Capacitated Arc Routing Problem over Sparse Underlying Graph under Travel Costs Uncertainty

Sara Tfaili[1], Abdelkader Sbihi[2], Adnan Yassine[1] and Ibrahima Diarrassouba[1]
[1]*Laboratoire de Mathématiques Appliquées du Havre, Université Le Havre Normandie, 25 Rue Philippe Lebon, Le Havre, France*
[2]*ESCEM, École de Management, Tours, France*

Keywords: Sparse Graph, Robust Optimization, Multiple Scenarios.

Abstract: In this paper, we study the capacitated arc routing problem over sparse underlying graphs under travel costs uncertainty. In particular, we work with Multiple-Scenario Min-Max CARP over sparse underlying graphs. We present a mathematical formulation of the problem. A greedy heuristic algorithm and an adapted tabu-search algorithm are developed to solve the problem. The computational experiments show the effectiveness of these two algorithms for different sizes of instances as well for different number of scenarios.

1 INTRODUCTION

Combinatorial optimization requires the advance knowledge of the data and the parameters of the problem. However, real life applications may have a high degree of uncertainty and thus any disturbance of the input data may affect the nature of the solution to be not optimal or even infeasible. In our study and mainly in the field of routing problems, such perturbations may appear in the uncertainty of travel times due to traffic for example (Sungur et al., 2008), required demands by clients, arrivals of new clients or even in the travel costs. In particular, we are interested in studying the sparse Capacitated Arc Routing Problem (CARP) with uncertain travel costs, and this requires to study the problem in the terms of theoretical and practical issues by investigating the extensions of the sparse CARP under uncertain data. Modeling the uncertainty in the problems of optimization has been done by several methods, and stochastic programming could be considered to be the most famous among these methods. In stochastic programming, the uncertain data is modeled as random variable with known probability distribution (Wets, 2002; Shapiro et al., 2009). The stochastic nature of the uncertainties and the possibility of identifying the probability distribution are two main conditions to apply the stochastic programming. However, the nature of the uncertainty is not always stochastic and it is not always possible to identify the probability distri bution of the data. Unto now, most of such studies are done with stochastic programming (Fleury et al., 2004). A Branch-and-Price algorithm for the capacitated arc routing problem with stochastic demands has been presented (Christiansen et al., 2009), and another study is given in (Mei et al., 2010). As a consequence of the difficulty of attaining both conditions of stochastic programming, an alternative of the latter must be sought. Such alternative is the robust optimization which finds robust solutions that remain feasible upon facing unpleasant impacts which are caused by the ambiguity or the imprecision of the input data.
This paper is organized as follows. In Section 2 we give a brief review about the robust optimization (definitions and criteria). We introduce a min-max mathematical model of our problem, a heuristic algorithm and an adapted and metaheuristic algorithm to determine a solution of the problem and improve it in Section3 . Computational experiments are performed in Section 4. Conclusions are drawn out in Section 5.

2 REVIEW ABOUT ROBUST OPTIMIZATION

Starting from the definition of the word "robust" which means strong, robust optimization is sufficient to give a solution that is strong enough to face any disruption in the data. The data may be exposed to some unpredictability especially in future, and this may affect the optimal solution that is computed before to

Tfaili, S., Sbihi, A., Yassine, A. and Diarrassouba, I.
Capacitated Arc Routing Problem over Sparse Underlying Graph under Travel Costs Uncertainty.
DOI: 10.5220/0006751001440151
In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems (ICORES 2018)*, pages 144-151
ISBN: 978-989-758-285-1

be at last infeasible or not optimal. Thus a robust solution is a solution that resists as much as possible disturbances. When the uncertain data has a probabilistic description, robustness can be attained by using the stochastic optimization which has been given in (Dantzig and Ramser, 1959). However, this type of programming has two drawbacks:

1. The underlying probability distributions must be already known and this is not always the case.
2. The solutions can become infeasible upon facing some random events or disruptions.

To avoid these drawbacks, robust optimization which is not stochastic but rather deterministic and set-based is considered to be a suitable framework. The aim of such optimization is to optimize the worst case value under all uncertain data. In our work, we represent the uncertain data by generating discrete scenarios. Robustness criteria include several families where the decisions can be made according to min-max, min-max regret, min-max relative regret and lexicographical min-max, etc. For more details about different robustness criteria, the reader may refer to (Coco et al., 2014;; Kouvelis and Yu, 1997; Kasperski and Zieliński, 2011). The robustness criterion which we follow is the min-max criterion in which we minimize the cost whenever the worst scenario occurs.

3 CAPACITATED ARC ROUTING PROBLEM UNDER UNCERTAINTY

In the following, we study the capacitated arc routing problem over sparse underlying graphs and we design new algorithms that can find more robust solutions. The uncertain capacitated arc routing problem that we study is characterized by the uncertainty in the travel costs and by the sparse network for which it is defined over. This uncertainty is represented by a finite set of scenarios where each required edge of the network has a different cost with respect to each scenario. We aim at determining a robust solution i.e. in an uncertain environment, the problem objective is no longer to find a single global optimal solution, but to find a solution with the best expected quality under all possible environments. We present a mathematical modeling of the capacitated arc routing problem over sparse graph under travel costs uncertainty. In other terms, we are concerned with the Multiple-Scenario Min-Max Capacitated Arc Routing Problem in sparse graphs which has a number of vertices equal to n, number of edges equal to $m = n + \alpha$ with $1 \leq \alpha \leq \frac{n}{2}$, number of required edges equal to r and the maximum vertex degree held in G is 3.

3.1 A Brief Survey about CARP

One of the most important variants of the arc routing problem is the Capacitated Arc Routing Problem. This problem is NP-hard, and we recall here the definition of it which can be stated as follows: Given a connected undirected graph $G = (V, E, C, D)$, where C is a cost matrix and Q is a demand matrix, and given a number of identical vehicles each with capacity Q (where $Q \geq maxd_e, e \in E$). Find a number of tours such that:

1. Each arc with positive demand is serviced by exactly one vehicle.
2. The sum of demands of those arcs serviced by each vehicle does not exceed Q.
3. The total cost of the tours is minimized.

3.2 Mathematical Formulation

Throughout the following, let $G = (V, E)$ be a graph where V denotes the set of vertices and E the set of edges. Denote by $R \subseteq E$ the set of the required edges i.e. the set of edges having strictly positive demands to be serviced. Consider the following notations and variables:

- K: the total number of the vehicles.
- Q: the capacity of each vehicle.
- $dem(e)$: the demand of the edge e.
- $\Delta_{accum}(e)$: the total demand served by the vehicle arriving at the service e including the demand of e itself which is by definition less than or equal to Q.
- c_e^s: the cost of the edge e.
- $N(e)$: the neighborhood of the edge e i.e. the set of adjacent edges to e.
- $S = \{1, 2, \ldots, P\}$: the set of scenarios.
- ω_e: the capacity of edge e i.e. the maximum number of times for which an edge can be traversed.
- $x_{e,f}^{e',f'}$: a binary variable which is equal to 1 if and only if the service at f' is successive to the service at e' by the same vehicle, and the chosen shortest path between e' and f' includes the consecutive adjacent edges e and f, and 0 otherwise.
- $y_{e',f'}$: a binary variable equal to 1 if f' is serviced directly after e', and 0 otherwise.

Recall that the graphs which we are working over are sparse with maximum degree equal to 3. Denote by 0 and 1 two incident edges to the depot node. The edge 0 denotes the edge of departure of the vehicle i.e. exit from the depot, and 1 denotes the edge of returning back of the vehicle i.e. entrance to the depot after accomplishing all the services of the corresponding vehicle. Moreover, we assume that these edges are required but with a null demand. Note that this model is adapted to all CARP in any network, but it is adaptable to our sparse network as it respects the criteria of generating these networks.

$$\min \max_{s\in S} \sum_{e',f'\in R, e\in E, f\in N(e)} c_e^s x_{e,f}^{e',f'} \tag{1}$$

Subject to

$$y_{e',f'} + y_{f',e'} \leq 1 \quad \forall e', f' \in R \tag{2}$$

$$\sum_{f'\in R} y_{0,f'} = K \tag{3}$$

$$\sum_{f'\in R} y_{1,f'} = 0 \tag{4}$$

$$\sum_{e'\in R} y_{e',1} = K \tag{5}$$

$$\sum_{f'\in R} y_{e',0} = 0 \tag{6}$$

$$\sum_{f'\in R} y_{e',f'} = 1 \quad \text{if} \quad e' \neq \{0,1\} \tag{7}$$

$$\sum_{e'\in R} y_{e',f'} = 1 \quad \text{if} \quad f' \neq \{0,1\} \tag{8}$$

$$\Delta_{accum}(f') \geq \Delta_{accum}(e') + dem(f') + (dem(f') + Q) \times (y_{e',f'} - 1) \quad \forall e', f' \in R, e' \neq f' \tag{9}$$

$$\sum_{f\in N(e)} x_{e,f}^{e',f'} - \sum_{f\in N(e)} x_{f,e}^{e',f'} = 0 \quad \text{if} \quad e \neq e', e \neq f', e', f' \in R \tag{10}$$

$$\sum_{f\in N(e)} x_{e,f}^{e',f'} - \sum_{f\in N(e)} x_{f,e}^{e',f'} = y_{e',f'} \quad \text{if} \quad e = e', e', f' \in R \tag{11}$$

$$\sum_{f\in N(e)} x_{e,f}^{e',f'} - \sum_{f\in N(e)} x_{f,e}^{e',f'} = -y_{e',f'} \quad \text{if} \quad e = f', e', f' \in R \tag{12}$$

$$\sum_{e',f'\in R, f\in N(e)} x_{e,f}^{e',f'} \leq \omega_e \quad \text{with} \quad \omega_e \geq 1 \quad \text{if} \quad e \neq 0 \tag{13}$$

$$\sum_{e',f'\in R, f\in N(0)} x_{0,f}^{e',f'} = K \tag{14}$$

$$\sum_{e',f'\in R, f\in N(e)} x_{f,e}^{e',f'} \leq \omega_e \quad \text{with} \quad \omega_e \geq 1 \quad \text{if} \quad e \neq 1 \tag{15}$$

$$\sum_{e',f'\in R, f\in N(1)} x_{f,1}^{e',f'} = K \tag{16}$$

$$x_{e,f}^{e',f'}, y_{e',f'} \in \{0,1\}, \Delta_{accum}(e') \leq Q \quad \forall e', f' \in R, e, f \in E \tag{17}$$

The objective function (1) aims to minimize the total costs under the worst case. Constraints (2) are trivial to show that either e' is serviced before f' or vice versa. Constraints (3) to (6) show that all the vehicles must depart from the depot and all the vehicles must return back to the depot after serving the required edges. The number of predecessors and the number of successors is given by the constraints (7) and (8). Constraints (9) assure that if f' is served directly after e', then the total demand done at the level of f' is greater than or equal to the total demand done at e'. Otherwise, the difference between these demands is less than Q which is trivial. Shortest path constraints are represented from (10) to (12). Constraints (13) to constraints (16) determine the capacity of each edge in G i.e. the maximum number of times an edge can be traversed, where this capacity is some ω for edges different from depot, and it is K for the edges which are incident to the depot to assure the passage of all the vehicles from and into the depot. Decision variables constraints are given in (17).

3.3 Efficient Algorithms for Solving Robust Sparse CARP

In this part, we present a heuristic algorithm for solving the robust sparse capacitated arc routing problem under travel costs uncertainty. The initial solution which is obtained by this algorithm is then ameliorated by a well adapted tabu search algorithm.

3.3.1 A Heuristic Algorithm for Solving the Robust Sparse CARP under Travel Costs Uncertainty

This heuristic ends with a feasible initial solution of the problem. The procedure locates a worst scenario $\bar{S}$ and computes $Z(\bar{X}) = \max_x \sum c_e^{\bar{S}} x_{e,f}$. Let $e_1, e_2, \ldots, e_r$ be the required edges, and denote by λ_i the efficiency of each edge e_i which is given by the formula

$$\lambda_i = \frac{\sum_{s\in S} c_{e_i}^s}{dem(e_i)}, \tag{18}$$

where $dem(e_i)$ denotes the demand of the required edge e_i. This algorithm is valid for the two cases of

$\omega = 1$, where each edge can be traversed one only time, and $\omega > 1$, where there is a constant maximal number for traversing an edge. The only difference between the two cases lies mainly in the function $Update$.

Algorithm GH (Greedy Heuristic)
Input: A Robust CARP Instance
Output: A feasible solution $\bar{X}$ and the corresponding worst scenario $\bar{S}$

Initialization
0. $E = R \cup NR$; $|E| = n + \alpha$;
   $R = \{e_1, e_2, e_3, \ldots, e_{r-1}, e_r\}$;
   $NR = \{f_1, f_2, \ldots, f_{m-r}\}$;
1. For $j = 2$ to $r-1$ Do
   Sort the required edges in the non-decreasing order of the efficiencies λ_{e_j};
   End For
2. Set $\bar{X} \leftarrow 0; Z(\bar{X}) = +\infty$; $\bar{S} = 1$

Main Steps
1. $k \leftarrow 1$.
2. While $(R <> \phi\}$ and $k <= K)$ Do
3. $\Delta \leftarrow dem_{e_2}; j \leftarrow 3; P(1) \leftarrow e_1; P(2) \leftarrow e_2; i \leftarrow 3;$ $P \leftarrow \{e_1, e_2\}$;
4. While $((\Delta <= Q)$ and $j < r)$ Do
5. $Subpath(e_j)$;
6. End While
7. $P(i) \leftarrow e_r$;
8. $\dim P \leftarrow i$;
9. $l(i) \leftarrow \dim P$;
10. $Complete(P)$;
11. $\bar{X} \leftarrow \bar{X} \cup P$;
12. Let $\bar{S} = \max_{1 \le s \le |S|} \{Z^s(\bar{X})\}$;
13. $Update(R, NR)$;
14. $k \leftarrow k + 1$;
15. End While
16. Exit with a feasible solution $\bar{X}$ and with the corresponding worst scenario $\bar{S}$

Figure 1: A Greedy Heuristic Algorithm for determining a starting feasible solution of the Robust CARP

```
1. If (Δ + dem_{e_j} <= Q) then
2.     Δ ← Δ + dem_{e_j};
3.     P(i) ← e_j;
4.     P ← P ∪ {e_j};
5.     i ← i + 1;
6. End If
7. j ← j + 1;
```

Figure 2: The function *Subpath*.

Some concept of scheduling is encountered in our heuristic algorithm. This can be viewed in the solution representation where the services are arranged according to their order of being done. A solution of the problem is formed first of the services according to their order of being serviced, then we apply Dijkstra algorithm to determine a shortest path between each couple of services. In the following, we explain the steps of the previous Greedy algorithm Figure 1.

```
1. While (l(i) > 1) Do
2.     If (P(l) and P(l-1) are not adjacent) then;
3.         Dijkstra(P(l), NR, P(l-1));
4.         P' ← Dijkstra(P(l), NR, P(l-1));
5.         P ← P ∪ P';
6.         l(i) ← l(i) - 1;
7.     End If
8. End While
```

Figure 3: The function *Complete*.

```
1. For (i = 2 to dim P - 1) Do
2.     For (j = 1 to |NR|) Do
3.         If (P(i) = (NR)_j) then
4.             ω((NR)_j) ← ω((NR)_j) - 1;
5.         End If
6.     End For
7.     For (j = 2 to r - 1) Do
8.         If(P(i) = e_j) then
9.             β ← ω(e_j);
10.            a ← e_j;
11.            For (l(i) = j to r - 1) Do
12.                e_{l(i)} ← e_{l(i)+1};
13.            End For
14.            |NR| ← |NR| + 1;
15.            NR(|NR|) ← a;
16.            ω(a) ← α - 1;
17.            r ← r - 1;
18.        End If
19.    End For
20. End For
```

Figure 4: The function $Update$ for $\omega > 1$.

```
1. For (i = 2 to dim P - 1) Do
2.     For (j = 1 to |NR|) Do
3.         If (P(i) = (NR)_j) then
4.             For (l(i) = j to |NR|) Do
5.                 (NR)_{l(i)} ← (NR)_{l(i)+1};
6.             End For
7.             |NR| ← |NR| - 1;
8.         End If
9.     End For
10.    For (j = 2 to r - 1) Do
11.        If(P(i) = e_j) then
12.            For (l(i) = j to r - 1) Do
13.                e_{l(i)} ← e_{l(i)+1};
14.            End For
15.            r ← r - 1;
16.        End If
17.    End For
18. End For
```

Figure 5: The function $Update$ for $\omega = 1$.

- **Step 1**: `k` $\leftarrow$ `1` to start with the first vehicle.
- **Step 2**: `while` the set of the required edges is not empty, and the number of the vehicles is less than or equal to the number of the available ones.

- **Step 3**: the total accumulated demand is fixed at the demand of the first required edge with strictly positive demand and greatest efficiency. The first edge in the path is the depot. The second edge in the path is the first required edge with a strictly positive demand.
- **Step 4**: `while` the accumulated demand respects the capacity of the vehicle and there are still required non-serviced edges.
- **Step 5**: `call` the function $Subpath()$, Figure 2, that tests if adding the j^{th} required edge will not violate the capacity constraint. In this case, `update` the accumulated demand to be the last one added to the demand of j, and place this edge in the i^{th} rank of the constructed subpath. Then, move to the next rank and then to the next required edge.
-**Step 7** to **Step 9**: Once adding a required edge could violate the capacity, go back to the depot. The dimension of the constructed subpath is i where the depot entrance is the i^{th}-edge of this subpath i.e. the last edge. Assignment of the dimension of P to an auxiliary variable $l(i)$.
-**Step 10**: `call` the function *Complete*, Figure 3. The function *Complete* tests if the predecessor of each required edge in P say at rank $l(i)$ is not the required edge served in this path P and placed at the rank $l(i)-1$, then call $Dijkstra$ and insert a shortest path of made of edges of NR between the edge at rank $l(i)$ and the edge at rank $l(i)-1$. Each time an edge is inserted, the dimension of the path P is incremented by 1 and each edge at rank $j-1$ will be at the rank j to let the inserted edge compensate the emptied rank.
-**Step 11**: Update the constructed solution.
-**Step 12**: Determine the worst scenario that corresponds to the scenario giving the maximal solution cost.
-**Step 13**: `Update` the sets NR and R: in the case where ω ¿ 1, Figure 4, the capacity ω of any used edge in the path P is decremented by 1 each time the edge is used. The same is applied to the edges of the set R with the additional step that will be impose the removal of the used edges from R and then added to NR. For the case of $\omega = 1$, Figure 5, the edges of both sets are removed once traversed or served.
-**Step 14**: Move to the next vehicle.
We determine by this heuristic algorithm an initial robust solution of the problem and a corresponding worst scenario.

3.3.2 An Adapted Tabu Search Algorithm for Solving the Robust Sparse CARP under Travel Costs Uncertainty

In this part, we develop an adapted tabu search algorithm for the Robust Sparse CARP under travel costs uncertainty. This algorithm starts with the initial solution that is determined by the above greedy heuristic. Consider the following notations:

- $X^\star$: best feasible solution determined by the tabu search algorithm.
- L: the tabu list.
- $Iter$: number of iteration.
- $MaxIter$: maximum number of iterations.
- $N(X^\star)$: neighborhood about the solution $X^\star$.
- $S^\star$: the worst scenario determined by the algorithm.
- Th: a certain threshold.

Algorithm TS
Input: An Initial Feasible Solution $\bar{X}$
Output: A Best feasible solution $X^\star$ with the corresponding worst scenario $S^\star$.

Initialization
0. $X^\star \leftarrow \bar{X}$;
1. $L \leftarrow \phi$;
2. $Iter \leftarrow 0$;

Main Steps
1. `While` $(Iter < MaxIter)$ `Do`
2. `If` $(|L| \leq Th)$ `Then`
3. $\text{Build}_1(N(X^\star))$;
4. `If` $(|L| > Th)$ `Then`
5. $\text{Build}_2(N(X^\star))$;
6. `For` $(t = 1 \text{ to } |N(X^\star)|)$
7. `If` $((Z(X_t^\star) \leq Z(X^\star)) \&\& (X_t^\star \notin L))$ `Then`
8. $X^\star \leftarrow X_t^\star$;
9. `End If`
10. `Update` $L, Iter$;
11. `End For`
12. `End While`
13. `Exit with solution` $X^\star$ `and with the corresponding worst scenario` $S^\star$

Figure 6: A Tabu Search Algorithm for determining best feasible solution of the Robust CARP.

The algorithm contains several steps. Tabu search starts by an initial feasible solution obtained thanks to the greedy heuristic algorithm. All the visited solutions are feasible. The exploration of the solutions space is executed with some swaps. The elite solutions list is generated by improving the objective value where the worst scenario has already been identified. The core of the approach is to build neighborhoods

and perform several local searches in order to reach a best solution. In **Step 2** of the main steps of Figure 6, if a certain threshold Th is not attained, we diversify the search by using the function $\texttt{Build}_1$ to build the neighborhood. In this step, we choose randomly two vehicles, $vehicle_1$ and $vehicle_2$, and we select two services of each chosen vehicle i.e. $service_1^1$, $service_1^2$, $service_2^1$ and $service_2^2$. We check whether the swap of these services (the first service of the first vehicle with the first service of the second vehicle, and the second service of the first vehicle with the second service of the second vehicle) respects the capacity of the vehicles, and we swap them as explained. In this way, we explore the neighbors and we choose the best that minimizes the cost for the worst scenario. The search progresses by iteratively moving from the current solution to an improved solution. In **Step 4** of the main steps of this Figure, if the threshold Th is attained, we intensify the search by using the function $\texttt{Build}_2$ to build the neighborhood which allows the exchange of two services of the same vehicle. The tabu based strategy incorporates a tabu list in the selection mechanism that forbids the selection of the non-improving solution for a certain tabu tenure. Each visited explored solution is then settled in the tabu list L to not be visited again unless the tabu list reaches its expiration point i.e. the tabu status of a move is removed if it belongs to the list L and it exceeds $MaxIter$ iterations.
For the intensification and the diversification of the search, both are achieved via the functions $\texttt{Build}_1$ and $\texttt{Build}_2$.

4 COMPUTATIONAL EXPERIMENTS

In this section, we introduce a set of computational experiments for which we apply each of these algorithms; the heuristic algorithm and the tabu search one. The benchmark of instances which we deal with has not been treated before. We have run all these instances with different number of scenarios and different densities of their corresponding networks in order to evaluate the performance of each of the presented algorithms. The proposed algorithms are coded in C++ and run on HP intel(R) Core(TM) i7 laptop (with 2.80 Ghz and 16 GB of Ram). These test problem instances are studied for the first time and their optimal solution values are not known. None of the previous algorithms in the literature deals with such type of instances. We work with sparse graphs having a maximum degree of 3.

Consider the following notations:

- WS_H: the worst scenario which is determined by the heuristic algorithm.
- $Cost_H$: the cost of the solution which is determined by the heuristic algorithm.
- CPU_H: the time needed by the heuristic algorithm to determine a solution.
- WS_T: the worst scenario which is determined by the tabu algorithm.
- $Cost_T$:the cost of the solution which is determined by the tabu algorithm.
- CPU_T: the time needed by the tabu algorithm to determine a solution.

The robust optimization that we apply via the developed algorithms allows us not only to get a robust solution, but also it gives us the worst scenario that may change upon the improvement of the solution as we observe in the coming tables. In other terms, a worst scenario of a solution determined by the heuristic algorithm is not necessarily the same worst scenario of the solution obtained by the tabu algorithm after improvement. As a result, an improvement comes in two directions: (1) obtaining a better solution with a minimal cost and (2) improving the corresponding worst scenario.
In Table 1, NI means no improvement i.e. the tabu gives the same initial feasible solution determined by the heuristic and cannot improve this solution after passing 7 hours.

In Table 1, first of all, we notice that our greedy heuristic succeeds to have the access to all the studied instances with a very small CPU consuming time regardless the quality of the found solution, whereas the tabu algorithm did not succeed to have the access to the big size instances.

Recall that G denotes a graph where $V(G)$ denotes the set of vertices and $|V(G)|$ its cardinality, $E(G)$ denotes the set of edges and $|E(G)|$ its cardinality. The set of required edges is represented by R, and the set of different scenarios is represented by S. The used fleet of vehicles is homogeneous where K denotes the number of available vehicles and Q represents the capacity of each one. The different instances are generated randomly i.e. the sparse network is generated randomly but respecting that $|E(G)| = |V(G)| + \alpha$ with $1 \leq \alpha \leq \frac{|V(G)|}{2}$ and that the maximum degree in this network is 3. The costs over the scenarios are all generated randomly too. The numerical instances are divided into 3 groups: Groups A with 10 scenarios, Group B with 40 scenarios and Group C with 100 scenarios.

Table 1: Results of different families of the robust CARP problem instances.

Instance	$\|V(G)\|$	$\|E(G)\|$	$\|R\|$	S	K	Q	WS_H	$Cost_H$	$CPU_H(s)$	WS_T	$Cost_T$	$CPU_T(s)$
1A	10	13	5	10	2	15	10	1295	0.001	10	1059	0.5
2A	20	27	10	10	2	40	1	3104	0.008	1	2644	1.52
3A	50	70	25	10	3	50	6	10169	0.012	6	5658	8.326
4A	100	150	59	10	3	120	2	27370	0.056	10	17735	40.96
5A	231	331	121	10	4	120	3	46461	0.236	3	31074	124
6A	257	362	191	10	5	150	1	84875	0.392	5	50972	208
7A	307	439	260	10	7	150	5	124241	0.664	6	76389	357
8A	400	600	350	10	7	150	5	118039	1.168	5	71498	450
1B	10	13	5	40	2	15	20	1607	0.002	18	1529	2.216
2B	20	27	10	40	2	40	25	3594	0.052	15	1969	5.944
3B	50	70	25	40	3	50	1	9989	0.06	33	5439	35.152
4B	100	150	59	40	3	120	39	28120	0.305	37	16389	169
5B	231	331	121	40	4	120	22	43603	0.852	*NI*	*NI*	*NI*
6B	257	362	191	40	5	150	21	87199	1.723	*NI*	*NI*	*NI*
7B	307	439	260	40	7	150	8	119613	2.683	*NI*	*NI*	*NI*
8B	400	600	350	40	7	150	31	152818	2.8	*NI*	*NI*	*NI*
1C	10	13	5	100	2	15	48	1548	0.007	48	1216	4.872
2C	20	27	10	100	2	40	49	3655	0.27	49	2520	15.556
3C	50	70	25	100	5	60	32	9924	0.241	79	5689	101.084
4C	100	150	59	100	5	120	42	27870	0.452	54	18180	441.576
5C	231	331	121	100	7	120	66	45412	1.624	*NI*	*NI*	*NI*
6C	257	362	191	100	12	190	44	91616	3.81	*NI*	*NI*	*NI*
7C	307	439	260	100	12	225	13	149114	5.318	*NI*	*NI*	*NI*
8C	400	600	350	100	15	240	79	108750	8.375	*NI*	*NI*	*NI*

Consider the instances of Group *A* where we have a relatively small number of scenarios (10 scenarios). It is obvious that both algorithms perform well whatever the size of the instance is. So, for instances with small number of scenarios, the greedy heuristic algorithm behaves almost in the same way for all the instances where it determines a solution within a very small consuming time. However, as the size of the studied problem instance increases, the consuming time needed by the tabu search algorithm increases too, though it ameliorates the quality of the solution obviously.

For a medium number of scenarios (40 scenarios) represented by Group *B* instances, we observe that the performance of the heuristic algorithm is almost the same for all the instances, whereas the performance of the tabu search algorithm differs according to the size of the instance i.e. as the number of the vertices of the network increases, the CPU consuming time of this algorithm increases too although there is a high improvement of the quality of the solution. However, we see that there is no rapid improvement for big size instances.

Concerning the last group of instances; Group *C* with a big number of scenarios (100 scenarios), the heuristic algorithm performs rapidly and determines a solution within a very short time for all the instances, while the performance of the tabu search algorithm is affected by the size of the instance and it needs more time and memory to improve the solution found by the heuristic.

A general conclusion is drawn out, the heuristic algorithm is able to determine an initial solution for any problem instance and for any number of scenarios within a very short CPU consuming time. On the other hand, the performance of the developed tabu search algorithm is proportional to the number of scenarios and to the size of the studied problem instance. In other terms, as the number of scenarios and the size of the instance increase, the CPU consuming time of the tabu search algorithm increases too. However, this algorithm is able to improve very well the solution which is determined by the heuristic whenever it is able to ameliorate.

5 CONCLUSIONS

In this paper we studied the Robust Sparse Capacitated Arc Routing Problem under travel costs uncertainty where the uncertainty is represented by a set of scenarios. We study this problem over a sparse network whose maximum degree held by its vertices is 3. We presented a mathematical formulation of this problem, and we developed two algorithms to solve the problem. The first heuristic revealed to be effective

for the determination of a feasible solution for all the studied instances within a very short CPU consuming time whatever the number of scenarios is. The performance of the developed adapted tabu search algorithm is related to two factors: (1) the size of the studied instance i.e. the number of vertices of the network over which the problem is defined and (2) the total number of scenarios. This algorithm starts by an initial solution which is determined by the heuristic algorithm and attempts to improve it. As seen in the previous section, the CPU consuming time needed by the tabu search algorithm extends as the number of the vertices of the network increases and as the number of scenarios augments. The latter algorithm did not succeed to present an improvement of the initial solution for the big size instances under a medium and a big number of scenarios, but at the end we have a robust solution determined by the heuristic even if it is not of very good quality.

REFERENCES

Christiansen, C., Lysgaard, J., and Wfhlk, S. (2009). A branch-and-price algorithm for the capacitated arc routing problem with stochastic demands. *Operations Research Letters*, 37(6):392–398.

Coco, A. A., Solano-Charris, E. L., Santos, A. C., Prins, C., and Noronha, T. (2014). Robust optimization criteria: state-of-the-art and new issuses. Technical report, Université de Technologie de Troyes.

Dantzig, G. and Ramser, J. (1959). The truck dispatching problem. *Management Science*.

Fleury, G., Lacomme, P., and Prins, C. (2004). Evolutionary algorithms for stochastic arc routing problems.

Kasperski, A. and Zieliński, P. (2011). On the approximability of robust spanning tree problems. *Theoretical Computer Science*, 412(4-5):365–374.

Kouvelis, P. and Yu, G. (1997). *Robust discrete optimization and its applications*. Springer-Science+Business Media, B.V.

Mei, Y., Tang, K., and Yao, X. (2010). Capacitated arc routing problem in uncertain environments.

Sbihi, A. (2010). A cooperative local search-based algorithm for the multiple-scenario max-min knapsack problem. *European Journal of Operational Research*, 202(2):339–346.

Shapiro, A., Dentcheva, S., and Ruszczyński, A. (2009). *Lectures on stochastic programming: Modeling and Theory*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

Sungur, I., Ordóñez, F., and Dessouky, M. (2008). A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. *IIE Transactions*, 40(5):509–523.

Wets, R. (2002). Stochastic programming models: Waitand-see versus here-and-now. volume 128.