Alcom-FT Technical Report Series
ALCOMFTTR-03-105 (revised November 29, 2003)

Ant Colony Optimization for the maximum disjoint paths problem*

Maria Blesa[1] and Christian Blum[2]

[1] Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain
mjblesa@lsi.upc.es

[2] IRIDIA
Université Libre de Bruxelles, B-1050 Brussels, Belgium
cblum@ulb.ac.be

Abstract. One of the basic operations in communication networks consists in establishing routes for *connection requests* between physically separated network nodes. In many situations, either due to technical constraints or to quality-of-service and survivability requirements, it is required that no two routes interfere with each other. These requirements apply specially to routing and admission control in large-scale, high-speed and optical networks. The same requirements also arise in a multitude of other applications such as real-time communications, VLSI design, scheduling, bin packing, and load balancing.
Given a graph G representing a network topology, and a collection $T = \{(s_1, t_1) \ldots (s_k, t_k)\}$ of pairs of vertices in G representing connection request, the maximum *disjoint paths problem* is an NP-hard problem that consists in determining the maximum number of pairs in T that can be routed in G by mutually disjoint $s_i - t_i$ paths.
We propose an *Ant Colony Optimization* (ACO) algorithm to solve this problem. ACO algorithms are inspired by the foraging behavior of real ants, whose distributed nature makes them suitable for the applications both in static and dynamic network environments. Our current version is aimed for the application in static graphs. In comparison to a multi-start greedy approach, our algorithm has advantages especially when speed is an issue.

1 Introduction

One of the basic operations in communication networks consists in establishing routes for *connection requests* between physically separated network endpoints that wish to establish a connection for information exchange. Many connection requests occur simultaneously in a network, and it is desirable to establish routes for as many requests as possible. In many situations, either because of technical constraints or just to improve the communication, it is required that no two routes interfere with each other, which implies not to share network resources such as links or switches.

Let $G = (V, E)$ be an edge-weighted undirected graph representing a network in which the nodes represent the hosts and switches, and the edges represent the links. The weight $w(e) > 0$ of every edge in the graph corresponds to the distance between its endpoints. Let $T = \{(s_j, t_j) \mid j = 1, \ldots, |T|; s_j \neq t_j \in V\}$ be a list of *commodities*, i.e., pairs of nodes in G, representing endpoints demanding connection requests. T is said to be *realizable* in G if there exist mutually edge-disjoint (respectively vertex-disjoint) paths from s_j to t_j in G, for every $j = 1, \ldots, |T|$. Deciding whether a given set of pairs is realizable in a given graph is one of Karp's original NP-complete problems

* Partially supported by the FET Programme of the EU under contract numbers IST-1999-14186 (ALCOM-FT) and by the Spanish CICYT project TIC-2002-04498-C05-03 (TRACER). M. Blesa acknowledges support by the Catalan Research Council (grant 2001FI-00659). C. Blum acknowledges support by the *Metaheuristics Network*, a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

[12] (other references on the computational complexity of the problem are [15, 18]). The problem remains NP-complete even when the underlying graph is a two-dimensional mesh.

The combinatorial optimization version of the problem consists in satisfying as many of the requests as possible, which is equivalent to finding a realizable subset of T of maximum cardinality. A solution S to the combinatorial optimization problem is a set of edge-disjoint paths for a subset of the commodities. (Note that this includes the possibility of having a solution that contains a path for each commodity). The objective function value $f(S)$ of a solution S is defined as $f(S) = |S|$.

Disjointness can refer either to nodes or to edges, the latter being the most commonly studied version. In this work, we deal with the edge-disjoint version of the problem (from the point of view of combinatorial optimization), and refer to it as the maximum *edge-disjoint paths* (EDP) problem. In the extreme case in which the list of commodities is composed by repetitions of the same pair (s, t), the problem is known as *edge-disjoint Menger* problem. The EDP problem is a simpler version of the more general *unsplittable flow* problem, when demands, profits, and capacities are considered to be one.

The EDP problem is interesting from the point of view of different research fields such as combinatorial optimization, algorithmic graph theory and operations research. It has a multitude of applications in areas such as real-time communications, VLSI-design, scheduling, bin packing, load balancing, and it has recently been brought into focus in works discussing applications to routing and admission control in modern networks, namely large-scale, high-speed and optical networks [4, 16, 1, 2]. Concerning real-time communications, the EDP problem is very much related to survivability and information dissemination. Concerning survivability, it is especially desirable to establish disjoint paths in order to minimize the effect of possible failures occurring in the base network. Furthermore, to communicate via multiple disjoint paths can increase the effective bandwidth between pairs of nodes, reduce congestion in the network and increase the velocity and the probability of receiving the information [11, 17]. This becomes especially important nowadays due to the type of information that circulates over networks (e.g., media files), which requires fast, qualified and reliable connections.

To the best of our knowledge, there is a lack of efficient algorithms for the EDP problem that could be applied to real communication networks. In fact, much of the difficulty in establishing virtual circuits for routing information in those networks comes from a lack of understanding the problem and the lack of good heuristics for finding disjoint paths. With the aim of contributing on reducing this lack of efficient algorithms for the EDP problem, we have developed an Ant Colony Optimization (ACO) algorithm [7, 10, 8] to solve the task of finding edge-disjoint paths. Ant Colony Optimization is a relatively new metaheuristic approach for solving hard combinatorial optimization problems. One of the reasons that encouraged our choice of ACO is the successful application to adaptive routing in communication networks [6]. ACO approaches provide many advantageous features, such as for example the usage of only local information, that are useful when applications in dynamically changing environments are concerned. ACO algorithms are composed by independently operating computational units that generate a global perspective without the necessity of direct interaction. Our current approach is developed for the EDP problem in static graphs.

The paper is organized as follows. In Section 2 we present a (multi-start) greedy approach for the EDP problem. In Section 3 we present our Ant Colony Optimization approach. The set of benchmark instances used to test the performance of our approach are introduced in Section 4. The graphs considered represent real-life communication networks. In Section 5 we analyze the results obtained and point out possible lines for future research, since our ACO algorithm is a first approach to solve the problem via metaheuristics.

2 A greedy approach

The interest in greedy algorithms is often motivated by the fact that they usually provide quite good solutions in a reasonable computation time. In this section we propose a multi-start greedy algorithm, which is based on the simple greedy algorithm (SGA) proposed in [13] for the development of approximation algorithms for very specific types of graphs.

The SGA is a natural way of approximating the EDP problem that works as follows. It starts with an empty solution S. Then, it proceeds through the commodities in order. For routing each commodity $T_i \in T$, it considers the graph G without the edges that are already in the paths of the solution S under construction. The shortest path between s_i and t_i (with respect to the edge-weights) is assigned as path for the commodity $T_i = (s_i, t_i)$. Note that this off-line algorithm can be naturally considered an on-line algorithm and then, the lower bounds of [3] would imply that it does not achieve a good performance ratio on graphs such as trees and two-dimensional meshes. Note that the solutions it provides are very much dependent on the order in which the commodities are provided.

We extend this simple greedy algorithm to a multi-start version in which the input list of commodities is permuted randomly several times (i.e., N_{perm} times). SGA is then applied to each of these permutations. This new greedy approach is pseudo-coded in Algorithm 1 and, in the following, we refer to it as *multi-start greedy algorithm* (MSGA).

Algorithm 1 : Multi-start simple greedy algorithm (MSGA) for the EDP problem

input: a problem instance (G, T, N_{perm})
$S_{best} \leftarrow \emptyset$, $\pi \leftarrow (1, 2, \dots, |T|-1, |T|)$
for $i = 1$ to N_{perm} **do**
  $S_i \leftarrow \emptyset$, $\hat{E} \leftarrow E$ **// Simple greedy algorithm** [13]
  for $j = 1, \dots, |T|$ **do**
    if $s_{\pi(j)}$ and $t_{\pi(j)}$ can be connected by a path in $G = (V, \hat{E})$ **then**
      $P_{\pi(j)} \leftarrow$ shortest path from $s_{\pi(j)}$ to $t_{\pi(j)}$ in $G = (V, \hat{E})$
      $S_i \leftarrow S_i \cup P_{\pi(j)}$, $\hat{E} \leftarrow \hat{E} \setminus \{e \mid e \in P_{\pi(j)}\}$
    end if
  end for
  if $f(S_i) > f(S_{best})$ **then**
    $S_{best} \leftarrow S_i$
  end if
  if $i < N_{perm}$ **then** $\pi \leftarrow$ GenerateRandomPermutation($|T|$)
end for
output: S_{best}

In Algorithm 1, $e \in P_j$ denotes the fact that edge e is on path P_j, S_i denotes the solution under construction in the embedded SGA, and S_{best} denotes the best solution found so far. The method GenerateRandomPermutation($|T|$) provides the next permutation π of the commodities for every iteration of the external loop.

3 ACO approach

Ant Colony Optimization (ACO) [8] is inspired by the foraging behavior of real ants. This behavior enables them to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called *pheromone* on the ground. When they decide about a direction to go, they choose probabilistically paths marked by strong pheromone concentrations. This behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

In ACO algorithms, artificial ants incrementally construct a solution by adding appropriately defined solution components to a partial solution under consideration. Each of the construction steps is a probabilistic decision based on local information, which is represented by the *pheromone* information. The exclusive use of local information is certainly a desirable property for algorithms that are aimed for the application in large-scale networks. This property makes ACO algorithms a natural choice for the application to the EDP problem.

3.1 Main algorithm

In the following we outline our ACO approach, which is based on a decomposition of the EDP problem. Each problem instance $\mathcal{P} = (G, T)$ of the EDP problem can be naturally decomposed into $|T|$ subproblems $\mathcal{P}_j = (G, T_j)$, with $j \in \{1, \ldots, |T|\}$, by regarding the task of finding a path for a commodity $T_j \in T$ as a problem itself. With respect to this problem decomposition, we use a number of $|T|$ ants each of which is assigned to exactly one of the subproblems. Therefore, the construction of a solution consists of each ant building a path between the two endpoints of her commodity. Obviously, the subproblems are not independent as the set of $|T|$ paths constructed by the ants should be mutually edge-disjoint.

Ant solutions

A solution S constructed by $|T|$ ants is a set of not necessarily edge-disjoint paths that contains a path for each commodity. We henceforth refer to them as *ant solutions*. From each ant solution a valid EDP solution can be produced by iteratively removing the path which has most edges in common with other paths, until all remaining paths are mutually edge-disjoint.

The objective function $f(\cdot)$ of the problem is characterized by having many plateaus when it is applied to ants solutions, as many ant solutions have the same number of disjoint paths. Thus, a consequence of decomposing the EDP problem is the need to define an objective function $f^a(\cdot)$ for ant solutions. Therefore, referring to $f(S)$ as a *first criterion*, we introduce a *second criterion* $C(S)$, which is defined as follows:

$$C(S) = \sum_{e \in E} \left(\mathsf{max} \left\{ \left(\sum_{P_j \in S} \delta^j(S, e) \right) - 1, 0 \right\} \right) \;, \text{ where } \delta^j(S, e) = \begin{cases} 1 & : \; e \in P_j \in S \\ 0 & : \; \text{otherwise.} \end{cases}$$

This second criterion tries to quantify the degree of non-disjointness of an ant solution. If all the paths in a solution S are edge-disjoint, $C(S)$ is zero. In general, $C(S)$ increases when increasing the usage of common edges in S. Therefore, based on the idea that "the less edges are shared in a solution, the closer the solution is to disjointness", a function $f^a(\cdot)$ that differentiates between ant solutions can be defined as follows. For two ant solutions S and S', it holds that

$$f^a(S) > f^a(S') \Leftrightarrow \underbrace{(f(S) > f(S'))}_{1^{st} \text{ criterion}} \text{ or } \underbrace{((f(S) = f(S') \text{ and } (C(S) < C(S'))}_{2^{nd} \text{ criterion}} \;. \tag{1}$$

Pheromone models

The problem decomposition as described above implies that we use a pheromone model τ^j for each subproblem $\mathcal{P}_j$. Each pheromone model τ^j consists of a pheromone value τ^j_e for each edge $e \in E$. The set of $|T|$ pheromone models is henceforth denoted by $\tau = \{\tau^1, \ldots, \tau^{|T|}\}$. The pheromone values are bounded by $\tau_{\min}$ and τ_{max}, which prevent the algorithm from converging to a solution. These bound are set to 0.001 and 0.999, respectively.

Algorithm framework and components

Algorithm 2 is a high level description of our ACO algorithm. Three different solutions are kept in the algorithm. S_{ibest} is the *iteration-best* solution, i.e., the best ant solution generated in the current iteration. S_{pbest} is the *currently best* solution, i.e., the best ant solution generated since the last escape action (see below). S_{gbest} is the the *best-so-far* solution, i.e., the best ant solution found since the start of the algorithm.

To control the improvement of the criteria we introduce the counters c_{crit1} and c_{crit2}, and the boolean variable *all_update*. The counters c_{crit1} and c_{crit2} count the number of successive iterations without improvement of the first and the second criterion, respectively. The variable *all_update* is set to true if c_{crit1} exceeds a limit $c1_{\max}$.

In the following, we give a high-level description of the algorithm. The main procedures used by the algorithm are explained in detail in the following of the section. First, all the variables are initialized. In particular, the pheromone values are set to their initial value $\tau_{\min}$ by the procedure $\mathsf{InitializePheromoneValues}(\tau)$. Second, a number of N_{sols} ant solutions are constructed per iteration (where one ant solution is constructed in that each ant applies the function

$\mathsf{ConstructPath}((s_{\pi(j)}, t_{\pi(j)}), \tau^{\pi(j)})$). At each iteration, the first of those N_{sols} ant solutions is constructed by sending the ants in the order in which the commodities are given in T. For each further ant solution construction in the same iteration, the order in which the ants construct a path for their commodity is randomly generated by the function $\mathsf{GenerateRandomPermutation}(|T|)$. Third, the value of the variables S_{ibest}, S_{gbest} and S_{pbest} is updated. Finally, the pheromone values are updated.

The pheromone update depends on the search phase in which our algorithm is at the moment of the update. Our algorithm works in two phases based on the two criteria of function $f^a(\cdot)$ as outlined in Equation 1. First, it tries to improve only the first criterion. If for a number of $c1_{\max}$ iterations the first criterion could not be improved, then the algorithm tries to improve the second criterion. In case of success the algorithm jumps back to the first phase trying to improve again the first criterion. Otherwise, if for a number of $c2_{\max}$ iterations the second criterion could not be improved, then some of the paths from the EDP solution that can be produced from S_{pbest} are deleted from S_{pbest}. This action can be seen as a backtracking move in order to escape from the current area in the search space. The algorithm is iterated until some opportunely defined termination conditions are satisfied. The algorithm returns the EDP solution generated from S_{gbest}.

The components of our algorithm are outlined more in detail in the following.

Algorithm 2 : ACO for the EDP problem

```
input: a problem instance (G, T)
S_gbest ← ∅, S_pbest ← ∅, c_crit1 ← 0, c_crit2 ← 0
all_update ← FALSE
InitializePheromoneValues(τ)
while termination conditions not met do
  π ← (1, 2, ..., |T| − 1, |T|)
  for i = 1 to N_sols do
    S_i ← ∅
    for j = 1 to |T| do
      P_π(j) ← ConstructPath((s_π(j), t_π(j)),τ^π(j))
      S_i ← S_i ∪ {P_π(j)}
    end for
    if i < N_sols then π ← GenerateRandomPermutation(|T|)
  end for
  Determine S_ibest ∈ {S_i | i = 1, ..., N_sols} such that f^a(S_ibest) >= f^a(S), ∀ S ∈ {S_i | i = 1, ..., N_sols}
  if f(S_ibest) > f(S_gbest) then S_gbest ← S_ibest
  if f^a(S_ibest) > f^a(S_pbest) then
    c_crit2 ← 0
    if f(S_ibest) > f(S_pbest) then
      S_update ← ExtractDisjointPaths(S_ibest)        // First phase
      c_crit1 ← 0
      all_update ← FALSE
    else
      c_crit1 ← c_crit1 + 1
    end if
    S_pbest ← S_ibest
    if all_update then S_update ← S_pbest          // Second phase
  else
    c_crit2 ← c_crit2 + 1
  end if
  if all_update then
    if c_crit2 > c2_max then
      S_pbest ← DestroyPartially(S_pbest)          // Escape mechanism
      S_update ← ExtractDisjointPaths(S_pbest)
      c_crit2 ← 0, c_crit1 ← 0
    end if
  else
    all_update ← (c_crit1 > c1_max)
  end if
  UpdatePheromoneValues(τ,S_update,all_update)
end while
output: the EDP solution generated from the best solution S_gbest
```

· $\mathsf{InitializePheromoneValues}(\tau)$ initializes all the pheromone values $\tau_e^j \in \tau^j \in \tau$ to τ_{min}.

· $\mathsf{ConstructPath}((s_j, t_j), \tau^j)$: For constructing a path between the endpoints of the commodity (s_j, t_j), an ant first chooses randomly to start either from the source s_j or the target t_j. Then, the ant iteratively moves from node to node using available edges that are not already in the path P_j under construction, and that are not labelled forbidden by a backtracking move. Backtracking is done in case the ant finds itself in a node in which all the incident edges have been used, or if all the incident edges are labelled forbidden. Note, that with this strategy the ant will find a path between source and target, if there exists one. Otherwise, the ant returns an empty path. Therefore, this iterative process is stopped if either the goal node is reached, or if the path could not constructed because at some point all incident edges were labelled forbidden. In the following the current node is denoted by v_c, the goal node is denoted by v_g, and the set of allowed edges in G (i.e., those not used yet in the path and not labelled as forbidden) is denoted by $\mathcal{I}^\star_{v_c}$. At each construction step, the choice of where to move to has a certain probability to be done deterministically. This is a feature we adopt from a well-working ACO variant called Ant Colony System (ACS) which was proposed by Dorigo and Gambardella in [9]. We draw a random number d_{rate} between 0 and 1. If $d_{rate} \leq 0.8$ (where 0.8 was chosen by parameter tuning as outlined in Section 4), the next edge to join path P_j is chosen deterministically:

$$e^* = \{v_c, u\} \leftarrow \mathsf{argmax}\ \{\tau_e^j \cdot \mathbf{p}(D_e)^\beta \cdot \mathbf{p}(U_e)^\gamma \mid e \in \mathcal{I}^\star_{v_c}\}\ , \tag{2}$$

where $\mathbf{p}(D_e)$ is a value that determines the influence of the distance from v_c via u to the goal vertex v_g, and $\mathbf{p}(U_e)$ is a value that determines the influence of the overall usage of edge e, which is the information whether e is already used in another path for the same solution. The length of the shortest path between two vertices u and v in G is henceforth denoted by $\sigma(u, v)$. $\mathbf{p}(D_e)$ and $\mathbf{p}(U_e)$ are defined as follows:

$$\mathbf{p}(D_{e=\{v_c,u\}}) \leftarrow \frac{(\sigma(u,v_g)+w(e))^{-1}}{\sum\limits_{e'=\{v_c,u'\}\in\mathcal{I}^\star_{v_c}} (\sigma(u',v_g)+w(e'))^{-1}}$$

$$\mathbf{p}(U_e) \leftarrow \frac{U(e)^{-1}}{\sum\limits_{e'\in\mathcal{I}^\star_{v_c}} U(e')^{-1}}\ , \quad \text{in which} \quad U(e) = \begin{cases} 2 : e \text{ already used in } S_i \\ 1 : \text{otherwise} \end{cases}$$

If $d_{rate} > 0.8$, the next edge e^* is chosen according to the following transition probabilities:

$$\mathbf{p}(e \mid \mathcal{I}^\star_{v_c}) = \begin{cases} \dfrac{\tau_e^j \cdot \mathbf{p}(D_e)^\beta \cdot \mathbf{p}(U_e)^\gamma}{\sum\limits_{e'\in\mathcal{I}^\star_{v_c}} \tau_{e'}^j \cdot \mathbf{p}(D_{e'})^\beta \cdot \mathbf{p}(U_{e'})^\gamma} & : \text{ if } \quad e \in \mathcal{I}^\star_{v_c} \\ 0 & : \text{ otherwise.} \end{cases} \tag{3}$$

The high probability of doing a deterministic step leads the search already at the start of the algorithm to relatively good areas in the search space. However, on the other side there is the danger that the algorithm gets stuck in sub-optimal regions. The use of the pheromone information τ_e^j ensures the flexibility of the algorithm, whereas the use of $\mathbf{p}(D_e)^\beta$ ensures a bias towards short paths, and $\mathbf{p}(U_e)^\gamma$ ensures a bias towards disjointness of the $|T|$ paths constituting a solution.

After every ant has constructed its path and the solution S is completed, we apply another feature of ACS, namely the evaporation of some amount of pheromone from the edges that were used by the ants. Given a solution S, the evaporation is done as follows:

$$\tau_e^j \leftarrow \begin{cases} (1 - e_{rate}) \cdot \tau_e^j & : \quad e \in P_j \in S,\ \ j = 1, \ldots, |T| \\ \tau_e^j & : \quad \text{otherwise}\ . \end{cases}$$

The reason for this pheromone evaporation is the desire to diversify the search in the sense that for the last solution constructions per iteration, the probability to construct the same solution as was obtained by the first solution construction, is decreased. After parameter tuning we chose a setting of $e_{rate} = 0.05$.

· $\mathsf{ExtractDisjointPaths}(S_{pbest})$: This function produces a valid EDP solution S_{update} from the ant solution S_{pbest}. The solution S_{update} is then used for updating the pheromone values in the phase of the algorithm in which it tries to improve the first criterion (i.e., the number of disjoint paths in a

Table 1. Our new set of benchmark graphs.

Graph	$\|V\|$	$\|E\|$	Degree (min.,	avg.,	max.)	Diameter	Represents...
graph3	164	370	1	4.512	13	16	Deutsche Telekom's subnetwork
graph4	434	981	1	4.521	20	22	Deutsche Telekom's subnetwork
bl-wr2-wht2.10-50	500	1020	2	4.08	13	23	Internet topology (generated)

solution). Additionally, this procedure resets the pheromone values corresponding to commodities for which a path does not exist in S_{update}. A reset of the pheromone values consists in re-initializing them to $\tau_{\min}$.

· DestroyPartially(S_{pbest}): In cases in which neither the attempt of improving the first criterion, nor the attempt of improving the second criterion is successful (i.e., *all_update* is true and $c_{crit2} > c2_{\max}$) an escape mechanism is employed. This escape mechanism consists in removing one quarter of the disjoint paths of the EDP solution that corresponds to solution S_{pbest}. This mechanism is similar to the perturbation mechanism made in local search based methods such as Iterated Local Search or Variable Neighborhood Search. Finally, the pheromone values that correspond to the commodities of the removed paths are reset to $\tau_{\min}$.

· UpdatePheromoneValues(τ,S_{update},*all_update*): Our ACO algorithm uses at any time only one solution, namely the solution S_{update}, for updating the pheromone values. This solution is obtained from solution S_{pbest}. The way in which this solution is obtained depends on the current search phase (first or second) of the algorithm. When *all_update* is false, i.e., the algorithm is in the phase in which it tries to improve the first criterion, S_{update} is obtained from S_{pbest} as described in function ExtractDisjointPaths(S_{pbest}) above. In this case only the pheromone values of commodities for which disjoint paths exist are updated. When *all_update* is true, i.e., the algorithm is in the phase in which it tries to improve the second criterion, S_{update} is equal to S_{pbest}. In this case, the pheromone values of all commodities are updated. This introduces a bias towards ant solutions in which less edges are shared among the paths, which in turn might eventually lead to an improvement in terms of the first criterion. Given a solution S_{update} the pheromone values for all $j \in \{1, \ldots, |T|\}$ for which $P_j \in S_{update}$ are updated as follows:

$$\tau_e^j \leftarrow \max\left\{\tau_e^j + \rho \cdot \left(1 - \tau_e^j\right),\ \tau_{max}\right\} \quad \forall\ e \in P_j \ , \tag{4}$$

where $\rho \in (0, 1]$ is a constant value called *learning rate*. This rate ρ is set to 0.1 when *all_update* does not hold, i.e., in the first search phase, and it is set to 0.5 otherwise. The update rule as outlined above can be regarded as the update rule of ACS when implemented in the hyper-cube framework [5].

4 Benchmark and experiments

So far there is no set of benchmark instances available for the EDP problem. To experimentally evaluate our ACO approach we have considered three graphs representing different communication networks. The graphs graph3 and graph4 are parts of the communication network of the Deutsche Telekom AG, Germany.[3] Using the network topology generator BRITE [14], we have generated another graph representing a two-level top-down *hierarchical topology* (AS-level plus a router level). This graph, which we henceforth denote by bl-wr2-wht2.10-50, has 10 nodes placed randomly in the AS-level topology and, for each of these nodes, there is a router-level topology with 50 nodes which are placed according to *heavy-tailed* distributions. The nodes of the topologies in both levels are interconnected using the Waxman probability model [19]. Table 1 shows other features of these three chosen graphs.

[3] We would like to thank Martin Oellrich for providing these graphs.

Table 2. Comparison of the results obtained by MSGA and ACS. The table layout is as follows. The first column gives the name of the graph and the second column the number of commodities. For both algorithms there are three columns reporting on the results obtained for the 20 instances of each combination of graph and number of commodities. The first column shows the average of the values of the best solutions found for the 20 instances. Such an average is in boldface, if the result is the best in the comparison. In case of ties the average computation time decides. The second column provides the standard deviation, and the third column reports on the average time needed to find the best solution values for the 20 instances. Finally, the last column shows the computation time of MSGA, which was used as computation time limit for ACO.

Graph	nb.pairs	MSGA			ACS			
		$\overline{\text{Best }\lvert S\rvert}$	σ	$\overline{\text{Best } t}$	$\overline{\text{Best }\lvert S\rvert}$	σ	$\overline{\text{Best } t}$	$\bar{t}$
graph3	10	9.850	0.357	0.629	**9.850**	0.357	0.015	6.535
graph3	25	**23.150**	1.276	3.629	22.450	1.396	2.998	8.841
graph3	50	33.400	2.223	6.342	**33.85**	2.555	3.862	12.190
graph3	75	36.400	2.577	7.024	**39.45**	3.008	6.518	15.599
graph4	10	10	0	15.734	**10**	0	0.842	134.270
graph4	25	24.6	0.490	25.317	**24.75**	0.433	6.135	161.338
graph4	50	**45.85**	2.455	79.901	44.95	2.334	106.276	188.478
graph4	75	**57.1**	3.491	87.597	55.050	3.653	123.629	218.371
bl-wr2-wht2.10-50	10	8.950	0.92	61.504	**9**	0.894	9.070	240.559
bl-wr2-wht2.10-50	25	14.9	1.729	94.546	**16.150**	1.492	40.535	267.923
bl-wr2-wht2.10-50	50	22.200	2.657	129.614	**24.75**	2.165	48.880	313.972
bl-wr2-wht2.10-50	75	25.35	3.439	121.549	**29.35**	3.245	76.097	354.434

Instances and platform. For each of our three benchmark graphs we have randomly generated different sets of commodities. Henceforth we refer to the combination of a graph and a set of commodities as an *instance*. For each graph we generated 20 different instances with 10, 25, 50, and 75 commodities. This makes a sum of 80 instances for each graph, and 240 instances altogether. The name given to each instance identifies it uniquely and gives information about which is the underlying graph and how many commodities are requested to be connected, e.g., instance graph3-25p.8 is the eighth instance with graph3 as underlying topology, and for which there is a list of 25 commodities to route.

All the algorithms in this paper were implemented in the C++ programming language and compiled using GCC 2.95.2 with the -o3 option. The experiments have been run on a PC with Intel(R) Pentium(R) 4 processor at 3.06GHz and 900 Mb of memory running a Linux operating system. Moreover, they were all implemented on the same data structures. Information about the shortest paths in the respective graphs is provided to all of them as input. Notice however that, while the multi-start greedy algorithm MSGA needs to partially recompute this information after the routing of each commodity, ACO does not.

Results. First of all, the execution of the ACO approach requires the setting of parameter values (which we have already specified in Section 3). The values of parameters d_{rate} (determinism in the solution construction), β (influence of the distance to the destination), γ (influence of the edge usage) and ρ (learning rate) were set according to tuning experiments. First, we have tested different combinations of values for d_{rate} (i.e., $d_{rate} \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$), β and γ (i.e., $\beta, \gamma \in \{0.1, 1.0, 10.0\}$)). According to the results obtained in the tuning, we have set $d_{rate} = 0.8$ and $\beta = \gamma = 1.0$. Figure 2 shows as an example the tuning results for the 20 "50-commodities" instances of graph graph3.

We applied MSGA as well as our ACO approach to all 240 instances exactly once. Hereby, MSGA was applied with 10 restarts (i.e., $N_{perm} = 10$). For each of the 240 instances, the computation time of MSGA was used as a maximum CPU time limit for the ACO algorithm on that instance. We present the results as averages over the 20 instances of each combination of graph and commodity number in Table 2.

First, we observe that in 9 out of 12 cases the ACO approach beats MSGA. Furthermore, in general the ACO approach needs less computation time than MSGA. This advantage in computation time decreases with growing number of commodities. Exceptions for this evaluation are the results

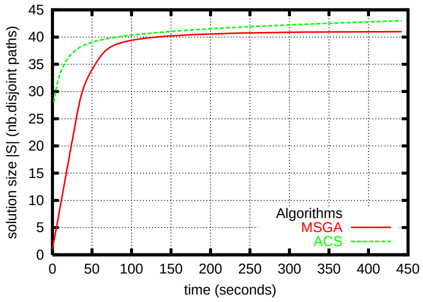


Fig. 1. A representative example of the run-time behavior of ACO in comparison to the run-time behavior of MSGA. In general, the ACO approach reaches good solutions much faster.

for the combinations of graph4 and commodity numbers 50 and 75. For this combination MSGA has advantages over the ACO approach. One possible reason for this behavior might be related to the high degree of some vertices of graph4 (see the maximum degree in Table 1). A possible conjecture is that the nodes with high degree bias the search process performed by ACO; first, because they are important connectivity points and are often in the shortest paths between the vertices of the graph and, second, because the higher the degree of a vertex, the less the probability of choosing a particular outgoing edge in the construction mechanism. In this case, for choosing a good outgoing edge the algorithm has possibly to be lucky, especially in early stages of the search process. Probably, in this situations it would be better to force a deterministic choice (see Equation 2). Apart from graph4, the average results for 50 and 75 commodities show a clear advantage of ACO in contrast to MSGA, both in quality and time. In general, the robustness of both algorithms are comparable.

A second analysis that we can apply to our algorithms concerns the development of the best solutions found over time. When observing the run-time behavior of our algorithms we notice that the ACO approach finds relatively good solutions (which might be sufficient in practice) already after a very short computation time. In general, already the first solutions produced by ACO are quite good, whereas the greedy approach reaches the same solution quality only much later in time. We show a representative example of the run-time behavior in Figure 1. This property of our ACO approach is a desirable feature in the context of communication networks.

Finally, Figure 3 shows an example for the usefulness of the two phases of our ACO approach trying to improve the first criterion, respectively the second criterion.

5 Conclusions and further work

In this paper we have proposed an Ant Colony Optimization (ACO) approach to tackle the maximum edge-disjoint path problem. To the best of our knowledge, this is the first application of this nature-inspired metaheuristic (in any of its existing versions) to this problem. Our approach is based on a decomposition of the maximum edge-disjoint path problem into subproblems. Furthermore, it includes features of other successful ACO versions. We have compared our algorithm to a multi-start greedy approach which is based on a greedy approach that was developed for approximation purposes.

First of all, the results showed that our ACO approach has in general advantages over the multi-start greedy approach in terms of solution quality as well as in terms of computation time. Furthermore, the results indicate that especially in the run-time behavior the ACO approach is superior to the multi-start greedy approach. Already in early stages of a run the ACO algorithm provides relatively good solutions, which might meet the requirements in communication networks, in which speed is an issue. In general, our ACO approach is well suited for the implementation in communication networks, as it only uses local information for building paths. This is in contrast to the greedy approach, which depends strongly on recomputations of shortest paths.

There are many possible directions for future work. From the algorithmic point of view, it would be interesting to study other possible greedy approaches including, for example, length restrictions on the routes. As observed in Kleinberg [13], length constraints can transform tractable disjoint paths problems into NP-hard variants (e.g., even the source-single sink case is NP-hard.) The relatively high degree of determinism used in our ACO approach might also suggest some new greedy approach.

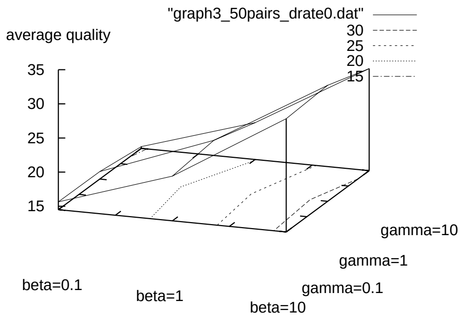
Then we plan to further analyse and improve our ACO approach. We also plan to develop different versions. One of the immediate things is to study the case in which ants construct the paths not one after the other, but in parallel. We also plan to apply our ACO approach to generalizations of the problem based on real-life features of nowadays networks (e.g., congestion, free bandwidth, adversarial traffic, etc.), which are specially interesting and challenging. This will require some changes to our approach, since ants will require a whole range of other types of locally available information. It will also be interesting to observe the behavior of our ACO approach when applied in dynamically changing networks in which the adaptation to a changing environment is required. The usage of that local information will be doubly important when working on dynamic networks in order to adapt the construction of solutions to the changing environment. However, the nature of ACO makes it a good metaheuristic candidate to be considered when dynamic environments are concerned.

References

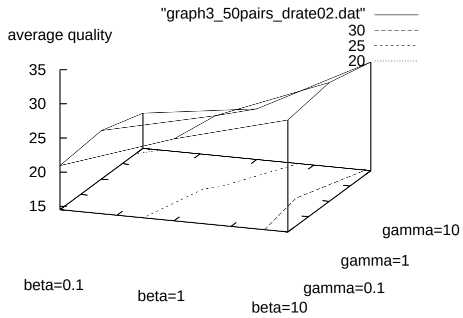
1. A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan. Efficient routing and scheduling algorithms for optical networks. In *5th. ACM-SIAM Symposium on Discrete Algorithms* (SODA'94), pages 412–423, 1994.
2. Y. Aumann and Y. Rabani. Improved bounds for all-optical routing. In *6th. ACM-SIAM Symposium on Discrete Algorithms* (SODA'95), pages 567–576, 1995.
3. B. Awebuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th Annual IEEE Symposium on Foundations of Computer Science* (FOCS'93), pages 32–40, 1993.
4. B. Awerbuch, R. Gawlick, F. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *35th. IEEE Symposium on Foundations of Computer Science* (FOCS'94), pages 412–423, 1994.
5. C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 2004. To appear.
6. G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
7. M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, DEI, Politecnico di Milano, Italy, 1992. (in Italian).
8. M. Dorigo, G. Di Caro, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.
9. M. Dorigo and L. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
10. M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26(1):29–41, 1996.
11. J. Hromkovič, R. Klasing, E. Stöhr, and H. Wagener. Gossiping in vertex-disjoing paths mode in d-dimensional grids and planar graphs. In T. Lengauer, editor, *1st Annual European Symposium on Algorithms* (ESA'93), volume 726 of *Lecture Notes in Computer Science*, pages 200–211, Bad Honnef, Germany, 1993. Springer Verlag.
12. R. Karp. *Compexity of Computer Computations*, chapter Reducibility among combinatorial problems, pages 85–103. Miller, R.E. and Thatcher, J.W. (Eds.). Plenum Press, New York, 1972.
13. J. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, MIT, Cambridge, USA, 1996.
14. A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Boston University Representative Internet Topoloy Generator. `http://cs-pub.bu.edu/brite/index.htm`, 2001.
15. M. Middendorf and F. Pfeiffer. On the complexity of the disjoint path problem. *Combinatorica*, 13:97–107, 1993.
16. P. Raghavan and E. Upfal. Efficient all-optical routing. In *26th. Annual ACM Symposium on Theory of Computing* (STOC'94), pages 134–143, 1994.

17. D. Sidhu, R. Nair, and S. Abdallah. Finding disjoint paths in networks. *ACM SIGCOMM Computer Communication Review*, 21(4):43–51, 1991.
18. J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61:83–90, 1995.
19. B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1671–1622, 1988.

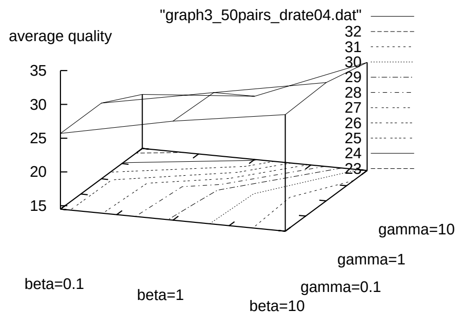
Appendix



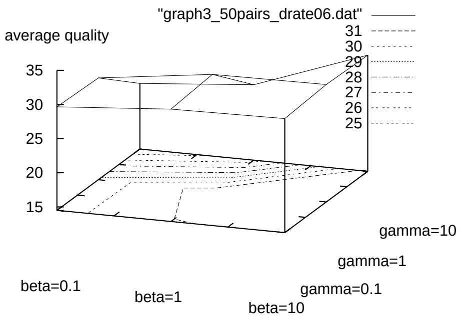
(a) graph3, 50 pairs, drate=0.0



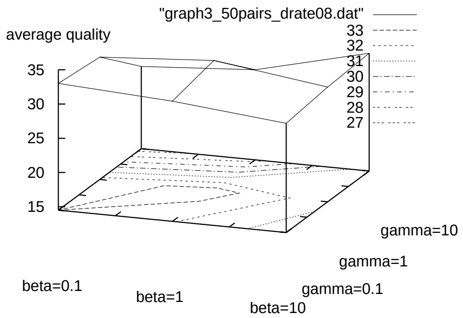
(b) graph3, 50 pairs, drate=0.2



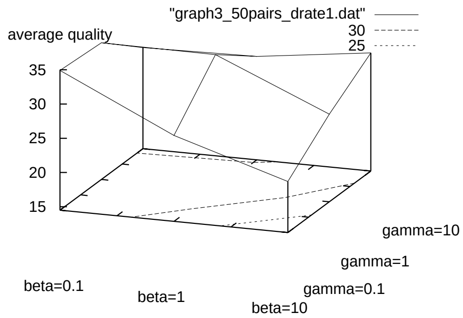
(c) graph3, 50 pairs, drate=0.4



(d) graph3, 50 pairs, drate=0.6

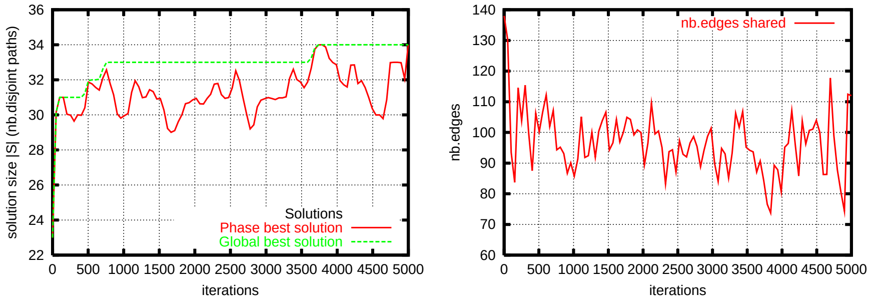


(e) graph3, 50 pairs, drate=0.8

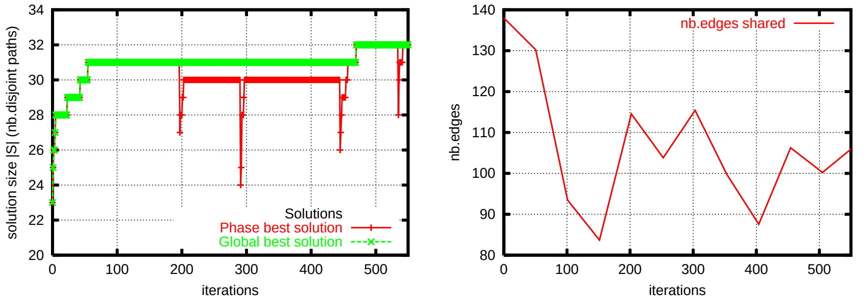


(f) graph3, 50 pairs, drate=1.0

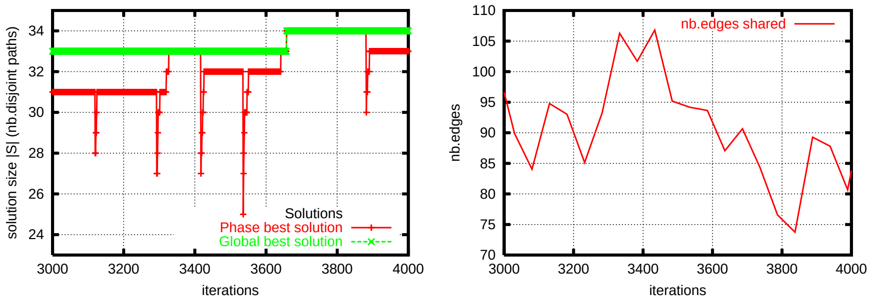
Fig. 2. Example of the tuning results. Results obtained for the tuning of the parameters d_{rate}, β and γ on the 20 instances of the combination of graph graph3 and 50 commodities. For low values of determinism, a high value of *beta* is required. This relation gets inverted when increasing the rate of determinism. Independently, a high value of γ is often good. However, the best results are obtained with medium-high values of determinism and unity values of β and γ. Basically the same tuning results are obtained for other numbers of commodities. We chose $d_{rate} = 0.8$, $\beta = \gamma = 1$.



(a) Representative example of the evolution of the quality of the solutions S_{pbest} and S_{gbest} in the search phases (**left**), and the number of shared edges (2^{nd} criterion) of the solution S_{pbest} (**right**). Note that all the curves are smoothed with gnuplots' *sbezier* function.



(b) Zoom on the first 500 iterations of Figure 3(a). In the first iterations, the best solution found is quickly improved (**left**). At about iteration 450 the algorithm destroys part of the S_{pbest} solution (with $f(S_{pbest}) = 30$, which produces an instantaneous worsening in the quality (**left**); however, this helps in achieving an improvement soon afterwards (**left**, improvement of S_{pbest} and S_{gbest} to 32).



(c) Zoom on the iterations 3500-4000 of Figure 3(a). An example of how the second criterion can help to improve can be observed between iteration 3550 and 3650 (approximately). The improvement of the second criterion (see **right**) finally leads to an improvement in the first criterion (see **left** at iteration 3650).

Fig. 3. Some details on the behavior of ACO: Search phases