Coloured Ant System and Local Search to Design Local Telecommunication Networks

Roberto Cordone[1] and Francesco Maffioli[2]

[1] *DEI - Politecnico di Milano*
(cordone@fusberta.elet.polimi.it)
[2] *DEI - Politecnico di Milano*
(maffioli@elet.polimi.it)

Abstract. This work combines local search with a variant of the *Ant System* metaheuristic recently proposed for partitioning problems with cardinality constraints. The *Coloured Ant System* replaces the classical concept of *trail* with p trails of different "colours", representing the assignment of an element to one of the classes in the partition. We apply the method with promising results to the design of local telecommunication networks. The combination of the *Coloured Ant System* with local search yields much better results than the two approaches alone.

1 Introduction

The design of local telecommunication networks (e. g. cable television companies providing internet access [16]) often requires to link demand nodes to a wide area network through a number of concentrator devices taking over the traffic flow. The demand nodes form tree-shaped subnetworks, as a higher connectivity is unjustified at this level. We model the situation as a *Weight Constrained Graph Tree Partition Problem* (*WC-GTPP*). Let $G(V, E)$ be an undirected graph of n vertices and m edges. Given a cost function $c : E \to \mathbb{N}$ on the edges of G and a weight function $w : V \to \mathbb{N}$ on its vertices, determine a spanning forest $F(V, X)$ of p trees $T_r(U_r, X_r)$, such that the weight of each tree belongs to a given interval $[W^-; W^+]$ and the cost of F is minimum.

This paper adapts the *Ant System* to the *WC-GTPP* and combines it to local search. As the purpose is to partition the vertices into p clusters, the ants release trails of p different colours, instead of a single undifferentiated trail. Hence the name of *Coloured Ant System* (*CAS*) [5]. The algorithm relaxes the weight bound, penalizing its violations with a factor tuned by feedback. The *CAS* interacts with *Swinging Forest*, a local search procedure moving vertices from tree to tree. Once in a local optimum, the procedure splits trees to create slack capacity and removes them, alternatively. At each iteration, the *CAS* submits the best current solution to local search before updating the trails. The two approaches take advantage of each other: the *CAS* provides a smart initialization to local search; local search provides a smart trail update to the *CAS*.

Sect. 2 briefly surveys the literature on local network design problems, and gives some references on the *Ant System* and its variants. Sect. 3 describes the

CAS algorithm. Sect. 4 describes *Swinging Forest* and its interactions with the *CAS*. The last section presents some computational results.

2 Survey

The field of local telecommunication networks (servers, cable TV companies, etc...) poses a number of practical problems in which secondary devices must be connected to a given number of primary ones, minimizing the total connection cost and keeping a balance between the traffic of the disjoint subnetworks. Similar trade-offs between cost and balance occur in electric or radio broadcasting networks, as well as in electoral districting and Cluster Analysis.

Though fairly general, the *WC-GTPP* is a good model for these cases. It is strongly $\mathcal{NP}$-hard, by direct reduction from *SAT*, and in general not approximable [6], though a $(2p-1)$ algorithm exists if the number of vertices in each tree is given and the triangle inequality holds [13]. Similar problems have been deeply studied. In the *Capacitated Minimum Spanning Tree* problem (*CMST*) all secondary devices are linked to a single primary one, with an upper bound on the traffic in each branch [2]. The lower weight constrained minimum spanning forest problem bounds from below the weight of the trees and does not fix their number. These problems are strongly $\mathcal{NP}$-hard, but approximable [1, 14].

The *Ant System* has been first proposed for the *Travelling Salesman Problem* [10], spreading to other problems in Combinatorial Optimization, such as *Graph Colouring* [7] and the *Quadratic Assignment Problem* [15]. Several variants propose effective ways to manage the trail function in run-time. In particular, the *Max-Min Ant System* limits it into a given interval [18], the *Ant-Q* approach updates it by reinforcement learning mechanisms [8], the *Ant Colony System* approach by moving averages [9]. The probability distribution of choices can be based on the rank of the alternatives, instead of their absolute value [4]. In the field of routing problems, the *Ant System* combined with local search has brought to strong improvements upon both approaches [11].

3 *Coloured Ant System*

The *Ant System* involves a population of agents (*ants*), who build in parallel greedy solutions to a problem. Their behaviour is not trivially greedy: they partly take random choices influenced by a *trail function*, which is shaped by the results of previous runs. To reformulate the natural analogy for our problem, an agent is a caste of ants divided into p colonies. Each colony builds a hive on a different vertex and takes possession of a connected subgraph from it. The colony patrols a network of edges, whose cost evaluates the necessary effort, to control the food sources located on the vertices, whose amount is measured by the weights. The global control effort must be minimum, the amount of food owned by each colony neither too small nor too large. Each colony marks the vertices with a trail of a typical "colour". Season after season, they retire into the hives and resume colonizing the graph, driven by costs, weights and the residual

trails. To correct bad initial choices, the ants periodically resettle their hives. Several castes operate independently on the graph: ants of different colonies can occupy the same vertex, if they belong to different castes. However, the castes share the p hives and the trail information.

Algorithm 1 gives an outline of the method. First, it selects p starting hives. Then, S agents build solutions gradually annexing all of the other vertices to the hives. Solutions violating the weight bound are admitted, but their cost is multiplied by a penalization factor π. As the basic choice assigns a vertex v to a tree T_r, the trails are released on the vertices and their colours refer to the trees. The algorithm updates the trail function τ_{rv} in various points, both strengthening and weakening it. Every I_{root} iterations, the colonies resettle. The process ends after $I_{\max}$ iterations or $T_{\max}$ seconds since the beginning, or I_{best} iterations since the discovery of the best known solution.

Algorithm 1. **CAS**(G, c, w, W, p)
$\{v_1, \ldots, v_p\} := \mathit{StartingHives}(G, c, p)$; { Choose p starting hives }
$\tau_{rv} := \tau_0 \quad \forall r, \forall v$ { Distribute p uniform trails }

$i := 1$; $i^* := 1$; $i^r := 1$; $c^* := +\infty$;
While $i \leq I_{\max}$ **and** $i - i^* \leq I_{\text{best}}$ **and** $\mathit{ElapsedTime}() \leq T_{\max}$ **do**
{ Periodically resettle the hives }
If $i - i_{\text{root}} > I_{\text{root}}$ **then** $i_{\text{root}} := i$; $v_r := \mathit{RootVertex}(U_r) \quad \forall r$;

{ Build S solutions in parallel }
$U_r^{(s)} := \{v_r\} \quad \forall r, \forall s$; { The ants retire }
While $\exists s : \left(V \setminus \cup_r U_r^{(s)}\right) \neq \emptyset$ **do** { While unassigned vertices exist... }
For $s := 1$ **to** S **do**
$(\tilde{v}, \tilde{r}) := \mathit{BestAnnexion}(G, c, w, W, p, U_r^{(s)})$; { ...take a choice... }
$U_{\tilde{r}}^{(s)} := U_{\tilde{r}}^{(s)} \cup \{\tilde{v}\}$; { ...perform it... }
$\tau_{\tilde{r}\tilde{v}} := (1 - \rho)\tau_{\tilde{r}\tilde{v}} + \rho\tau_0$; { ...and deter repetitions }
EndFor
EndWhile

{ Determine the best current solution }
$F := \mathit{BestSolution}(G, c, w, W, p, U_r^{(s)})$;
$\tau_{rv} := (1 - \rho)\tau_{rv} + \rho / c_F \pi_F \quad \forall r, \forall v \in U_r^{(\tilde{s})}$; { Intensify the trails }
If $c_F < c^*$ **then** $c^* := c_F$; $i^* := i$; { Update the best known solution }
$i := i + 1$;
EndWhile;

3.1 The Greedy Heuristic

The basic greedy heuristic derives from Prim's algorithm for the *Minimum Spanning Forest* problem [17]: find the unassigned vertex closest to a colony and assign it, repeatedly, until the solution spans the whole graph. Instead of simply

considering the edge costs, the algorithm takes into account the violations of the weight bound and the a posteriori information given by the trails. In other words, instead of minimizing the connection cost $\bar{c}_{rv}$ between vertex v and colony r

$$\bar{c}_{rv} = \min_{u \in U_r} c_{uv} \tag{1}$$

it maximizes a *request factor* f_{rv}

$$f_{rv} = \frac{\tau_{rv}}{\bar{c}_{rv} \pi_{rv}} \tag{2}$$

where π_{rv} is a penalization factor and τ_{rv} is the trail function.

The heaviest tree which could be thus obtained spans the whole graph minus its $p-1$ lightest vertices. Let w_{M} be its weight, and $w_{\mathrm{m}} = \min_{v \in V} w_v$ be the weight of the lightest tree achievable. Therefore, the penalization factor:

$$\pi_{rv} = 1 + \pi \max\left(\frac{w_{U_r} + w_v - W^+}{w_{\mathrm{M}} - W^+}, \frac{W^- - w_{U_r} - w_v}{W^- - w_{\mathrm{m}}}, 0 \right) \tag{3}$$

ranges from 1 for feasible assignments to $1 + \pi$ for the most unbalanced.

At each basic step of the greedy heuristic, the algorithm chooses at random, based on a *deterministic factor* q, whether to obey a

- *deterministic strategy* (probability q): perform the assignment with the highest request factor f_{rv}
- *stochastic strategy* (probability $1-q$): choose at random to assign vertex v to T_r with a probability $p_{rv} = f_{rv} / \sum_{s=1}^{p} \sum_{u \in V} f_{su}$, proportional to f_{rv}

As q increases, the algorithm favours the choices with stronger request factors.

3.2 The Penalization Factor

We assume that, most of the time, the optimal solutions are close to the frontier of the feasible space. So, when the current S solutions are prevailingly feasible, π should be decreased to drive the search to the unfeasible region; when they are unfeasible, π should be increased. If S_{f} current solutions are feasible,

$$\pi_{k+1} = \pi_k \, 2^{(S - 2S_{\mathrm{f}})/S} \quad . \tag{4}$$

3.3 The Trail Function

The trail function saves information from previous runs in order to repeat the good choices and avoid the wrong ones. Its effective management can be interpreted in terms of the *diversification* and *intensification* principles. On one hand, it is profitable to avoid sticking in already explored regions. On the other hand, it is profitable to focus on the regions close to the best known solutions, since the optimal solution is often surrounded by suboptimal ones. A correct balance of these complementary strategies is a key issue to create an efficient heuristic.

At the beginning, the ants release a uniform small trail τ_0 of each colour on each vertex, as a sort of "ground value". While the literature derives τ_0 from the cost of a heuristic solution [9], we employ the cost of the minimum spanning forest, c_{MSF}, which usually has the same order of magnitude:

$$\tau_0 = \frac{1}{n c_{\mathrm{MSF}}} \ . \tag{5}$$

Diversifying the Trail Function. During the greedy heuristic, if an agent assigns vertex v to colony r, it immediately updates the corresponding trail, drawing it closer to the ground level:

$$\tau_{rv} = (1 - \rho)\, \tau_{rv} + \rho \tau_0 \ . \tag{6}$$

The other agents consider this assignment less attractive and prefer different solutions. Equation (6) describes the "evaporation" of the trail: the *oblivion factor* ρ tunes the strength of greediness versus memory.

Intensifying the Trail Function. At the end of the greedy heuristic, the best performing agent increases the trails corresponding to its solution $\tilde{X}$. This encourages to repeat its choices in the following iterations, exploring similar solutions. The better is $\tilde{X}$, the stronger is the effect on the trail:

$$\tau_{rv} = (1 - \rho)\, \tau_{rv} + \rho\, \frac{1}{c_{\tilde{X}} \pi_{\tilde{X}}} \ . \tag{7}$$

3.4 The Root Choice

The position of the roots influences remarkably the final result. At first, we choose them so as to cover the graph uniformly. Given a seed root, the second root is the farthest vertex in the graph, the third is the vertex with the highest total distance from the first two, and so on. Then, alternatively, we apply the greedy heuristic with no random choices and a uniform trail, and we move the roots to the centroids of the trees obtained. The process ends when the roots stabilize or begin to repeat, but it is performed n times, using each vertex as the seed. The best root assignment overall initializes the *CAS*. Every I_{root} iterations of the *CAS*, the roots move the centroids of the trees in the best known solution.

4 The *Swinging Forest* Procedure

Local search is based on the concept of *neighbourhood*, a function associating to each feasible solution F a subset $\mathcal{N}(F)$ of "neighbour solutions". Commonly, it is the set of solutions obtained applying a given family of manipulations (*moves*) to F. Local search procedures start from a current solution, assume one of its neighbours as the *incumbent solution*, and replace the former with it.

Algorithm 2 outlines the behaviour of *Swinging Forest*. First, the *Exchange* procedure improves the starting solution F moving vertices from tree to tree, one

by one or in pairs. This neighbourhood merges two related ones: $\mathcal{N}_1(F)$ includes the feasible forests whose vertices belong, all but one, to the same trees as in F; $\mathcal{N}_2(F)$ includes those whose vertices belong, all but two, to the same trees. We adopt a *steepest descent* strategy, exploring the whole neighbourhood and choosing its best solution as the incumbent. The procedure ends when no neighbour solution is better than the current one. Since the weight bound severely limits these moves, the approach is not very effective.

Then, *Swinging Forest* splits one tree and reoptimizes the solution on and on, until the number of trees p_F reaches p_{M}. The aim is to redistribute the vertices in spite of the weight bound, thanks to the slack capacity of the new trees. Then, the algorithm tries to remove each tree displacing its vertices into the other ones. When it has processed all trees, *Exchange* reoptimizes the solution. This phase ends when the number of trees reaches p_{m}. Then, the algorithm splits some trees to retrieve p. If during a whole period the cost has decreased, a new one starts. If it has not or it is no longer possible to obtain p trees, the algorithm terminates.

Algorithm 2. **SwingingForest**(G, c, w, W, p, F)
$c^* := \infty$;
While $p_F = p$ **and** $c_F < c^*$ **do**
  $F :=$ *Exchange*(G, c, w, W, F); { Steepest descent }
  If $c_F < c^*$ **then** $F := F^*$; $c := c^*$;

  While $p_F < p_{\mathrm{M}}$ **do** { Increase the number of trees }
    $F :=$ *TreeSplitting*(G, c, w, W, F);
    $F :=$ *Exchange*(G, c, w, W, F);
  EndFor;

  $Stop := False$;
  While $p_F > p_{\mathrm{m}}$ **and** $Stop = False$ **do** { Reduce the number of trees }
    $F' :=$ *TreeRemoval*(G, c, w, W, F);
    If $p_{F'} = p_F$ **and** $c_F \leq c_{F'}$
      then $Stop := True$; **else** $F :=$ *Exchange*(G, c, w, W, F');
  EndFor;

  While $p_F < p$ **do** { Retrieve p trees }
    $F :=$ *TreeSplitting*(G, c, w, W, F);
    $F :=$ *Exchange*(G, c, w, W, F);
  EndFor;
  If $p_F = p$ **and** $c_F < c^*$ **then** $F^* := F$; $c^* := c_F$;
EndWhile;
Return F^*;

Vertex Exchanges. An exhaustive search in $\mathcal{N}_1(F) \cup \mathcal{N}_2(F)$ takes $O\left(n^2 p^2 \gamma(m, n)\right)$ time, where $\gamma(m, n)$ is the time to evaluate the minimum spanning forest after a move. For the sake of efficiency, we update the spanning forest as in [12], instead of recomputing it from scratch. Moreover, a feasible move is usually such along a sequence of steps. Keeping a list of the best feasible moves on each couple or triplet of trees, one can perform the best move in the list, cancel those involving the trees modified and evaluate them again. No other move need be considered.

Tree Splitting. The number of trees in the current forest is increased by splitting one of them in two. The algorithm removes the most expensive edge in the forest: at least one of the resulting trees (possibly both) has a large amount of weight slack, which makes the new neighbourhood larger.

Tree Removal. The procedure lists the vertices of a tree, from the leaves up to the root, as this seems the most natural way to "prune" a tree. It evaluates all feasible transfers of the first vertex and performs the best one, if any exists. Then, it considers the second vertex, and so on. In the end, if the tree is empty or the solution has improved, the new solution replaces the current one. Otherwise, the original solution is retrieved. To make the removal easier, trees are processed in increasing cardinality order and, in case of ties, in increasing weight order.

4.1 The *Coloured Ant System* and *Swinging Forest*

Local search is effective in finding good solutions, the *Ant System* in managing intensification and diversification: they can gain much from each other. Our algorithm improves the best solution found at each iteration of the *CAS* by local search before using it to update the trails. This provides a smarter management of the trail, which can drive the *CAS* better. Conversely, the best solution found by the *CAS* is a smarter starting point for *Swinging Forest*. To limit the computational burden, instead of applying *Swinging Forest* at each step, we simply explore the $\mathcal{N}_1$ or the $\mathcal{N}_1 \cup \mathcal{N}_2$ neighbourhood. *Swinging Forest* runs only in the end, on the best solution overall.

5 Computational Results

We led an experimental campaign on the *CAS* and *Swinging Forest*, both as stand-alone algorithms and combined. They were run on a Pentium 450 MHz with a Linux operating system. Benchmark instances for the *WC-GTPP* were not available in the literature: we built them by adapting the *CMST* instances of the OR-Library [3]. For each original instance, we set two values for the number of trees p, so as to obtain tightly and loosely constrained instances. The *TC* and *TE* problems have 41 or 81 vertices, homogeneous weights and euclidean costs. Three upper bounds limit the weight of each tree: $W^+ = 3$, 5 and 10 for $n = 41$, $W^+ = 5$, 10 and 20 for $n = 81$. The *CM* problems do not satisfy the triangle inequality and have non uniform weights. We consider two problem sizes: $n = 50$ and $n = 100$. There are three weight upper bounds: $W^+ = 200$, 400 and 800. Each combination of family, size, weight bound and number of trees corresponds to 5 instances, which leads to a total of 180.

5.1 Parameter Settings

In our experience, the *CAS* always runs n times, since the maximum number of iterations $I_{\max}$ is set to n, the maximum number of non improving iterations

Table 1. Results of the *CAS* with different parameter settings.

	$q = 0.0$			$q = 0.2$			$q = 0.5$		
Class	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$
TC40	8.8%	8.8%	8.6%	8.3%	8.2%	**8.1%**	8.4%	8.5%	8.3%
TC80	4.4%	4.4%	4.3%	4.0%	4.0%	3.8%	3.5%	3.5%	**3.3%**
TE40	10.5%	10.4%	10.5%	9.9%	9.9%	9.6%	9.5%	**9.3%**	9.6%
TE80	12.5%	12.4%	11.9%	9.1%	8.9%	8.8%	8.6%	7.9%	**7.6%**
CM50	32.4%	32.6%	31.4%	31.9%	30.4%	32.3%	31.3%	31.7%	**29.9%**
CM100	41.8%	41.6%	39.8%	37.5%	37.8%	37.1%	34.9%	**34.2%**	34.3%
Avg.	18.4%	18.4%	17.8%	16.8%	16.5%	16.6%	16.0%	15.8%	**15.5%**

I_{best} and the maximum time $T_{\max}$ to infinite. The number of agents S is set to n. The penalization coefficient is at first $\pi_0 = 100$, but in few steps it shifts to the correct range, usually stabilizing on a floating behaviour rather than a fixed value. Experience shows that a frequent update of the roots is advantageous; so, $I_{\text{root}} = 1$. In the end, the minimum and maximum number of trees used by *Swinging Forest* are set, respectively, to $p_{\text{m}} = \lfloor p/1.1 \rfloor$ and $p_{\text{M}} = \lceil p/1.1 \rceil$.

5.2 Experience on the *CAS* Parameters

Table 1 sums up our experience on the deterministic factor q and the oblivion factor ρ. The first column reports the problem classes, the following ones the average gap of the solution with respect to the minimum spanning forest lower bound. We consider three values for q: 0.0 (the choice is taken at random proportionally to the request factor), 0.2 and 0.5 (half of the time the choice is random, the other half it is greedy). As for ρ, we compare a longer ($\rho = 0.1$) a medium ($\rho = 0.2$) and a shorter memory ($\rho = 0.3$). The last two settings ($q = 0.5$ and $\rho = 0.2$ or 0.3) perform consistently better. This is confirmed by the number of best results obtained: $q = 0.0$ determines 39, 45 and 41 best solutions out of 180, respectively for $\rho = 0.1$, 0.2 and 0.3; $q = 0.2$ determines 58, 61 and 63 best solutions, $q = 0.5$ determines 79, 78 and 116. Rather likely, when $\rho = 0.1$ the starting solutions influence too strongly the algorithm: roughly speaking, this influence falls under 5% after 29 diversifying updates, since $(1 - \rho)^{29} \approx 0.047$ (see (6)). When $\rho = 0.2$ or 0.3, the same effect requires only 14 or 9 updates.

5.3 A Comparison between Algorithms

Table 2 compares the results of the algorithms described, namely the greedy heuristic run deterministically with an infinite penalization factor, the *CAS* with $q = 0.5$ and $\rho = 0.3$, *Swinging Forest* initialized with the five best solutions provided by the greedy heuristic (*SF(5)*) and the *CAS* followed by *Swinging Forest* applied on the best known solution (*CAS + SF*). Each column reports the average gap and the execution time T_{tot} in seconds. For the *CAS*, it also reports the time T_{best} required to find the best solution, in seconds. This is often

Table 2. The *CAS* and *Swinging Forest*, combined, improve upon both methods.

	Greedy		*CAS*			*SF(5)*		*CAS+SF(1)*	
Class	Gap	T_{tot}	Gap	T_{best}	T_{tot}	Gap	T_{tot}	Gap	T_{tot}
TC40	11.5%	0.1	8.3%	0.6	3.1	6.1%	14.0	4.1%	6.1
TC80	4.3%	0.8	3.3%	10.5	69.4	1.6%	116.5	1.5%	105.3
TE40	11.9%	0.1	9.6%	0.6	3.1	6.7%	15.5	4.2%	6.6
TE80	10.3%	0.7	7.6%	15.3	71.1	3.7%	282.6	3.0%	123.9
CM50	20.0%	0.2	29.9%	2.6	7.1	16.5%	59.2	5.5%	22.9
CM100	39.4%	3.0	34.3%	49.9	224.7	14.7%	1208.3	13.3%	477.6
Avg.	16.2%	0.8	15.5%	13.3	63.1	8.2%	282.7	5.3%	123.7

Table 3. The interaction between the *CAS* and *Swinging Forest*.

	CAS+SF		*CAS(N1)+SF*		*CAS(N2)+SF*	
Class	Gap	T_{tot}	Gap	T_{tot}	Gap	T_{tot}
TC40	4.1%	6.1	4.0%	6.2	2.9%	42.9
TC80	1.5%	105.3	1.4%	106.3	1.0%	786.9
TE40	4.2%	6.6	3.7%	6.9	2.8%	43.8
TE80	3.0%	123.9	2.8%	127.8	2.2%	995.6
CM50	5.5%	22.9	6.4%	21.0	4.4%	115.2
CM100	13.3%	477.6	15.5%	461.4	10.6%	2502.9
Avg.	5.3%	123.7	5.6%	121.6	4.0%	747.9

much lower, since the stopping conditions are not optimized. The first three algorithms do not dominate each other. The greedy heuristic is very fast, but unstable: in 7 cases out of 180 it could not find a feasible solution (this explains the seemingly better results on the *CM50* class). *Swinging Forest* performs better than the *CAS*, but in a much longer time. By contrast, their combination clearly gives the best results in much lower time.

5.4 Interaction between *CAS* and *Swinging Forest*

Table 3 takes these remarks further, by combining the *CAS* and local search more strictly. In all of the three cases, *Swinging Forest* runs on the best solution found by the *CAS*. In the second and third one, the best solution found at each iteration of the *CAS* is improved by a steepest descent exploration of, respectively, $\mathcal{N}_1$ (*CAS(N1)+SF*) and $\mathcal{N}_1 \cup \mathcal{N}_2$ (*CAS(N2)+SF*). Exploring $\mathcal{N}_1$ seems to bring no advantage, and even some loss on the *CM50* and *CM100* classes, confirming the poor quality of this neighbourhood. By contrast, the exploration of $\mathcal{N}_1 \cup \mathcal{N}_2$ remarkably improves the results, while the computational time, though much longer, is comparable to a multiple start (column *SF(5)* in Table 2).

References

1. K. Altinkemer and B. Gavish. Heuristics with constant error guarantees for the design of tree networks. *Management Science*, 32:331–341, 1988.
2. A. Amberg, W. Domschke, and S. Voß. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice*, 1:9–39, 1996.
3. J. E. Beasley. OR–Library. http:// mscmga.ms.ic.ac.uk/ info.html, 1999.
4. B. Bullnheimer, 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*, 7(1):25–38, 1999.
5. R. Cordone and F. Maffioli. A coloured ant system approach to graph tree partition. In *Proceedings of the ANTS' 2000 Conference*, Brussels, Belgium, September, 2000.
6. R. Cordone and F. Maffioli. On graph tree partition problems. In *Proceedings of EURO XVII*, Budapest, Hungary, July 16-19th, 2000.
7. D. Costa and A. Hertz. Ants can colour graphs. *Journal of Operational Research Society*, 48:295–305, 1997.
8. M. Dorigo and L. M. Gambardella. A study of some properties of Ant-Q. *Lecture Notes in Computer Science*, 1141:656–665, 1996.
9. M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.
10. M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.
11. L. M. Gambardella, E. Taillard, and G. Agazzi. Ant colonies for vehicle routing problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw–Hill, 1999.
12. M. Gendreau, J.-F. Larochelle, and B. Sansò. A tabu search heuristic for the Steiner tree problem. *Networks*, 34(2):162–172, September 1999.
13. N. Guttmann-Beck and R. Hassin. Approximation algorithms for minimum tree partition. *Discrete Applied Mathematics*, 87(1–3):117–137, October 1st, 1998.
14. C. Imielińska, B. Kalantari, and L. Khachiyan. A greedy heuristic for a minimum weight forest problem. *Operations Research Letters*, 14:65–71, September 1993.
15. V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 1999.
16. R. Patterson, E. Rolland, and H. Pirkul. A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. Working paper, University of California, Riverside, September 4th, 1998.
17. R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389, 1957.
18. T. Stützle and H. Hoos. The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 309–314, Piscataway, NJ, 1997. IEEE Press.