2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Benchmark Generator for Dynamic Capacitated Arc Routing Problems

Min Liu, Hemant Kumar Singh and Tapabrata Ray

***Abstract*— Capacitated arc routing problems (CARPs) are usually modeled as static problems, where information is known in advance and assumed to remain constant during the course of optimization. However, in practice, many factors such as demand, road accessibility, vehicle availability etc. change during the course of a mission and the route of each vehicle must be reconfigured dynamically. This problem is referred to as *dynamic capacitated arc routing problem* (DCARP) and there have been limited attempts to solve such problems in the past. Lack of standard DCARP benchmarks is one of the key factors limiting research in this direction. This paper introduces a benchmark generator for DCARPs considering interruptions/changes that are likely to occur in realistic scenarios. These benchmarks can be used to evaluate the strengths and the weaknesses of various optimization algorithms attempting to solve realistic DCARP problems.**

I. Introduction

The capacitated arc routing problem (CARP) is a challenging combinatorial problem, wherein a set of required arcs, each with a fixed demand, are served by a fleet of homogeneous vehicles of finite capacity while attempting to keep the total distance traveled to a minimum. However, in most real-life applications (winter gritting, street sweeping, inspection of pipe distribution networks etc.), certain unexpected events invariably happen while the vehicles are en route. Examples of such unpredicted events include change in tasks (i.e., new customer requests/cancellations), unexpected road blocks, traffic congestion and vehicle breakdowns. Under such situations, the originally planned vehicle routes need to be reconfigured, which translates to a dynamic optimization problem.

A. Review of Dynamic Benchmark Generators

In the context of dynamic optimization, few methods have been reported so far for constructing benchmarks. In [1]–[3], the environment parameters were switched over time to create a series of stationary problems. Multiobjective optimization concepts with dynamically changing objective weights were used to construct dynamic optimization test problems in [4]. There are also reports of methods which altered the multidimensional landscapes using shift and severity control schemes [5]–[8]. In 2008, a generalized dynamic benchmark generator (GDBG) was introduced in [9], [10], in which the environmental properties were altered to construct dynamic instances. Although the dynamic instances introduced in the above studies were meant to objectively evaluate the performance of optimization algorithms, they may not be adequate to construct DCARP instances as they have little or no control over practical features like mixed graphs, distribution of multiple lanes, traffic congestion, roadblocks or vehicle breakdowns. Changing environmental properties to construct dynamic instances in [9] [10] may be too simplistic to represent real life problems, as [10] only considered the dimensional changes (i.e., adding or removing variables from the optimization problem) and [9] only considered the non-dimensional changes (i.e., change in the value of variables). A generator that considers both dimensional and non-dimensional changes is required for modeling realistic DCARP. The motivation of the work presented here stems from the shortcomings described above.

Min Liu, Hemant Kumar Singh and Tapabrata Ray are with the School of Engineering and Information Technology, University of New South Wales, Canberra, Australia (email: min.liu@student.adfa.edu.au, {h.singh,t.ray}@adfa.edu.au).

B. Review of DCARP

There have been limited studies reported on modeling and solving DCARP. With the availability of advanced global positioning systems, the area is likely to gain greater attention in coming years. There are two principal forms of DCARP, a) *deterministic* and b) *stochastic*. In the deterministic form, there is no historical data available, and vehicle routes are generated based on current requirements. Part or all of the information (such as new tasks, vehicle availability etc.) is unknown and revealed dynamically en route. In the stochastic form, useful historical data about possible tasks, vehicle availability, roads maintenance etc. is available in order to schedule the vehicle routes. After initial scheduling, in both these forms, the vehicle routes are redefined continuously based on directions from a central decision maker.

The limited studies reported in solving DCARP include solving (deterministic) winter gritting problem by Tagmouti *et al.* [11]–[13]. In their study, three synthetic instances with 36, 76, and 162 tasks were used. To model the change in the problem over time, time-dependent service costs (demands) were assigned to each arc based on weather updates.

In order to develop a benchmark generator for DCARP, two fundamental capabilities need to be incorporated. The first is the generation of scalable, realistic road networks (initial/base graph), and the second is the classification of possible *interruptions/changes* and means to model them.

To overcome the first issue, it is theoretically possible to obtain road network details for specific region (e.g., from Google Maps). The existing instances (gdb [14], $kahs$ [15], val [16], egl [17]–[19], C, E of Beullens $et\ al.$ [20] and Set of Brandao and Eglese [21]) are all generated from real life networks of various sizes. However, they may represent isolated specific instances, whereas to thoroughly

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

test the performance of various CARP/DCARP algorithms, the user should be able to manipulate the properties of graphs to create customized instances. For example, in a waste collection problem, the waste pick-up sites must be distributed according to the distribution of waste amount (demand), which in turn is dependent on the density of population. The streets in the residential area may include one-way, two-way and multiple-lane roads. There could be one or more depots for dumping wastes. In the context of winter gritting, the service priority or demand distribution is closely dependent on the road surface temperatures [22] or current weather conditions [11]–[13]. For modeling such problems, the benchmark generator must be able to allow control of these characteristics (demand, directedness, task distribution etc.).

With regard to the second issue, there is a need for classification of interruptions/changes to represent realistic situations encountered in DCARP. Only varying demands have been considered in [11]–[13]. However, there are other additional factors that could possibly change over time, such as vehicle availability, road inaccessibility, addition/deletion of tasks and traffic congestion. This study aims to develop DCARP instances considering all these factors.

The remaining sections of this paper are organized as follows. Mathematical formulation of DCARP is presented in Section II. In Section III, the benchmark generator is described in detail and three DCARP benchmarks are generated using the proposed approach. A summary of presented work is given in Section IV.

II. Mathematical Formulation

A static CARP can be mathematically modeled as an undirected graph $G = (V, E)$, where V is a set of n nodes ($V = v_1, v_2, \ldots, v_n$), and E is a set of m edges ($E = e_1, e_2, \ldots, e_m$). A set of p tasks (edges) $Y = (y_1, \ldots, y_p)$ ($Y \in E$) needs to be served by a fleet of homogeneous vehicles with capacity C located at a depot s. Each task is associated with a distance $d(k) > 0 (k = 1, 2, \ldots, p)$ and a demand $q(k) \geq 0 (k = 1, 2, \ldots, p)$. The goal is to find a feasible set of trips with the minimum total traveled distance D, such that each vehicle trip starts and ends at the depot s, each required edge is serviced by one single trip, and the total demand handled by any vehicle must not exceed its capacity C.

In a DCARP, a planner is not aware of future events. The vehicle availability, the number/demand of tasks and road accessibility may change during the course of the mission. If the vehicles are unavailable due to breakdowns, the control center needs to re-assign other vehicle(s) to complete the remaining tasks. When some tasks are added or canceled, there is a need for rescheduling based on the position and availability of the vehicles. If certain roads are inaccessible (weather hazards) or travel times change due to traffic congestion, the control center needs to generate a new schedule to serve the tasks. The unexpected interruptions/changes encountered in a DCARP can be dimensional and/or non-dimensional changes. Dimensional changes include unavailability of vehicles, new customer requests, cancellation of planned tasks and inaccessible roads, while non-dimensional changes include traffic congestion and change in the demands for some tasks. More than one interruption/change may simultaneously occur at a time instant.

These unexpected events need to be modeled, and such a problem can be considered as a series of static CARPs. The occurrence of any interruption/change(s) results in a new instance. From the initial static CARP I_0 at time t_0, a series of instances I_G are generated at time t_G ($G = 1, 2, \ldots, G_{max}$), where G_{max} is the number of times the instance is updated.

III. Benchmark Generation

The proposed DCARP benchmark generator has two basic stages. First is the generation of a specific (base) CARP instance (I_0), and second is the generation of dynamically changing instances (I_1, I_2, I_3, …, $I_{G_{max}}$) from I_0. The process of modeling is shown in Figure 1. Stage 1 requires a set of 20 parameters ('Inputs 1'). It starts by generating a regular graph using random regular graph generator (RRGG) and then applying two operators (advanced graph generator (AGG) and repair operator (RO)) to incorporate desired problem specific characteristics based on Inputs 1. The graph generated from Stage 1 is then given as an input to Stage 2, along with a set of 9 parameters ('Inputs 2'). Subsequently, in Stage 2, instance modifying operator (IMO) acts on the basis instance I_0 to create modified instances I_1, I_2, I_3, … etc. The steps and the operators are detailed in the following subsections.

A. Stage 1: Creating basis instance

As evident, Stage 1 is designed to create a typical (static) base network (graph) for a given application. The inputs for this process control the character of the network (undirected, directed, mixed), distribution of multiple lanes, distance of the roads and their associated demands, and degree of nodes. In this case, degree of a node is defined as the number of connections terminating in that node, with multiple edges between two nodes counted as one connection[1]. The inputs for the process are listed in Table I. N_N denotes the number of nodes, which roughly translates to the size of the network, served by N_V vehicles starting from depot s. Based of the application, the graph could be undirected (e.g. pipeline inspection), directed (e.g. garbage collection) or mixed (e.g. winter gritting). For a mixed graph, the percentage of directed edges is prescribed using P_C (P_C percentage of undirected edges are replaced with two right-about directed edges). Subsequently, O_W percent of those directed edges are converted to one-way roads. The distribution of nodes' degree can be specified using parameters D_1, D_2, D_3, D_4, D_5 which correspond to percentages of nodes with degree 1, 2, 3, 4 and 5 respectively. Similarly, the percentage of roads with given number of lanes can be prescribed using parameters

[1]This is different from definition of degree used in graph theory, in which multiple edges between two nodes are counted separately.

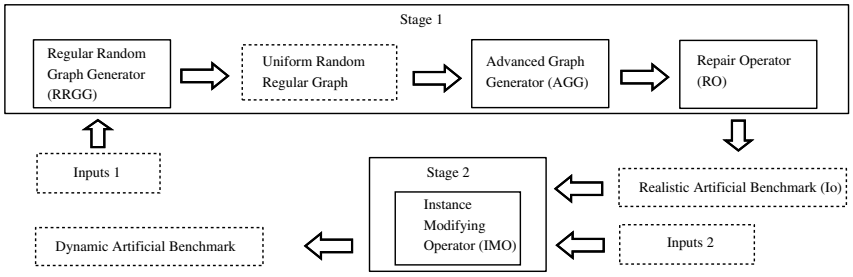


Fig. 1. DCARP benchmark generation

ML_1,ML_2,ML_3 and ML_4, and their distances using $Dist_s$, $Dist_m$, $Dist_l$. $Task_p$ denotes the percentage of total edges which need to be served. The parameter demand factor (D_F) determines the demand for a given task. This parameter has two settings. One is for the case where the demand is directly proportional to the road length (e.g. winter gritting). The demand for an edge (designated to be serviced) is calculated as D_F $\times$ edge length (road distance). The other setting corresponds to a case where the demand is proportional to the density of population. This is typically a case in a city, where central regions have shorter road distances, but are highly populated. In this setting, demand is calculated as D_F $\times$ $d_{longest}$(longest distance)/ d(road distance). Depending on specific application, other demand distribution can be used. The pseudo-code of Stage 1 is given in Algorithm 1 and the steps are detailed in following subsections.

TABLE I
LIST OF INPUTS FOR STAGE 1

Input	Value Range
No. of nodes (N_N)	An integer
No. of vehicles (N_V)	An integer
Character of graph (T)	u/d/m*
Depot of the fleet (s)	any node
% of directed edges (for mixed graph) (P_C)	[0, 100]
% of one-way roads (for directed/mixed graph) (O_W)	[0, 100]
% of nodes with degree 1 (D_1)	[0, 100]
% of nodes with degree 2 (D_2)	[0, 100]
% of nodes with degree 3 (D_3)	[0, 100]
% of nodes with degree 4 (D_4)	[0, 100]
% of nodes with degree 5 (D_5)	[0, 100]
% of edges with 1 lane (ML_1)	[0, 100]
% of edges with 2 lanes (ML_2)	[0, 100]
% of edges with 3 lanes (ML_3)	[0, 100]
% of edges with 4 lanes (ML_4)	[0, 100]
% of edges with distance between 100m-250m ($Dist_s$)	[0, 100]
% of edges with distance between 250m-1000m ($Dist_m$)	[0, 100]
% of edges with distance between 1000m-4000m ($Dist_l$)	[0, 100]
% of task ($Task_p$)	[0, 100]
Demand factor (D_F)	+ve real no.
if setting 1 (S1), demand = D_F $\times$ road distance	
if setting 2 (S2), demand = D_F $\times$ $d_{longest}$(longest distance)/ d(road distance)	

Notes: sum (D_1, D_2, D_3, D_4, D_5)= 100.
sum (ML_1, ML_2, ML_3, ML_4)= 100.
sum ($Dist_s$, $Dist_m$, $Dist_l$) =100.
*$u \equiv$ undirected,$d \equiv$ directed, $m \equiv$ mixed

Three operators are used in sequence in Stage 1 to create a realistic, static benchmark instance. These are described below, and their functionality is illustrated using a 10-node graph example, with parameter values set as $N_N = 10$, $T =$ undirected, $P_C = 0$, $O_W = 0$, $N_V = 2$, $s = 1$, $D_1 = 30$, $D_2 = 20$, $D_3 = 10$, $D_4 = 10$, $D_5 = 30$, $ML_1 = 50$, $ML_2 = 43$, $ML_3 = 3$, $ML_4 = 4$, $Dist_s = 80$, $Dist_m = 15$, $Dist_l = 5$, $Task_p = 35$, $D_F = 0.5$ (S1).

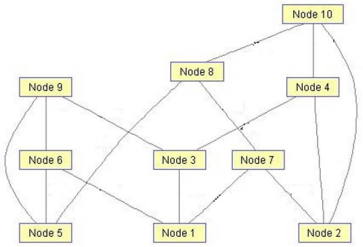


Fig. 2. 3-Regular graph generated using RRGG

1) RRGG: Regular random graph generator (RRGG) creates a uniform *3-regular* (each node is adjacent to 3 edges) undirected graph with the given number of the nodes and a random road distance between 100 to 4000. The graph for the 10-node example is shown in Figure 2. The reason why *3-regular* undirected graph is generated as the first level output is in the real life situations majority of the nodes are of degree 3 (T-junctions) and 4 (crossings). Further, it is more straightforward to get a connected graph with mixed degrees (1, 2, 3, 4, 5) starting from a 3-regular graph (maximum change in degree is 2) compared to 4-regular (maximum change in degree is 3).

2) AGG: The undirected 3-regular graph is then modified through the advanced graph generator (AGG) which first transforms the regular graph to a graph containing different nodes' degrees based on the required set of D_1, D_2, D_3, D_4 and D_5 and then transforms the updated graph to a directed or mixed graph containing one-way roads according to the input information T, P_C and O_W. The process of generating the desired distribution of nodes' degrees, starting from a 3-regular graph for a 10-node case is described below.

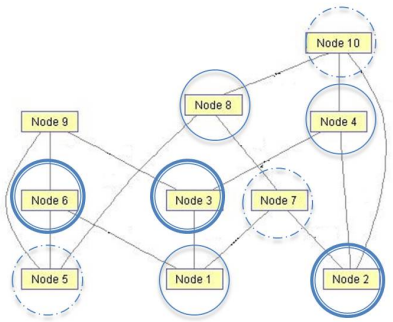
- **Generating degree-5 nodes:** First, the possible sets of 3 *disjoint* nodes are identified. *Disjoint* here means that there is no common edge between the given nodes. For the case shown in Figure 3(a), there are three such sets possible, which are {6, 2, 3}, {1, 8, 4} and {5, 10, 7}. If

Algorithm 1 Pseudo code of Stage 1

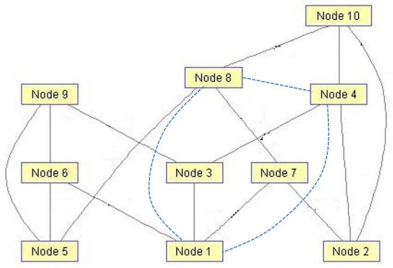
Require: N_N (even number), T, P_C, O_W, N_V, s, D_1, D_2, D_3, D_4, D_5, ML_1, ML_2, ML_3, ML_4, $Dist_s$, $Dist_m$, $Dist_l$, $Task_p$, D_F (for details of these symbols refer to Table I)
//RRGG Operator:
Initialize: generate a vector W (size $1\times 3N_N$) by repeating [1,2,..., N_N] three times;
A=sparse (N_N, N_N);
while isempty(W) ==0 **do**
  r_1 = rand[0,1], r_2 = rand[0,1];
  k_1 = round($r_1\times$ length(W)), k_2 = round($r_2\times$ length(W));
  Select two nodes, $v_1 \leftarrow W(k_1)$ and $v_2 \leftarrow W(k_2)$;
  if there are loops $v_1 = v_2$ or parallel edges $A(v_1, v_2) = 1$ **then**
    Re-select k_1 and k_2;
  else
    Add edges to graph, $A(v_1, v_2) = 1$ and $A(v_2, v_1) = 1$;
    Remove used edges (v_1, v_2) and (v_2, v_1) from W;
  end if
end while
//AGG Operator:
The depot of graph $\leftarrow$ s;
N_E = Number of edges;
Assign various degrees in terms of D_1, D_2, D_3, D_4 and D_5;
if the graph is a directed graph **then**
  Based on O_W value, randomly select round($N_E - N_E \times O_W \times$ 0.01) edges;
  Replace each edge with two right-about directed edges;
else if the graph is a mixed graph **then**
  Based on P_C value, randomly select round($N_E \times P_C \times$ 0.01) edges and save them into M;
  Based on O_W value, randomly select round(($N_E \times P_C \times 0.01) \times O_W \times$ 0.01) edges from M and save them into N;
  Replace each edge in N with one directed edge;
  Replace each edge in (M-N) with two right-about directed edges;
  Add a reverse edge for each of the selected edge into A;
end if
//RO Operator:
Generate the shortest distance DD_n between each pair of nodes;
for $j = 1 \rightarrow$ *Number of nodes* **do**
  if $0 \le DD_n(j,s) \le \infty || DD_n(s,j) = \infty$ **then**
    $path_{sj} \leftarrow$*reverse* $path_{js}$;
    Update DD_n;
  else if $0 \le DD_n(s,j) \le \infty || DD_n(j,s) = \infty$ **then**
    $path_{js} \leftarrow$*reverse* $path_{sj}$;
    Update DD_n;
  else if $DD_n(j,s) = \infty || DD_n(s,j) = \infty$ **then**
    Connect node j and s with an undirected edge;
    Update DD_n;
  end if
end for
Based on ML_1, ML_2, ML_3, ML_4, randomly select edges and add a parallel edge to each of them;
Based on $Dist_s$, $Dist_m$, $Dist_l$, randomly select edges and distribute distances to each of them;
Based on the percentage of tasks$Task_p$, randomly choose round($Task_p$ $\times$ (Total number of edges)) edges as tasks;
Use D_F to assign each task a demand;
Assign capacity of vehicles: ($\sum(demand)/N_V$ <capacity < ($\sum(demand)/N_V$)/N_V + $\sum(demand)/N_V$) and (capacity > max(demand))

N_{djn} = $\lfloor(N_N \times D_5/3)\rfloor$ is less then the number of sets identified (in this case 3), then N_{djn} sets are randomly chosen for assigning degree 5; otherwise all the sets are chosen. In this case, $N_{djn} = 1$, so 1 set ($\{1, 8, 4\}$) out of the possible three is randomly chosen. Thereafter, in each of the chosen sets, one edge is added between each pair of nodes, in order to raise their degree from 3 to 5. This process is shown in Figure 3. These sets of chosen nodes are thereafter excluded from the further steps (assignment of other degrees of nodes), and the remaining nodes are assigned to a set $RestNodes$.

- **Generating degree-1 nodes:** In *RestNodes*, possible sets of 3 *joint* nodes are identified. *Joint* here refers to a set of nodes that form a closed loop. In this example,



(a) Choosing sets of 3 disjoint nodes



(b) Graph after 5 degree distribution

Fig. 3. Process of generating nodes with degree 5

$\{6, 9, 5\}$ forms such a set. If $N_{jn} = \lfloor(N_N \times D_1/3)\rfloor$ is less than the number of such possible sets, N_{jn} sets are randomly chosen, otherwise all sets are chosen for degree 1 assignment. In this example, $N_{jn} = 1$, so the set $\{6, 9, 5\}$ is chosen. Among the chosen set(s), an edge is deleted between each of the nodes to reduce their degree to 1. This set of nodes is then removed from *RestNodes*, which now contains nodes $\{2, 3, 7, 10\}$ (Note: If the number of available sets is less than N_{jn}, then repeat the process using set(s) of 4 joint nodes instead).

- **Generating degree-4 nodes:** In the *RestNodes*, all possible sets of 2 disjoint nodes are identified. In this example, the possibilities are $\{2, 3\}$ and $\{7, 10\}$; If $N_{djn} = \lfloor(N_N \times D_4/2)\rfloor$ is less then the number of possible sets, then N_{djn} sets are randomly chosen, else all groups are chosen for degree 4 assignment. Thereafter, for each chosen set the nodes add an edge between the nodes to raise their degree to 4. For this particular case, N_{djn} = 0, so none of the sets are chosen (hence no nodes are generated with degree 4).
- **Generating degree-2 nodes:** In the *RestNodes* all possible sets of 2 joint nodes are identified. In this example, the only possibility is set $\{2, 7\}$. Thereafter, similar process as that for generating degree 1 nodes (described

above) is followed and the edge between the nodes is removed to generate degree 2 nodes.

Summarizing the above, if the degree needs to be raised, the process involves identifying disjoint nodes and adding edges between them, whereas if the degree needs to be reduced, joint nodes are identified and edges between them are deleted. After implementing the above process, the graph obtained with the required degree distribution is shown in Figure 4.

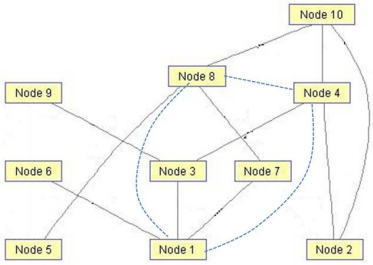


Fig. 4. Graph after assignment of degree distribution (AGG)

3) RO: After AGG operator, a repair operator (RO) is applied to the graph. The purpose of this operator is to maintain the connectedness of the network while maintaining (close to) desired properties of nodes' degrees. RO operates by connecting every task to the depot directly or indirectly by introducing an undirected edge or adding certain reverse edge within the shortest available path (between task and depot). In the example discussed here, node 1 is considered as the depot, which has a direct or indirect route to all other nodes, hence the graph remains unchanged after applying RO.

Finally, after RO operator, multiple lanes, distances (nearest integer), tasks and demands (nearest integer) and vehicle's capacity are assigned based on the specific input information to generate the base benchmark. When assigning the number of tasks, whether multiple lanes of an edge are considered as one task or multiple tasks depends on the specific real-life requirement and situation. The base graph obtained for the example is shown in Figure 5.

It is to be noted that in I_0, it may not be at times possible to generate *exact* number of required nodes with degree 1. The reason is that 3-regular graph may not contain 3 or 4 joint nodes. Hence, larger the D_1 value, more deviation from required degree distribution may be expected in the generated graphs. The percentage of O_W may also have an effect on the degree distribution. If O_W is set very high (most roads are one-way), then the graph may contain a large number of nodes which do not connect to the depot. In that case RO will add edges to create those connections which will alter the degree distribution of the graph.

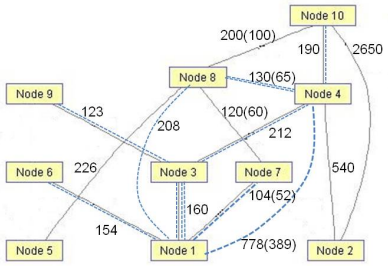


Fig. 5. The base instance (I_0) generated at Stage 1. The numbers along side the edges denote distance, and edges with numbers in brackets (denoting demand) are task edges.

B. Stage 2: Creating Dynamic Benchmarks

The instance generated from Stage 1 forms the basis instance I_0 at time t_0, and is given as an input for Stage 2 along with a set of parameters 'Inputs 2', listed in Table II. The parameter G_{max} defines the number of instances to be generated in series, starting from I_0. Each instance may incorporate one or more interruptions/changes from the previous instance. t_G defines the time between the instance updates. The parameters $w_1, w_2, \ldots, w_6$ indicate percent change to quantify six different types of interruptions/changes affecting the instance. Each of these parameters are a vector of length G_{max}, representing the % change at each time step. The parameter w_1 is used to prescribe the number of vehicles, and w_2 for number of roads that could possibly break down from one time step to other. The percentages w_1 and w_2 are always calculated based on the initial numbers (in I_0) of vehicles and roads respectively. w_3 denotes the percent of new added tasks at a time step. This percentage is calculated on the number existing tasks after applying w_2 changes. This is because some of the roads which broke down may have had tasks associated with them, which are automatically removed from the set of current tasks. Similarly, w_4 determines the number of tasks that are cancelled, which also is calculated based on current tasks. w_5 denotes the percent of existing roads whose (percieved) distance has changed due to congestion. In this model, the speed of vehicle is assumed constant, and hence distances have been manipulated (changed according to Gaussian distribution with current value as mean and 1% standard deviation) to reflect the affect of congestion. w_6 represents the percent of existing tasks for which the demands have changed. The demand is also assumed to change according to Gaussian distribution. The demand factor D_F is used to calculate demands for the added tasks. After calculating the percentage values (w_1, w_2, $\ldots$, w_6), the numbers are rounded to nearest integer for manipulating the instance. The pseudo-code for Stage 2 process is shown in Algorithm 2, and the description follows.

Instance Modifying Operator (IMO) is used to create

Algorithm 2 Pseudo code of Stage 2

Require: I_0, t_G, N_V, G_{max} , $w_1, w_2, \ldots, w_6$, D_F
  for $i = 1 \rightarrow G_{max}$ **do**
    $Vehicles \leftarrow$ Set of vehicles in I_0
    $N_{dv} \leftarrow$ round($N_V \times w_1(i) \times 0.01$)
    Delete N_{dv} randomly chosen vehicles from $Vehicles$
    //Update Edges:
    $Edges \leftarrow$ Set of all edges in I_0
    $N_E \leftarrow$ Number of edges in $Edges$
    $N_{de} \leftarrow$ round($N_E \times w_2(i) \times 0.01$)
    Delete N_{de} randomly chosen edges from $Edges$
    Update $Edges$, I_0, N_E
    //Update Tasks:
    $Tasks \leftarrow$ all tasks in I_0 ($Tasks \in Edges$)
    $NonTasks \leftarrow$ Edges without task in I_0 ($NonTasks \in Edges$)
    $N_t \leftarrow$ Number of tasks (edges) in $Tasks$
    $N_{nt} \leftarrow$ Number of edges in $NonTasks$
    $N_a \leftarrow$ round($N_t \times w_3(i) \times 0.01$)
    if $N_a < N_{nt}$ **then**
      Choose N_a edges from $NonTasks$
      Assign demands to the chosen edges using D_F
    else
      Assign demands to all edges in $NonTasks$ using D_F
    end if
    $N_{dt} \leftarrow$ round($N_t \times w_4(i) \times 0.01$)
    Delete N_{dt} randomly chosen tasks from $Tasks$
    Update $Tasks$, N_t
    //Update distances:
    N_{lc} = round($N_E \times w_5(i) \times 0.01$)
    Choose N_{lc} edges randomly from $Edges$
    Change the distances of chosen edges by values sampled from $\mathcal{N}(0, 0.01)$
    //Update demands:
    $N_{dc} \leftarrow$ round($N_t \times w_6(i) \times 0.01$);
    Choose N_{dc} tasks randomly from $Tasks$
    Change the demands of chosen tasks by values sampled from $\mathcal{N}(0, 0.01)$
  end for

TABLE II
LIST OF INPUTS FOR STAGE 2

Input	Value Range
No. of instances (G_{max})	An integer
Time duration for each interruption/change (t_G (mins))	+ve real no.
% of broken-down vehicles (w_1)	[0, 100]
% of broken-down roads (w_2)	[0, 100]
% of new added tasks (w_3)	[0, Max_{w_3}]*
% of cancelled tasks (w_4)	[0, 100]
% of roads changed distance (w_5)	[0, 100]
% of tasks changed demand (w_6)	[0, 100]
Demand factor (for new added tasks) (D_F)	+ve real no.
if setting 1 (S1), demand = D_F × road distance	
if setting 2 (S2), demand = D_F × $d_{longest}$(longest distance)/ d(road distance)	

*w_3 can go up to a value such that *all* edges become tasks

dynamic instances from the base graph I_0. The functioning of the operator is illustrated using the example used in the previous section. IMO uses the input parameters 'Inputs 2' (described in Table II) and applies the percent changes to the relevant attribute of the graph to create the new instance. In Table III, one such set of parameters is shown for creating two progressive instances from I_0.

1) Generation of I_1: Among the given parameters, t_G and w_1 indicate the time and the number of broken-down vehicles respectively, which do not affect the graph. However, they affect the instance since these parameters are required to completely define the problem. At t_1, the value of w_1 is 50%, which means 50% of N_V (the value rounds to 1 for this time step) number of vehicles are broken down. Also, the value of t_G dictates where the vehicles will be at the instant (while serving existing tasks), which will determine how they should be re-routed.

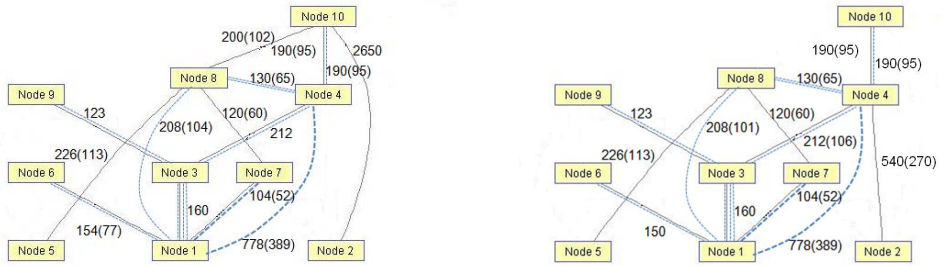
TABLE III
A SET OF INPUTS ('INPUTS 2') FOR STAGE 2

Parameter	Values (10 case)	Values (100 case)
G_{max}	2	2
t_G (mins)	{1, 3}	{4, 6}
w_1 (%)	{50, 20}	{10, 50}
w_2 (%)	{10, 15}	{10, 15}
w_3 (%)	{100, 120}	{5, 50}
w_4 (%)	{0, 0}	{5, 50}
w_5 (%)	{0, 10}	{5, 0}
w_6 (%)	{10, 10}	{10, 50}
D_F	{0.5 (S1), 0.5 (S1)}	{6 (S2), 6 (S2)} ($Instance_2$) {0.5 (S1), 0.5 (S1)} ($Instance_{1,3}$)

The percentage w_2 of broken-down roads for t_1 is set as 10%. I_0 has 14 roads, and hence *round* (10% of 14) = 1 road will break down (equivalently, removed from graph) at t_1. For this case, 1 random road (edge 2-4) is removed from the graph. The percentage w_3 of added tasks for t_1 is set as 100% which means there are *round* (100% of 5) = 5 new added tasks, e.g., edges 4-10, 4-10 (each lane of edge 4-10 is considered as a task here), 1-8, 1-6 and 5-8. The demands of new added tasks are assigned based on the D_F value. Due to low values of w_4 and w_5 (both will result in rounded value of zero), there are no deleted tasks and no change in road distance (congestion) at t_1. For the case of change in task demand, w_6 is 10%, which results in the change in demand of 1 task (edge 8-10) modified using Gaussian distribution as discussed earlier. The resultant isntance at t_1 is shown in Figure 6(a).

2) Generation of I_2: Following the similar process as for creating I_1, the values of change for various attributes are calculated. In this case, the number of broken-down vehicles comes out to be zero (20% of 2). Hence after this instant, both vehicles will be available to complete the tasks. The rounded values of broken-down roads, added tasks, deleted tasks, roads with changed distance and tasks with changed demands come out to be 2, 6, 0, 1 and 1 respectively. Given these numbers, two roads (edges 2-10 and 8-10) are removed from the graph I_0. It is to be noted here that the roads are removed from I_0 and not I_1. This implies that some (or all) roads which were broken/inaccessible in previous time step (edge 2-4 in this case) have become operable again. Thereafter, 6 tasks are added (at random) which are edges 1-8, 5-8, 4-10, 4-10, 3-4, 2-4, and no task is deleted. One road (edge 1-6) has changed distance from 154 to 150 at this instant, while 1 task (edge 1-8) has changed demand from 104 to 101. The resultant instance at t_2 is shown in Figure 6(b).

It is to be noted here that the selection of roads, vehicles, tasks etc. to be modified is done randomly using proportions ($w_1 - w_6$ parameters) in the example above. However, it is possible for the user to input exactly which of these (roads, vehicles, tasks etc.) he/she needs to change to generate specific instances, in which case these parameters will not be required. Hence, this model can be easily used to generate a specific dynamic benchmark which to meet the user's requirement.



(a) I_1 (b) I_2

Fig. 6. Dynamic instances generated during Stage 2

C. Example: 100-node benchmarks

In this section, three benchmarks are generated using the proposed benchmark generator: $Instance_1$ (undirected, representing problems like pipeline inspection), $Instance_2$ (directed, e.g., garbage collection) and $Instance_3$ (mixed, e.g., winter gritting) are briefly discussed. The parameters used for Stage 1 are listed in Table IV shown come after. Twenty independent instantiations were done for each of the base graphs $I0_1$, $I0_2$ and $I0_3$ in order to show the deviations in the output graphs from desired properties. The statistics are shown in Table V. It can be seen that on an average the properties of the generated graphs are close to the inputs prescribed. Slight deviations are caused due to parameters D_1 and O_W, as discussed earlier at the end of Section III-A.

TABLE IV
PARAMETERS INPUT OF EACH INSTANCE FOR STAGE 1

	$Instance_1$	$Instance_2$	$Instance_3$
N_N	100	100	100
N_G	1	1	1
T	undirected	directed	mixed
P_C	0	100	30
O_W	0	10	10
N_V	8	8	8
s	1	1	1
D_1	3	3	3
D_2	2	2	2
D_3	60	60	60
D_4	30	30	30
D_5	5	5	5
ML_1	50	50	50
ML_2	43	43	43
ML_3	3	3	3
ML_4	4	4	4
$Dist_s$	80	80	80
$Dist_m$	15	15	15
$Dist_l$	5	5	5
$Task_p$	10	10	10
D_F	0.5 (S1)	6 (S2)	0.5 (S1)

One of the random base graphs for each instance is chosen to represent $I0_1$, $I0_2$ and $I0_3$ at t_0 for further creating instances at t_1 and t_2. The parameters used for Stage 2 are listed in Table III. The properties of generated instances are shown in Table VI. As evident from the properties, w_2 (the number of broken-down roads) affects the connectedness and degree distribution of the graph (as some edges are deleted). Deleting the edges also *indirectly* affects the proportion of lanes (ML_1, ML_2, ML_3, ML_4) and distances ($Dist_s, Dist_m, Dist_l$). Based on *which* edges are deleted, this can have a significant effect on $D_1, D_2 \dots D_5$, as seen from changes in values for $Instance_1$ from t_0 to t_1. In real life situations, it is unlikely that w_2 will assume high values (as many roads broken-down at the same time is an unlikely occurrence). Changes in $Dist_s, Dist_m$ and $Dist_l$ largely depend on standard deviation used (in addition to w_5), which in this case is 1%, resulting in only slight variations in the road length distribution. The overall change in the number of tasks is determined by the number of broken-down roads with tasks associated with them, as well as the prescribed added/deleted tasks. The parameter w_1 affects the problem severity by changing the number of vehicles that are available to complete the tasks. In the given examples, the instances become progressively harder to solve with time as the number of vehicles decrease at both time steps (without proportional reduction in number of tasks).

IV. SUMMARY

DCARP is a challenging problem which is of great interest in both research and industry due to its applicability in many real world problems. In this study, a benchmark generator for DCARP is presented. The benchmark generator is capable of modeling interruptions/changes closely resembling those likely to be encountered in real life situations. Generic or very customized CARP and DCARP instances can be generated by manipulating the input parameters for the benchmark generator. The flexibility and control offered by the generator make it more suitable to create benchmarks in order to thoroughly test the DCARP optimization algorithms, compared to the limited set of specific instances currently available in the literature.

REFERENCES

[1] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in

TABLE V

BASE GRAPH STATISTICS FOR $Instance_1$, $Instance_2$ AND $Instance_3$

Output parameters	$I0_1$				$I0_2$				$I0_3$			
	Min.	Max.	Mean	Std.	Min.	Max.	Mean	Std.	Min.	Max.	Mean	Std.
Nodes	100	100	100	0	100	100	100	0	100	100	100	0
Roads	164	167	164.3	0.95	312	328	315.6	5.02	208	212	209.6	2.07
One-way roads	0	0	0	0	31	33	31.6	0.7	6	6	6	0
Vehicles	8	8	8	0	8	8	8	0	8	8	8	0
Capacity	183	767	318.9	176.9	333	450	392.1	34.1	319	940	511.9	167.5
Tasks	16	17	16.1	0.32	31	33	31.6	0.7	21	21	21	0
D_1	0	3	2.7	0.95	0	3	2.1	1.45	0	3	1.8	1.55
D_2	2	2	2	0	2	2	2	0	2	2	2	0
D_3	62	65	62.3	0.95	62	65	62.9	1.45	62	65	63.2	1.55
D_4	30	30	30	0	30	30	30	0	30	30	30	0
D_5	3	3	3	0	3	3	3	0	3	3	3	0
ML_1	50	50.3	50.03	0.09	50	50.16	50.05	0.08	50	50.24	50.02	0.08
ML_2	43.11	43.29	43.27	0.06	42.86	43.13	42.95	0.07	42.79	43.06	42.87	0.09
ML_3	2.99	3.05	3.04	0.02	2.86	3.15	2.93	0.12	2.83	2.88	2.86	0.02
ML_4	3.59	3.66	3.65	0.02	3.79	4.17	4.05	0.17	3.83	4.33	4.25	0.15
$Dist_s$	79.88	80.24	79.92	0.03	79.81	80.18	79.85	0.09	79.72	80.38	80.07	0.3
$Dist_m$	14.97	15.24	15.21	0.09	14.92	15.14	15.06	0.07	14.83	15.09	14.97	0.11
$Dist_l$	4.79	4.88	4.87	0.03	4.88	5.13	5.08	0.06	4.78	5.19	4.96	0.2

TABLE VI

DYNAMIC BENCHMARKS $Instance_1$, $Instance_2$ AND $Instance_3$

Output parameters	$Instance_1$			$Instance_2$			$Instance_3$		
	t_0	t_1	t_2	t_0	t_1	t_2	t_0	t_1	t_2
Nodes	100	100	99	100	100	100	100	100	100
Roads	164	148	139	328	295	279	208	187	177
One-way roads	0	0	0	33	27	31	6	6	4
Vehicles	8	7	4	8	7	4	8	7	4
Capacity	244	244	244	450	450	450	319	319	319
Tasks	16	15	13	33	25	28	21	19	21
D_1	3	6	6	3	3	3	3	6	4
D_2	2	18	26	2	4	4	2	12	20
D_3	62	52	52	62	62	63	62	58	53
D_4	30	22	13	30	28	28	30	24	22
D_5	3	2	3	3	3	2	3	0	1
ML_1	50	49.32	49.64	50	51.19	51.61	50	52.41	49.72
ML_2	43.29	43.24	43.88	42.99	41.69	40.5	42.79	41.18	41.48
ML_3	3.05	3.38	3.6	3.05	3.05	3.58	2.88	2.67	3.39
ML_4	3.66	4.05	2.88	3.96	4.07	4.3	4.33	3.74	5.08
$Dist_s$	79.88	79.73	82.01	80.18	80.34	81.72	80.29	80.21	80.23
$Dist_m$	15.24	14.86	15.11	14.94	14.58	14.34	14.9	14.97	16.38
$Dist_l$	4.88	5.41	2.88	4.88	5.08	3.94	4.81	4.81	3.39

Parallel Problem Solving from Nature–PPSN V. Springer, 1998, pp. 139–148.

[2] N. Mori, H. Kita, and Y. Nishikawa, "Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm," *Transactions of the Institute of Systems, Control and Information Engineers*, vol. 14, no. 1, pp. 33–41, 2001.

[3] K. P. Ng and K. C. Wong, "A new diploid scheme and dominance change mechanism for non-stationary function optimization," in *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1995, pp. 159–166.

[4] Y. Jin and B. Sendhoff, "Constructing dynamic optimization test problems using the multi-objective optimization concept," in *Applications of Evolutionary Computing*. Springer, 2004, pp. 525–536.

[5] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *IEEE Congress on Evolutionary Computation (CEC)*, vol. 3. IEEE, 1999.

[6] J. J. Grefenstette, "Evolvability in dynamic fitness landscapes: A genetic algorithm approach," in *IEEE Congress on Evolutionary Computation (CEC)*, vol. 3. IEEE, 1999.

[7] K. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *IEEE Congress on Evolutionary Computation (CEC)*, vol. 3. IEEE, 1999.

[8] R. W. Morrison and K. A. De Jong, "A test problem generator for non-stationary environments," in *IEEE Congress on Evolutionary Computation (CEC)*, vol. 3. IEEE, 1999.

[9] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Simulated Evolution and Learning*. Springer, 2008, pp. 391–400.

[10] C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, H. Beyer, and P. Suganthan, "Benchmark generator for CEC 2009 competition on dynamic optimization," University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep., 2008.

[11] M. Tagmouti, M. Gendreau, and J.-Y. Potvin, "A dynamic capacitated arc routing problem with time-dependent service costs," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 1, pp. 20–28, 2011.

[12] G. M. Tagmouti, M. and J. Potvin, "A Variable Neighborhood Descent Heuristic for Arc Routing Problems with Time Dependent Service Costs," *Computers & Industrial Engineering*, vol. 59, no. 4, pp. 954–963, 2010.

[13] M. Tagmouti, M. Gendreau, and J. Potvin, "Arc routing problems with time-dependent service costs," *European Journal of Operational Research*, vol. 181, no. 1, pp. 30–39, 2007.

[14] J. DeArmon, "A comparison of heuristics for the capacitated chinese postman problem," Master's thesis, University of Maryland, College Park, MD, 1981.

[15] M. Kiuchi, Y. Shinano, R. Hirabayashi, and Y. Saruwatari, "An exact algorithm for the capacitated arc routing problem using parallel branch and bound method," in *Spring National Conference of the Operational Research Society of Japan*, 1995, pp. 28–29.

[16] 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.

[17] R. Eglese, "Routeing winter gritting vehicles," *Discrete Applied Mathematics*, vol. 48, no. 3, pp. 231–244, 1994.

[18] R. W. Eglese and L. Y. Li, "A tabu search based heuristic for arc routing with a capacity constraint and time deadline," in *Meta-Heuristics*. Springer, 1996, pp. 633–649.

[19] L. Y. Li and R. W. Eglese, "An interactive algorithm for vehicle routeing for winter-gritting," *Journal of the Operational Research Society*, pp. 217–228, 1996.

[20] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden, "A guided local search heuristic for the capacitated arc routing problem," *European Journal of Operational Research*, vol. 147, no. 3, pp. 629–643, 2003.

[21] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1112–1126, 2008.

[22] H. Handa, D. Lin, L. Chapman, and X. Yao, "Robust solution of salting route optimisation using evolutionary algorithms," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2006, pp. 3098–3105.