This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

Estimation of Distribution Algorithm with Stochastic Local Search for Uncertain Capacitated Arc Routing Problems

Juan Wang, Ke Tang, *Senior Member, IEEE,* Jose A. Lozano, and Xin Yao, *Fellow, IEEE*

***Abstract*—The Uncertain Capacitated Arc Routing Problem (UCARP) is a challenging problem where the demands of tasks, the costs of edges, and the presence of tasks and edges are uncertain. The objective of this problem is to find a robust optimal solution for a finite set of possible scenarios. In this paper, we propose a novel robust optimization approach, called an Estimation of Distribution Algorithm with Stochastic Local Search (EDASLS), to tackle this problem. The proposed method integrates an estimation of distribution algorithm with a novel two phase stochastic local search procedure to minimize the maximal total cost over a set of different scenarios. The stochastic local search procedure avoids excessive fitness evaluations of unpromising moves in local search. Our experimental results on two sets of benchmark problems (a total of 55 problem instances) showed that the proposed approach outperformed existing state-of-the-art algorithms.**

***Index Terms*—Uncertain Capacitated Arc Routing Problem; Robust Optimization; Estimation of Distribution Algorithm; Local Search.**

I. Introduction

THE Capacitated Arc Routing Problem (CARP) [1] is a well-known combinatorial optimization problem. A CARP is defined over a directed graph $G = (V, E)$ with vertex set V and edge set E. Each edge $e \in E$ has a traversal cost $c(e)$ and a demand $d(e)$. Both $c(e)$ and $d(e)$ are non-negative. The edges with positive demand constitute the task set T, i.e., $T = \{e \in E | d(e) > 0\}$. A fleet of vehicles, each of which is with capacity Q and located in the depot v_0, are assigned to serve all the tasks. The aim of CARP is to determine a set of routes with minimal total costs and satisfying the following constraints: each route starts and ends at the depot; each task is served exactly once (but can be traveled more than once); the total load of a route, i.e., the total demand of tasks served by a route, cannot exceed the vehicle capacity Q.

CARP has been shown to be NP-hard [1]. Hence, exact methods are limited to small-scale instances and the mainstream approaches for solving CARP are heuristics and meta-heuristics. In the past few decades, in addition to some exact algorithms [2-4], a number of heuristics and meta-heuristics have been proposed. For instance, augment-merge [1], path-scanning [5], construct-strike [6] and route-first, cluster-second [7] are constructive heuristics that can produce fairly good solutions. More recently, variants of famous meta-heuristic methods, such as Simulated Annealing [8], guided local search [9] and memetic algorithms [10-12], have also been developed for CARP and showed even better performance in comparison to classical constructive heuristics. Moreover, research on CARP is not restricted to new algorithms, but has also been advanced in terms of problem formulations. Quite a few extensions of CARP, such as CARP with multiple depots [8], time windows [13] and alternative objective functions [14], have arisen from practical applications and have all been studied.

So far, most investigations on CARP (or extensions of CARP) are formulated as deterministic problems. However, real-world applications of CARP are usually stochastic. For instance, in urban waste collection, a typical real-world application of CARP, the amount of garbage on a street may change every day. In street salting, the amount of desired salt of a road changes with the road surface temperature [15]. Therefore, the stochastic version of CARP would have more practical value. In [16] and [17], the robustness of a solution to CARP with respect to stochastic demands was considered. In [18], a stochastic version of CARP, namely Stochastic CARP (SCARP) was approached directly and tackled with a memetic algorithm. In SCARP, demands of tasks are assumed to be random variables that follow normal distributions. The objective is to seek a robust solution that minimizes the expected total cost. Since then, SCARP has received some attention. In [19], the authors provided a mathematical analysis of SCARP and proposed a Hybrid Genetic Algorithm (HGA) to solve it. In [20], a bi-objective approach, which is an extension of Non-dominated Sorting Genetic Algorithm (NSGA-II), was proposed to optimize both the total cost and the duration of the longest trip for SCARP. In [21] , demands of tasks in SCARP are assumed to follow Poisson distribution and are revealed gradually as the vehicle progresses along the task. SCARP was formulated as a Set Partitioning Problem and a Branch-and-Price algorithm was proposed to solve it.

Although random variables in a robust optimization problem [22, 23] are usually assumed to follow specific distributions, this common practice might be inappropriate for CARP. In a real-world CARP, the distributions of the random variables (i.e., the demand on an edge) are usually unknown in advance, but can only be estimated using a number of different scenarios that

J. Wang, K. Tang (Corresponding Author) and X. Yao are with the USTC-Birmingham Joint Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (e-mail: jingze@mail.ustc.edu.cn; ketang@ustc.edu.cn).

J.A. Lozano is with the Intelligent Systems Group, Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, Paseo Manuel de Lardizabal, 1, 20018 Donostia, Giupzkoa, Spain (email: ja.lozano@ehu.eus).

X. Yao is also with the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (email: x.yao@cs.bham.ac.uk).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

are collected in the real world. Hence, solving a CARP with the estimated (and error-prone) distribution of random variables may not be meaningful. Alternatively, a more direct way is to seek a robust solution for a sample of deterministic scenarios. In this way, the underlying distributions of stochastic variables do not need to be known a priori. Based on this consideration, the Uncertain CARP (UCARP) problem is proposed in [24]. Compared with SCARP, UCARP considers variability in more parameters and does not assume predefined distributions for these parameters. Concretely, four random factors are considered in UCARP: demands of tasks, costs of edges, presence of tasks, and presence of edges. The first two indicate that values of costs and demands in UCARP are stochastic. The latter two mean that the task set and edge set are not fixed. These stochastic factors all correspond to uncertainties in practical situations. For instance, in salting route optimization problem [25], the salting demand and traversal cost of a street vary with the road surface temperature. As the temperature increases, some of the streets may not require salting, i.e., the tasks corresponding to these streets will disappear. Likewise, if a road becomes inaccessible due to construction or traffic jams, the corresponding edges will be absent. The primary aim of UCARP is to find a solution that works satisfactorily when the problem parameters change. Such a solution is called a robust solution [26]. It has been shown that UCARP has quite different characteristics from CARP and cannot be well addressed by simply adapting some existing CARP solver [24]. However, to the best of our knowledge, only one approach has been designed for UCARP so far [27].

In this article, a new approach to UCARP, namely Estimation of Distribution Algorithm with Stochastic Local Search procedure (EDASLS), is proposed to seek robust solutions over a set of deterministic CARP instances. The EDASLS is designed based on Edge Histogram Based Sampling Algorithm (EHBSA) [28], a variant of the Estimation of Distribution Algorithm (EDA) and is equipped with a novel Stochastic Local Search (SLS) procedure tailored to manage the stochastic characteristics of UCARP. Since a solution of UCARP can be represented as a permutation of tasks, UCARP is also in nature a permutation optimization problem. Tasks with tight correlation in the graph have a high probability to be adjacent in the final robust solution sequence. EHBSA fits this characteristic well because it generates new offspring by sampling a probabilistic model that learns the adjacency of elements in permutations. Therefore, an intuitive idea is that EHBSA can generate high-quality solutions to UCARP. However, the original EHBSA does not handle uncertainties explicitly and thus is not efficient for UCARP. Inspired by the idea of Memetic Algorithms (MAs), the SLS procedure is developed to address this challenge.

The remainder of this paper is organized as follows: Section II introduces the problem definition of UCARP and the corresponding notations. In Section III, details of EDASLS are presented. Section IV presents the computational experiments on two benchmark sets adapted from the UCARP literature. The results are compared to those obtained by two other state-of-the-art MAs. Finally, conclusions and future work is presented in Section V.

II. Problem Definition and Notations

Let $G(V,E)$ be a complete and undirected graph where V and E represent the sets of vertices and edges, respectively. Associated with each edge $e \in E$ there is a cost $c(e) > 0$ and a demand $d(e) \geq 0$. Both $c(e)$ and $d(e)$ are stochastic variables. The value of $c(e)$ could be infinite, which means that edge e is missing. The edges with positive demands form the task set T, i.e., $T = \{\tau \in E | d(\tau) > 0\}$. Note that tasks in T are not fixed. A task will turn into a non-required edge when its demand changes to 0. A fleet of vehicles with capacity Q and located at the depot $v_0 \in V$ is assigned to serve the task set. Similar to CARP, UCARP also involves three constraints: 1) each vehicle starts and ends at the depot; 2) each task is served by one and only one vehicle; 3) the total demand of the tasks served by each vehicle should not exceed the vehicle capacity. The objective of UCARP is to determine a set of vehicle routes that minimize the maximal total cost of the routes over a set of scenarios. That is, the UCARP in this article is a minimax optimization problem.

As mentioned in Section I, in this article, UCARP is approximated by a set of deterministic CARP instances. Concretely, an UCARP instance Θ is expressed as $\Theta = \{I_1, I_2, \dots, I_N\}$. For any $1 \leq j \leq N$, I_j is a CARP instance. All $I_j \in \Theta$ are related but different from each other. Intuitively, I_j can be regarded as a status of Θ in a specific scenario. The task sets of all the instances comprise of the *union task set* UT, i.e., $UT = \bigcup_{j=1}^{N} T_j$, where T_j is the task set of instance I_j. Each $\tau \in UT$ is associated with two task IDs which are integers within the range 1 to $2|UT|$. Each ID represents one direction of the task.

The main issue to be addressed in UCARP is that a planned feasible solution might be inappropriate (or infeasible) for some instances because of the stochastic nature of UCARP. Therefore a recourse strategy Φ, which guarantees the solution's feasibility and flexibility, is required. Particularly, Φ tackles the following three conditions that might occur when a solution is executed on an instance:

1) A travelled edge is missing;
2) An original task turns into a non-required edge;
3) A violation of vehicle capacity comes up.

For the first case, the missing edge is replaced by the shortest path connecting its two endpoints. For the second case, as a task becomes non-required, it is no longer necessary to serve it. The vehicle would ignore the "was-required" task and skip to the next task to continue service. For the last case, when the capacity of a vehicle is exceeded, which is commonly called a route failure in Stochastic Capacitated Vehicle Routing Problem (SCVRP) [29], the node counterpart of SCARP, the recourse strategy amends the routes by returning to the depot to get replenished before serving the next task. Since we are not aware of the distributions of demands on the tasks, it is intractable to determine where a route failure occurs if the actual demand of an edge is revealed only when the vehicle arrives. Hence, an assumption made in [30] for SCVRP is adopted in this work as well, that is, a task's actual demand is known before it is served. Under this assumption, when a task τ_i is served and the remaining capacity of the vehicle is not enough for the next task τ_{i+1}, we consider that a route failure

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

occurs right after τ_i is served. It is worth mentioning that after replenishment, the vehicle heads directly to serve τ_{i+1} instead of going back to the vertex where the route failure occurs (which is an endpoint of task τ_i).

A solution of UCARP can be represented by a set of routes $s = (R_1, \dots, R_m)$ that partition the task set UT into m routes $R_k = (\tau_{k,1}, \dots, \tau_{k,l_k})$, where $\tau_{k,t}$ and l_k denote the tth task and the number of tasks served in route R_k, respectively. The contiguous tasks are connected by implicit shortest paths which are not represented in the routes.

Let $head(\tau)$ and $tail(\tau)$ represent the two endpoints and $inv(\tau)$ the reverse direction of task τ, i.e., $head\big(inv(\tau)\big) = tail(\tau)$, $tail\big(inv(\tau)\big) = head(\tau)$, and vice versa, $d(\tau, I_j)$ represents the demand of τ in instance I_j, UCARP can be stated as follows:

$$\text{minimize } R(s) \tag{1}$$

$$\text{s.t. : } head\big(\tau_{k,1}\big) = tail\big(\tau_{k,l_k}\big) = v_0,\ \forall k = 1,2,\dots,m \tag{2}$$

$$\sum_{k=1}^{m} l_k = |UT| \tag{3}$$

$$\tau_{k_1,i_1} \neq \tau_{k_2,i_2},\ \forall (k_1,i_1) \neq (k_2,i_2) \tag{4}$$

$$\tau_{k_1,i_1} \neq inv(\tau_{k_2,i_2}),\ \forall (k_1,i_1) \neq (k_2,i_2) \tag{5}$$

$$\sum_{i=1}^{l_k} d\left(\tau_{k,i}, I_j\right) \leq Q,\quad \forall k = 1,2,\dots,m;\ \forall I_j \in \Theta \tag{6}$$

$$\tau_{k,i} \in UT \tag{7}$$

In constraints (4) and (5), the inequality $(k_1,i_1) \neq (k_2,i_2)$ means that the two equalities $k_1 = k_2$ and $i_1 = i_2$ do not hold simultaneously. Eq. (1) requires minimizing the maximal total cost $TC(s', I_j)$ for all $I_j \in \Theta$, i.e.,$R(s) = \max_{I_j \in \Theta} TC(s', I_j)$. In other words, the optimal solution for UCARP is $s^* = \arg\min_{s \in S}\{\max_{I_j} TC(s', I_j)\}$, where $s' = \Phi(s, I_j)$ denotes the modified solution obtained by the recourse strategy Φ when executing s on instance I_j and $TC(s', I_j)$ is the total cost of s' on instance I_j:

$$TC\big(s', I_j\big) = \sum_{k=1}^{m} C(R_k) \tag{8}$$

where $C(R_k)$ is the cost of route R_k:

$$C(R_k) = \sum_{i=1}^{l_k} \Big(c\big(\tau_{k,i}\big) + dis\big(tail(\tau_{k,i-1}), head(\tau_{k,i})\big)\Big) + dis\big(tail(\tau_{k,l_k}), v_0\big) \tag{9}$$

where $dis(a,b)$ is the cost of the shortest path from vertex a to vertex b. $tail(\tau_{k,i-1})$ is set to v_0 when $i = 1$.

Constraint (2) indicates the priority that each route starts and ends at the depot v_0. Constraints (3) to (5) make sure that all the tasks are served exactly once. Constraint (6) means that the total demand of each route should not exceed the capacity Q on any instance. Constraint (7) defines the domain of variables.

III. An EDA With Stochastic Local Search for UCARP

Most EDAs [31-33] focus on optimization problems based on integer or real domains. The EHBSA was proposed in 2002 for permutation optimization problems [28]. In each generation of EHBSA, an edge histogram matrix is calculated. Subsequently, new offspring are generated based on the matrix. By learning the adjacency of the vertices in the parent population, EHBSA can produce new individuals in which the adjacent elements would have a stronger correlation.

A UCARP can be transformed to a permutation optimization problem and solved by EHBSA. However, the basic and simple EHBSA does not have particularly significant advantage over other well-established EAs for deterministic problems, not to mention stochastic problems. To enhance the performance of EHBSA, a natural idea is hybridizing it with local search. However, because of the stochastic characteristics, conventional local search is inappropriate when applying to UCARP. In UCARP, an intermediate solution should be evaluated on all the instances, which is computationally costly. Thus, we propose a novel stochastic local search for UCARP and couple it with EHBSA, forming a new approach named EDA with Stochastic Local Search (EDASLS).

The pseudo-code of EDASLS is presented in Fig. 1. Several main ingredients are described in detail in Sections III.A-III.D: chromosome structure and evaluation, edge histogram based sampling method, local search procedure, population structure, initialization and replacement.

A. Chromosome structure and evaluation

Solving routing problems essentially involves determining a sequence of tasks (edges) or customers (nodes) with route delimiters partitioning the sequence into routes compatible with vehicle capacity. In 1983, Beasley [34] first introduced route-first cluster-second heuristics. This method first produces a giant route, also called chromosome, by temporarily ignoring the vehicle capacity. Then, the giant route is decomposed into feasible routes for vehicles. Subsequent to Beasley's work, a large amount of methods adopting the route-first split-second idea have been proposed for CARP [7, 10, 13, 18, 35, 36] as well as other routing problems [29, 37-41]. More related works can be found in a review [42]. Inspired by these literatures, our chromosome c is a permutation of $|UT|$ tasks (each task is represented by its task ID), without route delimiters, and with implicit shortest paths between consecutive tasks. We adopt this kind of chromosome representation because: (1) EHBSA can be directly applied to this encoding scheme; (2) an optimal solution can be extracted from a chromosome by the split procedure proposed in [7] for the deterministic CARP; (3) since our approach is developed on a set of CARPs, this chromosome is more convenient for evolution than a solution (routes with route delimiters). Without the capacity constraint, we do not need to frequently execute the recourse operator to repair a chromosome.

To evaluate the performance of a chromosome c, its fitness needs to be defined. In EDASLS, the fitness of c is defined as the maximum of its optimal (i.e., minimal) costs on all the instances and the lower fitness indicates the better performance:

$$F(c) = \max_{I_j \in \Theta} TC(s_{I_j}, I_j) \tag{10}$$

where s_{I_j} is the solution obtained by partitioning chromosome c on instance I_j using Ulusoy's splitting procedure [7].

Ulusoy's splitting procedure is an exact method that seeks the optimal way to split a chromosome into several feasible

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation


```
EDASLS(Θ, UT, p_ls, s_r)
Input: a set of instances Θ, the task set UT, the probability of local search p_ls
Output: A robust solution s_r
begin
  construct the initial population pop using heuristics and random generation.
  for each c_i ∈ pop do
    F(c_i) ← CEvaluate(c_i, Θ).
    update s_r
  end
  sort pop in non-descending order according to the fitness value.
  until stopping criterion is reached do
    develop the edge histogram matrix ehm based on the best half individuals of pop.
    randomly select an individual T[] from pop.
    c ← EHBSA(T[], n).
    Sample a random number r from the uniform distribution between 0 and 1.
    if r < p_ls then
      c' ← StochasticLS(c, ehm, UT, Θ, s_r);
      if c' is not a duplicate of any c_i ∈ pop then
        c ← c'
      end
    end
    CEvaluate(c, Θ).
    update s_r
    if c is better than T[] and c is not a duplicate of any c_i ∈ pop then
      T[] ← c
    end
    sort pop in non-descending order according to the fitness value.
  end
  return s_r.
end
```

Fig. 1. The pseudo-code of EDASLS(Θ, UT, p_{ls}, s_r).

routes. For a given chromosome c= $(\tau_1, \tau_2, \dots, \tau_{|UT|})$, it builds an auxiliary graph G^* with $(|UT|+1)$ vertices indexed from 0 to $|UT|$. Each arc (i,j) models a subsequence $(\tau_{i+1}, \tau_{i+2}, \dots, \tau_j)$ corresponding to a feasible route and is weighted by the route cost. The shortest path from vertex 0 to vertex t indicates an optimal partition of the chromosome c. Bellman's algorithm [43] can be used to compute the shortest path in $O(|UT|^2)$. Note that for the same chromosome, the solution s_{I_j} obtained by Ulusoy's splitting procedure on different I_j might be different, leading to different costs.

Fig. 2 shows the pseudo code of the procedure CEvaluate(c, Θ) which calculates the fitness value of chromosome c.

B. Edge histogram based sampling algorithm

This section briefly explains the EHBSA [28], which is used to generate offspring in EDASLS. EHBSA comprises of two phases: (1) modeling promising individuals, and (2) generating new individuals by sampling the model.

In each generation, an edge histogram matrix ehm for the promising individuals in the population is calculated. Let $p^* = \{c_1, c_2, \dots, c_l\}$ represent the l promising individuals. The elements in ehm are defined as:

$$ehm_{i,j} = \begin{cases} \sum_{k=1}^{l} (\delta_{i,j}(c_k) + \delta_{j,i}(c_k)) + \epsilon & if\ i \neq j \\ 0 & if\ i = j \end{cases} \quad (11)$$

where $\delta_{i,j}(c_k)$ is a delta function defined as:

$$\delta_{i,j}(c_k) = \begin{cases} 1 & if\ there\ is\ a\ subsequence (i,j)\ in\ c_k \\ 0 & otherwise \end{cases} \quad (12)$$

and ϵ is a bias to control the pressure in sampling elements and can be computed as:

$$\epsilon = \frac{2l}{L-1} B_{ratio} \quad (13)$$

where B_{ratio} is a bias ratio ($B_{ratio} > 0$) and L is the length of each permutation (chromosome). Intuitively, $ehm_{i,j}$ counts the number of adjacent element pairs (i,j) and (j,i) in the promising individuals plus a bias. Thus, the matrix ehm is symmetrical.

When applying EHBSA to UCARP, there exist some issues to be addressed. First, the elements in a permutation are required edges (tasks) rather than nodes. Since ehm learns the adjacency of consecutive elements, subsequence (i,j) is considered to be the same as $\big(inv(j), inv(i)\big)$ rather than (j,i). The reason is that in our permutation structure, the implicit shortest path between elements i and j is the same as that between $inv(j)$ and $inv(i)$ (both are from $tail(i)$ to $head(j)$ or in the opposite direction). Since the edges in the path are undirected, a path can be traveled along either direction). Moreover, a task should be served only once and can be served in either direction of an undirected arc. Hence, task τ and $inv(\tau)$ cannot appear in the same chromosome.

Based on the characteristics of UCARP described above, the definition of ehm matrix in our approach differs from (11):

$$ehm_{i,j} = \begin{cases} \sum_{k=1}^{l} (\delta_{i,j}(c_k) + \delta_{inv(j),inv(i)}(c_k)) + \epsilon\ if\ i \neq j \\ 0 \qquad\qquad if\ i = j \end{cases} \quad (14)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

CEvaluate(c,Θ)
Input: a set of instances Θ, a chromosome c
Output: the fitness of the chromosome $F(c)$
begin
  $F(c) \leftarrow 0$
  for each $I_j \in \Theta$ **do**
    $s_{I_j} \leftarrow$ partition c on I_j using Ulusoy's splitting procedure.
    if $TC\left(s_{I_j}, I_j\right) > F(c)$ **then**
      $F(c) \leftarrow TC\left(s_{I_j}, I_j\right)$
    end
  end
  return $F(c)$
end

Fig. 2. The pseudo-code of CEvaluate(c,Θ).

where $\delta_{i,j}(c_k)$ shares the same definition as that of the original EHBSA as defined in (12). ϵ is defined as:

$$\epsilon = \frac{l}{|UT|-1} B_{ratio} \quad (15)$$

With the edge histogram matrix ehm, EHBSA can generate a new chromosome (permutation) by sampling ehm. In [28], the author proposed two variants of EHBSA. One is without a template (EHBSA/WO) and the other with a template (EHBSA/WT). The latter is shown to have better performance [28] and thus is employed in this work.

To summarize, EHBSA works as follows: in each generation, a template chromosome $T[]$ is randomly chosen from the best half individuals of the current population and then randomly divided into n sub-segments. Subsequently, one of the sub-segments is randomly selected. The elements of the selected sub-segment are resampled one by one based on ehm. Fig. 3 shows the pseudo-code of the EHBSA employed in this work.

C. Stochastic local search procedure

A novel local search procedure, called Stochastic Local Search (SLS), is adopted with a predefined probability p_{ls} in EDASLS to improve the chromosome c obtained by EHBSA. Recall that in EDASLS, an individual is encoded as a sequence of task IDs. Hence, the local search is carried out with so-called move operators. In SLS, four traditional move operators are used:

Single insertion (u, v): Single insertion moves a task u after another task v. In UCARP, the edges, both non-required and required edges (tasks) are undirected. Therefore, both directions of u are considered when inserting it to another position. The direction leading to a better improvement is chosen.

Double insertion (u, v): *Two* consecutive tasks (u, w) are moved instead of a single task. Likewise, both directions (u, w) and $(inv(w), inv(u))$ are considered.

Swap (u, v)*:* Two candidate tasks u, v are selected and exchanged. Both directions of the tasks are considered.

2-opt (u, v): For a chromosome $c = (\tau_1, \tau_2, \dots, \tau_t)$, a sub-route $(\tau_u, \tau_{u+1} \dots, \tau_v)(u \leq v)$ is selected and reversed. That is, $(\tau_u, \tau_{u+1} \dots, \tau_v)$ is replaced by $(inv(\tau_v), \dots, inv(\tau_{u+1}), inv(\tau_u))$. Unlike the previous three operators, we do not need to consider the direction because the 2-opt operator itself is an operator changing the direction of tasks.

To determine whether an intermediate chromosome obtained by a move operator is better, its fitness should be evaluated. For deterministic problems such as CARP, this can be done in constant time. Because only the change in cost of a chromosome and (or) the capacity violation of demand caused by a move need to be calculated. In SLS, a perturbation to a chromosome would change the order of a subsequence of tasks, thus affecting the optimal solution obtained by Ulusoy's splitting procedure. Consequently, evaluating the impact of a move on the total cost can be rather time consuming. As defined in (10), to evaluate a chromosome's fitness, we need to partition it on all the instances and calculate the total costs of the corresponding solutions. That means, one fitness evaluation requires calling the splitting procedure N times (N is the number of instances as defined in Section II). As the time complexity of Ulusoy's splitting is $O(|UT|^2)$, it would take $O(N|UT|^2)$ to evaluate one intermediate chromosome. To scan all the $|UT|^2$ pairs of tasks, the time complexity would be $O(N|UT|^4)$. This is obviously quite time consuming compared with that of a local search procedure in CARP, which is $O(|UT|^2)$. To tackle this problem, the SLS procedure is developed to avoid expensive evaluations of unpromising moves.

Recall that the value of $ehm_{i,j}$ in the edge histogram matrix reflects the adjacency of task i and j in the promising individuals. Tasks τ_i and τ_j with a larger value of $ehm_{i,j}$ are more likely to be adjacent in the individuals with high performance. Intuitively, an improving local search move should increase the ehm value of adjacent tasks in the individual sequence. SLS is based on this idea. If a move reduces the adjacency of tasks, it will be considered to be an

EHBSA(*T[],ehm,n*)
Input: a template chromosome $T[]$, the edge histogram matrix ehm, number of sub segments n
Output: a child chromosome $C[]$
begin
  randomly divide the template individual $T[]$ into n sub segments $T_0[], \dots T_n[]$.
  randomly choose one sub segment $T_i[]$.
  $\Omega \leftarrow$ all the tasks in $T_i[]$.
  $ct \leftarrow$ the task before $T_i[0]$.
  for $p \leftarrow 0$ to $length(T_i[]) - 1$ **do**
    construct a roulette wheel vector $rw[]$ as $rw[k] = ehm_{ct,k}(k = 1,2, \dots 2t)$.
    for $k \leftarrow 1$ to $2t$ **do**
      if $k \notin \Omega$ && $inv(k) \notin \Omega$ **then**
        $rw[k] \leftarrow 0$
        $rw[inv(k)] \leftarrow 0$
      end
    end
    sample $T_i[p] = x$ with probability $rw[x] / \sum_{j=1}^{2t} rw[j]$.
    $ct \leftarrow T_i[p]$
    $p \leftarrow p + 1$
    $\Omega \leftarrow \Omega - \{ct\}$
  end
  $C[] \leftarrow T[]$
  return $C[]$
end

Fig. 3. The pseudo-code of our EHBSA(*T[],ehm,n*).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

un-improving move. Hence, it is unnecessary to evaluate the fitness of the intermediate chromosome obtained by this move.

The SLS procedure can be described as follows: Given a chromosome c generated by EHBSA, the SLS scans each pair of tasks (u, v) in $O(|UT|^2)$ time. The previously mentioned four move operators are separately applied to c. To avoid excessive computation, every move is pre-evaluated on the basis of the edge histogram matrix ehm in the first phase of SLS, which costs constant time. For instance, for the move *single insertion* (u, v), SLS first computes the value of $\Delta ehm = ehm_{i,j} + ehm_{v,u} + ehm_{ul} - ehm_{i,u} - ehm_{u,j} - ehm_{v,l}$, where i and j represent the tasks before and after u, respectively, and l is the task after v. If Δehm is positive, this move is considered to be promising and the fitness of the corresponding chromosome will be evaluated with Eq. (10) in the second phase. Otherwise, the corresponding chromosome will be discarded directly. The first improved chromosome produced by each move operator is preserved. By this means, we will obtain four improved intermediate chromosomes c_{si}, c_{di}, c_{swap} and c_{2-opt}. The one with optimal fitness will be chosen to replace the current chromosome c. This process repeats until the current chromosome can no longer be improved. Fig. 4 gives the pseudo-code of SLS.

D. Other components of EDASLS

Initialization: during the initialization phase, an initial population with $popsize$ chromosomes is constructed. Chromosomes in the initial population are generated either by heuristics or at random. A constructive heuristic called path-scanning (PS) [5] is executed to generate five solutions (task sequences with route delimiters) according to the five rules in PS. The solutions are then concatenated into five chromosomes by removing the route delimiters from the solution sequences. Since PS is a well-known heuristic, we omit its details in this paper. Interested readers may refer to the original paper [5]. The remaining $popsize - 5$ chromosomes are generated by a best-insertion heuristic (BIH). BIH starts with an empty route. In each iteration, the task satisfying the capacity constraint and with the minimal increased cost is inserted into the route. If no task satisfies the capacity constraint, a new empty route is initialized and the procedure repeats until all the tasks are inserted. The solution obtained is then also concatenated into a chromosome and inserted into the population. Note that both PS and BIH are deterministic algorithms and cannot be directly executed on UCARP instances. As our UCARP is approximated by a set of CARP instances, the aforementioned two heuristics are executed on randomly selected CARP instances. In order to increase the diversity of the population, no duplicates are allowed to appear in the population. The maximal trials for generating non-redundant initial individuals by using BIH are bounded to *ubtrial*. If the number of trials exceeds *ubtrial*, individuals are directly generated at random.

Stopping criterion: The EDASLS stops when a maximal number of fitness evaluations, FE_m, is met.

Update of robust solutions: As described in the previous sections, a population is composed of chromosomes (giant routes without route delimiters). However, the final solution is a sequence of tasks with route delimiters. Thus, a chromosome should be converted into a solution. A difficult issue to be addressed here is how to partition a chromosome c into a solution s with minimal $R(s)$ (defined in Section II). As the vehicle number is unfixed, the solution space of this problem is very large. To determine the optimal partition scheme with respect to $R(s)$ would be another hard optimization problem. To get around this problem, we follow the idea in Section III.A. A chromosome is first partitioned into $|\Theta|$ solutions on the $|\Theta|$ CARP instances in order to evaluate its fitness. The one with the minimal $R(s)$ among the $|\Theta|$ solutions is then compared to the current best robust solution. The one with minimal $R(s)$ will be retained. By this means, only a small part of the partition schemes will be checked. Assume the number of chromosome evaluations is FE, the number of solutions being checked is then $|\Theta| \cdot FE$. We do this for two reasons: (1) it saves much time to calculate R of $|\Theta| \cdot FE$ solutions rather than all the solutions in the solution space; (2) different instances in Θ share a similar structure and hence may have approximate values of parameters.

StochasticLS$(c, ehm, UT, \Theta, s_r)$
Input: a chromosome c, the edge-histogram matrix ehm, the task set UT, a set of instances Θ
Output: an improved chromosome c', a robust solution s_r
begin
  until c can no longer be improved **do**
    $c_{si} \leftarrow$ SI$(c, ehm, UT, \Theta, s_r)$
    $c_{di} \leftarrow$ DI$(c, ehm, UT, \Theta, s_r)$
    $c_{swap} \leftarrow$ Swap$(c, ehm, UT, \Theta, s_r)$
    $c_{2-opt} \leftarrow$ 2_opt$(c, ehm, UT, \Theta, s_r)$
    $c \leftarrow$ the best one of c_{si}, c_{di}, c_{swap} and c_{2-opt}
  end
  return c'
end

SI$(c, ehm, UT, \Theta, s_r)$
Input: a chromosome c, the edge-histogram matrix ehm, the task set UT, a set of instances Θ
Output: an improved chromosome c', a robust solution s_r
begin
  $c_{si} \leftarrow c$
  for all the pair of tasks (u, v) in the task set UT **do**
    move task u after task v in c_{si}
    $\Delta ehm \leftarrow \sum_{i=1}^{t-1} ehm_{c_{si}[i],c_{si}[i+1]} - \sum_{i=1}^{t-1} ehm_{C[i],C[i+1]}$.
    if $\Delta ehm > 0$ **then**
      $F(c_{si}) \leftarrow$CEvaluate(c_{si}, Θ)
      update s_r
      if $F(c_{si}) < F(c)$ **then**
        return c_{si}
      end
    end
  end
  return c_{si}
end

Note: The procedures of DI, Swap and 2-opt are similar to that of SI except that the move operations are different. Hence we omit the presentation of these algorithms.

Fig. 4. The pseudo-code of StochasticLS(c,ehm,UT,Θ).

IV. Experimental Studies

A. Benchmark instances

The UCARP test instances generated in [24], i.e., *ugdb* and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

TABLE I
THE PARAMETERS OF EDASLS

Name	Description	Value
psize	Population size	120
n	Number of sub segments in EHBSA	*2*
ubtrial	Maximum trials for generating chromosomes	100
p_{ls}	Probability of executing local search	0.1
FE_m	Maximum number of fitness evaluation	300000

TABLE II
RESULTS ON THE *ugdb* BENCHMARK TEST SET IN TERMS OF THE WORST-CASE SOLUTION COSTS

Name	EDASLS			EDA/NoLS			MA1			MA2			MAILS			*p*-value
	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	
ugdb1	91.03	368.40*	2.00	61.14	370.34	1.75	36.62	377.58	3.05	95.54	373.27	2.74	624.82	460.03	21.80	0
ugdb2	106.43	396.53*	1.73	95.36	397.58	2.76	38.97	407.84	3.23	115.41	403.41	3.43	923.23	488.29	22.27	0
ugdb3	94.45	323.17*	1.04	96.28	323.43	1.37	38.84	334.59	3.62	95.60	330.79	2.81	644.47	410.69	18.55	0
ugdb4	84.79	349.22	3.37	86.13	349.08*	4.22	34.67	356.38	6.73	81.92	352.33	4.11	436.57	421.66	22.85	0
ugdb5	107.13	468.84*	2.86	110.51	469.14	3.36	28.68	483.60	3.97	123.33	474.92	4.24	973.35	577.42	29.43	0
ugdb6	94.07	359.16*	1.94	96.95	362.36	3.54	28.20	365.79	2.78	101.23	360.57	2.97	670.80	434.73	22.91	0
ugdb7	93.50	370.03*	2.16	97.06	371.61	1.09	35.18	382.17	4.81	102.19	375.39	3.58	640.57	471.76	15.99	0
ugdb8	174.98	447.38*	4.88	189.58	457.23	5.94	45.48	482.21	6.33	443.32	465.80	3.94	4499.27	527.40	18.44	0
ugdb9	197.71	375.89*	3.48	214.48	382.39	5.10	49.92	404.45	8.06	515.56	391.97	3.48	5323.11	416.09	8.65	0
ugdb10	118.77	302.95	0.83	121.50	303.76	1.88	52.21	304.27	2.25	108.79	**302.05***	1.26	842.90	365.42	24.66	0.0004
ugdb11	245.87	448.72*	0.94	214.72	452.79	2.82	74.51	459.07	5.30	251.93	450.31	1.76	3538.27	467.26	5.87	0
ugdb12	89.65	615.88*	7.68	92.33	621.68	8.03	44.65	662.24	7.00	127.03	650.51	8.11	759.79	762.32	28.69	0
ugdb13	110.92	596.54*	2.31	116.14	597.64	1.34	44.89	601.55	2.15	135.63	597.23	1.95	920.71	622.20	12.14	0
ugdb14	90.40	111.88*	0.13	93.05	111.92	0.22	45.59	113.02	0.91	91.55	112.34	0.71	425.01	123.31	4.16	0
ugdb15	102.67	66.11*	0.00	105.99	66.11*	0.00	50.42	66.11*	0.00	95.12	66.11*	0.00	547.41	66.11	0.00	1
ugdb16	119.78	134.92*	0.60	126.55	135.09	0.55	48.24	135.89	0.58	120.00	135.18	0.27	1012.38	140.63	2.58	0
ugdb17	128.92	96.55*	0.09	135.10	96.65	0.07	50.36	96.58	0.09	121.98	96.58	0.10	1331.11	98.61	0.94	0.0041
ugdb18	183.16	177.69*	1.21	191.91	178.62	1.08	60.45	180.20	0.95	147.38	178.35	1.06	1536.15	189.30	4.46	0
ugdb19	58.27	76.33*	0.00	58.86	76.33*	0.00	37.00	76.33*	0.00	35.78	76.33*	0.00	123.78	76.82	1.36	1
ugdb20	94.84	138.48*	0.35	102.77	138.59	0.00	46.77	138.59	0.00	100.06	138.55	0.18	601.43	143.07	3.62	0.2417
ugdb21	131.33	169.43	1.38	138.92	170.08	1.11	47.24	171.91	1.91	161.92	169.15*	1.06	1747.61	178.91	2.88	0
ugdb22	159.41	217.54	0.57	176.55	217.60	0.74	47.91	219.21	0.94	249.96	**217.07***	0.67	3095.28	221.45	2.35	0
ugdb23	176.21	250.00*	0.64	194.85	250.49	0.98	49.50	258.06	1.37	420.31	253.08	0.83	5087.66	259.32	1.40	0
#. of 'w-d-l'					10-13-0			18-5-0			12-9-2			21-2-0		
mean	124.10	298.33	1.75	126.82	300.02	2.08	45.06	307.72	2.87	167.02	303.1	2.15	1361.46	344.47	12.00	

Time: average execution time (in CPU seconds) over 20 independent executions;
Average: the average value over the 20 executions;
Std: standard deviation over the 20 executions;
p-value: *p*-values of ANOVA tests for comparing EDASLS, EDA/NoLS, MA1 and MA2.
The minimal average results are with "*". Bold (underlined) results indicate that the corresponding algorithm is better (worse) than EDASLS based on Wilcoxon rank-sum test with the level of significance $\alpha = 0.05$.
#. of 'w-d-l': the number of 'win-draw-lose' of EDASLS versus the other algorithms; mean: the average values on all test instances.

uval sets, are used in the experimental studies. The *ugdb* instances are small-scale instances with 7-27 vertices and 11-55 edges, and *uval* instances are medium-scale instances with 24-50 vertices and 34-97 edges. All the edges in the corresponding CARP instances are required edges (tasks). Each UCARP instance contains 30 CARP instances generated by extending the same CARP instance. Thus, the 30 CARP instances share a similar graph structure and have similar (but not the same) values of parameters.

When Mei et al. [24] proposed UCARP, they also provided a procedure to generate UCARP instances by extending the well-known deterministic CARP instances. For each deterministic CARP instance, 30 UCARP instances are generated using the instance generator[1], with the starting random seed of 0. If necessary, more uncertain instances can be generated by the generator with other random seeds. The known random seeds enable us to re-produce the instances for our comparative studies.

[1] Available at http://staff.ustc.edu.cn/~ketang/codes/UCARPGen.zip

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

TABLE III
RESULTS ON THE *uval* BENCHMARK TEST SET IN TERMS OF THE WORST-CASE SOLUTION COSTS

Name	EDASLS			EDA/NoLS			MA1			MA2			MAILS			*p*-value
	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	
uval1A	332.44	182.43	1.18	289.11	183.26	1.52	83.49	183.29	2.31	139.83	**180.71***	1.27	3136.85	188.10	5.74	0
uval1B	256.01	190.56*	0.68	261.60	193.21	1.57	64.01	195.34	1.86	171.95	192.08	1.59	2842.67	204.66	3.69	0
uval1C	151.72	301.61*	2.77	161.08	305.06	3.95	36.06	323.11	4.08	285.63	311.70	2.62	3254.86	344.69	10.74	0
uval2A	268.15	246.12*	3.50	273.70	252.94	5.07	80.39	246.15	2.84	116.78	243.58	0.81	1767.45	436.45	48.57	0
uval2B	215.06	298.95*	1.90	226.19	301.06	2.36	69.62	312.87	3.51	142.40	308.83	2.47	1666.90	532.89	60.49	0
uval2C	120.42	659.46*	6.07	129.78	661.85	7.13	33.07	705.79	12.51	256.86	680.73	6.96	2334.98	825.33	25.52	0
uval3A	240.37	86.67*	0.30	257.09	88.13	1.36	87.11	87.21	0.77	122.48	86.70	0.42	1748.86	132.13	11.17	0
uval3B	178.83	97.16*	1.84	188.49	100.62	1.87	73.70	104.28	1.38	151.80	102.10	1.23	1898.44	143.48	10.22	0
uval3C	139.33	179.47*	1.31	116.24	181.86	1.98	36.90	196.64	1.73	234.74	191.38	1.67	2413.70	230.26	9.03	0
uval4A	706.12	445.04*	3.41	510.92	462.27	4.59	256.24	460.18	4.18	523.68	446.74	2.20	10441.06	934.17	58.67	0
uval4B	560.40	466.21*	3.29	566.90	496.19	7.38	194.13	489.61	5.11	577.09	473.37	4.62	9489.51	781.73	165.77	0
uval4C	387.68	507.67*	3.89	511.67	536.25	6.84	138.01	535.78	8.85	648.27	517.57	3.75	9695.79	630.17	102.66	0
uval4D	251.70	678.00*	6.43	353.80	712.63	10.83	74.07	747.59	13.74	866.51	713.14	8.31	11636.18	787.77	46.79	0
uval5A	401.16	464.53*	3.09	516.67	485.47	3.02	212.13	484.83	4.12	421.06	468.03	4.38	8423.93	975.68	41.29	0
uval5B	308.49	505.07*	3.14	371.72	518.76	7.73	153.28	527.87	5.41	473.56	507.22	3.88	8404.97	949.09	47.96	0
uval5C	255.24	547.70*	4.01	296.26	573.69	6.23	115.15	574.24	9.88	525.13	556.78	4.99	9377.18	889.89	84.82	0
uval5D	307.94	737.44*	7.82	202.28	775.84	13.07	66.03	794.18	11.84	721.85	761.63	7.80	15209.00	936.12	59.37	0
uval6A	412.91	246.34*	1.56	246.59	251.25	3.57	110.63	257.39	3.05	252.77	247.75	0.99	5266.45	259.42	13.07	0
uval6B	332.54	279.79*	1.76	297.54	282.52	1.49	83.69	288.48	2.84	292.99	281.36	1.31	4955.11	308.62	6.49	0
uval6C	193.39	430.19*	3.41	214.21	443.26	5.88	43.85	463.44	8.47	476.87	444.01	5.76	6218.83	510.99	24.7	0
uval7A	544.62	304.75*	1.40	674.30	308.43	2.79	163.98	316.42	3.18	452.78	306.28	0.78	12376.88	307.78	2.15	0
uval7B	374.66	309.15*	1.68	498.30	312.58	2.50	130.85	325.85	4.42	507.14	310.56	1.00	13412.82	315.71	2.93	0
uval7C	209.56	390.71*	2.31	263.64	407.28	5.14	62.06	424.97	7.49	776.15	406.64	3.23	11751.34	420.41	8.06	0
uval8A	364.99	408.88*	2.64	433.81	424.63	6.81	185.10	423.11	4.00	390.94	411.46	2.02	7476.15	443.25	43.72	0
uval8B	278.53	433.55*	2.97	318.25	457.63	6.60	142.88	457.32	5.53	431.68	443.26	2.01	8263.57	455.28	17.10	0
uval8C	281.36	647.20*	5.98	273.73	677.51	8.39	58.67	696.25	8.57	644.76	677.14	6.35	12615.85	717.47	17.19	0
uval9A	1171.13	343.73*	2.55	1224.33	368.63	5.84	445.15	357.44	3.54	918.19	343.80	1.14	27019.55	740.35	24.58	0
uval9B	712.24	353.31	2.33	763.90	384.73	6.17	330.82	370.57	3.64	1008.89	352.55*	1.82	34530.30	702.47	30.07	0
uval9C	502.62	366.01*	3.03	536.79	401.10	3.91	248.99	388.73	4.22	1091.80	366.42	3.28	31575.71	602.96	132.54	0
uval9D	297.88	472.01*	4.63	312.33	512.83	9.30	105.17	512.24	7.89	1543.39	489.15	5.72	27229.68	487.42	7.93	0
uval10A	1306.87	450.58	2.76	1326.30	475.16	5.98	574.53	460.62	4.29	1056.84	**448.32***	1.51	31391.86	892.19	39.11	0
uval10B	824.17	464.09	4.32	879.44	499.04	6.57	409.44	476.96	5.11	1126.66	462.75*	2.38	28942.32	868.32	33.66	0
uval10C	907.48	483.36*	3.99	933.61	522.50	7.94	298.72	506.44	3.75	1243.82	488.36	2.43	27829.49	820.89	83.86	0
uval10D	523.32	620.14*	5.83	580.32	670.10	6.20	115.11	664.40	7.48	1678.30	638.88	4.86	28387.79	649.85	14.09	0
#. of 'w-d-l'					33-1-0			32-2-0			24-8-2			34-0-0		
mean	421.16	399.94	3.54	441.50	418.48	5.34	155.38	480.66	6.14	596.28	462.74	3.47	12440.76	637.92	38.44	

Time: average execution time (in CPU seconds) over 20 independent executions;
Average: the average value over the 20 executions;
Std: standard deviation over the 20 executions;
p-value: *p*-values of ANOVA tests for comparing EDASLS, EDA/NoLS, MA1 and MA2.
The minimal average results are with "*". Bold (underlined) results indicate that the corresponding algorithm is better (worse) than EDASLS based on Wilcoxon rank-sum test with the level of significance $\alpha = 0.05$.
#. of 'w-d-l': the number of 'win-draw-lose' of EDASLS versus the other algorithms; mean: the average values on all test instances.

B. Algorithms tested and parameter settings

To evaluate the efficacy of EDASLS, a series of experiments on the aforementioned benchmark test sets are conducted. In order to evaluate our algorithm, we compared it against the following baseline approach: solve all CARP instances using a state-of-the-art approach for CARP, and the most robust solution among them is selected as the final solution. One of such state-of-the-art algorithms is a recently proposed meta-heuristic, named memetic algorithm with iterated local search (MAILS) for CARP [44]. MAILS is executed on all the CARP instances $I_j \in \Theta$ to obtain $|\Theta|$ solutions $S_{dcarp} = \{S_1, S_2, \ldots S_{|\Theta|}\}$, each $S_i \in S_{dcarp}$ is executed on all $I_j \in \Theta$ to calculate $R(S_i)$, the one with the minimal $R(S_i)$ is selected as the final robust solution.

Other compared algorithms are the two variants of MA for UCARP proposed in [27] (MA1 and MA2). By replacing the expectation-based fitness function with the minimax fitness function defined in Section III.A, these approaches are modified to suit the minimax fitness function.

To investigate the contributions of the SLS to EDASLS, an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

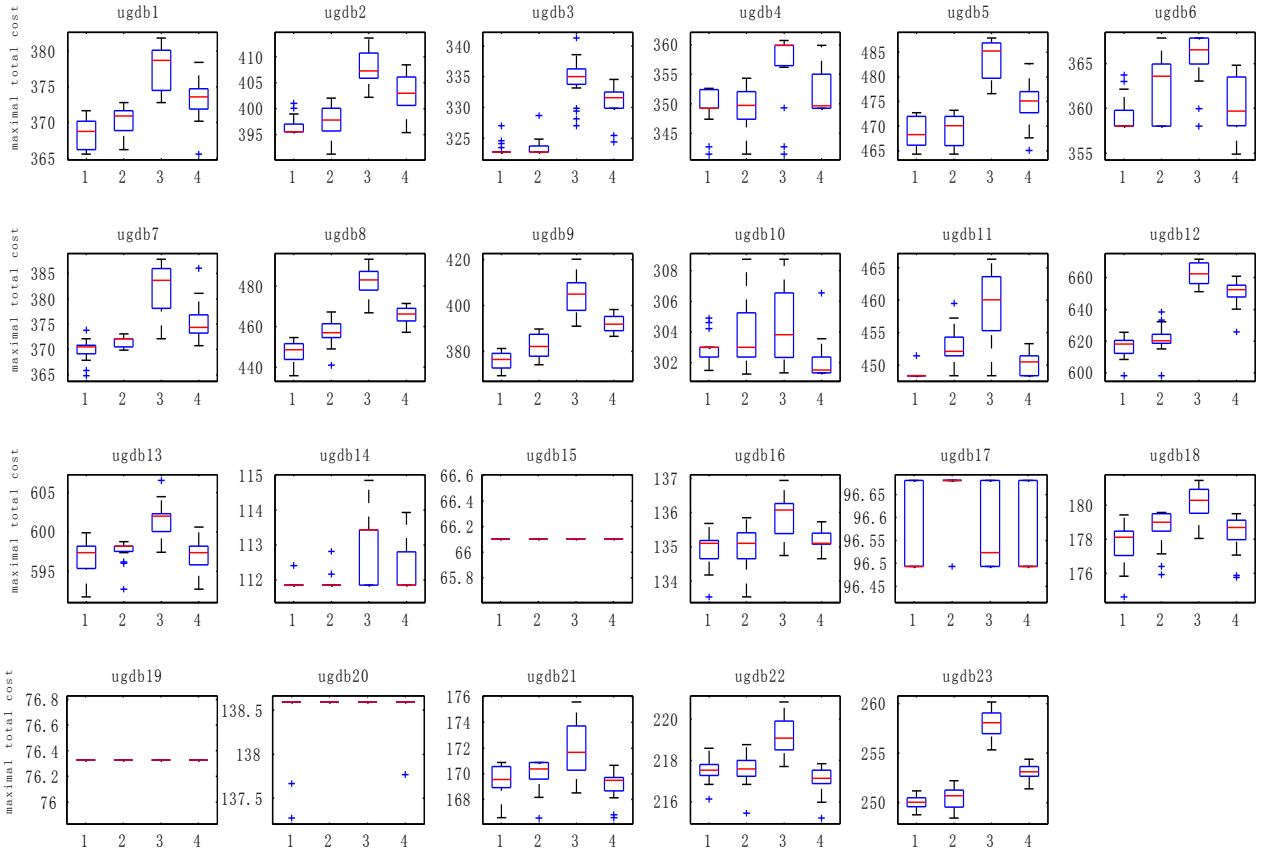

Fig. 5. The box plots on the *ugdb* benchmark test set in terms of the worst-case solution costs, where 1, 2, 3 ,4 in x-axis stand for EDASLS, EDA/NoLS, MA1, MA2, respectively.

EDA without SLS (EDA/NoLS), in which SLS is removed while all the other components of EDASLS is kept, was also involved in the comparison.

In order to ensure the fairness of the comparison, all algorithms except MAILS were assigned with the same number of fitness evaluation number FE_m. The termination conditions of MAILS, as well as all its other parameters, were set to the values presented in the original publication.

TABLE I summarizes the parameter settings of EDASLS, determined after some preliminary experiments. All the experiments were conducted for 20 independent executions, and the averages and standard deviations of the results are reported in this paper.

C. Comparing EDASLS to existing algorithms

1) Performance metrics

TABLE II and TABLE III report the results on *ugdb* and *uval* benchmark sets, respectively. The columns 'Time', 'Average' and 'Std' stand for the average execution time, average cost and standard deviation of cost over 20 independent executions. Results marked with "*" indicate the minimal average cost. In our experiments, all the algorithms are coded in C++ language using an Intel(R) Xeon(R) E5620 2.40GHz. Other algorithms are compared to EDASLS by using Wilcoxon rank-sum test with the level of significance $\alpha = 0.05$ over 20 executions. For EDA/NoLS, MA1, MA2 and MAILS, the average results highlighted in bold indicate that, statistically, the corresponding algorithm is significantly better than EDASLS. The underlined results indicate the corresponding algorithm is significantly worse than EDASLS. Results without any symbol indicate that the difference between EDASLS and the corresponding algorithm is statistically insignificant. Take TABLE II as an example, for *ugdb1*, the average result of EDA/NoLS (370.34) is underlined, indicating that EDASLS performed significantly better than EDA/NoLS on *ugdb1*. The average result of EDASLS (368.40) is marked with "*", indicating that EDASLS was the best-performing algorithm on *ugdb1*. For *ugdb10*, the average result of MA2 (302.05) is highlighted in bold, which means MA2 performed significantly better than EDASLS on *ugdb10*. The average result of MA1 (304.27) is not highlighted, meaning that statistically, there is no significant difference between EDASLS and MA1 on *ugdb10*.

For each table, two additional rows are included at the bottom. The first row represents the number of 'win-draw-lose' of EDASLS versus the other algorithms. The second row summarizes the average values for each algorithm on all the test instances.

Considering the columns headed 'Average' in TABLE II and TABLE III, we can observe that EDASLS outperforms the compared algorithms in almost all cases. Specifically, EDASLS outperforms MA1 on 18 out of 23 *ugdb* instances and 32 out of 34 *uval* instances. The gap between EDASLS and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

Fig. 6. The box plots on the *uval* benchmark test set in terms of the worst-case solution costs, where 1, 2, 3 ,4 in x-axis stand for EDASLS, EDA/NoLS, MA1, MA2, respectively.

MA2 is also significant. On 12 out of 23 *ugdb* instances and 24 out of 34 *uval* instances, EDASLS is the winner. Only on four instances (*ugdb 10*, *ugdb 22*, *uval1A* and *uval10A*) EDASLS is inferior to MA2 with small gaps (0.9, 0.47, 1.72 and 2.26). When compared to MAILS, EDASLS shows more significant advantage, outperforming MAILS on 21 of 23 *ugdb* instances and on all *uval* instances. The gaps between the average values are quite large (46.14 for *ugdb* and 183.82 for *uval*). Compared with EDA/NoLS, EDASLS is also competitive, winning on 10 instances of the *ugdb* set and 33 instances of the *uval* set.

In terms of the standard deviations, TABLE II shows that the average standard deviation of the results obtained by EDASLS over all the instances of *ugdb* set is just 1.75, while those obtained by EDA/NoLS, MA1, MA2 and MAILS are 2.08, 2.87, 2.15 and 12.00, respectively. The results on the *uval* set reported in TABLE III are similar. Although the average standard deviation obtained by EDASLS is not the lowest (3.54), it is quite close to the lowest value of 3.47 obtained by MA2. A closer look at TABLE II and TABLE III also shows that the largest standard deviation is 7.82 (*uval 5D*) for EDASLS, while for EDA/NoLS, MA1, MA2 and MAILS are 13.07 (*uval 5D*), 13.74 (*uval 4D*), 8.31 (*uval 4D*) and 165.77 (*uval 4B*), respectively. These observations highlight the advantage of EDASLS in terms of solution robustness and stability.

There are some particular phenomena that are worth noticing in TABLE II and TABLE III. In *ugdb15*, the results show that solutions obtained by all the algorithms have the same performance over 20 executions. This is because the graph of *ugdb15* is a very simple and complete one with only eight vertices. Despite the uncertainties, it is still quite easy to solve *ugdb15*. Another interesting observation is that in some of the *uval* instances, such as *uval9A~uval9C* and *uval10A~uval10C*, MAILS performs poorly. This phenomenon is due to the characteristics of the instances. On these instances, the values of demands vary greatly over tasks. And the absence of a task would affect the problem instance. Thus, an optimal solution for one instance (one scenario) might collapse on another instance in which a "was-required" task with a large demand disappears or an originally absent task emerges, leading to a large recourse cost.

Additionally, ANOVA tests were also conducted to compare the performance of EDASLS, EDA/NoLS, MA1 and MA2. MAILS was excluded because the previous analysis showed that MAILS was uncompetitive. The *p*-values of ANOVA tests are presented in the '*p*-value' columns in TABLE II and TABLE III. It can be observed that the compared approaches are significantly different for the 95% confidence interval on all of the instances except for *ugdb15, ugdb19* and *ugdb20*. To depict the performance of the tested approaches, the box plots of EDASLS, EDA/NoLS, MA1 and MA2 on all instances are presented in Fig. 5 and Fig. 6. Fig. 5 shows that solutions

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

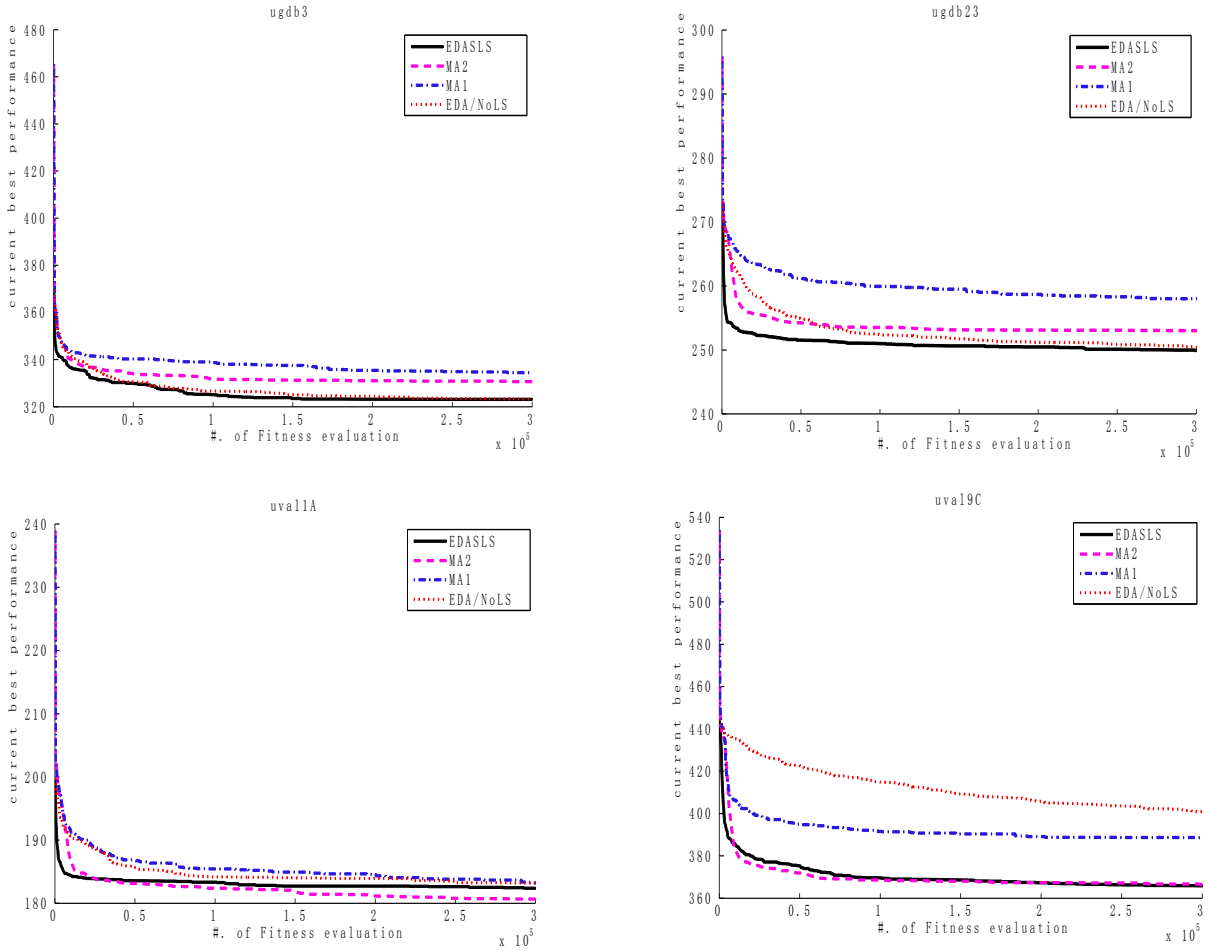

Fig. 7. The convergence curves of EDASLS, MA1, MA2 and EDA/NoLS on instances *ugdb3*, *ugdb23*, *uval1A* and *uval9C*.

obtained by EDASLS are more stable than those obtained by other tested approaches on most of the *ugdb* test instances. For the *uval* test instances, EDASLS clearly performed better than EDA/NoLS. This indicates that the SLS procedure indeed greatly improved the algorithm's performance. When compared to MA1 and MA2, EDASLS also showed advantages.

With regard to the execution time, EDASLS is also very competitive, although it is not the most efficient one among all the tested algorithms. EDASLS is only inferior to MA1.

To study the convergence speed of the tested algorithms, the convergence curves of the compared algorithms are plotted on four typical instances in Fig. 7. Pictures on other instances are similar. MAILS is not included in the figure because of its different basic framework from other algorithms. It can be observed that EDASLS converged the fastest among the compared algorithms. However, due to its fast convergence, it could converge to a local optimum (e.g., on *uval 1A*). To avoid this situation, the diversity of individual population needs to be increased, which will be a topic of our future research.

The observation that EDASLS outperforms the other algorithms demonstrates the efficiency of EDASLS. The reasons of EDASLS's excellent performance can be summarized as follows: (1) the quality of a solution to UCARP heavily depends on the order of tasks being served. In a robust solution, closely related tasks are more likely to be served adjacently to each other. In EDASLS, we build the edge histogram matrix to learn the adjacency information of tasks. New individuals are generated based on this information. By this means, the closely related fragments of solutions have a higher chance to be retained, making it easier to find the robust solution. (2) In the stochastic local search procedure, only the moves that put related tasks closer and produce chromosomes with better fitness are implemented. This strategy not only improves the quality of chromosomes but also avoids a large amount of excessive computations of fitness.

2) Structure metrics

Further analysis on the structures of the solutions obtained by EDASLS may deepen our understanding of UCARP. As mentioned in Section II, stochastic demands of tasks might incur route failures (violations of vehicle capacity constraints). When a route failure occurs, the recourse strategy amends the infeasible route by returning to the depot to get vehicle replenished before serving the next task. This "return-and-back" replenishment process would incur additional cost, which could be a large proportion of the total cost if the route failure occurs frequently. Since our goal is to minimize the total costs of solutions, it is meaningful to investigate the frequency of occurrence of route failure. Hence, a new structure metric Rf is defined. Rf is the route failure ratio of a solution in all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

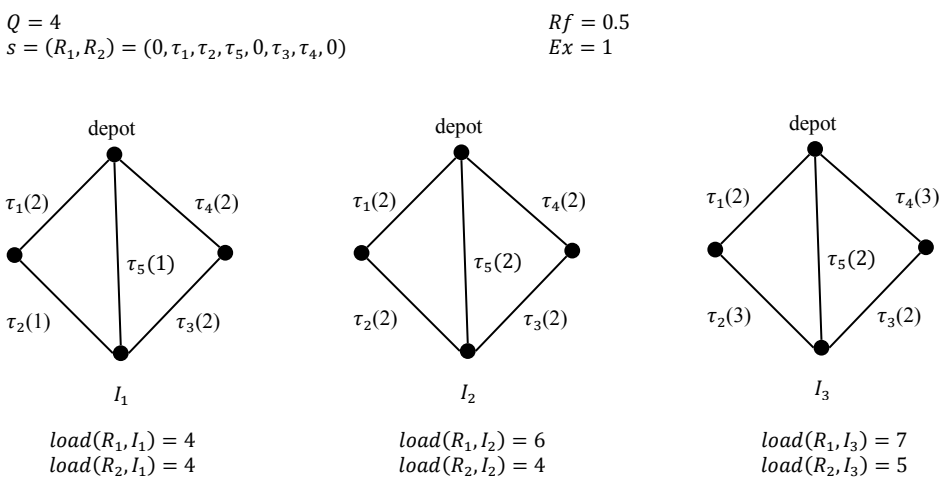

Fig. 8. An illustrative example of the structure metrics Rf and *Ex*.

scenarios of scenario set Θ. For solution $s = (R_1, R_2, \dots, R_m)$, Rf is defined as:

$$Rf = \frac{\sum_{I_j \in \Theta} \sum_{k=1}^{m} sgn(max\{load(R_k, I_j) - Q, 0\})}{m \cdot |\Theta|} \quad (16)$$

where $sgn(x)$ is the sign function defined as:

$$sgn(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if\ x < 0 \end{cases} \quad (17)$$

and $load(R_k, I_j)$ denotes the total demand of route R_k in scenario I_j.

As can be seen from (16) and (17), Rf is concerned with the number of route failures while ignoring the quantity of capacity constraint violation. In real-world situations, when route failure is inevitable, one would hope the quantity of violation to be as small as possible. Thus, we also define another structure metric Ex, which is the ratio of capacity violation. Similar to Rf, Ex is defined as:

$$Ex = \frac{\sum_{I_j \in \Theta} \sum_{k=1}^{m} max\{load(R_k, I_j) - Q, 0\}}{m \cdot |\Theta|} \quad (18)$$

Rf and Ex are similar but are concerned with different aspects of a solution. The former measures the number of route failures while the latter is concerned with the quantity of capacity violation when a route failure occurs. Fig. 8 gives a simple example to illustrate the two metrics. Each graph corresponds to a scenario. The tasks (edges) are labeled as task IDs with demands in brackets. Assuming a solution $s = (0, \tau_1, \tau_2, \tau_5, 0, \tau_3, \tau_4, 0)$ with two routes R_1 and R_2 and a capacity $Q = 4$. Due to the uncertain demands of tasks, route failures occur in route R_1 (with a capacity violation of 2) in scenario I_2 and R_1 (with capacity violation 3) and R_2 (with capacity violation 1) in scenario I_3. The total number of route failures is 3 and the total violation of capacity is 6. Hence, the values of Rf and Ex are $3/(2 \times 3) = 0.5$ and $6/(2 \times 3) = 1$, respectively.

Fig. 9 depicts the histograms of the Rf and Ex on instances *uval8C* and *uval5D*. For the convenience of cross-comparison, the histograms of costs are also shown. All values are the averages of results of 20 independent executions. On both instances, EDASLS achieved the lowest average costs among the five algorithms. It can be observed that for EDASLS, the values of *Rf* and *Ex* generally coincide with the costs. To be specific, the solution obtained by EDASLS corresponds to the lowest *Rf* and *Ex* on *uval8C*, and the second lowest *Rf* and *Ex* on *uval5D*. These observations support our conjecture that less route failures could save more additional replenishment cost and lead to lower total cost. Although the two structure metrics of the solution obtained by EDASLS are slightly higher than those achieved by EDA/NoLS, the differences between the two methods are relatively small, e.g., 0.14 to 0.12 in terms of *Rf*. In this case, the additional cost induced by the slight increase in number of route failures may be insignificant in comparison to the total cost. Hence, EDASLS still managed to achieve a lower total cost.

3) Further investigation of SLS

To further investigate the influence of the first phase of SLS, i.e., the calculation of ehm value to pre-evaluate intermediate

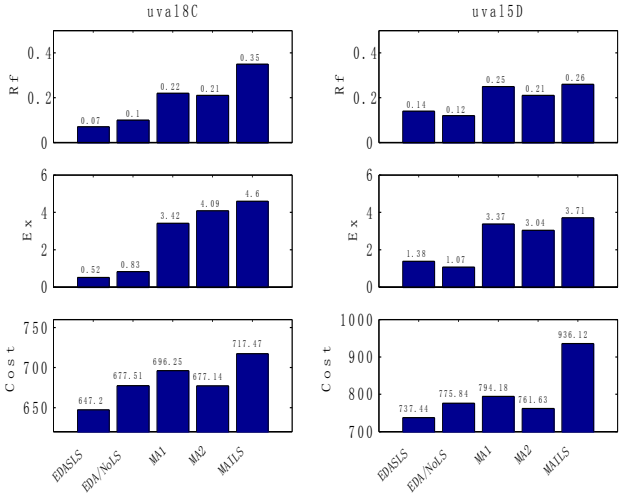

Fig. 9. The histograms of Rf, Ex and cost on instances *uval8C* and *uval5D*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

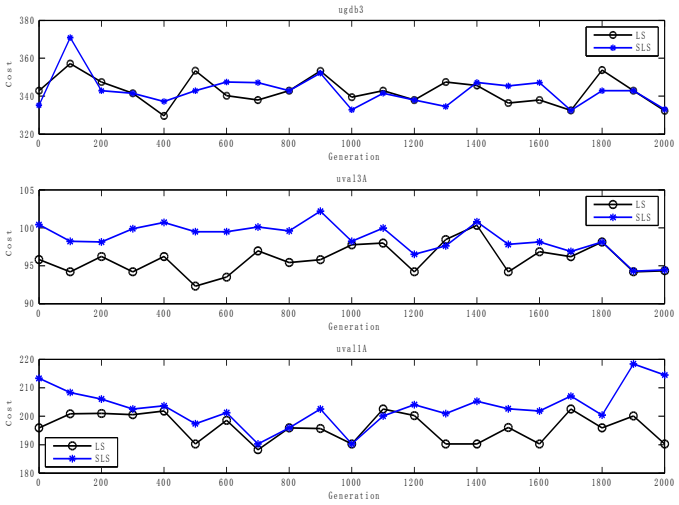

Fig. 10. Average costs of the "after-local-search" individuals obtained by LS and SLS versus generations.

individuals, we removed it from SLS while keeping the other parts unchanged. By this means, SLS turns into traditional LS. The resultant LS was applied to the same individuals that undergo the SLS procedure. That is, in each local search process, LS and SLS were executed on the same offspring and two improved individuals were produced and compared with each other. Fig. 10 gives the results of average cost of the "after-local-search" individual versus generation on three typical instances. Results on other instances are not reported because of the page limit. Note that each pair of "after-local-search" individuals was generated from the same neighborhood of an individual to make a fair comparison.

The results on instance *ugdb3* show that there is no apparent gap between the results obtained by EDASLS and those obtained by EDALS. This observation indicates that the pre-evaluation phase of SLS does not misjudge the intermediate individual, i.e., intermediate individuals with good fitness are not discarded in the first phase of SLS. Taking a closer look at the results on instance *uval3A*, we observe that in early generations, LS achieved solutions with lower costs than EDASLS. However, in later generations, EDASLS caught up with EDALS. The reason is that the quality of the edge histogram matrix could greatly affect the performance of SLS. In early generations, the population is not well evolved, thus the edge histogram matrix is not very reliable, leading to misjudgments in the pre-evaluation phase of SLS. In later generations, individuals in the population are generally of higher quality. In consequence, the pre-evaluation phase of SLS is more correct. Results on instance *uval1A* can be divided into three stages. The first two show the similar trend to that on *uval3A*. While in the last stage, the line of SLS is higher than that of LS, indicating a higher ratio of misjudgment in the pre-evaluation phase of SLS. This may be because the edge histogram matrix learned from the population overfitted the convergent population. Thus, new individuals (might have good performance) with different structures from those in the population are discarded because they cannot enhance the correlation of adjacent tasks.

TABLE IV summarizes the average execution time of EDAs with SLS and with LS on all tested instances over 20 independent runs. The results show that SLS is much faster than LS, especially on the medium-sized *uval* set.

TABLE IV AVERAGE RUNTIME OVER 20 RUNS (IN CPU SECONDS) OF THE COMPARED ALGORITHMS ON THE TEST SETS.

Test set	EDASLS	EDALS
Ugdb	124.10	274.7
uval	421.16	2224.4

V. CONCLUSION AND FUTURE WORK

In this paper, we formulate UCARP as a minimax optimization problem. An EDA with a novel stochastic local search (EDASLS) is devised for solving UCARP. The EDASLS algorithm is featured with: (1) a problem-dependent chromosome structure and evaluation strategy, and (2) a two-phase stochastic local search procedure (SLS) to avoid redundant fitness evaluations in local search. From the experimental results, three conclusions can be drawn. First, EDASLS is capable of obtaining solutions with better "worst-case" performance. Second, the SLS plays an important role in EDASLS. It prevents excessive fitness evaluations of unpromising moves in traditional local search and hence accelerates the algorithm without reducing the performance. Third, although a UCARP can be approximated by a set of CARP instances, it is not a good choice to separately solve the CARP instances using existing CARP algorithms, because the optimal solution to one single CARP instance might have poor robust performance.

Despite its excellent performance in terms of both solution quality and efficiency, EDASLS is by no means perfect. The main disadvantage of EDASLS is that it is confined to a finite number of deterministic instances. Hence, a potential direction to be explored in the future is to develop approaches that can achieve robust solutions over an infinite set of CARP instances, e.g., representing the uncertainty of the demand of tasks using intervals. Since UCARP takes into account of the uncertainties that might occur in reality, it is much closer to real-world applications, than deterministic CARPs. Extending EDASLS to tackle relevant practical problems, such as path planning of mobile robot or unmanned aerial vehicles, is also an important and interesting future work.

ACKNOWLEDGMENT

This work was supported in part by the 973 Program of China under Grant 2011CB707006, the National Natural Science Foundation of China under Grants 61175065 and 61329302, the Program for New Century Excellent Talents in University under Grant NCET-12-0512, the Science and Technological Fund of Anhui Province for Outstanding Youth under Grant 1108085J16, the European Union Seventh Framework Programme under Grant 247619, and an EPSRC grant (No. EP/I010297/1). Jose A. Lozano is partially supported by the Basque Government (IT609-13), and Spanish Ministry of Economy and Competitiveness MINECO (BCAM Severo Ochoa excellence accreditation SEV-2013-0323 and TIN2013-41272P). Xin Yao was support by a Royal Society Wolfson Research Merit Award.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

REFERENCES

[1] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks,* vol. 11, pp. 305-315, 1981.

[2] R. Hirabayashi, Y. Saruwatari and N. Nishida, "Tour constructive algorithm for the capacitated arc routing problem," *Asia-Pacific Journal of Operational Research,* vol. 9, pp. 155-175, 1992.

[3] H. Longo, M. P. de Aragão and E. Uchoa, "Solving capacitated arc routing problems using a transformation to the CVRP," *Computers & Operations Research,* vol. 33, pp. 1823-1837, 2006.

[4] R. Baldacci and V. Maniezzo, "Exact methods based on node‐routing formulations for undirected arc‐routing problems," *Networks,* vol. 47, pp. 52-60, 2006.

[5] 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, pp. 47-59, 1983.

[6] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Computers & Operations Research,* vol. 16, pp. 589-600, 1989.

[7] G. Ulusoy, "The fleet size and mix problem for capacitated arc routing," *European Journal of Operational Research,* vol. 22, pp. 329-337, 1985.

[8] R. W. Eglese, "Routeing winter gritting vehicles," *Discrete Applied Mathematics,* vol. 48, pp. 231-244, 1994.

[9] 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, pp. 629-643, 2003.

[10] P. Lacomme, C. Prins and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Annals of Operations Research,* vol. 131, pp. 159-185, 2004.

[11] 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, pp. 1151-1166, 2009.

[12] Y. Mei, X. Li and X. Yao, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *Evolutionary Computation, IEEE Transactions on,* vol. 18, pp. 435-449, 2014.

[13] M. Reghioui, C. Prins and N. Labadi, "GRASP with path relinking for the capacitated arc routing problem with time windows," in *Applications of Evolutionary Computing*: Springer, 2007, pp. 722-731.

[14] P. Lacomme, C. Prins and M. Sevaux, "A genetic algorithm for a bi-objective capacitated arc routing problem," *Computers & Operations Research,* vol. 33, pp. 3473-3493, 2006.

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

[16] G. Fleury, P. Lacomme, C. Prins, and W. Ramdane-Chérif, "Robustness evaluation of solutions for the capacitated arc routing problem," in *Conference, AI Simulation and Planning in High Autonomy Systems*, 2002, pp. 290-295.

[17] G. Fleury, P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Improving robustness of solutions to arc routing problems," *Journal of the Operational Research Society,* vol. 56, pp. 526-538, 2005.

[18] G. Fleury, P. Lacomme and C. Prins, "Evolutionary algorithms for stochastic arc routing problems," *Applications of Evolutionary Computing,* pp. 501-512, 2004.

[19] G. Fleury, P. Lacomme, C. Prins, and W. Ramdane-Chérif, "Stochastic capacitated arc routing problems," *Document de travail interne. Article en cours de rédaction,* 2005.

[20] G. Fleury, P. Lacomme, C. Prins, and M. Sevaux, "A bi-objective stochastic approach for stochastic CARP," *Research Report LIMOS/ RR-08-06,* 2008.

[21] C. H. Christiansen, J. Lysgaard and S. Wøhlk, "A branch-and-price algorithm for the capacitated arc routing problem with stochastic demands," *Operations Research Letters,* vol. 37, pp. 392-398, 2009.

[22] A. Ben-Tal and A. Nemirovski, "Robust optimization‐methodology and applications," *Mathematical Programming,* vol. 92, pp. 453-480, 2002.

[23] H. G. Beyer and B. Sendhoff, "Robust optimization-a comprehensive survey," *Computer Methods in Applied Mechanics and Engineering,* vol. 196, pp. 3190-3218, 2007.

[24] Y. Mei, K. Tang and X. Yao, "Capacitated arc routing problem in uncertain environments," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010, pp. 1-8.

[25] H. Handa, L. Chapman and X. Yao, "Robust route optimization for gritting/salting trucks: A CERCIA experience," *Computational Intelligence Magazine, IEEE,* vol. 1, pp. 6-9, 2006.

[26] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *Evolutionary Computation, IEEE Transactions on,* vol. 9, pp. 303-317, 2005.

[27] J. Wang, K. Tang and X. Yao, "A memetic algorithm for uncertain Capacitated Arc Routing Problems," in *Memetic Computing (MC), 2013 IEEE Workshop on*, 2013, pp. 72-79.

[28] S. Tsutsui, M. Pelikan and D. E. Goldberg, "Using edge histogram models to solve permutation problems with probabilistic model-building genetic algorithms," *IlliGAL Report,* 2003.

[29] J. E. Mendoza, B. Castanier, C. Guéret, A. L. Medaglia, and N. Velasco, "A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demands," *Computers & Operations Research,* vol. 37, pp. 1886-1898, 2010.

[30] D. J. Bertsimas, "A vehicle routing problem with stochastic demand," *Operations Research,* vol. 40, pp. 574-585, 1992.

[31] P. Larranaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation* vol. 2: Springer, 2002.

[32] R. Santana, P. Larrañaga and J. A. Lozano, "Protein folding in simplified models with estimation of distribution algorithms," *Evolutionary Computation, IEEE Transactions on,* vol. 12, pp. 418-438, 2008.

[33] Q. Zhang, J. Sun, E. Tsang, and J. Ford, "Hybrid estimation of distribution algorithm for global optimization," *Engineering Computations,* vol. 21, pp. 91-107, 2004.

[34] J. E. Beasley, "Route first—cluster second methods for vehicle routing," *Omega,* vol. 11, pp. 403-408, 1983.

[35] C. Prins, N. Labadi and M. Reghioui, "Tour splitting algorithms for vehicle routing problems," *International Journal of Production Research,* vol. 47, pp. 507-535, 2009.

[36] Y. Mei, K. Tang and X. Yao, "A Memetic Algorithm for Periodic Capacitated Arc Routing Problem," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 41, pp. 1654-1667, 2011.

[37] B. Golden, A. Assad, L. Levy, and F. Gheysens, "The fleet size and mix vehicle routing problem," *Computers & Operations Research,* vol. 11, pp. 49-66, 1984.

[38] S. Salhi and M. Sari, "A multi-level composite heuristic for the multi-depot vehicle fleet mix problem," *European Journal of Operational Research,* vol. 103, pp. 95-112, 1997.

[39] C. Prins, "A simple and effective evolutionary algorithm for the vehicle routing problem," *Computers & Operations Research,* vol. 31, pp. 1985-2002, 2004.

[40] N. Labadi, C. Prins and M. Reghioui, "A memetic algorithm for the vehicle routing problem with time windows," *RAIRO-Operations Research,* vol. 42, pp. 415-431, 2008.

[41] P. Lacomme, H. Toussaint and C. Duhamel, "A GRASP× ELS for the vehicle routing problem with basic three-dimensional loading constraints," *Engineering Applications of Artificial Intelligence,* vol. 26, pp. 1795-1810, 2013.

[42] C. Prins, P. Lacomme and C. Prodhon, "Order-first split-second methods for vehicle routing problems: A review," *Transportation Research Part C: Emerging Technologies,* vol. 40, pp. 179-200, 2014.

[43] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*: The MIT press, 2001.

[44] T. Liu, Z. Jiang and N. Geng, "A memetic algorithm with iterated local search for the capacitated arc routing problem," *International Journal of Production Research,* pp. 1-10, 2013.

Juan Wang received the B.S. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, Anhui, China, in 2010, and is currently working towards her Ph.D. degree.

Her current research interests include memetic algorithm and other metaheuristics for solving capacitated arc routing problems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2015.2428616, IEEE Transactions on Evolutionary Computation

Ke Tang (M'07-SM'13) received the B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, in 2002, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2007, respectively.

Since 2007, he has been with the School of Computer Science and Technology, University of Science and Technology of China, where he is currently a Professor. He has authored/co-authored more than 100 refereed publications. His major research interests include computational intelligence, evolutionary computation, machine learning, and their real-world applications.

Dr. Tang is an Associate Editor or Editorial Board Member of the IEEE Computational Intelligence Magazine, Computational Optimization and Applications (Springer), Natural Computing (Springer) and Memetic Computing (Springer).

Jose A. Lozano (M'07-SM'13) received a PhD degree in Computer Science from the University of the Basque Country in 1998. Since 2008 he is a full professor in the same university, where he leads the Intelligent Systems Group. He is the co-author of more than 70 ISI journal publications and co-editor of 4 books. His major research interests include machine learning, pattern analysis, evolutionary computation, data mining and metaheuristic algorithms. Prof. Lozano is associate editor of IEEE Transactions on Evolutionary Computation and member of the editorial board of other 5 journals.

Xin Yao (F'03) is a Chair (Professor) of Computer Science and the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications) at the University of Birmingham, UK. He is an IEEE Fellow and the President (2014- 15) of IEEE Computational Intelligence Society (CIS). His major research interests include evolutionary computation and ensemble learning. He published 200+ refereed international journal papers. His papers won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards. He received the prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE CIS Evolutionary Computation Pioneer Award in 2013. He was the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. He has published frequently on the topic of capacitated arc routing problems (CARP) since his first paper in 2006 (H. Handa, L. Chapman and Xin Yao, ``Robust route optimisation for gritting/salting trucks: A CERCIA experience," IEEE Computational Intelligence Magazine, 1(1):6-9, February 2006.).