An Enhanced Ant Colony System for the Team Orienteering Problem with Time Windows

R. Montemanni, D. Weyland, L.M. Gambardella
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Galleria 2, CH-6928 Manno, Switzerand
{*roberto, dennis, luca*}*@idsia.ch*

***Abstract*—The Team Orienteering Problem with Time Windows (TOPTW) is a combinatorial optimization problem arising both in industrial scheduling and in transportation. Ant Colony System (ACS) is a well-known metaheuristic framework, and many efficient algorithms for different optimization problems - among them the TOPTW - have been derived from this general framework. In this paper some directions for improving an ACS algorithm for the TOPTW recently appeared in the literature are identified. The resulting algorithm, called Enhanced Ant Colony System is experimentally shown to be extremely effective on some well-known benchmark instances available in the literature.**

***Keywords*-combinatorial optimization; metaheuristics; ant colony optimization; team orienteering problems.**

I. Introduction

The *Team Orienteering Problem* (TOP) was first studied in Butt and Cavalier [1] and in Chao et al. [2]. With some elementary considerations (see Montemanni and Gambardella [5]), it is easy to transform paths into tours. Therefore, the TOP might be seen as a member of the wide family of Vehicle Routing Problems with Profits. For this reason, competitors are often referred to as vehicles, the team as the fleet, points as customers, and the starting/ending point as the depot. The TOP has been recognized as a model of many different real applications, such as the multi-vehicle version of the home fuel delivery problem, the recruiting of college football players, the sport game of team orienteering, some applications of pickup or delivery services involving the use of common carriers and private fleets and the service scheduling of routing technicians (Chao et al. [2]). Several variants of the TOP exist and one of them takes Time Window constraints into account (TOPTW). Some metaheuristic approaches for the TOPTW are discussed in Vansteenwegen et al. [8] and in Tricoire et al. [7].

Ant Colony System (ACS) is a well-known metaheuristic metaphor, and has been successfully applied to many combinatorial optimization problems, and to the TOPTW in particular (Montemanni and Gambardella [5]). Although successful, the ACS paradigm presents some drawbacks when applied to the TOPTW. In this paper we critically analyze the method, and we propose some refinements to overcome the weak point identified.

II. Problem description

The TOPTW is defined as follows. We are given a complete undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes, and E is the set of edges. A positive weight t_{ij} is associated with each edge, representing the travel time between nodes i and j. For each node $i \in V$ we have the following data: p_i is a positive profit that is collected the first time the node is visited, $[a_i, b_i]$ is a time window defining the feasible arrival time at the node and s_i is a non-negative service time, that is the amount of time which is spent to visit the node. Two special nodes, numbered 1 and n, where $n = |V|$, are the endpoints of the path to be computed. We have $p_1 = p_n = 0$, $s_1 = s_n = 0$, $a_1 = a_n = 0$ and $b_1 = b_n = T$, where T is the maximum duration of the tour, which is equal to the maximum feasible arrival time at node n, that is $T = \max_{i \in V \setminus \{1,n\}}\{b_i + s_i + t_{in}\}$. The TOPTW requires the computation of a set $\mathcal{P}$ of m non-overlapping (apart from origin and destination) elementary paths, such that each path $k \in \mathcal{P}$ is defined as an ordered sequence of nodes starting from node 1 and ending at node n, and the total price collected at the visited nodes is maximized. Given a solution, we indicate with v_i the arrival time at node i.

III. An ACS algorithm for the TOPTW

A. General description of the ACS algorithm

The ACS algorithm is based on a computational paradigm inspired by the way real ant colonies function. The main underlying idea was that of parallelizing search over several constructive computational threads. A dynamic memory structure, which incorporates information on the effectiveness of previously obtained results, guides the construction process of each thread. The behavior of each single agent is inspired by the behavior of real ants. The main element of this metaheuristic algorithm are *ants*, simple computational agents that individually and iteratively construct solutions for the problem. Intermediate partial problem solutions are seen as *states*; each ant moves from a state a to another state b, corresponding to a more complete partial solution. At each step, every ant computes a set of feasible expansions to its current state and moves to one of these probabilistically, according to a probability distribution specified as follows. For ant k the probability p^k_{ab} of moving from state a to state b depends on the combination of two values: the *attractiveness* μ_{ab} of the move, as computed by some heuristic, indicating the *a priori* desirability of that move; the *trail level* τ_{ab} of the move, indicating how proficient it has been in the past to make that particular move; it represents therefore an *a*

978-0-7695-4443-4/11 $26.00 © 2011 IEEE
DOI 10.1109/ISCCS.2011.95

posteriori indication of the desirability of that move. Trails are updated at each iteration, increasing the level of those that corresponding to moves included in good solutions, while decreasing all the others. The specific formula for defining the probability distribution at each move makes use of a set $tabu_k$ which indicates a problem-dependent set of infeasible moves for ant k.

With probability q_0 the next node to visit after node a for ant k is chosen as the node b, $b \notin tabu_k$, for which the result of $(\tau_{ab})\xi+(\mu_{ab})(1-\xi)$ is highest (deterministic rule), while with probability $1-q_0$ the node b is chosen with a probability given by

$$p_{ab}^k = \begin{cases} \frac{(\tau_{ab})\xi+(\mu_{ab})(1-\xi)}{\sum_{c\notin \text{tabu}_k}((\tau_{ac})\xi+(\mu_{ac})(1-\xi))} & \text{if } b \notin tabu_k \\ 0 & \text{otherwise} \end{cases}$$

where the sum is over all the feasible moves, ξ is a parameter controlling the relative importance of the trail τ_{ab} of move (a,b) versus the actual attractiveness μ_{ab} of the same move. In this manner p_{ab}^k is a trade-off between the apparent desirability and information from the past. The phase in which ants build up their solutions is usually referred to as *constructive phase*. Every time all the m ants forming the population have completed a solution, the trail level of each move (a,b) of the best solution retrieved so far is reinforced. The rationale is that in this way a *preferred solution* is memorized in the pheromone trail matrix, and future ants will use this information to generate new solutions in a neighborhood of this preferred solution. We refer to the most profitable solution generated since the beginning of the computation as $BestSol$, and to its cost as $CostBest$. For all $(a,b) \in BestSol$, we have the following formula for pheromone update:

$$\tau_{ab} = (1-\rho)\tau_{ab} + \rho \cdot CostBest$$

where ρ is a parameter regulating how strong the pheromone trace left by the best solution is. Pheromone is also removed from visited arcs during solution building, according to the following rule:

$$\tau_{ab} = (1-\psi)\tau_{ab} + \psi \cdot \tau_{\text{init}}$$

where ψ is a parameter regulating the evaporation of the pheromone trace over time, while τ_{init} is the initial value of trails.

In the current view of ACS algorithms, once an ant has finished constructing its solution, and prior to the evaluation of the solution, a *local search* is generally applied to improve this solution.

B. Application to the TOPTW

The ACS algorithm discussed in Montemanni and Gambardella [5] is based on a modification of the TOPTW problem. The *Hierarchical TOPTW* (HTOPTW) requires the computation of an ordered list of non-overlapping (apart from origin and destination) elementary paths $\mathcal{L} = (P_1, P_2, \ldots, P_{|\mathcal{L}|})$, where each $P_k \in \mathcal{L}$ is defined as an ordered sequence of nodes starting from node 1, ending at node n, such that $a_i \leq v_i \leq b_i \ \forall i \in P_k$; $\forall P_k, P_h \in \mathcal{L}$, P_k and P_h have in common only nodes 1 and n and $\bigcup_{P_k \in \mathcal{L}} P_k = V$. For each pair of nodes $\{i,j\}$ consecutively visited along the same $P_k \in \mathcal{L}$ we have $v_j = \max\{v_i + s_i + t_{ij}, a_j\}$, with $v_1 = 0$ on each path $P_k \in \mathcal{L}$. Finally note that we set $|\mathcal{L}|$ equal to an upper bound of the number of paths required to visit all the customers (e.g. n), and therefore empty tours are allowed in solutions. The objective function of HTOPTW can be defined as follows:

$$\max \left(\sum_{P_k \in \mathcal{L}, k \leq m} \sum_{(i,j)\in A} p_i x_{ij}^{P_k} + \sum_{P_k \in \mathcal{L}, k > m} M^{m-k} \sum_{(i,j)\in A} p_i x_{ij}^{P_k} \right)$$

such that $a_i \leq v_i \leq b_i$, $\forall P_k \in \mathcal{L}, \forall i \in P_k$. Note that M is an arbitrarily large number. The problem HTOPTW can be seen as a generalization of TOPTW where a set of non-overlapping tours are optimized in a hierarchical fashion. Note that the first m tours represent the solution to the original TOPTW problem, while the remaining tours optimize over the feasible nodes that are not in the solution of the TOPTW, in a hierarchical way. From an optimization point of view, the rationale behind the introduction of the problem HTOPTW for solving the TOPTW, is that keeping a hierarchic set of non-overlapping tours helps to have good fragments of tours placed in the $(m+1)$-st to last tours. These prepared fragments are used by the local search procedure to perform exchanges/insertions, aiming at improving the quality of the hierarchy of tours, and - as a side effect - of the first m tours.

The algorithm uses a solution model in which each ant builds a single, giant tour (see Gambardella et al. [4]). A solution is represented as follows: nodes 1 and n are unified into a unique depot node, with outgoing travel times corresponding to node 1, and incoming travel times corresponding to node n. The depot with all its connections is then duplicated a number of times equal to the number of feasible nodes of the problem. Distances between copies of the depot are set to zero. In such a model, each time a copy of the depot is reached, tour duration is set back to zero. The use of the giant tour representation makes the HTOPTW problem closer to a traditional traveling salesman problem. A feasible solution is a tour that visits all the nodes exactly once. Note that in case of too many duplicated depots, we will have dummy arcs of type *(depot, depot)* visited by the solution.

Given the implementation choices above, the construction phase is directly borrowed from that of the ACS for the *Vehicle Routing Problem with Time Windows* described in Gambardella et al. [4]. In practice, each ant iteratively starts from node 1 and adds new nodes until all nodes have been visited. At node i, an ant applies the transition rule described before, where the set of feasible nodes takes of course into account time windows. We refer the interested reader to Montemanni and Gambardella [5] for a detailed description of the heuristic criterion for nodes desirability implemented in the algorithm.

The local search procedure implemented is a specialized version of the CROSS exchange procedure (Gambardella et al. [4]). The procedure is based on the exchange of

two sub-chains of customers of the giant tour. One of the two sub-chains can eventually be empty, implementing therefore a more traditional insertion routine.

IV. An Enhanced ACS for the TOPTW

With reference to the general ACS paradigm described in Section III-A, we can observe that the constructive phase of the algorithm carries out both diversification (exploring new regions of the search space) and intensification (searching very deeply a given region of the search space): the original ACS algorithm did not consider any local search. The constructive phase of ACS is therefore able to generate improving solutions that are in a neighborhood of the best solution computed so far. On the other hand, the local search procedure is considered as a strong intensification process, able to bring each solution computed by the artificial ants to its local minimum. However, in presence of a strong local search procedure - this is the case for the TOPTW - the role of the construction phase is less prominent and has to be revised, since the local search itself is able to efficiently cover vast regions of the search space in its intensification process. In particular the constructive phase could drop any kind of intensification, concentrating only on a (milder) form of diversification. This is the main consideration at the basis of the EACS algorithm we propose for the TOPTW, together with the observation that the local search itself can be better integrated within the method.

Two operations to enhance the performance of ACS are suggested for the TOPTW: the first one regards the constructive phase, and the second the integration between the constructive phase itself, and the local search procedure. Thanks to the two enhancements we propose to heuristically limit the search space considered by the algorithm, the quality of the solution produced by the EACS will be shown to be higher than those of the classic ACS for the TOPTW, due to the remarkable speed-up guaranteed by the enhancements introduced.

One known drawback of the ACS approach is the large total running time required to build new solutions by each artificial ant. Let n be the number of steps necessary to build a solution. The constructive process takes time $O(n)$ to choose the next node to visit for each of the $O(n)$ steps required. This can be too time consuming for some problems, like the TOPTW considered in this study. Taking into account the general considerations made before about the role of the constructive phase, which should drop any intensification capability and only guarantee a form of diversification, we propose to modify the ACS algorithm in two directions to overcome the computational speed issue, as described in the remainder of this section.

A. An improved constructive phase

Our proposal to speed up the constructive phase is based on a new approach which - in contrast with the classic ACS algorithm - directly considers the best solution computed so far already during the constructive phase. As seen before, in the classic ACS algorithm an ant in state i selects the next node j to visit according to a probabilistic criterion. With probability q_0 the state selected is that with the best weighted compromise between pheromone trail and heuristic desirability, while with probability $1-q_0$ the next node is selected according to a Monte Carlo sampling mechanism. In our new proposal, the state selected with probability q_0 is the state reached after the current state i in the best solution computed so far (in case this state is not feasible, the classic mechanism described above is applied). Since probability q_0 is usually greater than or equal to 0.9, the new approach drastically reduces the running time required to select the next edge to visit (typically from $O(n)$ to something that can be approximated by a constant).

B. A better integration between the constructive phase and the local search procedure

In the conventional ACS framework described in Section III, after the constructive phase is completed, each solution is brought to its local minimum using a local search procedure. The second enhancement we propose is again in the direction of speeding up the whole algorithm, and concerns a better integration between the constructive and the local search phases of the algorithm, leading to a faster overall method: the local search procedure is applied (probabilistically) only on those solutions on which the local search has not been recently applied, in order to avoid searching the neighborhood of the same solution over and over again. Let cnt be an iteration counter initialized at 0 in the beginning, and increased by one at each iteration. The probability of running the local search on a current solution encountered and optimized for the last time at iteration itr is given by $\frac{cnt-itr}{cnt}$. According to such a strategy the same solution is not likely to be optimized too often. On the other hand, running the local search on a same solution from time to time is desirable due to the heuristic nature of the local search implemented: different runs can lead to different final solutions.

V. Experimental results

A. Benchmark problems

We tested our algorithms on two classes of instances, originally adopted in Montemanni and Gambardella [5] and obtained from the well-known Solomon's data-set of VRPTW instances (problems *c*_50*, *c*_100*, *r*_100* and *rc*_100*, see Solomon [6]) and from instances proposed by Cordeau et al. [3] (problems *pr**) for the Multi-Depot Periodic Vehicle Routing Problem (MDPVRP). Leaving out some toy-size instances not considered anymore in the literature, he first data-set is composed by 48 instances with 100 nodes. Depending on the displacement of the customers, this data-set is divided into clustered (*c*_100*), random (*r*_100*) and random clustered (*rc*_100*) categories. The second data-set (*pr*_* instances) has been derived from Cordeau's data-set (20 instances), considering all customers active in the same day. These instances have up to 288 customers. For each instances available, the number of vehicles m considered is between one and four, leading to a total of 388 problems.

Table I
PERCENTAGE DEVIATIONS FROM BEST KNOWN SOLUTIONS. INSTANCES GROUPED BY NUMBER OF VEHICLES m AND CLASS.

m	Class	ILS [8]	VNS [7]	ACS [5]	EACS
1	*c*_100*	1.6596	0.0000	0.1865	0.1865
1	*r*_100*	2.1879	0.0000	0.8552	0.8552
1	*rc*_100*	3.1368	0.0000	0.5782	0.5782
1	*pr*_**	6.4629	0.0000	5.4442	2.8234
2	*c*_100*	1.4980	-	0.4794	0.3977
2	*r*_100*	2.3892	-	1.5806	1.2348
2	*rc*_100*	3.1027	-	1.3323	1.0593
2	*pr*_**	6.0988	-	2.8455	3.0949
3	*c*_100*	2.2008	-	0.5771	0.6410
3	*r*_100*	1.0456	-	0.3171	0.2154
3	*rc*_100*	3.0332	-	1.5760	0.2786
3	*pr*_**	6.4565	-	2.8214	2.7393
4	*c*_100*	1.6062	-	0.3069	0.1949
4	*r*_100*	1.6056	-	0.4242	0.2257
4	*rc*_100*	1.4765	-	0.4012	0.1274
4	*pr*_**	5.9984	-	2.3281	2.3052
AVERAGE		3.1224	-	1.3784	1.0598

B. Results

The EACS algorithm has been coded in ANSI C, and the experiments presented have been run on a Dual AMD Opteron 250 2.4 GHz/4GB computer. The parameter settings adopted for EACS are as follows: $m = 10$, $\xi = 0.5$, $\rho = \psi = 0.1$, $q_0 = 0.9$ and $\tau_{\text{init}} = \frac{1}{CostFirst \cdot |V|}$, where $CostFirst$ is the cost of a solution generated by the ant colony following the ACS constructive algorithm without using the pheromone trails. These parameters are those adopted in [5], apart from q_0 for EACS, which has been selected according to some preliminary experiments.

In Table I the results obtained by some reference algorithms recently appeared in the literature are presented. The algorithms considered are the Iterated Local Search (ILS) method discussed in [8], the Variable Neighborhood Search (VNS) discussed in [7], the ACS presented in [5], and the EACS discussed in this paper. The maximum computation time allowed for each run on one instance is 3600 seconds (comparable hardware is used in the different experiments). For each method we report the percentage deviation from best-known results of the best solution retrieved over 10 runs. Data are aggregated by number of vehicles and instance class. It is important to stress that ILS was designed to be a fast, reactive method, and its execution time is never longer than a few seconds, and that the results of VNS are only available for $m = 1$. In Table I it is possible to notice that EACS was able to improve the results of ACS in average, and anyway for the great majority of the instance classes. Moreover, EACS was also able to retrieve some new best known results, that are summarized in Table II, and therefore to become a reference method for the TOPTW.

Table II
IMPROVED BEST KNOWN SOLUTIONS.

m	Instance	Previous best known method	val	EACS
2	c103_100	ACS [5]	710	720
2	r104_100	ACS [5]	544	552
2	r107_100	ACS [5]	529	535
2	r112_100	ACS [5]	543	544
2	rc106_100	ACS [5]	481	483
3	r106_100	ACS [5]	722	729
3	r111_100	ACS [5]	770	774
3	r112_100	ACS [5]	769	776
3	rc102_100	ACS [5]	710	714
3	rc103_100	ACS [5]	747	754
3	rc106_100	ACS [5]	695	705
3	rc107_100	ACS [5]	755	773
3	rc108_100	ACS [5]	783	795
3	rc206_100	ACS [5]	1722	1724
3	pr08_144	ACS [5]	1118	1124
3	pr11_48	ACS [5]	649	654
4	c105_100	ACS [5]	1060	1070
4	c106_100	ACS [5]	1070	1080
4	c108_100	ACS [5]	1120	1130
4	r105_100	ACS [5]	771	778
4	r109_100	ACS [5]	879	885
4	r112_100	ACS [5]	954	971
4	rc102_100	ACS [5]	903	906
4	rc103_100	ACS [5]	948	963
4	rc106_100	ACS [5]	908	909
4	pr07_72	ACS [5]	872	876

VI. CONCLUSIONS AND FUTURE WORK

An Ant Colony System algorithm for the Team Orienteering Problem with Time Windows has been critically analyzed to discover its potential weaknesses. Based on this analysis, an Enhanced Ant Colony System has been developed. The new algorithm has been empirically shown to outperform the original one, and to be a reference method for the problem considered.

REFERENCES

[1] S.E. Butt and T.M. Cavalier. A heuristic for the multiple path maximum collection problem. *Computers and Operations Research*, 21:101–111, 1994.

[2] I-M. Chao, B.L. Golden, and E.A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:464–474, 1996.

[3] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.

[4] 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,* D. Corne et al. eds., McGraw-Hill, London, U.K., pages 63–76, 1999.

[5] R. Montemanni and L.M. Gambardella. An ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34(4):287–306, 2009.

[6] M. Solomon. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, 35:254–265, 1987.

[7] F. Tricoire, M. Romauch, K.F. Doerner, and R.F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers and Operations Research*, 37(2):351–367, 2010.

[8] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research*, 36(12):3281–3290, 2009.