviced and terminated. Whenever there are unserviced arcs remaining in the graph, Step 2 is repeated and this counts the $k+1$ trips. Before this global cycle starts once again, all collected demands, $q_{ij} \in E'_{(i, j)}$ in k trip is eliminated to ensure no redundant collection in trip $k+1$. However, the arc's cost $c_{ij} \in E'$ in the previous global cycle is kept to the next searching process, permitted the deadheading traverse. Step 3 is terminated when all arcs in the graph are serviced, $(E(i, j) \in R) - (E'(i, j) \in R') = 0$ or $(E(j, i) \in R) - (E'(j, i) \in R') = 0$. Finally, Step 4 is the terminating procedure that keeps all last values after the last global cycle stops.

Nearest Procedure of Highest Demand/Cost

In this study, nearest procedure of highest demand/cost (NPHDC) is the central engine in our entire heuristics algorithms. In general, the NPHDC heuristic is developed to generate the initial and feasible solution by satisfying all necessary constraints. This heuristic generates a good solution that gives an optimum (or near optimum) solution to the CARP model. NPHDC works in start-up and neighbourhood moves of loading function that finds the first node to begin the service and its successor edge.

The start-up move of NPHDC is likely similar to path-scanning greedy technique where the algorithm looks the unserviced successor arc, $x_{ijk} = 0$. However, NPHDC only generates start-up moves from any corner vertex after entire network region identified. We refer to our paper [14] for further explanation on the identification of the corner vertex in a special network graph. The start-up move algorithm of NPHDC is designed as follows:

```
Step 1: FOR $G = (V, E)$, scan $\forall V \in G$.
          if two successor arcs ($E_1$ and $E_2$) intercept $V$ then
                $V_c := \{V_1, \ldots, V_n\}$.

Step 2: FOR $\forall V_c$, compare $q_{mn1}/c_{mn1} \in E_1$ with $q_{st1}/c_{st1} \in E_2$.
          Assign $val_{mn1} := q_{mn1}/c_{mn1}$ and $val_{st1} := q_{st1}/c_{st1}$.
            if $val_{mn1} > val_{st1}$ then $tempv := val_{mn1}$,
            else $tempv := val_{st1}$.

Step 3: Compare all temporary value, $tempv$.
        Rank and choose the highest $tempv$ and assigned $x_{ijk} = 1$.
        Count new weight, $q_{new} := q_{init} + q_{ij}$, spare capacity, $q_{balnew} := W - q_{new}$
        and cost, $c_{new} := c_{init} + c_{ij}$.
        Count first trip, $k = 1$.
```

These three steps are described as follows. Firstly in Step 1, the algorithm scans all corner vertices in the graph. The set of corner vertex V_c is said to be found if one vertex has two intercepts of connected arcs. This character defines vertex corner clearly by omitting more than two connected arcs to every node. When these two options occur for every corner vertices, then the highest ratio of quantity/cost, q_{ij}/c_{ij} arc is selected as start-up move. For any network, this algorithm is compute as first trip, $k = 1$. In Step 2, the highest q_{ij}/c_{ij} for two adjacent arcs that intercepts a corner node will be compared and stores in temporary parameter, $tempv$. Finally in Step 3, when the arc is selected to move (based on highest q_{ij}/c_{ij} ratio), the algorithm will count the pickup weight, spare capacity and traversal cost; and then stored in their respective parameters (q_{new}, q_{balnew} and c_{new}).

Similar to the start-up move, neighbourhood move uses the start-up algorithms in Step 2 and Step 3 by extends it to two or three ratio comparison of arcs. This modification by means to escape the algorithm from start-up vertex corner and begins the search for all remaining nodes in the region graph. The algorithm builds one route at a time. In each iteration, the current route is extended by adding the most promising edge according to highest q_{ij}/c_{ij}. This move behaves likely similar to path-scanning which is the most commonly used greedy heuristic approach to CARP. In each iteration, the algorithm scans the remaining unserviced edges to determine which edge should be visited next. Then the feasible edge will be added if its demand satisfies the remaining spare capacity q_{balnew}. The modification of neighbourhood move algorithm is drawn as follows.

```
for $G = (V, E)\backslash\{V_c = y_{ijk} = 1\}$ repeat the following.
        if $E_{1'}, E_2 \in V$ then select unserviced route $E_2$.
        if $E_{1'}, E_2, E_3 \in V$ then compute $q_{mnk}/c_{mnk} \in E_2$ and $q_{stk}/c_{stk} \in E_3$.
        if $E_{1'}, E_2, E_3, E_4 \in V$ then compute $q_{mnk}/c_{mnk} \in E_2$, $q_{stk}/c_{stk} \in E_3$ and $q_{uvk}/c_{uvk} \in E_4$.
        Assign $val_{mnk} := q_{mnk}/c_{mnk}$, $val_{stk} := q_{stk}/c_{stk}$ and $val_{uvk} := q_{uvk}/c_{uvk}$.
        Rank $val_{mnk}$, $val_{stk}$ and $val_{uvk}$ in ascending order.
        Choose the first value in ranking. Count $y = y_{init} + 1$.
        Count new quantity, $q_{new} := q_{new} + q_{ij}$ and spare capacity, $q_{balnew} := W - q_{new}$.
        Count new cost, $c_{new} := c_{new} + c_{ij}$.
Mark the covered $x_{ijk}$ until $s_l \le q_{new} \le s_u$.
```

Note that the above algorithm stops when $s_l \le q_{new} \le s_u$ is TRUE. Next, we apply switching rule where two pre-capacity buffers, namely lower capacity s_l and upper capacity s_u are imposed to utilize the pickup quantity and ignoring the edge cost. This approach is a modification from [14], where least insertion cost was applied when the collected quantity surpass or equal to a critical value. However in this work, the algorithm selects the highest quantity of candidate edges to move on rather than their costs. We set $s_l = 0.7W$ and $s_u = 0.8W$ respectively. We illustrate this modification in the following example.

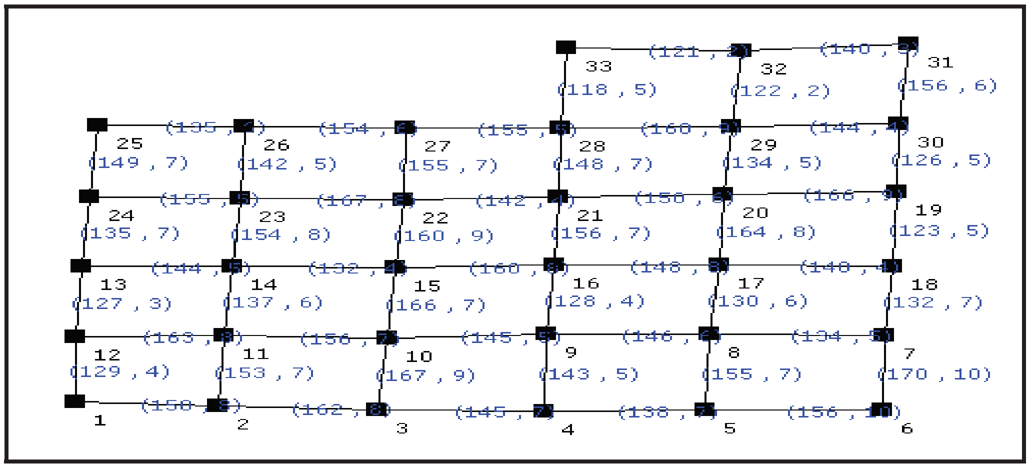


FIGURE 2. An example of CARP network.

Consider FIGURE 2 as a CARP network graph. Let say the vehicle at node 15; and the lower and upper buffer pre-capacity, $s_l = 420$ kg and $s_u = 480$ kg respectively. Next, let say the initial carried quantity $q = 430$ kg (which is $s_l \le q \le s_u$) and maximum capacity, $W = 600$ kg. At this point, the spare capacity left is only 170 kg. The options at node 15 are shown in TABLE 1. Arc $X(15, 10)$ is chosen with aim to maximize the collected demand without exceeding truck's capacity W which is $q = 596$ kg (less than W). Instead of choosing arc $X(15, 14)$ due to its highest demand/cost ratio, the capacity is not fully utilize since we get the total loading quantity, $q = 562$ kg only.

TABLE 1. Options at node 15.

Node, *n*	Edge, *X(i, j)*	Demand, q_{ij}	Cost, c_{ij}	q_{ij}/c_{ij}
15	***X(15, 10)***	***166***	7	23.7
	X(15, 14)	132	4	33.0
	X(15, 16)	160	8	20.0
	X(15, 22)	160	9	17.8

COMPUTATIONAL RESULTS

In this section, the computational experiments are divided into two sub-sections: real-life and generated instances. The program is coded in C# language and run on notebook CeleronM 1.73GHz processor with 504MB RAM

Real-life Instances

We conducted a real-life study onto four vehicles involved in solid waste collection. The observation was done onto 15 residential areas in Johor Bahru, the capital city of Johor, Malaysia. The road networks are modeled as undirected CARP, meanwhile the customers' demands followed Uniform distribution [15]. The computational results are shown in TABLE 2 and the explanations are given as follows. For column 1, instances 1-4 are considered as moderate network. Meanwhile instances 5-8 are considered small and 9-15 are large CARP. Note again, in CARP we consider the number of edges as our customers to be serviced, not the number of nodes. Column 2 and 3 initiates the number of nodes and edges respectively. Column 4 lists four different vehicles used in waste collection with capacity W_1 = 5000 kg, W_2 = 5100 kg, W_3 = 9000 kg and W_4 = 10000 kg. Column 5 summarizes the total least cost which is the objective solutions of our problem model. Column 6 displays the actual cost gained by the company. Finally column 7 depicts the quality of the algorithms in term of running time (in millisecond or in second as stated otherwise).

TABLE 2. Computational results on 15 real instances.

Instances	Nodes, *n*	Edges, *E*	Capacity, *W*	Total cost, $c_{glocost}$	Actual cost	CPU time (ms)
1	33	54	5100	480	496	7
2	33	54	5000	480	496	24
3	33	54	9000	409	430	8
4	33	54	10000	409	420	9
5	25	38	5100	390	435	36
6	25	38	5000	390	435	37
7	25	38	9000	390	400	31
8	25	38	10000	390	410	31
9	40	66	5000	648	680	34
10	40	66	9000	648	720	39
11	55	95	5000	901	935	1065 s
12	55	95	9000	901	945	1065 s
13	68	120	5000	1106	1200	3120 s
14	68	120	9000	1106	1260	3180 s
15	68	120	10000	1106	1278	3360 s

Secondary Instances

In this study, secondary instances were obtained from a benchmark dataset of *gdb* files created by [3], defined on artificial graphs. This dataset contains 23 instances *(gdb1-gdb23)* where 10 is the smallest number of nodes and 27 is the largest. In addition, all edges are required to be serviced, where 11 is the minimum and 55 is the maximum. In this file, all vertices are connected to the depot node in instance *gdb14*, *gdb15*, *gdb16*, *gdb17*, *gdb18*, *gdb22* and *gdb23*. Moreover, *gdb18*, *gdb22* and *gdb23* are defined on a complete connected graph; which means all vertices are connected to each other. The complete dataset and its explanation are available on the web (http://www.uv.es/~belengue/carp.html). We summarize our results in the following TABLE 3.

TABLE 3. Computational results for 23 benchmark instances of *gdb* files.

Instances	Vertices	Edges	Vehicles	Capacity	Total cost	IS	CPU time (ms)
gdb1	12	22	5	5	252	392	2
gdb2	12	26	6	5	291	395	2
gdb3	12	22	5	5	233	310	2
gdb4	11	19	4	5	238	320	1
gdb5	13	26	6	5	316	475	8
gdb6	12	22	5	5	260	354	1
gdb7	12	22	5	5	262	355	2
gdb8	27	46	10	27	210	423	21
gdb9	27	51	10	27	219	369	16
gdb10	12	25	4	10	252	302	1
gdb11	22	45	5	50	356	451	11
gdb12	13	23	7	35	334	627	3
gdb13	10	28	6	41	509	585	7
gdb14	7	21	5	21	96	106	2
gdb15	7	21	4	37	56	60	1
gdb16	8	28	5	24	119	144	2
gdb17	8	28	5	41	84	101	2
gdb18	9	36	5	37	158	178	5
gdb19	8	11	3	27	45	63	0
gdb20	11	22	4	27	105	134	1
gdb21	11	33	6	27	149	176	2
gdb22	11	44	8	27	191	211	5
gdb23	11	55	10	27	223	251	6

IS: Initial solutions obtained from our heuristics.

Next, we compare our initial solutions of heuristics with one recent work by Letchford and Oukil [16] that used the same instances of *gdb* files. They solved CARP by using column generation algorithms and produced two lower bounds (LB_1 and LB_2). The comparison is shown in the following TABLE 4.

TABLE 4. Result comparison with Letchford and Oukil [16].

Instances	LB_1	LB_2	IS	Total cost	Opt.	Dev. (%)
gdb1	282	285	392	252	316	24.1
gdb2	313	314	395	291	339	16.5
gdb3	248	250	310	233	275	12.7
gdb4	266	272	320	238	287	11.5
gdb5	358	359	475	316	377	26.0
gdb6	282	284	354	260	298	18.8
gdb7	288	293	355	262	325	9.2
gdb8	319	330	423	210	348	21.6
gdb9	291	294	369	219	303	21.8
gdb10	254	254	302	252	275	9.8
gdb11	364	364	451	356	395	14.2
gdb12	422	444	627	334	458	36.9
gdb13	525	525	585	509	536	9.1
gdb14	98	98	106	96	100	6.0
gdb15	56	56	60	56	58	3.4
gdb16	122	122	144	119	127	13.4
gdb17	85	85	101	84	91	11.0
gdb18	159	159	178	158	164	8.5
gdb19	47	55	63	45	55	14.5
gdb20	107	114	134	105	121	10.7
gdb21	151	151	176	149	156	12.8
gdb22	196	196	211	191	200	5.5
gdb23	233	233	251	223	233	7.7

LB_1: Lower bound with non-elementary column generation by Letchford and Oukil [16].
LB_2: Lower bound with elementary column generation by Letchford and Oukil [16].
Total cost: The sum of all servicing costs of all required edges.
Opt.: Optimal solution values known in literature.

Dev.: Deviation percentage measured by $\left| \frac{IS - Opt.}{Opt.} x100 \right|$.

ANALYSIS AND DISCUSSIONS

The result from real-life instances in TABLE 2 is described as follows. In term of computational time, our heuristics algorithms produced very fast results (less than one second) for intermediate and large networks. However, this running time increased exponentially when tested to very large customers. We conjecture that the algorithms trapped at one point in the search cycles or too many deadheading paths occurred in neighborhood moves. This phenomenon is currently being investigated. The total costs produced are considerably acceptable and reasonable when compared to actual operational costs by the company.

For secondary instances, our algorithms are capable to produce very competitive results when compared to [16] and best known optimum. Note that in this entire work, purposely we build initial solution of heuristics algorithms that are capable to produce near optimum, not the best optimum. Overall, the algorithms produced less than 10% deviation in eight instances, which is considerably very close to optimum. We also obtained ten instances below 20% deviation, which is also close to optimum. Four instances are below 30% close to optimum; meanwhile only one instance *(gdb12)* is quite far. On average, when measured on 23 *gdb* established benchmark instances, our algorithms are approximately 14% close to optimum. This is a big improvement of Belenguer *et al.* [12] whereby they can achieve only 23% (in average) of solutions cost for the mixed CARP.

CONCLUSIONS

In this paper, we presented the algorithms of various heuristics to determine the initial solution for the capacitated arc routing problem. In general, we developed these heuristics to generate the initial and feasible solution by satisfying all necessary constraints in the mathematical model of undirected CARP. Mainly, NPHDC is the central engine in our entire heuristics algorithms. In general, our NPHDC heuristics algorithms are capable to generate a start-up and neighborhood moves in various CARP networks. NPHDC works in loading function that finds the first node to begin the service and its successor edge. We also imposed switching rule to fully utilize the loading function when the collected loads are nearly full.

We tested these heuristics on real-life CARP in waste collection and a set of benchmark instances (*gdb* files). For real-life cases, the results are reasonably acceptable when compared to actual operating cost by the company. For benchmark instances, our algorithms seem competitive; which is 14% close to best optimum gained by other studies.

ACKNOWLEDGMENTS

The authors would like to thank Pusat Pengurusan Penyelidikan & Inovasi, Universiti Malaysia Perlis (UNIMAP) for their administration support. The paper is a contribution of Fundamental Research Grant Scheme (FRGS/1/2013/ST06/UNIMAP/02/1, No.: 9003-00359), awarded by the Ministry of Education (MOE), Malaysia.

REFERENCES

1. Golden, B. L. and Wong, R. T., *Networks* **11**, 305-315 (1981).
2. Hirabayashi, R., Saruwatari, Y. and Nishida, N., *Asia-Pacific Journal of Operational Research* **9**, 155-175 (1992).
3. Golden, B. L., DeArmon, J. S. and Baker, E. K., *Computers and Operations Research* **10**, 47-59 (1983).
4. Ulusoy, G., *European Journal of Operational Research* **22**, 329-337 (1985).
5. Eglese, R. W. and Li, L. Y. O., "A Tabu Search Based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline", in *Metaheuristics: Theory and Applications*, edited by Osman, I. H. and Kelly, J. P., Kluwer, 1996, pp. 633-650.
6. Lacomme, P., Prins, C. and Ramdane-Cherif, W., "A Genetic Algorithm for the Capacitated Arc Routing Problem and its Extensions", in *Applications of Evolutionary Computing. Lecture Notes in Computer Science*, edited by Boers, E. J. W. et al., Springer, Berlin, 2001, pp. 473-483.
7. Lacomme, P., Prins, C. and Ramdane-Cherif, W., *Annals of Operations Research* **131**, 159-185 (2004).
8. Beullens, P., Muyldermans, L., Cattrysse, D. and Van Oudheusden, D., *European Journal of Operational Research* **147**, 629-643 (2003).
9. Gendreau, M., Hertz, A. and Laporte, G., *Management Science* **40**, 1276-1290 (1994).
10. Irnich, S., Funke, B. and Grünert, T., *Computers & Operations Research* **33**, 2405-2429 (2006).
11. Nanry, W. P. and Barnes, J. W., *Transportation Research Part B* **34**, 107-121 (2000).
12. Belenguer, J-M., Benavent, E., Lacomme, P. and Prins, C., *Computers & Operations Research* **33**, 3363–3383 (2006).
13. Zuhaimy, I. and M. Fadzli, Ramli, *American Journal of Applied Sciences* **8**, 382-392 (2011).
14. Amponsah, S. K. and Salhi, S., *Waste Management* **24**, 711–721 (2004).
15. Zuhaimy, I. and M. Fadzli, Ramli, "Modelling Two Variances of Capacitated Arc Routing Problem Based on Rainy Weather Condition in Solid Waste Collection", in *Proceeding 4th International Conference on Mathematics and Statistics (ICoMS 2009)*, 13-15 August 2009, Universitas Malahayati, Bandar Lampung, Indonesia.
16. Letchford, A. N. and Oukil, A., *Computers & Operations Research* **36**, 2320-2327 (2009).