An Ant Colony Approach to the Orienteering Problem

Yun-Chia Liang and Alice E. Smith, Senior Member, IEEE
Department of Industrial and Systems Engineering
Auburn University
207 Dunstan Hall
Auburn, AL 36849 USA

Submitted to *IEEE Transactions on Evolutionary Computation*

October 2001

An Ant Colony Approach to the Orienteering Problem

Abstract

This paper develops an ant colony optimization approach to the orienteering problem, a general version of the well-known traveling salesman problem with many relevant applications in industry. Based on mainstream ant colony ideas, an unusual sequenced local search and a distance based penalty function are added to result in a method that is convincingly shown to be the best heuristic published for this problem class. Results on 67 test problems from the literature show that the ant colony method performs as well or better in all cases and does so at very modest computational cost. Furthermore, the ant colony method is insensitive to seed, problem instance, problem size and degree of constraint.

Keywords: ant colony, ant system, orienteering problem, traveling salesman problem, routing, optimization

1. Introduction

The orienteering problem (OP) can be formulated as follows: given *n* nodes, each node *i* has a score $S_i \geq 0$ and the scores of the starting node denoted by 1 and the ending node denoted by *n* are set to 0; i.e., $S_1 = S_n = 0$. A score can be considered as sales, customer satisfaction or any other measure of profitability. Each node can be visited at most once. A path between nodes *i* and *j* has a cost c_{ij} associated with it. This cost can be interpreted as time, money spent or distance traveled. Usually *n* nodes are considered in the Euclidean plane. Since the distance and travel time between nodes are determined by the geographical measure, they are assumed as to be known quantities and distance is used as the representative of cost in the following sections. Therefore, the objective of the OP is to maximize the score of a route that consists of a subset of nodes starting from node 1 and finishing at node *n* without violating the cost (time/distance) constraint $T_{\max}$. Generally, the mathematical model of the OP is formulated as

follows:

$$\text{Max} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} S_i x_{ij} \tag{1}$$

$$\text{Subject to} \quad \sum_{j=2}^{n} x_{1j} = \sum_{i=1}^{n-1} x_{in} = 1, \tag{2}$$

$$\sum_{i=2}^{n-1} x_{ik} = \sum_{j=2}^{n-1} x_{kj} \le 1, \qquad k = 2,...,n-1, \tag{3}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \le T_{\max}, \tag{4}$$

$$x_{ij} \in \{0,1\}, \qquad i, j=1,\ldots n. \tag{5}$$

The OP is NP-hard. While it was originally modeled for the sport of orienteering, it has practical applications in vehicle routing and production scheduling, as discussed in Golden et al. [28] and Keller [33]. It should also be noted that the OP is equivalent to the Traveling Salesman Problem (TSP) when the time is relaxed just enough to cover all nodes and where start and end nodes are not specified.

2. Previous Approaches to the Orienteering Problem

There has been work on exact methods for the OP. In Laporte and Martello [36], a linear programming (LP) relaxation of a 0-1 integer programming model within a branch-and-bound scheme is presented. The algorithm starts by relaxing the constraints, and the resulting problem is then solved through LP and the violated conditions are gradually solved through a branch-and-bound process. Upper and lower bounds are derived to fathom nodes of the search tree. Leifer and Rosenwein [38] relax the 0-1 constraints and drop the connectivity constraints. Thereafter, certain valid inequalities are added to the model. After solving the LP relaxation, a cutting plane algorithm is added and the LP is solved again. Fischetti, et al. [24] propose a branch-and-cut algorithm by adding inequalities such as the matching inequality, the cover inequality, and the path inequality. Then, the overall branch-and-cut algorithm is used to find the optimal solution. Hayes and Norman [31] are the first to use dynamic programming to solve the OP. There are no scores associated with nodes in this paper, thus, the authors define the objective as minimizing the total travel time given the constraint that the competitors must visit some pre-specified control nodes. The travel time between nodes depends on distance and geographical factors,

such as uphill takes more time than downhill. Ramesh, et al. [50] use Lagrange relaxation along with improvement procedures within a branch-and-bound method. The solution procedure consists of two phases. In the first phase, starting with an initial set of Lagrange multipliers, the relaxed problem is solved by improving the multipliers at each iteration. If the optimal solution is found, the algorithm is terminated; otherwise, a second phase, branch-and-bound search, is conducted. Although these approaches have yielded solutions to smaller sized problems, as in other NP-hard problems, the computational limitations of exact algorithms encourage the exploration of heuristic procedures.

The first heuristics, the S-algorithm and the D-algorithm, were proposed by Tsiligirides in 1984 [54]. The S-algorithm uses the Monte Carlo method to construct routes using probabilities correlated to the ratio of node score to node distance from the current node. The D-algorithm is built based upon the vehicle scheduling method proposed by Wren and Holiday [58]. This approach operates by dividing the search area into sectors that are determined by two concentric circles and an arc of known length. Sectors are varied by changing the two radii of the circles and by rotating the arcs. A route is built when all nodes in a particular sector have been visited, or it is impossible to visit any other node of the same circle without violating the $T_{\max}$ constraint. In these papers, Tsiligirides also devises the most well known test problems for the OP, which have 21, 32 and 33 nodes.

Golden, Levy and Vohra [28] propose an iterative heuristic for the OP which consists of three steps: route construction using a greedy method, route improvement using a 2-opt swap, and center-of-gravity which guides the next search step. Golden, Wang and Liu [29] combine Tsiligirides's S-algorithm concept (randomness), the center of gravity, and learning capabilities into another approach to solve the OP. To provide probabilities for node selection, the score of neighboring nodes are also considered. Keller [33] uses his algorithm for the multi-objective vending problem (MVP) to solve the OP. A path construction phase uses a measure identical to that of the S-algorithm. This is followed by a three step improvement phase that uses node insertion and identification of node clusters. Wang, et al. [56] propose an artificial neural network approach to solve the OP. A Hopfield-like neural network is formulated and a fourth order convex energy function is devised. Ramesh and Brown [49] propose a four-phase heuristic for the generalized orienteering problem, i.e., the cost function is not limited to a Euclidean

function. The four phases consist of node insertion, cost improvement, node deletion and maximal insertions. The route is improved by a 2-opt procedure followed by a 3-opt procedure in the second phase. In the third phase, one node is removed from the current route and one node is then inserted in an attempt to decrease the length of the route. Finally, as many unassigned nodes as possible are inserted onto the current route in order to increase the total score.

Chao, et al. [8] introduce a two-step heuristic to solve the OP. In the first step, initialization, by using the starting and ending nodes as the two foci of an ellipse and the $T_{\max}$ constraint as the length of the major axis, several routes are generated and the one with the highest score is the initial solution. The initial route is then improved by a 2-node exchange in the cheapest-cost way, and then improved by a 1-node improvement that tries to increase the total score. They apply this algorithm to Tsiligirides's [54] problems and 40 new test problems. The authors also point out a mistake in Tsiligirides's data set and suggest the correction.

Tasgetiren and Smith [53] propose a genetic algorithm (GA) to solve the orienteering problem. A permutation representation is used and a penalty function is employed to help search the infeasible region. Four test sets, the three originally from Tsiligirides [54] and the one corrected by Chao, et al. [8], are used. Tasgetiren's results are competitive to the best known heuristics, though the computational time is relatively high.

3. An Ant Colony Approach

3.1 Background of the Ant Colony Method

Because of the route structure and the lack of dominant heuristic for the OP, this problem class appears to be a good candidate for ant colony optimization (ACO) methods. The Ant System (AS) was first introduced by Dorigo and his colleagues [18, 19, 20, 23]. Since then, ACO algorithms have been applied to different problems such as the traveling salesman problem (TSP) [4, 6, 9, 10, 11, 21, 22, 23, 25, 30, 46], the quadratic assignment problem (QAP) [40, 52], the generalized assignment problem [48], the vehicle routing problem [5, 7, 16, 17, 27], telecommunication networks [15], graph coloring [13, 34, 35], scheduling [1, 12, 14, 32, 42, 43], the shortest supersequence problem [44, 45], the Hamiltonian graph problem [55], the multiple knapsack problem [37], the sequential ordering problem [26], the redundancy allocation problem [39], water distribution network design [41], the constraint satisfaction problem [51], and continuous function problems [2, 3, 57]. In an ACO algorithm, after setting parameter values

and initializing pheromone trails, the ant colony constructs solutions by applying a state transition rule. Local search, if applicable, and a pheromone update rule are employed during each iteration, and the process continues until a stopping criterion is reached. The ACO procedure can be summarized as follows:

Set all parameters and initialize pheromone trails

Loop

Sub-Loop

Construct solutions based on the state transition rule

Apply the online pheromone update rule (*optional*)

Continue until all ants have been generated

Apply local search (*optional*)

Evaluate all solutions and record the best solution so far

Apply the offline pheromone update rule

Continue until the stopping criterion is reached

A local heuristic, η_{ij}, is a key component of the state transition rule. It is problem-dependent, such as $\eta_{ij} = \frac{1}{d_{ij}}$ in TSP, $\eta_{ij} = \frac{1}{s_{ij}}$ in QAP, or $\eta_{ij} = \frac{1}{MDD_{ij}}$ in a scheduling problem. Local search plays an important role in improving the solution quality of ACO algorithms. Problem-dependent local search methods are used in different applications, such as the 2-opt for the symmetric TSP and the 3-opt for the asymmetric TSP [21, 22], the 2-opt and tabu search for the QAP [52], descent local search and tabu search for the generalized assignment problem [48], and adjacent pairwise interchange for the single machine total tardiness problem [1].

3.2 An ACO for the OP

3.2.1 Representation

Since the orienteering problem can be translated to a Generalized Traveling Salesman Problem (GTSP), the graphic representation can easily correspond to the underlying topographical graph as in the TSP. The main difference of the graphic representations between these two problems is that the OP does not necessarily visit all control nodes except the starting and ending ones while the TSP is required to visit all. The graph representation consists of a set of nodes denoted by *N* representing different control nodes, and a set of arcs denoted by *E*

representing transitions between nodes in *N*. Then $G = \{N, E\}$ is a graph. While constructing solutions, ants move from one node to another using the connecting arcs and laying down their pheromone trails on these arcs. Thus, arcs represent adding a specific node to the tour in the OP and each arc has an associated score with it. Therefore the OP may be represented by a weighted acyclic, directed graph with exactly one starting node and one ending node. The starting node is denoted by 1 and a tour always starts from here; the ending node is denoted by *n* and a tour is required to end here. Given the total number of *n* nodes, including the starting and ending nodes, the starting node has $n-1$ outgoing arcs and each arc represents a choice of an unvisited node. The ending node has $n-1$ incoming arcs and no outgoing arcs. Before the tour construction starts, all nodes except the starting and ending nodes have $n-2$ outgoing arcs and $n-2$ incoming arcs. In each transition exactly one node is added to the tour. After a node is chosen, the rest of the incoming arcs to this specific node will be prohibited in order to satisfy the constraint that each node can be only visited at most once. During the construction process, the total distance of current tour is updated with time. Any path from the starting node to the ending node represents an initial solution for the ACO-OP algorithm.

Figure 1 shows an example of the OP graphic representation with five nodes, i.e., $n = 5$. Node 1 represents the starting node and the node 5 denotes the ending one. Nodes 2, 3 and 4 are the remaining control nodes. Therefore, node 1 has four outgoing arcs and no incoming arcs while node 5 has four incoming arcs and no outgoing arcs. Nodes 2, 3 and 4 each have three outgoing arcs and three incoming arcs. Figure 2 shows an example of the tour construction process. This process starts from the starting node, i.e., node 1, and node 2 is selected as the next node to visit. Thereafter, all other outgoing arcs from node 1, which are represented by dashed lines, are prohibited. In next transition, node 3 is chosen, so the rest of node 2's outgoing and incoming arcs are forbidden as shown in Figure 2(b). The final transition assumes that the ending node has to be selected, and the tour construction is completed, with the path 1-2-3-5 formed.

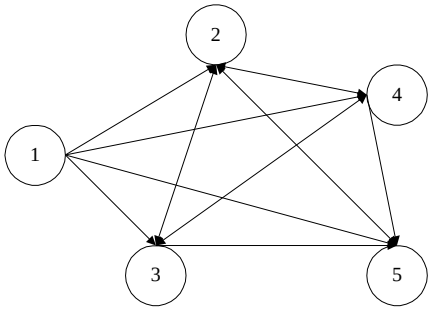
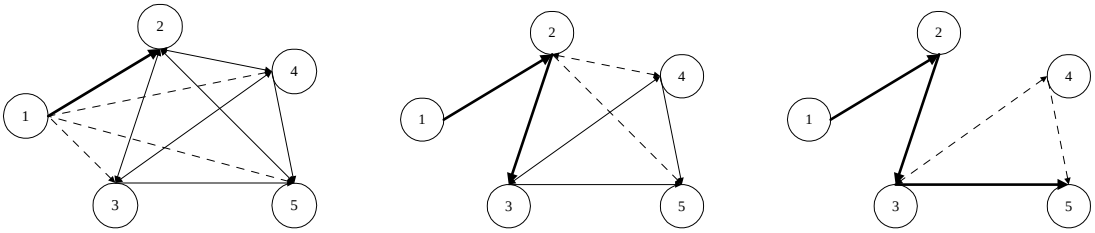


Figure 1. A 5-node example of an initial OP graphic representation.



(a) (b) (c)

Figure 2. Solution construction process. (a) node 2 is chosen as the next node to visit, (b) node 3 is selected as the next node to visit, (c) the ending node is chosen to complete the tour.

3.2.2 The State Transition Rule

The Ant Colony System (ACS) algorithm proposed by Dorigo and Gambardella [21] is used as the main structure of the ACO-OP algorithm here. To construct a solution the ants successively choose nodes to be appended to the current tour. If the sum of current total distance at a node and the distance from this node to the ending node has reached or exceeded the constraint $T_{\max}$, the tour will be terminated after connecting the path from current node to the ending node. For node selection, the ants use problem specific (local) heuristic information, denoted by η_{ij}, as well as pheromone trails, denoted by τ_{ij}, specific to an arc connecting nodes *i* and *j*. The former is an indicator of how good the choice of that node seems to be in general, and the latter indicates how good the choice of the node was during this specific search so far.

In order to balance the exploitation of good solutions and the exploration of the search space, the state transition rule shown as follows is used for the solution construction process where node *v* is selected to be next node visited with the current position at node *i*.

$$v = \begin{cases} \arg \max_{l \in US} [(\tau_{il})^{\alpha} (\eta_{il})^{\beta}] & q \le q_0 \\ V & q > q_0 \end{cases} \quad (6)$$

and V is selected according to the transition probability given by

$$P_{iv} = \begin{cases} \dfrac{(\tau_{iv})^{\alpha} (\eta_{iv})^{\beta}}{\sum_{l \in US} (\tau_{il})^{\alpha} (\eta_{il})^{\beta}} & v \in US \\ 0 & Otherwise \end{cases} \quad (7)$$

where α and β are parameters that control the relative weight of pheromone (τ) and local heuristic (η), respectively, *US* is the set of unvisited nodes, q is a random number uniformly generated between 0 and 1, and q_0 is a parameter which determines the relative importance of exploitation versus exploration. When $q \le q_0$ an exploitation of the knowledge available about the problem (the local heuristic knowledge about the choice of nodes) and the learned knowledge memorized in the form of pheromone trails are used, whereas $q > q_0$ favors more (random) exploration. The problem specific heuristic used is $\eta_{ij} = \dfrac{S_j}{c_{ij}}$ where S_j represents the associated score of node *j* and c_{ij} denotes the distance associated with the path between nodes *i* and *j*.

3.2.3 The Pheromone Trail Update Rule

The pheromone update rule consists of two phases – online (step-by-step) updating and offline (delayed) updating. The purpose of online updating is to decay the pheromone intensity of the selected move to encourage exploration. Online updating occurs after an ant makes a move by

$$\tau_{ij}^{new} = \rho \cdot \tau_{ij}^{old} + (1 - \rho) \cdot \tau_o \quad (8)$$

where $\rho \in [0,1]$ is a parameter that controls the pheromone persistence, i.e., $1 - \rho$ represents the proportion of the pheromone evaporated. Initial trail intensities (τ_0) are set to $\dfrac{1}{n \cdot T_{\max}}$ where n

is the total number of nodes and $T_{\max}$ is the distance constraint.

After all ants have followed the selection process described above and constructed a tour, local search, as described in the next section, is used to improve the solutions. The objective function (1) for each ant, *k*, is calculated, as is the degree of infeasibility, T_k - T_{max}. Most ACO algorithms avoid infeasible solutions during the process of solution construction by means of, for example, a tabu list in the TSP application. However, Ramalhinho and Serra [48] suggest that a penalty function can be used in the objective function evaluation. For a solution violating a constraint, a penalty is added to the objective, and during pheromone updating, infeasible ants contribute less. Since the OP is a constrained problem and search can benefit from considering mildly infeasible solutions, a penalty function for infeasible solutions is used:

$$S_{kp} = S_k \cdot \frac{T_{\max}}{T_k} \tag{9}$$

where T_k and S_k are the total distance and the total score of ant *k*, respectively. If a solution is feasible, $S_{kp} = S_k$; if a solution falls in the infeasible region, the penalized total score, S_{kp}, is calculated by multiplying the unpenalized objective, S_k, with a penalty factor $\frac{T_{\max}}{T_k}$ that correlates with the magnitude of infeasibility. This penalty function will encourage the ACO-OP algorithm to explore the feasible region and infeasible region near the border of the feasible area, and discourage, but permit, search further into the infeasible region, since the global optimum solution is close to or at $T_{\max}$.

Then, the best feasible solution is updated and is used to contribute pheromone in offline updating. At the same time evaporation reduces pheromone trails. The offline trail update can formally be expressed as follows:

$$\tau_{ij}^{new} = \rho \cdot \tau_{ij}^{old} + (1-\rho) \cdot \Delta\tau_{ij} \tag{10}$$

where $\Delta\tau_{ij}$ is the amount of pheromone trail added to τ_{ij} by the ants. This paper uses the elitist approach [22] where only the best ant contributes, i.e., $\Delta\tau_{ij} = S_B$ for all combinations (*i*,*j*) belonging to the best feasible solution found so far, where S_B is the total score of that solution.

3.2.4 The Local Search

Since the solutions generated in each generation before local search are either on or close to the border of the feasible region, local search is very effective in this problem class. The Variable Neighborhood Search (VNS) metaheuristic [47] provides an appropriate idea for local search. VNS does not follow a specified trajectory, but explores increasingly distant neighborhoods of the current solution, and jumps to a new one if, and only if, an improvement is made. A VNS algorithm starts by determining a set of neighborhood structures, i.e., a set of local search methods. Beginning with a neighborhood of the set, i.e., using one of the local search methods, local search is applied and if the solution obtained is better than the incumbent, it replaces the old one and search continues from there. The VNS algorithm continues until all neighborhoods defined have been searched, that is, all local search methods used. den Besteb, et al. used this concept to good effect with an ACO approach to a class of scheduling problems [14]. In the ACO-OP method, the concatenation of several iterative descent local search methods are used as the VNS:

- Eliminate method (ELM): Except the starting and the ending nodes, beginning with the 2^{nd} node on a tour, eliminates a node from the tour.
- Interchange method (INT): Starting with the 2^{nd} node on a tour, interchanges this node with an unvisited node if there is any existing.
- Swap method (SWAP): Swaps a pair of nodes existing on a tour, except the starting and the ending nodes.
- Forward Insert method (FI): Takes a node from its current position and inserts it into a position after its current one but before the ending node.
- Backward Insert method (BI): Takes a node from its current position and inserts it into a position before its current one but after the starting node.
- Add method (ADD): Adds an unvisited node, if it exists, to a position between the starting node and the ending node of the current tour.

During the local search, if the penalized total score improves or the total distance decreases, the new tour replaces the current solution. This sequence of local search mechanisms can be segregated into three parts: INT-ELM, FI-BI-SWAP and ADD. The first part aims at reducing the magnitude of infeasibility and increasing the total score, the second part tries to decrease the

total distance, and the last part of sequence is mainly responsible for increasing the total score of a tour. The iteration-best feasible ant is updated during the local search process above and if it is better than the globally best feasible ant, the globally best feasible ant is updated and another concatenated local search is applied to the new globally best feasible solution by the sequence of INT-ADD-FI-BI-SWAP. ELM is not used since this ant is, by definition, feasible.

4. Computational Experience

The ACO-OP algorithm is coded in Borland C++ and all experiments are run using an Intel Celeron 433 MHz PC with 128 MB RAM. All computations use real float point precision without rounding or truncating values. The length of the final path and the CPU time (in seconds) are rounded to three digits behind the decimal point.

From preliminary explorations, these parameter values are established: colony size = 30, $\alpha = 1$, $\beta = 3$, $q_0 = 0.2$ and $\rho = 0.9$. There is some sensitivity of ACO-OP to changes in these parameters, particularly values of q_o greater than 0.7. With $\beta > \alpha$ and a lower q_o there is more emphasis on the local heuristic as opposed to the pheromone and more emphasis on exploration rather than exploitation. The stopping criteria are when the total number of iterations reaches 200, the best ant has not changed for 100 consecutive iterations or the best feasible ant has reached the upper bound provided by [38]. The test problems are those most studied in the literature [8, 54]. These are three sets of size 32, 21, and 33 nodes with 18, 11, and 20 instances, varying by T_{max} value, respectively. Chao et al. [8] found a mistake in the original data set of the size 32 problem, corrected the mistake and created a new data set, named data set 4, which is different from the old set at node 30. The search spaces are 1.2×10^{17} for the 21 node, 2.7×10^{32} for the 32 node and 8.2×10^{33} for the 33 node problems.

The best results of the ACO-OP algorithm over 10 runs on each instance are compared with the best results of the other heuristics and the upper bound [38] as follows:

- UB: upper bound on score from Leifer and Rosenwein [38] (for data sets 1, 2 and 3 only).
- T: Tsiligirides' S-algorithm [54] (for data sets 2, 3 and 4 only).
- TC: Tsiligirides' S-algorithm coded by Chao et al. [8] (for data sets 1 and 4 only).
- MVP: Keller heuristic [33] (for data sets 1, 2 and 3 only).
- GLV: Golden, Levy, and Vohra heuristic [28] (for data sets 1, 2 and 3 only).

- GWL: Golden, Wang, and Liu heuristic [29] (for data sets 1, 2 and 3 only).
- ANN: Wang, Sun, Golden and Jia's neural network [56].
- CGW: Chao, Golden, and Wasil heuristic [8].
- GA: Tasgetiren and Smith's genetic algorithm [53].

In Tables 1 through 4, the "+" symbol means that ACO-OP produces a better score, the "–" symbol means that ACO-OP produces a worse score, and an empty cell means ACO-OP generates the same score as the comparing heuristic.

For the 49 instances listed in Tables 1 through 3, ACO-OP produces better scores in 10 instances compared to the ones produced by MVP, in 26 instances by GLV, and in 1 instance by GWL. In Tables 2, 3 and 4, ACO-OP performs better in 34 instances compared to the ones produced by T. ACO-OP also outperforms TC in 19 of 49 instances in Tables 1 and 4. Over all 67 instances, ACO-OP is superior in 7 instances compared to ANN, in 1 instance for CGW, and in 1 instance for GA. The most competitive heuristics are CGW and GA, which are improved upon by ACO-OP in only one instance each. Note that in no case does ACO-OP fail to find the best known solution.

Table 1. Comparison of results on test problem set 1 (32 nodes and 18 instances).

	Heuristic Methods from Literature									**ACO-OP vs. Previous Heuristic Methods**						
T_{MAX}	**UB**	**TC**	**MVP**	**GLV**	**GWL**	**ANN**	**CGW**	**GA**	**ACO-OP**	**TC**	**MVP**	**GLV**	**GWL**	**ANN**	**CGW**	**GA**
5	10	10	10	10	10	10	10	10	10							
10	20	15	15	15	15	15	15	15	15							
15	45	45	45	45	45	45	45	45	45							
20	70	65	65	65	65	65	65	65	65							
25	95	90	90	90	90	90	90	90	90							
30	120	110	110	110	110	110	110	110	110							
35	140	135	130	125	135	135	135	135	135		+	+				
40	160	150	155	140	155	155	155	155	155	+		+				
46	180	170	175	165	175	175	175	175	175	+		+				
50	195	185	185	180	190	190	190	190	190	+	+	+				
55	210	195	200	200	205	205	205	205	205	+	+	+				
60	230	220	225	205	225	225	225	225	225	+		+				
65	245	235	240	220	240	240	240	240	240	+		+				
70	260	255	260	240	260	260	260	260	260	+		+				
73	270	260	265	255	265	265	265	265	265	+		+				
75	270	265	270	260	270	270	270	270	270	+		+				
80	285	270	280	275	280	280	280	280	280	+		+				
85	285	280	285	285	285	285	285	285	285	+						
Summary of ACO-OP vs. Previous Heuristic Methods									+	11	3	11	0	0	0	0
									-	0	0	0	0	0	0	0

Table 2. Comparison of results on test problem set 2 (21 nodes and 11 instances).

	Heuristic Methods from Literature									ACO-OP vs. Previous Heuristic Methods						
T_{MAX}	UB	T	MVP	GLV	GWL	ANN	CGW	GA	ACO-OP	T	MVP	GLV	GWL	ANN	CGW	GA
15	145	120	120	120	120	120	120	120	120							
20	200	190	200	200	200	200	200	200	200	+						
23	215	205	210	210	205	205	210	210	210	+			+	+		
25	240	230	230	230	230	230	230	230	230							
27	265	230	230	230	230	230	230	230	230							
30	275	250	260	260	265	265	265	265	265	+	+	+				
32	305	275	300	260	300	300	300	300	300	+		+				
35	350	315	320	300	320	320	320	320	320	+		+				
38	375	355	360	355	360	360	360	360	360	+		+				
40	400	395	380	380	395	395	395	395	395		+	+				
45	450	430	450	450	450	450	450	450	450	+						
Summary of ACO-OP vs. Previous Heuristic Methods									+	7	2	5	1	1	0	0
									-	0	0	0	0	0	0	0

Table 3. Comparison of results on test problem set 3 (33 nodes and 20 instances).

	Heuristic Methods from Literature									ACO-OP vs. Previous Heuristic Methods						
T_{MAX}	UB	T	MVP	GLV	GWL	ANN	CGW	GA	ACO-OP	T	MVP	GLV	GWL	ANN	CGW	GA
15	175	100	170	170	170	170	170	170	170	+						
20	210	140	200	200	200	200	200	200	200	+						
25	290	190	260	250	260	250	260	260	260	+		+		+		
30	340	240	320	320	320	320	320	320	320	+						
35	395	290	370	380	390	390	390	390	390	+	+	+				
40	445	330	430	420	430	420	430	430	430	+		+		+		
45	490	370	460	450	470	470	470	470	470	+	+	+				
50	535	410	520	500	520	520	520	520	520	+		+				
55	575	450	550	520	550	550	550	550	550	+		+				
60	605	500	570	580	580	580	580	580	580	+	+					
65	635	530	610	600	610	610	610	610	610	+		+				
70	665	560	640	640	640	640	640	640	640	+						
75	695	590	670	650	670	670	670	670	670	+		+				
80	725	640	700	690	710	700	710	710	710	+	+	+		+		
85	750	670	740	720	740	740	740	740	740	+		+				
90	785	690	760	770	770	770	770	770	770	+	+					
95	800	720	790	790	790	790	790	790	790	+						
100	800	760	800	800	800	800	800	800	800	+						
105	800	770	800	800	800	800	800	800	800	+						
110	800	790	800	800	800	800	800	800	800	+						
Summary of ACO-OP vs. Previous Heuristic Methods									+	20	5	10	0	3	0	0
									-	0	0	0	0	0	0	0

Table 4. Comparison of results on test problem set 4 (32 nodes and 18 instances, corrected).

	Heuristic Methods from Literature						ACO-OP vs. Previous Heuristic Methods				
T_{MAX}	T	TC	ANN	CGW	GA	ACO-OP	T	TC	ANN	CGW	GA
5	10	10	10	10	10	10					
10	15	15	15	15	15	15					
15	45	45	45	45	45	45					
20	65	65	65	65	65	65					
25	90	85	90	90	90	90		+			
30	110	110	110	110	110	110					
35	135	135	130	135	135	135			+		
40	150	150	155	155	155	155	+	+			
46	175	175	175	175	175	175					
50	190	185	190	190	190	190		+			
55	205	200	205	205	205	205		+			
60	220	220	220	220	225	225	+	+	+	+	
65	240	240	240	240	240	240					
70	255	250	260	260	260	260	+	+			
73	260	265	265	265	265	265	+				
75	270	265	270	275	270	275	+	+	+		+
80	275	270	280	280	280	280	+	+			
85	280	285	285	285	285	285	+				
Summary of ACO-OP vs. Previous Heuristic Methods						+	7	8	3	1	1
						-	0	0	0	0	0

It is difficult to make a solid computational comparison. CPU seconds will vary according to hardware, software and coding. The CGW heuristic [8] and the GA [53] are used here for CPU runtime comparison purposes since they are the primary competitors when considering performance. CGW is coded in FORTRAN and executed on a SUN 4/370 workstation and GA is coded in Borland C++ and run on a Dell 450 PC. In Tables 5 through 8, CPU time in seconds for CGW, GA and ACO-OP are given, with the quickest method highlighted in gray. The CPU time of ACO-OP is the mean time over those runs reaching the best solution and ranges from 0.011 seconds to 25.274 seconds. The mean CPU time of ACO-OP is lower than one second in over half of the 67 instances, is lower than CGW in all but 3 of the 67 instances and is considerably faster than GA in all instances. Tables 5 through 8 also show the best tour generated by ACO-OP, the *T* value of the tour, and the maximum, mean and standard deviation over the ten runs. In most instances, the ACO-OP found the optimum in each of the ten runs and in no case was the worst performance very bad.

Table 5. CPU time comparison and sequence found by the ACO-OP for problem set 1.

	CPU Time			Score					
T_{MAX}	CGW	GA	ACO-OP	Max	Min	Avg	SD	T_d	ACO-OP Tour
5	0.67	5.36	0.013	10	10	10	0.0000	4.143	1 28 32
10	0.80	10.28	0.011	15	15	15	0.0000	6.867	1 28 18 32
15	2.28	14.98	0.207	45	35	43	4.2164	14.264	1 27 31 26 20 19 32
20	17.49	20.47	0.064	65	65	65	0.0000	19.595	1 27 31 26 22 21 20 19 32
25	9.07	25.89	0.733	90	85	89.5	1.5811	24.816	1 27 31 26 22 21 12 11 10 9 32
30	31.92	31.36	0.080	110	110	110	0.0000	29.711	1 27 31 26 25 24 23 22 21 12 19 32
35	25.25	38.48	0.808	135	135	135	0.0000	34.081	1 28 27 31 26 25 23 22 21 12 11 10 8 9 13 32
40	16.76	42.06	6.227	155	155	155	0.0000	38.974	1 28 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 32
46	21.58	48.68	0.677	175	175	175	0.0000	44.512	1 28 27 31 26 25 24 23 22 21 12 11 10 9 8 2 3 7 6 32
50	24.91	58.94	0.795	190	190	190	0.0000	49.534	1 28 27 31 26 25 24 23 22 21 12 11 10 9 8 2 3 7 5 6 13 32
55	24.67	60.39	15.732	205	200	203.5	2.4152	54.797	1 28 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 5 4 14 15 18 32
60	24.28	65.55	11.464	225	220	223.5	2.4152	59.888	1 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 5 4 14 15 16 17 28 32
65	23.26	69.56	0.495	240	240	240	0.0000	63.822	1 27 31 26 25 24 23 22 21 12 11 10 8 2 3 7 6 5 4 14 15 16 17 28 32
70	25.09	73.56	11.781	260	260	260	0.0000	69.127	1 28 29 17 16 15 14 4 5 6 7 3 2 8 10 11 12 21 22 23 24 25 26 31 27 20 19 32
73	25.24	75.29	1.352	265	265	265	0.0000	70.731	1 28 29 17 16 15 14 4 5 6 7 3 2 8 9 10 11 12 21 22 23 24 25 26 31 27 20 19 32
75	28.53	77.72	4.075	270	270	270	0.0000	73.507	1 28 29 17 16 15 14 4 5 6 7 3 2 8 10 11 12 21 22 23 24 25 30 31 26 27 20 19 32
80	26.84	80.58	3.738	280	280	280	0.0000	78.720	1 28 29 17 16 15 14 4 5 6 13 7 3 2 8 9 10 11 12 21 22 23 24 25 30 31 26 27 20 19 32
85	21.71	88.12	2.739	285	285	285	0.0000	81.784	1 19 20 27 26 31 30 25 24 23 22 21 12 11 10 9 8 2 3 7 13 6 5 4 14 15 16 17 29 28 18 32

Table 6. CPU time comparison and sequence found by the ACO-OP for problem set 2.

	CPU Time			Score					
T_{MAX}	CGW	GA	ACO-OP	Max	Min	Avg	SD	T_d	ACO-OP Tour
15	1.29	11.53	0.023	120	120	120	0.0000	14.543	1 7 12 11 8 9 10 14 21
20	2.24	15.35	0.402	200	200	200	0.0000	19.88	1 12 7 6 5 3 2 8 9 10 11 13 14 21
23	4.45	19.04	0.685	210	200	206.5	4.7434	22.648	1 7 6 5 4 3 2 8 9 10 11 14 21
25	5.65	20.53	0.111	230	230	230	0.0000	24.128	1 12 7 6 5 4 3 2 8 9 10 11 13 14 21
27	6.37	22.63	0.152	230	230	230	0.0000	24.895	1 11 10 9 8 2 3 4 5 6 7 12 13 14 21
30	6.18	23.9	0.172	265	250	259.5	3.6893	29.849	1 7 6 2 8 17 16 15 9 10 11 13 14 21
32	7.21	26	0.561	300	300	300	0.0000	31.625	1 7 6 5 3 2 8 17 16 15 9 10 11 13 14 21
35	7.81	27.74	1.344	320	310	312	4.2164	34.989	1 7 6 5 4 20 19 18 17 8 9 10 11 13 14 21
38	6.84	28.74	2.797	360	355	356.5	2.4152	37.842	1 7 6 5 2 3 4 20 19 18 17 8 9 10 11 13 14 21
40	7.14	28.71	3.818	395	385	394	3.1623	39.778	1 7 6 5 3 4 20 19 18 16 15 17 8 9 10 11 13 21
45	0.61	29.01	0.731	450	450	450	0.0000	44.438	1 12 7 6 5 2 3 4 20 19 18 16 15 17 8 9 10 11 13 14 21

Table 7. CPU time comparison and sequence found by the ACO-OP for problem set 3.

	CPU Time			Score					
T_{MAX}	CGW	GA	ACO-OP	Max	Min	Avg	SD	T_d	ACO-OP Tour
15	4.37	25.30	0.036	170	170	170	0.0000	14.573	1 24 22 7 5 28 14 4 23 33
20	5.16	31.87	2.194	200	190	199	3.1623	19.792	1 24 22 7 5 28 14 4 3 27 23 33
25	9.40	39.38	0.209	260	260	260	0.0000	24.458	1 24 22 7 5 14 4 20 13 3 23 33
30	9.96	44.05	0.465	320	310	319	3.1623	28.770	1 24 22 7 5 28 20 17 13 3 4 14 27 23 33
35	15.38	47.68	0.164	390	390	390	0.0000	34.791	1 24 22 7 5 28 14 4 20 17 16 15 13 3 23 33
40	18.65	52.36	0.711	430	430	430	0.0000	38.881	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 32 33
45	26.84	66.89	2.223	470	470	470	0.0000	44.259	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 29 26 33
50	28.74	72.78	1.528	520	520	520	0.0000	48.936	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 29 26 33
55	30.27	75.40	5.377	550	550	550	0.0000	53.546	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 29 30 26 32 33
60	27.68	79.01	0.402	580	580	580	0.0000	59.341	1 24 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 29 30 33
65	25.02	79.20	18.369	610	600	603	4.8304	63.236	1 24 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 29 30 26 32 23 33
70	29.82	84.69	12.888	640	640	640	0.0000	69.139	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 11 10 9 30 29 26 32 33
75	29.25	90.24	0.934	670	670	670	0.0000	74.227	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 11 19 18 10 9 25 33
80	30.14	100.42	9.406	710	710	710	0.0000	79.710	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 29 30 11 19 18 10 9 25 33
85	28.30	103.64	14.800	740	740	740	0.0000	84.862	1 24 22 7 5 28 14 4 20 17 16 15 13 3 6 2 8 31 12 11 19 18 10 9 30 29 26 32 23 27 33
90	24.43	103.91	12.106	770	760	769	3.1623	89.313	1 24 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 11 19 18 10 9 30 29 26 32 33
95	22.33	106.47	1.491	790	790	790	0.0000	92.791	1 24 23 27 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 11 19 18 10 9 30 29 26 32 33
100	0.67	105.55	2.105	800	800	800	0.0000	97.078	1 24 23 27 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 11 19 18 10 9 25 30 29 26 32 33
105	0.60	103.01	0.550	800	800	800	0.0000	97.240	1 24 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 11 19 18 10 9 25 30 29 26 32 23 27 33
110	0.72	102.27	0.403	800	800	800	0.0000	97.078	1 24 23 27 22 7 5 28 14 4 20 17 21 16 15 13 3 6 2 8 31 12 11 19 18 10 9 25 30 29 26 32 33

Table 8. CPU time comparison and sequence found by the ACO-OP for problem set 4.

	CPU Time			Score					
T_{MAX}	CGW	GA	ACO-OP	Max	Min	Avg	SD	T_d	ACO-OP Tour
5	0.22	6	0.014	10	10	10	0.0000	4.143	1 28 32
10	0.27	10.87	0.013	15	15	15	0.0000	6.867	1 28 18 32
15	0.72	16.98	0.752	45	45	45	0.0000	14.264	1 27 31 26 20 19 32
20	4.76	23.42	0.056	65	65	65	0.0000	19.686	1 27 31 26 20 21 12 19 32
25	2.47	29.42	0.119	90	90	90	0.0000	24.816	1 27 31 26 22 21 12 11 10 9 32
30	10.86	35.64	0.081	110	110	110	0.0000	28.797	1 28 27 31 26 22 21 12 11 10 8 9 13 32
35	14.11	44.57	1.239	135	135	135	0.0000	34.081	1 28 27 31 26 25 23 22 21 12 11 10 8 9 13 32
40	21.81	50.51	6.443	155	150	154.5	1.5811	38.974	1 28 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 32
46	21.62	55.58	0.323	175	175	175	0.0000	44.512	1 28 27 31 26 25 24 23 22 21 12 11 10 9 8 2 3 7 6 32
50	22.76	71.78	0.414	190	190	190	0.0000	49.776	1 28 27 31 26 25 24 23 22 21 12 11 10 9 8 2 3 4 5 6 32
55	24.81	90.31	5.902	205	200	203.5	2.4152	54.797	1 28 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 5 4 14 15 18 32
60	20.39	90.74	11.229	225	220	224	2.1081	59.888	1 27 31 26 25 23 22 21 12 11 10 8 2 3 7 6 5 4 14 15 16 17 28 32
65	26.78	96.45	0.811	240	240	240	0.0000	64.311	1 28 17 16 15 14 4 5 7 3 2 8 10 11 12 21 22 23 24 25 26 31 27 19 32
70	25.51	85.44	8.163	260	260	260	0.0000	69.127	1 28 29 17 16 15 14 4 5 6 7 3 2 8 10 11 12 21 22 23 24 25 26 31 27 20 19 32
73	27.04	85.16	2.158	265	265	265	0.0000	71.867	1 28 30 29 17 16 15 14 4 5 7 3 2 8 10 11 12 21 22 23 24 25 26 31 27 20 19 32
75	27.47	89.59	25.274	275	270	272	2.5821	74.661	1 28 30 29 17 16 15 14 4 5 6 7 3 2 8 9 10 11 12 21 22 23 24 25 26 31 27 20 19 32
80	28.17	90.12	5.067	280	280	280	0.0000	77.725	1 19 20 27 31 26 25 24 23 22 21 12 11 10 9 8 2 3 7 6 5 4 14 15 16 17 29 30 28 18 32
85	21.64	87.77	8.005	285	285	285	0.0000	81.482	1 19 20 27 31 26 25 24 23 22 21 12 11 10 8 9 13 2 3 7 6 5 4 14 15 16 17 29 30 28 18 32

5. Conclusions

This paper presented the first known application of an ant colony optimization method to the orienteering problem. The OP problem has many important parallels in problems found in industry, along with its original inspiration of the sport of orienteering. An ACO that uses both on line and off line pheromone updating and employs local search each iteration is effective and efficient. The local search is atypical in that it involves a sequence of simple heuristics rather than either a single heuristic or a rotating choice of heuristics. This variable neighborhood concept may be of value in other ACO implementations. Computational experience shows that the ACO approach is dominant to all published heuristics in quality of solution obtained and is modest indeed in its computational requirements. Furthermore, sensitivity to seed is small and robustness to problem instance, size and degree of constraint is great. In summary, as in other path based problems such as network routing and TSP, ACO clearly shows its merit in the orienteering problem.

References

1. Bauer, A., B. Bullnheimer, R. F. Hartl, and C. Strauss, "Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization," *Central European Journal of Operations Research*, vol. 8, no. 2, 2000, pp. 125-141.
2. Bilchev, G. and I. C. Parmee, "The Ant Colony Metaphor for Searching Continuous Design Spaces," in T. Fogarty (ed.), *Lecture Notes in Computer Science*, vol. 993, 1995, Springer Verlag, pp. 25-39.
3. Bilchev, G. and I. C. Parmee, "Constrained Optimisation with an Ant Colony Search Model," *Proceedings of the Adaptive Computing in Engineering Design and Control Second Conference (ACEDC'96)*, Plymouth, UK, March 1996, pp. 145-151.
4. Bullnheimer, B., R. F. Hartl, and C. Strauss, "Parallelization Strategies for the Ant System," in R. De Leone, A. Murli, P. Pardalos, and G. Toraldo (eds.), *High Performance Algorithms and Software in Nonlinear Optimization; Series: Applied Optimization*, vol. 24, 1998, pp. 87-100.
5. Bullnheimer, B., R. F. Hartl, and C. Strauss, "Applying the Ant System to the Vehicle Routing Problem," in S. Voss, S. Martello, I. H. Osman, and C. Roucairol (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, 1999, Kluwer, pp. 285-296.
6. Bullnheimer, B., R. F. Hartl, and C. Strauss, "A New Rank Based Version of the Ant System – A Computational Study," *Central European Journal for Operations Research and Economics*, vol. 7, no. 1, 1999, pp. 25-38.
7. Bullnheimer, B., R. F. Hartl, and C. Strauss, "An Improved Ant System Algorithm for the Vehicle Routing Problem," *Annals of Operations Research*, vol. 89, 1999, pp. 319-328.
8. Chao, I. M., B. L. Golden, and E. A. Wasil, "A Fast and Effective Heuristic for the Orienteering," *European Journal of Operational Research*, vol. 88, 1996, pp. 475-489.
9. Colorni, A., M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, "Heuristics from Nature for Hard Combinatorial Optimization Problems," *International Transactions in Operational Research*, vol. 3, no. 1, 1996, pp. 1-21.
10. Colorni, A., M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies," *Proceedings of the First European Conference on Artificial Life (ECAL91)*, Paris, France, 1991, pp. 134-142.
11. Colorni, A., M. Dorigo, and V. Maniezzo, "An Investigation of Some Properties of an Ant Algorithm," *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN92)*, Brussels, Belgium, 1992, pp. 509-520.
12. Colorni, A., M. Dorigo, V. Maniezzo, and M. Trubian, "Ant System for Job-Shop Scheduling," *Belgian Journal of Operations Research, Statistics and Computer Science (JORBEL)*, vol. 34, no. 1, 1994, pp. 39-53.
13. Costa, D. and A. Hertz, "Ants Can Colour Graphs," *Journal of the Operational Research Society*, vol. 48, 1997, pp. 295-305.

14. den Besteb, M., T. Stützle, and M. Dorigo, "Ant Colony Optimization for the Total Weighted Tardiness Problem," *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN VI)*, LNCS 1917, Berlin, 2000, pp. 611-620.

15. Di Caro, G. and M. Dorigo, "Ant Colonies for Adaptive Routing in Packet-Switched Communication Networks," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, Amsterdam, The Netherlands, September 1998, pp. 673-682.

16. Doerner, K., R. F. Hartl, and M. Reimann, "Ant Colony Optimization Applied to the Pickup and Delivery Problem," *POM Working Paper No. 8/2000*, 2000, University of Vienna, Austria.

17. Doerner, K., R. F. Hartl, and M. Reimann, "Cooperative Any Colonies for Optimizing Resource Allocation in Transportation," *POM Working Paper No. 9/2000*, 2000, University of Vienna, Austria.

18. Dorigo, M., *Optimization, Learning and Natural Algorithms*, Ph.D. Thesis, 1992, Politecnico di Milano, Italy.

19. Dorigo, M. and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill, pp. 11-32.

20. Dorigo, M., G. Di Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, 1999, pp. 137-172.

21. Dorigo, M. and L. M. Gambardella, "Ant Colonies for the Travelling Salesman Problem," *BioSystems*, vol. 43, 1997, pp. 73-81.

22. Dorigo, M. and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 53-66.

23. Dorigo, M., V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 26, no. 1, 1996, pp. 29-41.

24. Fischetti, M., J. J. S. Gonzalez, and P. Toth, "Solving the Orienteering Problem through Branch-and-Cut," *INFORMS Journal on Computing*, vol. 10, no. 2, 1998, pp. 133-148.

25. Gambardella, L. M. and M. Dorigo, "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem," *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, Tahoe City, California, 1995, pp. 252-260.

26. Gambardella, L. M. and M. Dorigo, "An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem," *INFORMS Journal on Computing*, vol. 12, no. 3, 2000, pp. 237-255.

27. Gambardella, L. M., E. Taillard, and G. Agazzi, "MACS-VRPTW A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows," in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill, pp. 63-76.

28. Golden, B. L., L. Levy, and R. Vohra, "The Orienteering Problem," *Naval Research*

Logistics, vol. 34, 1987, pp. 307-318.

29. Golden, B. L., Q. Wang, and L. Liu, "A Multifaceted Heuristic for The Orienteering Problem," *Naval Research Logistics*, vol. 35, 1988, pp. 359-366.
30. Guntsch, M. and M. Middendorf, "Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP," *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, LNCS 2037, Lake Como, Italy, 2001, pp. 213-222.
31. Hayes, M. and J. M. Norman, "Dynamic Programming in Orienteering: Route Choice and the Siting of Controls," *Journal of the Operational Research Society*, vol. 35, no. 9, 1984, pp. 791-796.
32. Iredi, S., D. Merkle, and M. Middendorf, "Bi-Criterion Optimization with Multi Colony Ant Algorithms," *Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO '01)*, LNCS 1993, Zurich, 2001, pp. 359-372.
33. Keller, P. C., "Algorithms to Solve the Orienteering Problem: A Comparison," *European Journal of Operational Research*, vol. 41, 1989, pp. 224-231.
34. Kuntz, P. and P. Layzell, "An Ant Clustering Algorithm Applied to Partitioning in VLSI Technology," *Proceedings of the 4th European Conference on Artificial Life*, 1997, pp. 417-424.
35. Kuntz, P. and D. Snyers, "New Results on an Ant-based Heuristic for Highlighting the Organization of Large Graphs," *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, pp. 1451-1458.
36. Laporte, G. and S. Martello, "The Selective Traveling Salesman Problem," *Discrete Applied Mathematics*, vol. 26, 1990, pp. 193-207.
37. Leguizamon, G. and Z. Michalewicz, "A New Version of Ant System for Subset Problems," *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, pp. 1459-1464.
38. Leifer, A. C. and M. B. Rosenwein, "Strong Linear Programming Relaxations for the Orienteering Problem," *European Journal of Operational Research*, vol. 73, 1993, pp. 517-523.
39. Liang, Y.-C. and A. E. Smith, "An Ant System Approach to Redundancy Allocation," *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, pp. 1478-1484.
40. Maniezzo, V. and A. Colorni, "The Ant System Applied to the Quadratic Assignment Problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, 1999, pp. 769-778.
41. Mariano, C. E. and E. Morales, "MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems," *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, vol. 1, 1999, pp. 894-901.
42. Merkle, D. and M. Middendorf, "A New Approach to Solve Permutation Scheduling Problems with Ant Colony Optimization," *Applications of Evolutionary Computing:*

Proceedings of EvoWorkshops 2001, LNCS 2037, Lake Como, Italy, 2001, pp. 484-493.

43. Merkle, D., M. Middendorf, and H. Schmeck, "Ant Colony Optimization for Resource-Constrained Project Scheduling," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, Las Vegas, Nevada, July 2000, pp. 893-900.
44. Michels, R. and M. Middendorf, "An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, Amsterdam, The Netherlands, September 1998, pp. 692-701.
45. Michels, R. and M. Middendorf, "An Ant System for the Shortest Common Supersequence Problem," in D. Corne, M. Dorigo and F. Glover (eds.), *New Ideas in Optimization*, 1999, McGraw-Hill, pp. 51-61.
46. Middendorf, M., F. Reischle, and H. Schmeck, "Information Exchange in Multi Colony Ant Algorithms," *Parallel and Distributed Computing: Proceedings of the 15th IPDPS 2000 Workshops, the 3rd Workshop on Biologically Inspired Solutions to Parallel Processing Problems (BioSP3)*, LNCS 1800, Cancun, Mexico, May 2000, pp. 645-652.
47. Mladenovic, N. and P. Hansen, "Variable Neighborhood Search," *Computers and Operations Research*, vol. 24, no. 11, 1997, pp. 1097-1100.
48. Ramalhinho, H. and D. Serra, "Adaptive Approach Heuristics for the Generalized Assignment Problem," *Economic Working Paper 288*, 1998, Universitat Pompeu Fabra, Spain.
49. Ramesh, R and K. M. Brown, "An Efficient Four-Phase Heuristic for the Generalized Orienteering Problem," *Computers and Operations Research*, vol. 18, no. 2, 1991, pp. 151-165.
50. Ramesh, R., Y. S. Yoon, and M. H. Karwan, "An Optimal Algorithm for the Orienteering Tour Problem," *ORSA Journal on Computing*, vol. 4, no. 2, 1992, pp. 155-165.
51. Schoofs, L. and B. Naudts, "Ant Colonies are Good at Solving Constraint Satisfaction Problems," *Proceedings of the 2000 Congress on Evolutionary Computation*, San Diego, CA, July 2000, pp. 1190-1195.
52. Stuetzle, T. and M. Dorigo, "ACO Algorithms for the Quadratic Assignment Problem," in D. Corne, M. Dorigo and F. Glover (eds.), New Ideas in Optimization, 1999, McGraw-Hill.
53. Tasgetiren, M. F. and A. E. Smith, "A Genetic Algorithm for the Orienteering Problem," *Proceedings of the 2000 Congress on Evolutionary Computation*, San Diego, CA, July 2000, pp. 1190-1195.
54. Tsiligirides, T., "Heuristic Methods Applied to Orienteering," *Journal of Operational Research Society*, vol. 35, no. 9, 1984, pp. 797-809.
55. Wagner, I. A. and A. M. Bruckstein, "Hamiltonian(t)-An Ant Inspired Heuristic for Recognizing Hamiltonian Graphs," *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, pp. 1465-1469.
56. Wang, Q., X. Sun, B. L. Golden, and J. Jia, "Using Artificial Neural Networks to Solve the

Orienteering Problem," *Annals of Operations Research*, vol. 61, 1995, pp. 111-120.

57. Wodrich, M. and G. Bilchev, "Cooperative Distributed Search: The Ants' Way," *Journal of Control and Cybernetics*, vol. 26, no. 3, 1997, pp. 413-446.

58. Wren, A. and A. Holiday, "Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points," *Operations Research Quarterly*, vol. 23, 1972, pp. 333-344.