An Ant Algorithm for the Steiner Tree Problem in Graphs

Luc Luyet[1], Sacha Varone[2], and Nicolas Zufferey[3,⋆]

[1] Independent consultant, Geneva, Switzerland
[2] Haute École de Gestion de Genève, 1127 Carouge, Switzerland
[3] Faculté des Sciences de l'Administration,
Université Laval, Québec (QC), Canada, G1K 7P4
nicolas.zufferey@fsa.ulaval.ca

Abstract. The Steiner Tree Problem (STP) in graphs is a well-known NP-hard problem. It has regained attention due to the introduction of new telecommunication technologies, since it is the mathematical structure behind multi-cast communications. The goal of this paper is to design an ant algorithm (called ANT-STP) for the STP in graphs which is better than TM, which is a greedy constructive method for the STP proposed in [34]. We derive ANT-STP from TM as follows: each ant is a constructive heuristic close to TM, but the population of ants can collaborate by exchanging information by the use of the trail systems. In addition, the decision rule used by each individual ant is different from the decision rule used in TM. We compare TM and ANT-STP on a set of benchmark problems of the OR-Library.

1 Introduction

Let $G = (V, E, w)$ be a graph, V its vertex set, E its edge set, and w be a cost function which associates a positive cost $w[x, y]$ with each edge $[x, y] \in E$. We define a *tree* in a graph G as a subgraph s of G without any cycle which connects a subset of vertices of V. Given a subset R of V, the Steiner Tree Problem (STP) in graph G consists in finding a tree s in G which connects all the vertices in R (the mandatory vertices) at a minimum cost. Thus, the objective function f to minimize is simply $f(s) = \sum_{[x,y] \in s} w(x, y)$. The vertices in $V - R$ are called the *Steiner vertices*. The decision version of this problem has been shown in [22] to be NP-complete in the general case. As a result, the existing exact methods can not solve large instances and heuristic approaches are required to such instances likely to be encountered in real-life applications of the problem.

In telecommunication networks, data may have to be sent from one or more source(s), which are vertices in R and are called *terminals*, to multiple destinations, as in the case of conference calls or other applications sharing activities. The problem to send data to multiple destinations is known as the multicasting

⋆ Corresponding author.

M. Giacobini et al. (Eds.): EvoWorkshops 2007, LNCS 4448, pp. 42–51, 2007.

routing problem [5], which has often additional delay constraints [27]. Online version of this problem, in which vertices can appear or disappear, can be found in [20], [29]. The weight function w is generally given by a multiple of a transmission capacity unit. Multicast routing problems occur in a variety of fields such as massive multi-player online role playing game [6], video or voice conferences [14], and collaborative virtual environments [13]. Several surveys describe data multicast techniques [25] or their associated combinatorial optimization methods [26].

STP in graphs also occurs in other fields, such as VLSI interconnect layout [4], [18]. VLSI designs generally consists in the interconnection of a set of pins, which must be made electrically connected by a set of wire segments. Although rectilinear metrics have to be considered in VLSI, there is a straightforward equivalence between STP with rectilinear metric and STP in graphs, since Hanan showed in [17] that only Steiner vertices formed by the intersections of vertical and horizontal lines through the vertices have to be considered.

Well known constructive heuristics for the STP in graphs are e.g. the ones proposed in [24], and in [34]. The latter starts with an empty solution s and successively connects a terminal vertex $x \notin s$ to s by the use of the shortest paths of value $SP(x, s)$ from x to s, where $SP(x, s) = \min_{y \in s} d(x, y)$, and where $d(x, y)$ is the length (or cost) of the path connecting x and y in G. At each step, the terminal vertex x such that $SP(x, s) = \min_{y \in R-s} SP(y, s)$ is added to the current partial solution s. Distributed heuristics have also been studied by several authors [23], [3], [32]. Among the most efficient algorithms, we can mention for example [28] and implementations of meta-heuristics such as genetic algorithms [21], [12], GRASP [31], tabu search [16], [30] or simulated annealing [11]. Heuristics for the STP are reviewed e.g. in [19], [36].

The solution space of the tabu search methods proposed in [16] consists in all the minimum spanning trees covering the mandatory vertices. Each element of such a solution space can thus be identified by its subset of Steiner vertices. In order to generate a neighbor solution s' from a solution s, the idea is to add in or remove from s a single Steiner vertex. The reverse of such a move is then tabu for some iterations. In the genetic algorithm presented in [12], the encoding is based on the use of the Distance Network Heuristic (DNH), the deterministic heuristic for the STP proposed in [24]. The genotype *gen* specifies a set of selected Steiner vertices, and a Steiner tree can be build from *gen* by applying DNH on the terminals and selected Steiner vertices.

In this paper, we present a distributed approach using an ant colony system to tackle the STP in graphs. We first describe a preprocessing step which hopefully will decrease the size of the solution space. We then present the main ideas of the ant algorithms and our ant heuristic for the STP. Finally, we show and discuss the obtained results on a set of benchmark instances.

2 Preprocessing Step

Often, the STP in graphs can be reduced using either exclusion or inclusion tests. The former identifies edges or non terminal vertices which do not belong

to at least one minimal Steiner tree, whereas the latter identifies edges or non terminal vertices which belong to all minimal Steiner trees. We choose some of the tests described in [19] for effectiveness reasons.

Four exclusion tests have been selected: non terminal of degree one, non terminal of degree two, long edge and paths with many terminals. The degree of a vertex x is the number of adjacent vertices to x. The Non Terminal of Degree One test (NTD1) removes vertices of degree one which are not in R, since minimal Steiner tree should not contain any non terminal vertices of degree one. The Non Terminal of Degree Two (NTD2) test is based on the following property: let v be a non terminal vertex of degree two, $[x, v]$ and $[v, y]$ be its two associated edges. Then

- if there is no edge $[x, y]$, then v and its two incident edges can be replaced by the edge $[x, y]$ with a weight $w[x, y] = w[x, v] + w[v, y]$;
- if there is an edge $[x, y]$ such that $w[x, y] \leq w[x, v] + w[v, y]$, then there exists a minimal Steiner tree which does not contain v; therefore v and its incident edges can be removed;
- if there is an edge $[x, y]$ such that $w[x, y] > w[x, v] + w[v, y]$, then no minimal Steiner tree contains $[x, y]$; therefore $[x, y]$ can be removed.

The Long Edge (LE) test removes each edge whose weight is strictly greater than the length of a shortest path between its extremities. The Path with many Terminals (PTm) test is a generalization of the LE test which involves the bottleneck Steiner distance. If all the vertices in a path, but its extremities, are non terminals, then this path is said to be elementary. Let P be a path between x and y. P is composed of one or more elementary paths. The longest elementary path in P is said to be the Steiner distance between x and y. The bottleneck Steiner distance $b(x, y)$ between x and y is the shortest Steiner distance among all the paths between x and y. The PTm test removes each edge whose weight is strictly greater than the bottleneck Steiner distance between its extremities.

Two inclusion tests have been selected: the Terminal of Degree One and the Nearest Vertex. The Terminal of Degree One (TD1) contracts each terminal vertex of degree one, since they belong to any (minimal) Steiner tree. The Nearest Vertex (NV) contracts the shortest edge $[z, x]$ incident to a terminal vertex z if the length of a second shortest edge $[z, y]$ is greater than or equal to $w[z, x] + SP[x, z']$, where z' is a terminal vertex.

The order in which those tests are performed is important since it has a strong impact on the computational time. Let SPM be a matrix, called the *shortest path matrix*, such that $SPM(x, y)$ equals the value of the shortest path between x and y. We propose the following preprocessing algorithm, which is close to the one proposed in [12].

While one of the following test is able to reduce G, do:
(a) perform TD1, NTD1, and NTD2;
(b) compute the shortest paths matrix SPM;
(c) perform PTm and LE;

Table 1. Reductions on B instances

	Initial Graph			Reduced Graph		
Instance I	$\|V\|$	$\|R\|$	$\|E\|$	$\|V\|$	$\|R\|$	$\|E\|$
B1	50	9	63	1	1	0
B2	50	13	63	7	4	11
B3	50	25	63	1	1	0
B4	50	9	100	25	6	42
B5	50	13	100	11	4	19
B6	50	25	100	20	9	36
B7	75	13	94	1	1	0
B8	75	19	94	1	1	0
B9	75	38	94	1	1	0
B10	75	13	150	44	9	92
B11	75	19	150	36	5	76
B12	75	38	150	18	9	33
B13	100	17	125	27	9	43
B14	100	25	125	21	8	37
B15	100	50	125	12	8	19
B16	100	17	200	60	9	135
B17	100	25	200	31	8	60
B18	100	50	200	15	7	23

Table 2. Reductions on C and D instances

	Initial graph	Gendreau *et al.*	Our reduced graph
I	$(\|R\|; \|E\|)$	$(\|V\|; \|R\|; \|E\|)$	$(\|V\|; \|R\|; \|E\|)$
C1	(5 ; 625)	(145 ; 5 ; 263)	(138 ; 5 ; 246)
C2	(10 ; 625)	(130 ; 8 ; 239)	(126 ; 8 ; 231)
C3	(83 ; 625)	(125 ;39 ;237)	(95 ;34 ; 178)
C4	(125 ; 625)	(116 ;42 ;233)	(74 ;29 ; 134)
C5	(250 ; 625)	(47 ;24 ;117)	(20 ;13 ; 36)
C6	(5 ; 1000)	(369 ;5 ;847)	(369 ;3 ;841)
C7	(10 ; 1000)	(382 ;9 ;869)	(380 ;9 ;857)
C8	(83 ; 1000)	(335 ;53 ;817)	(334 ;52 ;815)
C9	(125 ; 1000)	(351 ;80 ;834)	(322 ;75 ;711)
C11	(125 ; 2500)	(499 ;5 ;2184)	(498 ;5 ;2036)
C12*	(10 ; 2500)	(498 ;9 ;2236)	(498 ;9 ;2236)
C14	(5 ; 2500)	(414 ;68 ;1886)	(360 ;64 ;1000)
C16†	(5 ; 12,000)	(500 ;5 ;4740)	(500 ;5 ;4740)
C17*	(10 ; 12,000)	(500 ;10 ;4704)	(498 ;8 ;4685)
C18	(83 ; 12,000)	(483 ;67 ;4637)	(461 ;47 ;2575)
C19	(125 ; 12,000)	(433 ;58 ;3431)	(415 ;41 ;2016)
D1	(5 ; 1250)	(274 ;5 ;510)	(273 ;5 ;506)
D2	(10 ; 1250)	(285 ;10 ;523)	(283 ;10 ;519)
D3	(167 ; 1250)	(228 ;10 ;445)	(166 ;58 ;307)
D4	(250 ; 1250)	(180 ;81 ;376)	(105 ;47 ;196)
D12	(10 ; 1250)	(999 ;9 ;4669)	(994 ;9 ;3890)
D7	(10 ; 2000)	(754 ;10 ;1735)	(747 ;10 ;1709)
D8†	(167 ; 2000)	(732 ;124 ;1711)	(773 ;149 ;1753)
D9†	(250 ; 2000)	(660 ;157 ;1613)	(774 ;227 ;1749)
D16*	(5 ; 25,000)	(1000 ;5 ;10595)	(1000 ;5 ;10595)

We present the results of the above reduction procedure in Tables 1 and 2. We consider a set of the benchmark instances available from the OR-library [1], which can be found in [35]. Among the instances of type B, five have been solved to optimality using only the above reduction procedure, and the size of all instances but one has been reduced by half at least. All of them were preprocessed in less than two minutes on a computer *Silicon Graphics Indigo2 (195 MHz, IP28 processor)*. Instances of type C (characterized by $|V| = 500$) and D (characterized by $|V| = 1000$) have also been tested and compared favorably with those obtained in [16]. The drawback of our reduction procedure could rely on computational time, so that PTm may be inadequate for some instances. Therefore, on instances labeled by "*" in Table 2, the PTm test has been replaced by the LE test in order to avoid the computation of the bottleneck Steiner distance. In three cases labeled by "†", the procedure has been stopped after 1 hour of CPU time. In Table 2, we compare our reduction procedure with the one proposed in [16]. We can observe that we obtain better results except for instances D8 and D9.

3 Ant Algorithms

Evolutionary heuristics encompass various algorithms such as genetic algorithms, scatter search, ant systems and adaptive memory algorithms [2]. They can be defined as iterative procedures that use a central memory where information is collected during the search process. Ant colonies were first introduced in [9] and in [10]. In these methods, the central (long term) memory is modeled by a trail system. In the usual ant system, a population of ants is used, where each ant is a constructive heuristic able to build a solution step by step. At each step, an ant adds an element to the current partial solution. Each decision or *move* m is based on two ingredients: the *greedy force* $GF(m)$, which is the short term profit for the considered ant, and the *trails* $Tr(m)$, which represent the information obtained from other ants. Let M be the set of all the possible moves. The probability $p_k(m)$ that ant k chooses move m is given by

$$p_k(m) = \frac{GF(m)^\alpha \cdot Tr(m)^\beta}{\sum_{m' \in M_k(adm)} GF(m')^\alpha \cdot Tr(m')^\beta} \quad (1)$$

where α and β are parameters and $M_k(adm)$ is the set of admissible moves that ant k can perform at that time. In some ant algorithms [7], at each step and for a fixed value of parameter $p \in [0;1]$, a random number r is generated in $[0;1]$. If $r < p$, the chosen move is selected according to Equation (1), otherwise it is the one maximizing $p_k(m)$. When each ant of the population has built a solution, in which case we say that a generation has been performed, the trails are updated, for example as follows: $Tr(m) = \rho \cdot Tr(m) + (1-\rho) \cdot \Delta Tr(m)$, $\forall m \in M$, where $0 < \rho < 1$ is a parameter representing the evaporation of the trails, which is generally close or equal to 0.9, and $\Delta Tr(m)$ is a term which reinforces the trails left on move m by the ant population. That quantity is usually proportional to the number of times the ants performed move m, and to the quality of the obtained solutions when move m has been performed.

In some systems [7], the trails are updated more often (e.g. each time a single ant has built its solution) or are updated by only considering a subset of the ant population (e.g. the ants which provided the best solutions at the end of the current generation). In *hybrid ant systems*, the solutions provided by some ants may be improved using a local search technique. In the *max-min ant systems* [33], the authors proposed to normalize $GF(m)$ and $Tr(m)$ in order to better control these ingredients and thus the search process. An overview of ant algorithms can be found in [8].

4 Proposed Ant Algorithm

In order to propose an ant algorithm for the STP, we mainly have to define: a move, the greedy force of a move, the way to update the trails and the way to select a move.

Let N be the number of ants in the considered population. The role of a single ant k is to build a solution s_k step by step, starting with an empty solution s_k. At each step, as proposed in [34], we perform a move by connecting a terminal vertex $x \notin s_k$ to s_k using the shortest path between x and s_k. Thus, it is straightforward to define the greedy force of a terminal vertex $x \notin s_k$ as $GF(x) = \frac{1}{SP(x,s_k)}$.

To define $Tr(x)$, we associate a trail value $t(v)$ to each non terminal vertex v. $Tr(x)$ is then defined as the average value of t computed by considering each non terminal vertex v which belongs to the shortest path between x and s_k. We chose the average value of t instead of a summation in order to avoid to give too much importance to the terminal vertices which can be connected to s_k by a shortest path containing lots of non terminal vertices. At the end of each generation, the trails are *globally* updated as follows:

$$t(v) = (1 - \rho_g) \cdot t(v) + \rho_g \cdot \Delta t(v),\ \forall v \notin R,$$

where $\rho_g \in [0; 1]$ is a parameter. The reinforcement term $\Delta t(v)$ is updated as follows: $\Delta t(v) = \sum_k \Delta t_k(v)$, where the summation only holds for the N_{best} best ants of the current generation (i.e. the N_{best} ants with the smallest values of the objective function f), where N_{best} is a parameter. It is straightforward to set

$$\Delta t_k(v) = \begin{cases} \frac{1}{f(s_k)} & \text{if } v \in s_k; \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, if $t(v)$ is large, it means on the one hand that most of the N_{best} best ants of the current generation have chosen v in their associated solutions, and on the other hand that these solutions have smaller values of f when compared to the other solutions built during the current generation.

In addition, assuming ant 1 to ant $k-1$ have respectively built their own solutions $s_1, \ldots, s_{k-1}$, when ant k has then built its solution s_k, the trails are *locally* updated as follows:

$$t(v) = \begin{cases} t(v) + \mu \cdot [1 - t(v)] \text{ if } f(s_k) \leq \min\limits_{i \in \{1,\ldots,k-1\}} f(s_i); \\ t(v) \text{ otherwise but } f(s_k) < \frac{1}{k-1} \sum\limits_{i=1}^{k-1} f(s_i); \\ \rho_l \cdot t(v) \text{ otherwise;} \end{cases}$$

where $\rho_l, \mu \in [0; 1]$ are parameters. We can observe that if s_k is the best solution of the current partial generation, the trails of non terminal vertices in s_k are reinforced. If it is not the case but s_k is better than the average quality of the solutions of the current partial generation, the trails of non terminal vertices in s_k keep the same value. Otherwise, such trails are reduced by an evaporation factor ρ_l.

At each step of the construction of a solution by a single ant, a random number r is generated in $[0; 1]$. If $r < p$ (where $p \in [0; 1]$ is a parameter), the chosen move is selected according to Equation (1) using "x" instead of "m" (where $x \in R - s_k$), otherwise (i.e. if $r \geq p$) it is the one maximizing $p_k(x)$. Note that in order to put some diversification in the general process, the first terminal vertex is always randomly selected when an ant starts to build its solution. In addition,

at the very beginning of the process, all the $t(v)$ values are initialized to 0.5. Thus, only the greedy force will guide the choices of the first ant in the first generation. Equivalently, such an ant simply applies the method proposed in [34].

We have now all the ingredients which are necessary to design our general heuristic for the STP in graphs.

1. Initialize $t(v) = 0.5$ for all $v \notin R$;
2. Initialize the parameters: $N, N_{best}, \rho_g, \rho_l, \mu, p, \alpha, \beta$;
3. Set $f^* = \infty$;
4. While 500 generations without improvement of f^* have not been performed, do:
   (a) for $k = 1$ to N, do:
      i. initialize s_k with a randomly chosen terminal z;
      ii. successively connect a terminal vertex x to s_k by the use of the shortest path from x to s_k; repeat this step until all terminal vertices are in s_k;
      iii. update the trails (locally);
      iv. if $f(s_k) < f^*$, set $f^* = f(s_k)$ and $s^* = s_k$;
   (b) update the trails (globally);
5. Return solution s^* of value f^*;

5 Numerical Results

It is important to mention that all the ingredients introduced in the previous section (namely $GF(x), Tr(x), t(v), \Delta t(v)$) are always normalized in interval $[0; 1]$. Such normalization leads to a better control on the search. In addition, preliminary experiments showed that the following parameter setting is appropriate: $N = 30$, $N_{best} = 3$ (thus only the best 10% of the ants are involved to update the trails), $\rho_g = \rho_l = 0.9$ as it is the case in most of the ant algorithms, $\mu = 0.1, p = 0.5, \alpha = 1$ and $\beta = 0.02$. Therefore, it is better to give more weight to the greedy force rather than to the trails. The stopping condition of our algorithm is a maximum number of generations without improvement of the best solution found so far during the search. Such a number is fixed to 500 and all the experiments were performed on a *Silicon Graphics Indigo2 (195 MHz, IP28 processor)*.

We always start our general heuristic by performing the reductions proposed above. For the B instances presented in Table 1, we always got the optimal solution in less than one second. Thus, we will not focus anymore on these instances. Such observation does not hold for the instances presented in Table 2. For these instances, the results are given in Table 3, in which we provide the optimal value of f for each instance (this is denoted by OPT). In this Table, we compare three methods: TM, which is a multi-start constructive method proposed in [34]; TabuGLS, which is an efficient tabu search method proposed in [16]; and ANT-STP, which is our ant algorithm.

We observed that instances C1, C2, C3, C5, C6, C7, C11, C12, C16, C17, D1, D7, D11, D12, and D16 were optimally solved in a few seconds by TM, TabuGLS and ANT-STP. Thus, we do not provide any information on such instances in

Table 3. For each method and each remaining instance I, we only indicate a result if the considered method is not able to find the optimal solution.

On the one hand, it is interesting to compare our method with TM because if we ignore the trails in ANT-STP, we obtain a method which is close to TM. In Table 3, we actually provide the best obtained results if we perform TM $|V|$ times. Each time, we restart the method by initializing the current partial solution with a different vertex in V. In other words, if we allow the same amount of CPU time to TM and ANT-STP, we show that all the ingredient we add to TM in order to derive ANT-STP are useful. Consequently, the collaboration between the ants is well defined.

On the other hand, as the methods TabuGLS and ANT-STP start to work on their associated reduced instances, we will be able to measure the effectiveness of our method when compared to one of the best state-of-the-art method. In the two last columns, we respectively give the CPU times (in seconds and ignoring the time spent for the reductions) T-TabuGLS and T-ANT-STP associated with TabuGLS and ANT-STP. Note that we only indicate such times if they are larger than 60 seconds, and that the TabuGLS method was performed on a computer *Ultra Sparc 1 (170)*, which is comparable to the computer we used for our experiments.

Table 3. Obtained results

I	OPT	TM	TabuGLS	ANT-STP	T-TabuGLS	T-ANT-STP
C4	1079	1080				
C8	509	510			23	74
C9	707	714	708	708	50	3189
C14	323	326	324		37	150
C18	113	122		116	53	943
C19	146	153		169	39	463
D2	220	221			7	
D3	1565	1570	1567	1565	16	266
D4	1935	1936				
D8	1072	1088	1076	1084	244	2312
D9	1448	1471	1451	1465	331	1929

We can first observe that ANT-STP always obtained better results than TM (except on instance C19). If we compare ANT-STP with TabuGLS, we can remark that ANT-STP performed better on instances C14 and D3, but TabuGLS was better on instances C18, C19, C8, and D9. On every other instance, both methods got similar results. However, TabuGLS generally consumes less CPU time than ANT-STP.

6 Conclusion

In this paper, we have presented an ant algorithm, called ANT-STP, to solve the Steiner tree problem in graphs. The role of each ant is to build a solution step by step as in the constructive method TM proposed in [34]. If we ignore the trails, ANT-STP and TM could be considered as similar methods, because at each step of the construction, a terminal vertex x is added to the current partial solution s by the use of the shortest path between x and s. Because ANT-STP was very favorably compared to TM, it is obvious that all the ingredients we

add in TM in order to design ANT-STP are useful. These ingredients mainly are the short term and long term memory, i.e. the trail systems (remember that trails are updated locally and globally), and the way to select the next terminal to add in the current partial solution.

To assess the efficiency of the procedure, ANT-STP was applied to a set of benchmark instances for which the optimal solution is known, and the results were compared to TabuGLS, which is a tabu search heuristic proposed in [16]. We observed that ANT-STP and TabuGLS obtained comparable results but ANT-STP needs more time to get such results.

Finally, we would like to mention that, to improve overall system efficiency, ant algorithms are often enriched with extra capabilities such as lookahead, local optimization, backtracking, and so on (which cannot be found in real ants). This is the case in many implementations, where constructive ant systems have been hybridized with local optimization procedures (see, e.g., [7], [15], [33]]). In contrast with such hybrid ant heuristics, ANT-STP is competitive without using any local search procedure to improve the solutions built by the ants.

References

1. Beasley, J.: Or-library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41** (1990) 1069–1072
2. Calegari, P., Coray, C., Hertz, A., Kobler, D., and Kuonen, P.: A taxonomy of evolutionary algorithms in combinatorial optimization. Journal of Heuristics **5** (1999) 145–158
3. Chen, G., Houle, M., and Kuo, M.: The steiner problem in distributed computing systems. Information Sciences **74(1)** (1993) 73–96.
4. Cong, J., He, L., Koh, C., and Madden, P.: Performance optimization of vlsi interconnect layout. Integration: the VLSI Journal **21** (1996) 1–94
5. Deering, S., and Cheriton, D.: Multicast routing in datagram internetworks and extended lans. ACM Transaction on Computer Systems **8(2)** (1990) 85–110
6. Diot, C., and Gautier, L.: A distributed architecture for multiplayer interactive applications on the internet. IEEE Network **13(4)** (1999) 6–15
7. Dorigo, M., and Gambardella, L.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation **1** (1997) 53–66
8. Dorigo, M., Di Caro, G., and Gambardella, L.: Ant algorithms for discrete optimization. Artificial Life **5** (1999) 137–172
9. Dorigo, M., Maniezzo, V., and Colorni, A.: Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Dipartimento di Elettronica, Italy (1991)
10. Dorigo, M.: Optimization, learning and natural algorithms (in Italian). Ph.D. Dissertation, Politecnico di Milano, Dipartimento di Elettronica, Italy (1992)
11. Dowsland, K.: Hill-climbing simulated annealing and the steiner problem in graphs. Engineering Optimization **17** (1991) 91–107
12. Esbensen, H.: Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm. Networks: An International Journal **26** (1995) 173–185
13. Fisher, H.: Multicast issues for collaborative virtual environments. IEEE Computer Graphics and Applications **22(5)** (2002) 68–75

14. Foreman, D.: Managing data in distributed multimedia conferencing applications. IEEE Multimedia 9(4) (2002) 30–37
15. Gambardella, L. M., and Dorigo, M.: HAS-SOP: An hybrid ant system for the sequential ordering problem. Tech. Rep. 11-97, Lugano, Switzerland: IDSIA (1997)
16. Gendreau, M., Larochelle, J.-F., and Sansò, B.: A tabu search heuristic for the steiner tree problem. Networks **34(2)** (1999) 162–172
17. Hanan, M.: On steiner's problem with rectilinear distance. SIAM Journal of Applied Mathematics **14(2)** (1966) 255–265
18. Hu, J., and Sapatnekar, S. S.: A survey on multi-net global routing for integrated circuits. Integration: The VLSI Journal **31** (2001) 1–49
19. Hwang, F., Richards, D., and Winter, R.: The Steiner tree problem, volume 53 of Ann. of Discrete Math. Amsterdam: North-Holland (1992)
20. Imase, M., and Waxman, B. M.: Dynamic steiner tree problem. SIAM J. Discrete Math. **4(3)** (1991) 369–384
21. Kapsalis, A., Rayward-Smith, V., and Smith, G.: Solving the graphical steiner tree problem using genetic algorithms. Journal of the Operational Research Society **44** (1993) 397–406
22. Karp, R. M.: Reducibility among combinatorial problems. Plenum Press. (1972) 85–103
23. Kompella, V., Pasquale, J., and Polyzos, G.: Two distributed algorithms for the constrained steiner tree problem. In Proceedings of the Second International Conference on Computer Communications and Networking (ICCCN'93) (1993) 343–349
24. Kou, L., Markowsky, G., and Berman, L.: A fast algorithm for steiner trees. Acta Informatica **15** (1981) 141–145
25. Mir, N.: A survey of data multicast techniques, architectures, and algorithms. IEEE Communications Magazine **39(9)** (2001) 164–170
26. Oliveira, C. A. S., and Pardalos, P. M.: A survey of combinatorial optimization problems in multicast routing. Comput. Oper. Res. **32(8)** (2005) 1953–1981
27. Pasquale, J., Polyzos, G. C., and Xylomenos, G.: The multimedia multicasting problem. Multimedia Systems **6(1)** (1998) 43–59
28. Polzin, T., and Daneshmand, S. V.: The multimedia multicasting problem. Multimedia Systems **6(1)** (1998) 43–59
29. Raghavan, S., Manimaran, G., and Murthy, C. S. R.: A rearrangeable algorithm for the construction delay-constrained dynamic multicast trees. IEEE/ACM Trans. Netw. **7(4)** (1999) 514–529
30. Ribeiro, C. C., and Souza, M. C. D.: Tabu search for the steiner problem in graphs. Networks **36(2)** (2000) 138–146
31. Ribeiro, C. C., Uchoa, E., and Werneck, R. F.: A hybrid grasp with perturbations for the steiner problem in graphs. INFORMS J. on Computing **14(3)** (2002) 228–246
32. Shaikh, A., and Shin, K. G.: Destination-driven routing for low-cost multicast. IEEE Journal of Selected Areas in Communications **15(3)** (1997) 373–381
33. Stuetzle, T., and Hoos, H.: The MAX-MIN Ant System and local search for the traveling salesman problem. In T. Baeck, Z. Michalewicz, and X. Yao (Eds.) Proceedings of IEEE-ICEC-EPS97. Piscataway, NJ: IEEE Press (1997) 309-314
34. Takahashi, H., and Matsuyama, A.: An approximate solution for the steiner problem in graphs. Math. Japonica **24(6)** (1980) 573–577
35. Voss, S., Martin, A., and Koch, T.: Steinlib testdata library. online
36. Voss, S.: Steiner's problem in graphs: Heuristic methods. Discrete Applied Mathematics **40** (1992) 45–72