A Multiple Ant Colony System for a Vehicle Routing Problem with Time Windows and Uncertain Travel Times

Nihat Engin Toklu, Luca Maria Gambardella, and Roberto Montemanni
Dalle Molle Institute for Artificial Intelligence (IDSIA - USI/SUPSI), Galleria 2, 6928 Manno, Switzerland
Email: {engin, luca, roberto}@idsia.ch

Abstract—**In this paper, we study the capacitated vehicle routing problem with time window constraints, under travel time uncertainty. The uncertainty here represents the perturbation on the data caused by the effects of the unpredictable events in the reality, like traffic jams, road constructions, etc. To be able to near-optimally solve the large-instances of this problem without encountering memory errors or without taking too much time, we propose a heuristic approach based on ant colony optimization, which generates multiple solutions at the end of its execution, each solution with a different protection against the uncertainty. The trade-off between robustness and cheapness shown by these generated multiple solutions are then discussed.**

Index Terms—**vehicle routing problem, time window constraints, robust optimization, metaheuristics**

I. Introduction

In the vehicle routing problem (VRP; see the related reading material [1]-[6]) the goal is to transport the goods stored in a depot to the customers waiting at various locations, where the routes of the available are decided in such a way that the total cost of the travels is as low as possible. An extension to VRP is the capacitated VRP (CVRP), where each vehicle has a carrying capacity. In CVRP, one must schedule the vehicles such that, the capacity of each vehicle will be enough to serve all the customers on its route without intermediate visits to the depot. A further extension on CVRP is CVRP with time window constraints (CVRPTW). In CVRPTW, each customer must be served within a time window. If a vehicle arrives earlier than the beginning of a time window, it must wait at that location until the beginning of that time window. If, according to a CVRPTW solution, a vehicle arrives later than the ending of a time window, that solution is declared infeasible.

A classical approach in the field of optimization is to assume that the data is known exactly. With this assumption, the problem data are expressed as fixed numbers. An argument against this kind of assumption is that, in the reality the problem data is subject to uncertainty because of the unpredictable factors in the environment (which are, in the case of VRP, traffic jams, road constructions, etc.). Ignoring this uncertainty might cause undesirable situations: a solution which is optimal according to the optimization model might turn out to be far from optimality, or even infeasible, when applied in the reality. To prevent this undesired situation, alternative schools of thoughts have been gaining popularity in the last decade. One such such school of thought is robust optimization [7]-[10]. In robust optimization, the existence of uncertainty is accepted, and the uncertain data is expressed as discrete collections or continuous intervals of possible value outcomes. Given that a *scenario* is a set of assumptions on which values will be encountered out of these discrete collections or intervals, the goal of robust optimization is to find a solution which does not go too bad and/or become infeasible on most or all scenarios.

In modern robust optimization, an important concept is the *degree of conservativeness*, which means the level of pessimistic thinking of the decision maker, during the optimization process. A non-conservative approach would be to assume that there is no uncertainty-caused data perturbation at all. On the other hand, A fully conservative approach would be to assume that all of the problem data is perturbed by the uncertainty, towards their worst possible values. Finally, a partially conservative approach would be to assume that some of the problem data is perturbed towards their worst possible values, and the rest will be at their better (or best) possible values. As the conservativeness degree increases, solutions that are better protected against the uncertainty can be found. However, in a too conservative solution, the opportunity of being close to optimality is given up in more optimistic scenarios, as the focus is on the pessimistic scenarios. Therefore, it is a trade-off.

In this paper, we follow the school of robust optimization, and we study a CVRPTW with uncertain travel times (CVRPTWU). In CVRPTWU, we express the uncertain travel times as intervals. The challenge in CVRPTWU is to handle the time window constraints, as the feasibility of a solution depends on the uncertain travel times.

Here, we propose an ant colony optimization (ACO; see [11] and [12]) algorithm. ACO can be defined as a

Manuscript received January 5, 2014; revised March 25, 2014.

doi: 10.12720/jtle.2.1.52-58

class of metaheuristic algorithms. The purpose of using a metaheuristic is to make sure that larger problem instances can be solved to near-optimality without demanding too much memory and/or too much execution time. An ACO algorithm simulates the behavior of the ants in the nature on a solution space. The inspiration of ACO is as follows. In the nature, the ants get out of their nests to reach a food source. In the beginning, various ants reach to the food source by using various paths and they mark the path they choose by leaving their pheromones. The ants that choose shorter paths can go back and forth more frequently to the food source, increasing their pheromones on their paths. The other ants that are influenced by the pheromones also get attracted to these shorter paths, leaving their own pheromones, thus increasing the total pheromones on shorter paths even more. Therefore, as the better solutions (shorter paths) get more pheromones, in the end, most of the ants gather around the best solution known so far.

For analyzing the effects of the uncertainty on the problem at hand, a useful thing to have for the decision maker, is a solution pool, in which each solution has its own conservativeness degree. By comparing the solutions in a solution pool to each other, and by evaluating the trade-off shown by these solutions, a decision maker can pick the most practical solution. A popular study in this field is [13], there the authors consider the problems with random variables behaving according to known probability distributions, and they propose a multiobjective, non-dominated sorting genetic algorithm (NSGA-II) based approach generating a population of solutions with various reliabilities (i.e. probabilities of staying feasible). In [14], a robust multiple ant colony system (RMACS) was proposed for generating solution pools for CVRP with uncertain travel costs. In RMACS, ant colony systems (ACS, an elitist ACO variation; see [15]) work concurrently, each colony focusing on a different conservativeness degree. As they work concurrently, these colonies share their best solutions periodically with each other, so that a colony becomes aware if it is stuck on a dominated solution and gets unstuck by importing a better solution from another colony. In the end, the best solutions of these multiple ACSs are collected in a solution pool. In this study, we use RMACS for solving CVRPTWU.

Previous studies on VRP with robust optimization considerations are available in the literature. In [16], a mathematical programming approach is taken to solve a VRP where there is uncertainty in the customer demands. In [17], a robust mathematical model for VRP with deadlines is proposed, and, in [18], robust mathematical models for CVRPTWU are proposed. The methods listed above are exact methods, as they are designed to find the optimal values, given enough time. However, for larger instances, exact methods might end up taking too much time and memory (for example, in [18], instances only up to 50 customers were reported, possibly a limitation brought by the time and memory requirements). In the situations where the decision maker wants a near-optimal solution, within a limited amount of time, without demanding for too much memory, metaheuristic approaches can be used. Within the category of metaheuristics, a previous study is [19], where a VRP with uncertain demands is solved by a particle swarm approach. Our approach that we propose here belongs to the category of metaheuristics. Differently from the study presented in [19], we consider that the uncertainty is in the time windows, not in the demands.

The structure of this paper is as follows. In section II, a more formal problem definition is given. In section III, our proposed metaheuristic is explained. In section IV, we present our experimental results. Finally, in section V, the conclusions are drawn.

II. Problem Definition

Let us now make more formal definitions of CVRPTW and CVRPTWU. We can express these problems in terms of $G = (L, E)$, where L is the set of locations and E is the set of edges representing the paths between locations. The set of locations is defined as $L = \{0, 1, 2, ..., |L|-1\}$, 0 representing the depot, and the other contained integers representing the index numbers of the customer locations. The set of edges is defined as $E = \{(i, j) \mid i, j \in L, i \neq j\}$. For each edge $(i, j) \in E$, the associated data are c_{ij} which represents the cost of traveling from location i to location j; and t_{ij} which represents the time requirement for traveling from location i to location j. Each customer at location $i \in (L \setminus \{0\})$ has a demand d_i. At the depot, the demand is 0, therefore, $d_0 = 0$. A customer at location $i \in (L\setminus\{0\})$ must be served within the time window $[\underline{T}_i;\overline{T}_i]$. If a vehicle arrives at the customer location i before $\underline{T}_i$, it has to wait until $\underline{T}_i$. If a vehicle arrives at the customer location i after $\overline{T}_i$, the solution is declared infeasible. Also, serving a customer at location i takes $\ddot{T}_i$ amount of time. We call $\ddot{T}_i$ the service time at location i. At the depot, the service time is 0, therefore, $\ddot{T}_0 = 0$; also, the time window of the depot is $[0;\infty]$. The set of vehicles is V, and the number of vehicles is $|V|$. The capacity of a vehicle is denoted by Q.

Let us denote a solution for CVRPTW and CVRPTWU by x. According to x, the route of the vehicle $v \in V$ is denoted by $x[v]$. The number of visits by the vehicle v is denoted by $|x[v]|$. The k-th visited location of the vehicle v within the solution x is denoted by $x[v, k]$. In terms of a solution x, let us now define the constraints. A vehicle v must start and end its route at the depot:

$$x[v, 1] = x[v, |x[v]|] = 0 \quad (1)$$

The non-depot locations visited by the vehicle v must be valid customer locations:

$$x[v, k] \in (L\setminus\{0\}) \quad \forall v\in V;\ k \in \{2,3,...,|x[v]|-1\} \quad (2)$$

A vehicle v must not visit a customer more than once:

$$x[v, k] \neq x[v, k'] \quad \forall v \in V;\ k, k' \in \{2,3,...,|x[v]|-1\};\ k \neq k' \quad (3)$$

Two vehicles v, v' must not visit the same customer:

$$x[v, k] \neq x[v', k']$$
$$\forall v, v' \in V; v \neq v';$$
$$k \in \{2,3,...,|x[v]|-1\}; k' \in \{2,3,...,|x[v']|-1\} \quad (4)$$

The total demand of the customers which will be visited by a vehicle $v \in V$ must not exceed Q, the capacity of a vehicle:

$$\sum_{k \in \{2,3,...,|x[v]|-1\}} d_{x[v,k]} \leq Q \quad \forall v \in V \quad (5)$$

In addition to the constraints (1), (2), (3), (4), and (5), there are time window constraints which need to be satisfied. To understand the basic idea, let us first describe the time windows in deterministic (i.e. without the consideration of uncertainty) CVRPTW. Let us express the appearance time of a vehicle $v \in V$ at its k-th visited location, as A_k^v. For the depot, we define the appearance time of a vehicle as 0:

$$A_1^v = 0 \quad \forall v \in V \quad (6)$$

The appearance time of vehicle v to its k-th location ($x[v, k]$) is the sum of its appearance time on the location $x[v, k-1]$, service time of the location $x[v, k-1]$, and the travel time from $x[v, k-1]$ to $x[v, k]$; or the beginning of the time window of the location $x[v, k]$, whichever one is the higher one. So, the appearance times for the next locations of a vehicle v are formulated like this:

$$A_k^v = \max(A_{k-1}^v + \ddot{T}_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]}; \underline{T}_{x[v,k]})$$
$$\forall v \in V; k \in \{2,3,...,|x[v]|\} \quad (7)$$

A vehicle must not appear after the time window ending of a location:

$$A_k^v \leq \overline{T}_{x[v,k]} \quad \forall v \in V; k \in \{1,2,...,|x[v]|\} \quad (8)$$

The cost of a solution x is defined as follows:

$$\text{COST}(x) = \sum_{v \in V} \sum_{k=2}^{|x[v]|} c_{x[v,k-1],x[v,k]}$$

We are finally ready to make a complete definition of the deterministic CVRPTW:

$$\text{CVRPTW} \begin{cases} \text{minimize} & \text{COST}(x) \\ \text{subject to} & (1), (2), (3), (4), (5), (6), (7), \text{ and } (8) \end{cases}$$

In CVRPTWU, however, the constraints (7) and (8) can not be directly expressed, as they depend on the travel times t_{ij}, which are under uncertainty. In CVRPTWU, the uncertain travel times are expressed as $t_{ij} \in [\underline{t}_{ij}; \overline{t}_{ij}] \; \forall (i, j) \in E$.

For handling the time window constraints under uncertainty, let us first define the appearance time of a vehicle $v \in V$ at its k-th visited location, in the best-case scenario (in which all the edge costs are assumed to be at their minimum values), as $\underline{A}_k^v$. For the depot, we define the best-case appearance time of a vehicle as 0:

$$\underline{A}_1^v = 0 \quad \forall v \in V \quad (9)$$

The best-case appearance times for the next locations of a vehicle v are formulated like this (which is equivalent to (7) with A replaced by $\underline{A}$):

$$\underline{A}_k^v = \max(\underline{A}_{k-1}^v + \ddot{T}_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]}; \underline{T}_{x[v,k]})$$
$$\forall v \in V; k \in \{2,3,...,|x[v]|\} \quad (10)$$

At least in the best-case scenario, a vehicle must satisfy all the deadlines imposed by the time windows:

$$\underline{A}_k^v \leq \overline{T}_{x[v,k]} \; \forall v \in V; k \in \{1,2,...,|x[v]|\} \quad (11)$$

While the constraints (11) secure the solution feasibility in the best-case scenario, there might be other scenarios in which some deadlines are violated. Since we would like to minimize the deadline violation possibilities, we also need a function TIMEWINDOWVIOLATIONPENALTY, which returns a penalty value when a scenario in which a deadline is violated.

The existence of a time window violating scenario can be checked by calculating the maximum possible latency of a vehicle to a destination. In more details, considering a solution x, with maximum possible latency, if a vehicle v arrives to its k-th destination later than $\overline{t}_{x[v,k]}$, then it becomes apparent that there is indeed at least one scenario in which there is time window violation. For the purpose of calculating the maximum latency of a vehicle to a destination, let us use a function originally proposed in [18], that we call ML in this study. The function ML is configurable in terms of conservativeness: it depends on an argument Γ, which tells the function to assume that Γ number of the edges on the route will be perturbed towards their worst case values in terms of their travel time requirements, and the rest of the edges will stay at their best case values. The function ML can be formulated as follows:

$$\text{ML}(x, v, k, \Gamma) =$$
$$\begin{cases} 0 & \text{if } k = 1 \\ \max\left(\underline{T}_{x[v,k]}, \text{ML}(x, v, k-1, 0) + \ddot{T}_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]}\right) & \text{if } 2 \leq k \leq |x[v]| \text{ and } \Gamma = 0 \\ \max\Big(\underline{T}_{x[v,k]}, \\ \quad \text{ML}(x, v, k-1, \Gamma-1) + \ddot{T}_{x[v,k-1]} + \overline{t}_{x[v,k-1],x[v,k]}, \\ \quad \text{ML}(x, v, k-1, \Gamma) + \ddot{T}_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]} \Big) & \text{if } 2 \leq k \leq |x[v]| \text{ and } 1 \leq \Gamma \leq k-1 \\ -\infty & \text{if } 1 \leq k-1 \leq |x[v]| \text{ and } \Gamma \geq k \end{cases}$$

Apart from the special cases, it can be seen that ML is a recursive function: it depends on the results ML on the previous destinations of the route, to find the maximum latency for the k-th destination.

Now, let us define another function called ISLATE, which checks, by using the function ML, if there is a scenario in which the vehicle v of solution x is late for its k-th destination. For the function ISLATE, we define a

conservativeness degree relative to the length of the route leading to the k-th destination, controlled by the parameter Ψ. When $\Psi = 0$ there is no conservativeness at all and all the edges on the path to $x[v, k]$ are considered at their best case values in terms of travel time requirements. On the other hand, when $\Psi = 1$ there is full conservativeness and all the edges on the path to $x[v; k]$ are considered at their worst case values. When $\Psi = 0.5$, half of the edges on the path are considered at their worst case values, and the other half are considered at their best case values. The function IsLate can be formulated as follows:

$$\text{IsLate}(x, v, k, \Psi) = \begin{cases} 1 & \text{if } \text{ML}(x, v, k, \lceil \Psi \cdot (|k-1|) \rceil) > \overline{T}_{x[v,k]} \\ 0 & \text{otherwise} \end{cases}$$

Now, we are ready to make a definition of the function Time Window Violation Penalty as:

$$\text{TimeWindowViolationPenalty}(x, \Psi) = \sum_{v \in V} \sum_{k=1}^{|x[v]|} \left(\Pi_{x[v,k]} \cdot \text{IsLate}(x, v, k, \Psi) \right)$$

where Π_i is a penalty factor, to be decided by the decision maker, according to the importance of the location $i \in L$.

We can now sum up the definition of CVRPTWU as follows:

$$\text{CVRPTWU} \begin{cases} \text{minimize} \\ \quad (\text{Cost}(x) \\ \quad\quad + \text{TimeWindowViolationPenalty}(x, \Psi)) \\ \text{subject to} \\ \quad (1), (2), (3), (4), (5), (9), (10), \text{ and } (11) \end{cases}$$

III. The Approach

In this section, we first explain the ACS, and then the RMACS approach, in which multiple ACSs are executed concurrently.

A. The Ant Colony System

Let us start by making a general definition of an ACO algorithm. On a combinatorial optimization problem like the traveling salesman problem or the VRP, an ACO algorithm generates its solutions by using artificial ants. Artificial ants "walk" on the solution space, adding a new decision to the solution at each step. When an ant completes its walk, its solution is evaluated. After the evaluation, artificial pheromones are left on the path that was chosen by the ant. The amount of these artificial pheromones depends on the result of the evaluation: more pheromones are put for the better solutions. In the case of a transportation problem like the VRP, the pheromones are put on the edges that were chosen by the artificial ant, and more pheromones are put on the edges of a solution with lesser total travel cost. The function of the pheromones is as follows: the artificial ants of the next iterations, while making their decisions, are attracted towards the choices that are marked by the pheromones. More pheromones mean more attraction. This attraction causes the ants to do local searches around successful solutions known so far. After enough number of iterations, artificial ants converge to a successful near-optimal solution.

The ACS is an elitist ACO algorithm, based on the one discussed in [15]. The term elitism means that only the ants that have improved the best-known solutions during the execution of the algorithm, are allowed to put pheromones. The effect of this elitism is that, the ants are encouraged to do their local searches only around the best solutions.

We now give the technical details of the ACS. At first, an initial solution x^{init} is generated according to the nearest neighbourhood heuristic (NNH; see [20]). At the beginning, this solution x^{init} is also the best-known solution so far. Therefore, the variable that stores the best solution, x^{best}, is equivalent to x^{init} at the beginning. The ACS then iteratively activates generations of ants. At each generation, Ω ants are activated. If an ant finds a solution x^{better} that is better than x^{best}, then that better solution is assigned s the new best solution (i.e. $x^{best} \leftarrow x^{better}$). New generations are activated iteratively, until the execution time limit is reached.

An artificial ant constructs a solution by picking customer locations one by one, by also adding the depot location occasionally. The depot location must also be picked for the very beginning and ending of the solution. The resulting solution vector specifies which vehicle visits which customers. A visit to the depot in the middle of the solution signifies that the current vehicle is returning to the depot and another vehicle is now to be considered. For example, assuming that $L = \{0, 1, 2, 3\}$ and $|V| = 2$ if an artificial ant constructs a solution $x = [0, 1, 2, 0, 3, 0]$, this means that the ant is suggesting that the first vehicle should visit the customer 1 and then the customer 2, and the second vehicle should visit the customer 3. At each step, an artificial ant chooses its next location from the visitable locations, where a visitable location means a location that can be added without violating the capacity constraint of the current vehicle, and which has not been added to the solution vector yet if it is a customer location. Considering an artificial ant that has added location i most recently to the solution vector, and now has to decide its next location j, we now make the following definitions:

N_w: The set of visitable locations for the ant w;

λ_{ij}: Euclidean distance between the locations i and j;

$\eta_{ij} = 1 / \lambda_{ij}$: heuristic distance-wise attractiveness factor (smaller distances give higher attractiveness values);

τ_{ij}: The amount of pheromone left on the edge (i, j);

β: A parameter to configure the importance of distance-wise attractiveness factor while an ant chooses its location;

α: A parameter to configure the balance of importance between exploration and exploitation. With probability α, an ant chooses to do exploitation and

picks the location j by following the edge (i, j) that gives the maximum value for $\tau_{ij} \cdot [\eta_{ij}]^{\beta}$. On the other hand, with probability 1-α, the ant chooses to do exploration. In the case of exploration, the probability for an ant w to pick location j is formulated as follows:

$$p_{ij}^{w} = \begin{cases} \dfrac{\tau_{ij} \cdot [\eta_{ij}]^{\beta}}{\sum_{j' \in N_w} \tau_{ij'} \cdot [\eta_{ij'}]^{\beta}} & \text{if } j \in N_w \\ 0 & \text{otherwise} \end{cases}$$

At the beginning of the ACS execution, each edge (i, j) is given the same amount of pheromone $\tau_0 = 1/(|L| \cdot \text{Ç}(x^{init}, \Psi))$, where $\text{Ç}(x) = \text{COST}(x) + \text{TIMEWINDOW VIOLATIONPENALTY}(x, \Psi)$

During the execution of the ACS, the pheromones become updated in two ways: local update and global update. The local update is a decrease of pheromones over the edges that are walked by an ant, so that the other ants in the same generation will be discouraged to make very similar decisions. This update is done according to the formulation $\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0$, where ρ is a parameter which configures the amount of pheromone decrease imposed by the local update. The global update is an increase of pheromones over the edges used by x^{best} at the end of each iteration, to attract the ants of the next generations towards the choices of x^{best}. The global update is done according to the formulation: $\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho / \text{Ç}(x^{best}, \Psi)$.

When the construction of a solution is complete, a popular local-search algorithm called *3-opt* is executed on the solution. The details of *3-opt* can be found in [21], [22], [20].

B. The Robust Multiple Ant Colony System Approach

The RMACS approach depends on a set of parameters $S^{\Psi} = \{\Psi_1, \Psi_2, ..., |S^{\Psi}|\}$, which represents the collection of the conservativeness degrees that are interesting to the decision maker. In the RMACS approach, $|S^{\Psi}|$ number of ACSs are activated, each ACS focusing on a conservativeness degree within the set S^{Ψ}. Until Δ seconds have passed, each ACS works by itself. After Δ seconds, ACSs start communicating with each other by using a shared memory. This communication is done as follows. At each δ seconds, each ant colony *ColonyX* exports its best solution to the shared memory, and also scans the best solutions exported by the other ant colonies, to see if another colony *ColonyY* has a better solution than the best solution of *ColonyX* according to the conservativeness degree of *ColonyX*. If the best solution of *ColonyY* is indeed better, *ColonyX* imports the solution of *ColonyY* by making one of its artificial ants repeat the decisions of written in the solution of *ColonyY*. With the help of this solution sharing mechanism, an ant colony can realize that it is stuck on a dominated solution and can get unstuck by importing a better solution and by improving that solution according to its own conservativeness degree.

IV. EXPERIMENTAL RESULTS

In this section, we present our experiments. In these experiments, we generate solution pools over various CVRPTWU instances and analyze the effects of the travel time uncertainty.

For the experiments, the Homberger instances (available online at [23]) with 200 customers and with vehicle capacity of 200 were used. These instances were originally created for the deterministic CVRPTW in which travel time uncertainty is not considered. To convert these instances to CVRPTWU instances, the following procedure was used: for each edge (i, j), the interval for t_{ij} becomes $[c_{ij}; c_{ij} \cdot \text{RND}(1, 1.1)]$, where RND($a$, b) means a random real number between a and b. For each location i, the time window violation penalty factor is given according to: $\Pi_i = (20 \cdot \bar{t}_{0,i})$.

The RMACS approach was implemented in C, and was tested on the considered instances on a computer with Intel Core 2 Duo P9600 @ 2.66GHz with 4GB of RAM, with an execution time limit of 900 seconds. The settings are: α=0.99, β=1, ρ=0.1, Δ=700, δ=5. The considered conservativeness degrees are S^{Ψ}= {0, 0.25, 0.5, 1}.

The results are given in Table I. In the table, for each instance, a solution pool is presented, where each row represents a solution, Ψ represents the conservativeness degree configuration of the ant colony that gave the solution. To see how a solution would behave in different situations, different conservativeness degree assumptions, denoted by Υ, were made, and the solution was evaluated according to those assumptions. The results of the evaluations (i.e. the total travel cost plus the time window violation penalties, Ç(x) for each solution x) are reported. In the table, from the results, we can detect some patterns. In some instances like c1_210, rc1_210, r1_2_2, rc1_2_6, solution pools with solutions varying according to the conservativeness degree are found. The interesting behaviour here is that, for Υ = 0, the cheapest solution is found, but the objective value of this cheap solution is quickly burdened by the time window violation penalties as the Υ value increases. On the other hand, with Ψ=1, more expensive but completely robust solutions are found. Intermediate solutions can be seen with Ψ=0.25 and/or Ψ=0.5, Therefore, the trade-off between the cheapness and robustness is visible. This trade-off is also visible in instances like r1_2_1 and rc1_2_2, but the pattern in these examples is not exactly the same: with Ψ=1, the time window violation penalty is minimized, but not completely gone. In some cases, there is a single solution dominating the entire solution pool, as can be seen in, for example, r1_2_3 and rc1_2_9. Finally, it can be noted that, there might be some inconsistencies in the generated solution pools. For example, in rc1_2_5, the solution with Ψ=0.25 is better than the solution with Ψ=0 under the assumption Υ =0. The explanation for this behaviour might be that the ant colony working on Ψ=0.25 found a solution which dominates the solution of the ant colony working on Ψ =0, but the global execution time was reached before these colonies exchanged information. However, even with inconsistencies, a tradeoff can be

seen in the solution pool of rc1_2_5 between the solution Ψ=0.25 (which dominates the solution Ψ=0), and the solution Ψ=0.5 (which dominates the solution Ψ=1).

TABLE I: SOLUTION POOLS OBTAINED OVER INSTANCES WITH 200 CUSTOMERS

		Solution Pool						Solution Pool			
Instance	Ψ	Υ =0	Υ =0.25	Υ =0.5	Υ =1	Instance	Ψ	Υ =0	Υ =0.25	Υ =0.5	Υ =1
r1_2_1	0	5110	19494.35	20142.4	26099.18	r1_2_7	0	3853	10125.12	11581.65	13599.23
	0.25	5254	7849.96	10959.84	17071.08		0.25	3971	3971	3971	3971
	0.5	5441	8219.64	8219.64	11171.77		0.5	3971	3971	3971	3971
	1	5495	8273.64	8273.64	9746.38		1	3971	3971	3971	3971
rc1_2_2	0	3772	15635.07	15635.07	15635.07	c1_2_4	0	2754	4139.71	4139.71	4139.71
	0.25	4559	4559	9825.95	14576.69		0.25	2815	2815	2815	2815
	0.5	3792	6275.62	6275.62	6275.62		0.5	2815	2815	2815	2815
	1	3792	6275.62	6275.62	6275.62		1	2815	2815	2815	2815
r1_2_3	0	4175	4175	4175	4175	rc1_2_9	0	3724	3724	3724	3724
	0.25	4175	4175	4175	4175		0.25	3724	3724	3724	3724
	0.5	4175	4175	4175	4175		0.5	3724	3724	3724	3724
	1	4175	4175	4175	4175		1	3724	3724	3724	3724
c1_2_9	0	2711	4522.33	5876.63	5876.63	r1_2_8	0	3350	3350	3350	3350
	0.25	2734	2734	2734	2734		0.25	3350	3350	3350	3350
	0.5	2734	2734	2734	2734		0.5	3350	3350	3350	3350
	1	2734	2734	2734	2734		1	3350	3350	3350	3350
rc1_2_7	0	3797	5161.16	5161.16	5161.16	rc1_2_6	0	3838	5437.27	9803.87	9803.87
	0.25	3810	3810	3810	3810		0.25	3847	3847	6316.43	7742.8
	0.5	3810	3810	3810	3810		0.5	3849	3849	3849	3849
	1	3810	3810	3810	3810		1	3849	3849	3849	3849
c1_2_6	0	2633	7175.61	7175.61	8180.72	r1_210	0	3961	3961	3961	3961
	0.25	2710	2710	3704.32	4709.43		0.25	3961	3961	3961	3961
	0.5	2711	2711	2711	3716.11		0.5	3961	3961	3961	3961
	1	2784	3897.46	3897.46	3897.46		1	3961	3961	3961	3961
c1_2_3	0	2747	2747	2747	2747	rc1_2_8	0	3866	3866	3866	3866
	0.25	2747	2747	2747	2747		0.25	3866	3866	3866	3866
	0.5	2747	2747	2747	2747		0.5	3866	3866	3866	3866
	1	2747	2747	2747	2747		1	3866	3866	3866	3866
c1_2_1	0	2637	2637	4019.08	4019.08	c1_2_5	0	2634	3929.72	3929.72	5204.48
	0.25	2637	2637	4019.08	4019.08		0.25	2636	2636	2636	3910.76
	0.5	2643	2643	2643	2643		0.5	2636	2636	2636	3910.76
	1	2643	2643	2643	2643		1	2645	2645	2645	2645
rc1_2_5	0	4200	19002.08	21417.9	25143.77	rc1_210	0	3529	8361.52	9377.92	9377.92
	0.25	4148	4148	4148	5717.38		0.25	3619	3619	4950.9	4950.9
	0.5	4506	4506	4506	4506		0.5	3621	3621	3621	3621
	1	4514	4514	4514	4514		1	3621	3621	3621	3621
rc1_2_3	0	3547	3547	3547	3547	c1_210	0	2675	5820.03	5820.03	7138.31
	0.25	3547	3547	3547	3547		0.25	2770	2770	4097.4	5321.76
	0.5	3547	3547	3547	3547		0.5	2828	2828	2828	2828
	1	3547	3547	3547	3547		1	2828	2828	2828	2828
c1_2_8	0	2638	5567.98	5567.98	5567.98	c1_2_7	0	2628	30076.99	30076.99	30076.99
	0.25	2888	2888	2888	2888		0.25	2640	2640	2640	2640
	0.5	2888	2888	2888	2888		0.5	2640	2640	2640	2640
	1	2632	2632	2632	2632		1	2640	2640	2640	2640
r1_2_6	0	4577	22366.84	22366.84	22366.84	c1_2_2	0	2830	6037.26	7958.18	8666.53
	0.25	4708	4708	4708	5983.74		0.25	2945	2945	3604.84	5190.4
	0.5	4708	4708	4708	5983.74		0.5	3076	3076	3076	3076
	1	4713	4713	4713	4713		1	2905	2905	2905	2905
r1_2_4	0	3617	3978.65	3978.65	3978.65	rc1_2_1	0	3995	16403.61	17187.83	22312.59
	0.25	3629	3629	3629	3629		0.25	4265	4958.12	9095.4	11738.39
	0.5	3629	3629	3629	3629		0.5	4458	5848.23	5848.23	6668.15
	1	3618	3618	3618	3618		1	4390	6082.6	6082.6	6902.53
r1_2_2	0	4713	11063.71	17508.34	17508.34	r1_2_9	0	4453	10911.1	10911.1	10911.1
	0.25	4723	4723	7351.3	7351.3		0.25	4684	4684	4684	4684
	0.5	4775	4775	4775	4775		0.5	4684	4684	4684	4684
	1	4775	4775	4775	4775		1	4684	4684	4684	4684
r1_2_5	0	4769	15842.03	18544.94	21465.66	rc1_2_4	0	3395	3395	3395	3395
	0.25	5150	5150	5150	5150		0.25	3395	3395	3395	3395
	0.5	5150	5150	5150	5150		0.5	3395	3395	3395	3395
	1	5150	5150	5150	5150		1	3395	3395	3395	3395

V. CONCLUSIONS

An implementation of RMACS was proposed for solving the CVRPTWU. The RMACS approach generates solution pools in which each solution has a different robustness against the uncertainty. By analyzing these solution pools, a decision maker can see the tradeoff between cheapness and robustness, and this can help her/him in making a practical decision.

ACKNOWLEDGMENT

N. E. Toklu was supported by Hasler Stiftung through project 13002: "Matheuristic approaches for robust optimization".

REFERENCES

[1] G. B. Dantzig and J. H. Ramser. "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
[2] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 3, pp. 345–358, 1992.
[3] P. Toth and D. Vigo, *The Vehicle Routing Problem*, P. Toth and D. Vigo, Eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.
[4] R. Baldacci, N. Christofides, and A. Mingozzi, "An exact algorithm for the vehicle routing problem based on the set

partitioning formulation with additional cuts," *Mathematical Programming*, vol. 115, no. 2, pp. 351–385, 2008.
[5] B. L. Golden, S. Raghavan, and E. A. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*, vol. 43, Springer, 2008.
[6] R. Baldacci, P. Toth, and D. Vigo, "Exact algorithms for routing problems under vehicle capacity constraints," *Annals of Operations Research*, vol. 175, no. 1, pp. 213–245, 2010.
[7] A. L. Soyster, "Convex programming with set-inclusive constraints and applications to inexact linear programming," *Operations Research*, vol. 21, no. 5, pp. 1154–1157, 1973.
[8] P. Kouvelis and G. Yu, *Robust Discrete Optimization and Its Applications*, Kluwer Academic Publishers, 1997.
[9] A. Ben-Tal and A. Nemirovski, "Robust solutions of linear programming problems contaminated with uncertain data," *Mathematical Programming*, vol. 88, no. 3, pp. 411–424, 2000.
[10] D. Bertsimas and M. Sim, "Robust discrete optimization and network flows," *Mathematical Programming*, vol. 98, no. 1, pp. 49–71, 2003.
[11] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," Technical Report, Dipartimento di Elettronica, Politecnico di Milano, 1991.
[12] M. Dorigo, "Learning and natural algorithms," PhD dissertation, Politecnico di Milano, 1992.
[13] K. Deb, S. Gupta, D. Daum, J. Branke, A. K. Mall, and D. Padmanabhan, "Reliability-based optimization using evolutionary algorithms," *IEEE Transactions on Evalutionary Computations*, vol. 13, no. 5, 2009.
[14] N. E. Toklu, R. Montemanni, and L. M. Gambardella, "A robust multiple ant colony system for the capacitated vehicle routing problem," in *Proc. IEEE International Conf. on Systems, Man, and Cybernetics*, 2013.
[15] L. M. Gambardella, É. Taillard, and G. Agazzi, "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows," in *New Ideas in Optimization*, McGraw- Hill, 1999, pp. 63–76.
[16] I. Sungur, F. Ordóñez, and M. Dessouky, "A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty," *IIE Transactions*, vol. 40, no. 5, pp. 509–523, 2008.
[17] C. Lee, K. Lee, and S. Park, "Robust vehicle routing problem with deadlines and travel time/demand uncertainty," *Journal of the Operational Research Society*, vol. 63, no. 9, pp. 1294–1306, 2011.
[18] A. Agra, M. Christiansen, R. Figueiredo, Lars M. Hvattum, M. Poss, and C. Requejo, "The robust vehicle routing problem with time windows," *Computers and Operations Research*, vol. 40, no. 3, pp. 856–866, 2013.
[19] B. F. Moghaddam, R. Ruiz, and S. J. Sadjadi, "Vehicle routing problem with uncertain demands: An advanced particle swarm algorithm," *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 306–317, 2012.
[20] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study in local optimization," *Local Search in Combinatorial Optimization*, pp. 215–310, 1997.
[21] G.A. Croes. "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, nol. 6, pp. 791–812, 1958.
[22] S. Lin. "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.
[23] NEO Networking and Emerging Optimization. (2012). Capacitated vrp with time windows instances. [Online]. Available: http://neo.lcc.uma.es/vrp/vrp-instances/ capacitated-vrp-with-time-windows-instances/

Nihat Engin Toklu received his Bachelor's degree in Computer Engineering from Eastern Mediterranean University in Famagusta, Northern Cyprus in 2007, and his Master's degree in Mechatronics Engineering from Dokuz Eylül University in Izmir, Turkey in 2009. His research interests are metaheuristics, matheuristics and robust optimization.
Affiliated with IDSIA and the Informatics department of University of Lugano in Switzerland, he is currently pursuing a Ph.D. degree.

Luca Maria Gambardella received a Scientific License at Scientific Liceum G.B. Grassi, Saronno, Italy, in 1980, and a Laurea in Information Science in Università degli Studi di Pisa in 1985. His research on bio-inspired distributed optimization algorithms has led to the publication of several state-of-the-art methodologies for solving popular combinatorial optimization problems like the traveling salesman problem and the vehicle routing problem with time windows, the quadratic assignment problem, the sequential ordering problems, and the flexible job shop problem.
Since 1995, he is the co-director of IDSIA, and is also responsible for the Intelligence Systems masters program at the Informatics department of the University of Lugano.

Roberto Montemanni obtained a Laurea degree in Computer Science from the University of Bologna, Italy in 1999 and a Ph.D. in Applied Mathematics from the University of Glamorgan, UK in 2002. His research interests are mathematical programming modeling and development of heuristic algorithms for solving optimization problems.
He is Professor of Advanced Algorithms at the University of Applied Sciences of Southern Switzerland, and Lecturer at the University of Lugano, Switzerland. He also holds a Senior Researcher position at IDSIA.