A nature-inspired algorithm for the disjoint paths problem * †

Maria J. Blesa Christian Blum

ALBCOM research group, Universitat Politècnica de Catalunya
Jordi Girona 1-3 Ω building, Campus Nord, E-08034 Barcelona, Spain

E-mail: {mjblesa,cblum}@lsi.upc.edu

1. Introduction

The efficient use of modern communication networks depends on our capabilities for solving a number of demanding algorithmic problems. An example is the establishment of 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 due to 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. This scenario can be modeled as follows. 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. 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 to be connected by a path in G. 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 $j = 1, \ldots, |T|$. The question whether T is realizable was early known to be NP-complete in arbitrary graphs [14] as well as specific types of graphs [18, 21, 26, 22, 19].

The combinatorial optimization version of this 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 disjoint paths, in which each path satisfies the connection request for a different commodity. For any solution S, the objective function value $f(S)$ is defined as

$$f(S) = |S| \ . \tag{1}$$

We henceforth refer to our problem as the maximum *edge-disjoint paths* (EDP) problem.

The EDP problem is interesting for 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 large-scale, high-speed and optical networks [3, 23, 1, 2]. Concerning real-time communications, the EDP problem is very much related to survivability and information dissemination. Concerning survivability, having several disjoint paths available may avoid the negative effects 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 [24, 13].

In general, there is a lack of efficient algorithms for tackling the EDP problem. Only some greedy approaches (see Section 2) and a preliminary ant colony optimization (ACO) approach [4] exist for tackling the problem. The greedy approaches are used as approximation algorithms for theoretical purposes, but the quality of the solutions they obtain are susceptible to improvement. Based on the (basic) approach in [4], we have evolved a more sophisticated ACO algorithm.

2. A greedy approach

A greedy heuristic is a constructive algorithm that builds a solution step-by-step starting from an empty solution. At

* Partially supported by EU Programmes under contract no. IST-2004-15964 (AEOLUS), COST-295 (DYNAMO), and by the Spanish Ministry of Education and Science under CICYT projects no. TIN-2005-09198 (ASCE) and no. TIN-2005-08818-C04 (OPLINK). C. Blum also acknowledges support from the *"Juan de la Cierva"* postdoctoral program of the Spanish Ministry of Education and Science.

† The reader is addressed to [5], for more technical details on this work. The reader should take into account that the SGrA algorithm is named there as SGA, and the MSGrA algorithm is named as MSGA.

1-4244-0054-6/06/$20.00

Algorithm 1 SGrA for the EDP problem

INPUT: a problem instance $\{G, T\}$
$S \leftarrow \emptyset$, $\hat{E} \leftarrow E$
for $j = 1, \ldots, |T|$ **do**
  if s_j and t_j can be connected by a path in $G = (V, \hat{E})$ **then**
    $P_j \leftarrow$ shortest path from s_j to t_j in $G = (V, \hat{E})$
    $S \leftarrow S \cup P_j$, $\hat{E} \leftarrow \hat{E} \setminus \{e \mid e \in P_j\}$
  end if
end for
OUTPUT: the solution S

each construction step, an element from a finite set of solution components is added to the current partial solution. The element to be added is chosen at each step according to some greedy function. A characteristic feature of the greedy algorithms is that, once a decision is made on which element to add, this decision is never reconsidered again. Greedy heuristics are usually easy to implement and fast in execution. In contrast, the quality of the solutions provided is often far from being optimal.

Greedy algorithms are often used as *approximation algorithms* to solve optimization problems with a *guaranteed performance*. With this aim, some greedy algorithms were proposed for the EDP problem; examples are the *simple greedy algorithm* [15], its constrained variant the *bounded-length greedy algorithm* [15, 7, 17] and the *greedy path algorithm* [16, 8]. Due to its lower time complexity when compared to the other greedy approaches we decided to implement the simple greedy algorithm (henceforth denoted by SGrA) and a multi-start version of it.

The SGrA algorithm (see Algorithm 1) is a natural way of approximating the EDP problem. It starts with an empty solution S and then it proceeds through the commodities in the order that is given as input. For routing each commodity $T_j \in T$, it considers the graph G without the edges that are already in the paths of the solution S under construction. The shortest path (w.r.t. the number of edges) between s_j and t_j is assigned as path for the commodity $T_j = (s_j, t_j)$. Observe that the SGrA algorithm is deterministic and that the quality of the solutions it provides depends heavily on the order in which the commodities are treated. A simple way of overcoming that dependence on the order is to develop a multi-start version of the SGrA by permuting the order of the commodities for each restart. This approach is henceforth called *multi-start greedy algorithm* (MSGrA). The output of MSGrA is the best solution found in among all restarts.

Due to the deterministic decisions that greedy algorithms take during the solution construction, it is sometimes not possible for them to find an existing optimal solution. This is also the case for the SGrA and MSGrA greedy algorithms presented here. Consider for example the instance of the EDP problem depicted in Figure 1, which consists in the depicted graph and the set

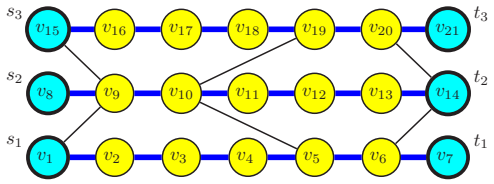


Figure 1. Instance of the EDP problem with $T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$. Neither SGrA nor MSGrA can find the optimal solution depicted in bold font.

$T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$ of three commodities to join. The optimal solution in which all three commodities are connected is also shown in bold font in Figure 1. This solution is found by our ACO algorithm, which is presented next, in a small amount of time (less than 30 ms.). In contrast, it is impossible for SGrA and MSGrA to find the optimal solution. The is because they are based on shortest paths, and therefore they will connect the commodities through non-consecutively-numbered vertices. For example, when trying to connect first the commodity (v_1, v_7), the SGrA algorithm will establish the path $\{v_1, v_9, v_{10}, v_5, v_6, v_7\}$. This excludes edge $\{v_9, v_{10}\}$ as a possibility for being used in other paths, which makes it impossible to build disjoint paths simultaneously for the remaining two commodities, independently of which one is built next. Analogous situations occur when starting from any of the other two commodities.

3. An ant colony optimization approach

Ant colony optimization (ACO) [10, 12] is a nature-inspired metaheuristic for solving hard combinatorial optimization problems. Apart from the application to static combinatorial optimization problems the method has also gained recognition for routing applications in communication networks [9]. ACO algorithms are composed by independently operating computational units, namely *artificial ants*, that generate a global perspective without the necessity of direct interaction. This exclusive use of local information is an advantageous and desirable feature when applications in large-scale environments are concerned in which the computation of global information is often too costly. This property makes ACO algorithms a natural choice for the application to the EDP problem.

ACO is inspired by the foraging behavior of real ants. While walking from food sources to the nest and vice versa, ants deposit a chemical 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 between food sources and their nest. In ACO algorithms, artificial ants incrementally construct a solution by adding appropriately defined solution components to the current partial solution. Each of the construction steps is a probabilistic decision based on local information, which is represented by the *pheromone* information.

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 P_j between the two endpoints of her commodity T_j. Obviously, the subproblems are not independent as the set of $|T|$ paths constructed by the ants should be mutually edge-disjoint.

3.1. Ant solutions and pheromone model

Our algorithm will deal with solutions that contain a path for each commodity. A solution S constructed by the $|T|$ ants is a set of non-necessarily edge-disjoint paths. We henceforth refer to them as *ant solutions*, in contrast to the EDP solutions, which only consist of disjoint paths. From each ant solution a valid EDP solution can be produced by iteratively removing the path which has most edges in common with the remaining paths, until all remaining paths are mutually edge-disjoint.

The objective function $f(\cdot)$ of the problem (see Equation 1) is characterized by having many plateaus when applied to ant solutions, because many ant solutions have the same number of disjoint paths. Thus, we define a more fine-grained 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 quantifies the degree of non-disjointness of an ant solution and is defined as follows:

$$C(S) = \sum_{e \in E} \left(\max \left\{ 0, \left(\sum_{P_j \in S} \delta^j(S, e) \right) - 1 \right\} \right), \text{ where}$$

$\delta^j(S, e) = 1$ when $e \in P_j \in S$, and $\delta^j(S, e) = 0$, otherwise. If all the paths in a solution S are edge-disjoint, $C(S)$ is zero. In general, $C(S)$ increases whith increasing numbers of edges in S that are used in more than one path. Therefore, based on the idea that *the fewer 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', $f^a(S) > f^a(S')$ if, and only if

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

Algorithm 2 ACO algorithm for the EDP problem

```
INPUT: a problem instance (G, T)
S_gb ← S_pb ← ∅, κ1 ← κ2 ← 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 ← ConstructSolution(G,π)
    if i < N_sols then
      π ← GenerateRandomPermutation(|T|)
    end if
  end for
  Choose S_ib ∈ {S_i | i = 1, ..., N_sols} s.t.
  f^a(S_ib) ≥ f^a(S), ∀S ∈ {S_i | i = 1, ..., N_sols}
  if f(S_ib) > f(S_gb) then S_gb ← S_ib end if
  if f^a(S_ib) > f^a(S_pb) then
    κ1 ← κ1 + 1, κ2 ← 0, S_psave ← S_pb, S_pb ← S_ib
    if f(S_ib) > f(S_psave) then
      S_update ← ExtractDisjointPaths(S_pb)
      κ1 ← 0, all_update ← FALSE
    end if
    if all_update then S_update ← S_pb end if
  else κ2 ← κ2 + 1
  end if
  if all_update and (κ2 > max κ2) then
    S_pb ← DestroyPartially(S_pb)
    S_update ← ExtractDisjointPaths(S_pb)
    κ2 ← 0, κ1 ← 0
  else if not all_update then all_update ← (κ1 > max κ1)
  end if
  UpdatePheromoneValues(τ,S_update)
end while
OUTPUT: the EDP solution generated from S_gb
```

The problem decomposition as described above requires 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 in $[\tau_{\min}, \tau_{\max}]$, where $\tau_{\min} = 0.001$ and $\tau_{\max} = 0.999$, since our ACO algorithm is implemented in the hyper-cube framework [6] with ideas borrowed from the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant Systems [25].

3.2. Algorithmic framework

Algorithm 2 is a high-level description of our ACO algorithm for the EDP problem. First, all the variables are initialized. In particular, the pheromone values are initialized by the procedure InitializePheromoneValues(τ), which sets all the pheromone values $\tau^j_e \in \tau^j \in \tau$ to the value

$\tau_{\min}$. Second, N_{sols} ant solutions are constructed per iteration. To construct a solution, each ant applies the function ConstructSolution(G,π) (see Subsection 3.2.1), where π is a permutation of T. At each iteration, the first of those N_{sols} ant solutions is constructed with the identity permutation, i.e., by sending the ants in the order in which the commodities are given in T. However, for each further ant solution construction in the same iteration, π is randomly generated by the function GenerateRandomPermutation($|T|$) in order to avoid bias.

Three different ant solutions are kept in the algorithm: S_{ib} is the *iteration-best* solution, i.e., the best ant solution generated in the current iteration, and S_{gb} is the *best-so-far* solution, i.e., the best ant solution found since the start of the algorithm. In addition to them, an ant solution S_{pb} is also kept, which is the *currently best* solution, i.e., the best ant solution generated since the last escape action (see Subsection 3.2.3). The values of these three variables are always kept updated. Additionally, the S_{update} solution is generated from S_{pb} and used for updating the pheromone values.

The search process has two differentiated phases (see Subsection 3.2.2) controlled by two variables, κ_1 and κ_2. The variable κ_1 controles the first phase by counting the number of successive iterations without improvement of the first criterion of the objective function. The variable κ_2 counts the number of successive iterations without improvement of the second criterion, thus controling the second phase. Limits $\max \kappa_1$ (for κ_1) and $\max \kappa_2$ (for κ_2) are used to determine when the algorithm should change phases.[1] The direct repercussion of the phase distinction is the selection of edges for the pheromone update, i.e., the construction of S_{update} from S_{pb}. When the algorithm is in the first phase only the disjoint paths of solution S_{pb} are used for updating, but when the algorithm is in the second phase all paths of S_{pb} are used for updating. Additionally, the escape mechanism might be applied by destroying S_{pb} partially (see Subsection 3.2.3).

Finally, the pheromone values are updated in the method UpdatePheromoneValues(τ,S_{update}) depending on the edges of the paths included in S_{update}. The algorithm is iterated until some opportunely defined termination conditions are satisfied, and it returns the EDP solution generated from the ant solution S_{gb}. In the following, we explain in more detail the features concerning the solution construction, the search procedure and its different search phases, and the escape mechanism of our algorithm.

3.2.1. Solution construction. The solution construction is performed in method ConstructSolution(G,π), whose high-level description is shown in Algorithm 3. At each construction step, each ant moves from the node where it is currently located to a neighboring node by traversing one

1 After parameter tuning we set $\max \kappa_1 = \max \kappa_2 = 20$.

Algorithm 3 Method ConstructSolution(G,π)

INPUT: a graph G from a problem instance (G, T), and a permutation π of T.
$S \leftarrow \emptyset$, $\kappa_{completed} \leftarrow 0$, $j \leftarrow 0$
for $i = 1$ to $|T|$ **do** $P_{\pi(i)} \leftarrow \emptyset$ **end for**
repeat
  if not isFinishedPath($P_{\pi(j+1)}$) **then**
    $P_{\pi(j+1)} \leftarrow$ ExtendOneStepPath($P_{\pi(j+1)}$,$\tau^{\pi(j+1)}$)
    if isFinishedPath($P_{\pi(j+1)}$) **then**
      $\kappa_{completed} \leftarrow \kappa_{completed} + 1$
      $S \leftarrow S \cup \{P_{\pi(j+1)}\}$
    end if
  end if
  $j \leftarrow (j + 1) \bmod |T|$
until ($\kappa_{completed} = |T|$)
EvaporatePheromone(τ, S)
OUTPUT: an ant solution S

of the available edges that is not already in its path $P_{\pi(j)}$ under construction, and that is not labelled forbidden by a backtracking move.[2] Otherwise, the ant returns an empty path. This way of constructing the solution emulates that the ants build concurrently their paths. The procedures of Algorithm 3 are:[3]

– isFinishedPath(P_i) returns a boolean value indicating whether a path P_i from s_i to t_i is finished.

– ExtendOneStepPath(P_i,τ^i) . For constructing a path between the endpoints of the commodity (s_i, t_i), an ant first chooses randomly from which endpoint to start; this is done when the path P_i is empty. The endpoint other than the one chosen for starting becomes the so-called *goal node* and will be denoted by v_g. Afterwards, this method either tries to extend the path P_i under construction, by adding exactly one edge or, it performs a backtracking step. Backtracking is done in case the ant finds itself in a node whose incident edges have all been used already, or if all the incident edges are labelled forbidden.

Let us denote by v_c the current node, and by $\mathcal{I}^\star_{v_c}$ the set of edges in G incident to v_c which are not used yet in P_i and not labelled as forbidden. The length of the shortest path between two vertices u and v in G is henceforth denoted by $\sigma(u, v)$ and it is measured in terms of the number of edges. From the set $\mathcal{I}^\star_{v_c}$ of allowed edges, only the two best edges will actually be considered as candidates. This is called a *candidate list strategy* in the context of ACO. The best two edges are those that maximize the value

$$\tau_e^j \cdot \mathbf{p}(D_e) \cdot \mathbf{p}(U_e), \quad \text{where}$$

2 Note, that with this strategy the ant will find a path between its source and its destination, if there exists one.

3 For the sake of readability, we substitute $\pi(j + 1)$ in the description of the functions by i.

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

$$\mathbf{p}(U_e) \leftarrow \frac{U(e)^{-1}}{\sum\limits_{e'\in\mathcal{I}^\star_{v_c}} U(e')^{-1}}$$

being $U(e) = 2$ when e is already used in S_i, and $U(e) = 1$, otherwise. $\mathbf{p}(D_e)$ determines the influence of the distance from v_c via u to the goal vertex v_g, and $\mathbf{p}(U_e)$ determines the influence of the overall usage of edge e in the paths of another ants for the same solution. Thus, using this candidate list strategy, we can just consider the set $\mathbb{I}^\star_{v_c} = \{e^*_1, e^*_2\}$, where e^*_1 is the best edge in $\mathcal{I}^\star_{v_c}$, i.e.,

$$e^*_1 = \{v_c, u\} \leftarrow \mathsf{argmax}\,\{\tau^j_e \cdot \mathbf{p}(D_e) \cdot \mathbf{p}(U_e) \mid e \in \mathcal{I}^\star_{v_c}\},$$

and e^*_2 is the second best edge in $\mathcal{I}^\star_{v_c}$.

At each construction step, the choice of where to move to has a certain probability p to be done deterministically, and probability $1-p$ to be chosen probabilistically among the elements in $\mathbb{I}^\star_{v_c}$. This is a feature that we adopt from a particularly effective ACO variant called Ant Colony System (ACS [11]). In 75% of the cases, the next edge to join the path $P_{\pi(k)}$ under construction will be e^*_1, while in the remaining 25% of the cases, the next edge is chosen from $\mathbb{I}^\star_{v_c}$ according to the following transition probabilities:

$$\mathbf{p}(e \mid \mathbb{I}^\star_{v_c}) = \frac{\tau^j_e \cdot \mathbf{p}(D_e) \cdot \mathbf{p}(U_e)}{\sum\limits_{e'\in\mathbb{I}^\star_{v_c}} \tau^j_{e'} \cdot \mathbf{p}(D_{e'}) \cdot \mathbf{p}(U_{e'})} \quad ,\forall\, e \in \mathbb{I}^\star_{v_c}$$

The use of the pheromone information τ^j_e ensures the flexibility of the algorithm, whereas the use of $\mathbf{p}(D_e)$ ensures a bias towards short paths, and $\mathbf{p}(U_e)$ ensures a bias towards disjointness of the $|T|$ paths constituting a solution.

– EvaporatePheromone(τ, S). Once the solution S is completed, some amount of pheromone from the edges that were used by the ants is evaporate. The reason for this pheromone evaporation is the desire to diversify the search in each iteration.[4] Given a solution S, the evaporation is done as follows:

$$\tau^j_e \leftarrow \begin{cases} (1-\varepsilon)\cdot\tau^j_e & : \quad e \in P_{\pi(j)} \in S,\ \ j \in [1,|T|] \\ \tau^j_e & : \quad \text{otherwise} \ . \end{cases}$$

3.2.2. Search with distinguished phases. The pheromone update procedure is an important component of every ACO algorithm and determines to a large degree the failure or the success of the algorithm. We propose a pheromone updating scheme that is based on the idea that, in order to maintain a higher degree of freedom for finding also edge-disjoint paths for the commodities that initially prove to be problematic, it might be better not to use the non-disjoint paths for updating the pheromone at the beginning of the search. Therefore, we propose a two-phases search process based on the two criteria of function $f^a(\cdot)$: in the 1st phase, the algorithm will try to improve the first criterion of $f^a(\cdot)$ (while disregarding the second one) and only disjoint paths are used for updating the pheromone values; the 1st phase is followed by a 2nd phase which is initiated when no improvements of the first criterion can be found over a certain number of iterations bounded by $\max\kappa_1$. The algorithm will try to improve the second criterion of $f^a(\cdot)$ in this 2nd phase, and all the paths are used for updating the pheromone values. Once the second phase leads to an improvement also in terms of the first criterion, the algorithm changes back to the first phase.

In the first phase, the solution S_{update} that is used for updating the pheromone values is obtained by applying function ExtractDisjointPaths(S_{pb}), which implements the process of returning a valid EDP solution from the ant solution S_{pb} as explained in Section 3.1. In the second phase, the solution S_{update} that is used for updating the pheromone values is a copy of the ant solution S_{pb}, including possibly non-disjoint paths. If for a number of $\max\kappa_2$ iterations the second criterion could not be improved neither, some of the paths from the EDP solution that can be produced from S_{pb} are deleted. This action can be seen as a mechanism to escape from the current area of the search space (see Subsection 3.2.3).

After determining solution S_{update}, the pheromone of the edges conforming its paths are updated in function UpdatePheromoneValues(τ,S_{update}) as follows:

$$\tau^j_e \leftarrow \max\left\{\tau^j_e + \rho\cdot\left(1-\tau^j_e\right),\ \tau_{\max}\right\}$$

for all the edges $e \in P_j \in S_{update}$, where $\rho \in (0,1]$ is a constant value which is called *learning rate*. We set ρ to 0.1.

3.2.3. Escape mechanism. One of the main problems of metaheuristic search procedures is to detect situations in which the search process gets stuck, i.e., when some local minimum is reached. Once detected, an algorithm might escape from such a situation by means of a so-called espace mechanism. For our algorithm, we propose as escape mechanism the partial destruction of the disjoint part of the solution which is used for updating the pheromone values. This escape mechanism is implemented through the function DestroyPartially(S_{pb}) of Algorithm 2. It works by deleting 25% of the longest paths in the EDP solution extracted from S_{pb}. The idea behind the destruction of the longest paths is that these paths probably have more conflicts with other paths. Thus, by removing the longest paths, the number of total edges available is maximized. The mechanism is triggered once the algorithm cannot improve the currently best solution in the second search phase.

4 After parameter tuning we chose a setting of $\varepsilon = 0.10$.

4. Experiments

We present the experimental evaluation of our ACO approach in comparison to the greedy approaches outlined in Section 2.[5] All the algorithms were implemented in C++ and compiled using GCC 2.95.2 (option -o3). The experiments have been run on a Linux PC with Intel(R) Pentium(R) 4 processor at 3.06GHz and 900 Mb of memory. Information about the shortest paths in the respective initial graphs is provided to all algorithms as input.

Problem instances. The set of benchmark instances used to experimentally evaluate our ACO approach includes graphs representing different communication network topologies. For example, the structures of graph3 and graph4 (from [4]) resemble parts of the communication network of the Deutsche Telekom AG, Germany. Furthermore, graph0, graph1, and graph2 were created with BRITE [20]. Their topology defines a 2-level top-down hierarchy (autonomous system plus router level), which is typical in Internet topologies (see Table 1 for details).

For each of the five graphs we have randomly generated different sets of commodities. Hereby, we made the size of the commodity sets dependent on the number of vertices of the graph. For each graph $G = (V, E)$ we generated 60 instances: 20 different instances with $0.10|V|$, $0.25|V|$ and $0.40|V|$ commodities. This makes 300 instances altogether.

Results and conclusions. We applied the SGrA, the MSGrA, and the ACO algorithm to all 300 instances exactly once. First, we applied MSGrA with 50 restarts to each of the 300 instances. The computation time of MSGrA was used as a maximum CPU time limit for the ACO algorithm. We present the results as averages over the 20 instances of each combination of graph and commodity number in Table 2.

Concerning the comparison between SGrA and MSGrA, we observe a clear advantage of MSGrA. This means that the order in which the commodities are treated is crucial for SGrA. However, as there is no obvious way of determining a good commodity order beforehand, the only way of exploiting this knowledge is by randomly permuting the commodity list and running MSGrA. The price we have to pay for exploiting this knowledge is the increased computation time.

When comparing the SGrA and the MSGrA with ACO, we can observe that in 11 out of 15 cases the ACO approach beats the greedy approaches. The ACO approach is on average 4.69% better than MSGrA, and in one case (graph4, 173 commodities) it is even 15.07% better. Additionally, the ACO approach needs in general less computation time

5 For a detailed description of the parameter settings see [5].

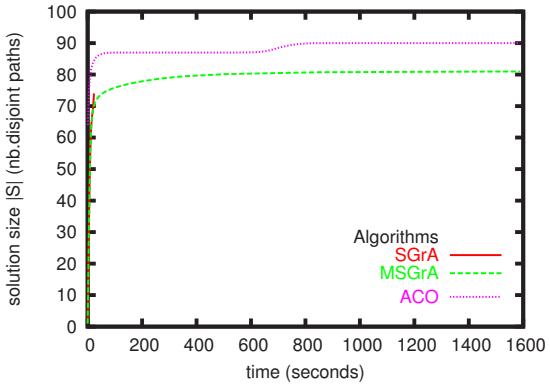


Figure 2. Representative example of the run-time behavior of our algorithms. The graphic shows the evolution in time of the quality of solution S_{gb} (graph4, 173 commodities).

than the greedy approaches. This advantage in computation time increases with increasing number of commodities. Exceptions are some of the results for small numbers of commodities. For this combination MSGrA has often slight advantages over the ACO approach. Therefore, we recommend to use a greedy approach when easy problem instances are concerned, but to use the ACO approach for instances with a higher number of commodities, since then a clear advantage of the latter is observed in comparison to MSGrA both in quality and time.

An additional analysis concerns the run-time behaviour of the algorithms (see Figure 2). The ACO approach finds relatively good solutions already after a very short computation time. In general, already the first solutions produced by the ACO are quite good, whereas the greedy approaches reach a comparable solution quality only much later in time. This property of our ACO approach is a desirable feature in the context of communication networks since the quality of the solutions that are found after a short execution time might be often sufficient in practice. Also of interest is to observe the usefulness of ACO's escape mechanism and how the second criterion evolves as a measure for disjointness (see Figure 3).

Acknowledgements. The authors would like to thank Martin Oellrich for providing graph3 and graph4, and the anonymous referees for their comments and suggestions.

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 Sympo-*

Graph	$\|V\|$	$\|E\|$	Degree			Diameter	Clustering coefficient	Resemblance
			min.	avg.	max.			
graph0	500	1020	2	4.08	13	23	0.102385	Internet-like topology
graph1	100	190	2	3.80	7	11	0.378524	Internet-like topology
graph2	100	217	2	4.34	8	13	0.411119	Internet-like topology
graph3	164	370	1	4.51	13	16	0.226161	Partial D-Telekom-like topology [4]
graph4	434	981	1	4.52	20	22	0.155547	Partial D-Telekom-like topology [4]

Table 1. Main quantitative measures of our benchmark graphs. For readability reasons, we have changed the name of these graphs w.r.t. the longer technical version [5] of this work, where graph0**,** graph1 **and** graph2 **were named** bl-wr2-wht2.10-50**,** AS-BA.R-Wax.v100e190 **and** AS-BA.R-Wax.v100e217**, respectively.**

Graph	number of commodities	SGrA $\overline{q}$	SGrA σ	SGrA $\overline{t}$	MSGrA $\overline{q}$	MSGrA σ	MSGrA $\overline{t}$	ACO $\overline{q}$	ACO σ	ACO $\overline{t}$	avg. CPU time (seconds)
graph0	50	19.70	2.238	17.926	22.55	2.397	318.518	**24.10**	1.947	155.899	971.488
graph0	125	34.15	4.464	46.387	38.10	4.369	1004.462	**42.30**	4.540	344.092	2425.090
graph0	200	46.70	4.961	62.158	50.85	4.892	1151.197	**56.30**	5.245	847.415	3124.550
graph1	10	8.75	0.942	0.114	**9.10**	0.943	0.579	8.95	0.973	0.611	6.665
graph1	25	12.30	1.900	0.280	14.25	1.374	4.809	**14.85**	1.195	3.718	16.740
graph1	40	15.45	2.500	0.443	17.95	1.624	7.796	**19.45**	1.936	4.121	26.850
graph2	10	7.00	1.225	0.103	**8.05**	0.921	0.427	7.88	0.927	0.164	6.892
graph2	25	11.40	1.882	0.300	13.60	1.463	4.330	**13.83**	1.579	1.816	17.622
graph2	40	14.60	1.685	0.497	17.00	1.949	9.833	**17.80**	1.646	2.212	28.318
graph3	16	15.30	0.781	0.566	15.70	0.557	0.960	**15.70**	0.557	0.457	30.582
graph3	41	29.00	2.864	1.298	**32.00**	2.302	25.235	31.80	1.990	27.953	79.619
graph3	65	33.70	2.777	2.156	37.60	2.577	49.267	**40.30**	2.571	57.899	126.945
graph4	43	40.50	1.628	12.121	**42.05**	1.024	95.744	41.45	1.284	168.871	237.520
graph4	108	58.10	4.194	31.138	64.10	3.064	697.456	**68.15**	2.725	730.436	1656.475
graph4	173	66.75	4.846	49.281	73.95	3.542	974.350	**85.10**	3.534	1111.982	2603.872

Table 2. Comparison of the results obtained by the SGrA, the MSGrA, and the ACO algorithm. The 1st and 2nd column provide the graph and the number of the commodities, respectively. For each algorithm, the average results obtained for the 20 instances of each combination of graph topology and number of commodities are reported in 3 columns. The one headed by $\overline{q}$ shows the average of the values of the best solutions; the 2nd column shows the standard deviation of the 20 values used to compute $\overline{q}$, and the 3rd one reports the average time needed to find the best solution values. The last column shows the average computation time of MSGrA. Best results in the comparisons are in boldface and, in case of ties, the computation time decides.

sium on Discrete Algorithms, pages 412–423. Society for Industrial and Applied Mathematics, 1994.

[2] Y. Aumann and Y. Rabani. Improved bounds for all-optical routing. In *6th. ACM-SIAM Symposium on Discrete Algorithms*, pages 567–576. Society for Industrial and Applied Mathematics, 1995.

[3] B. Awerbuch, R. Gawlick, F. Leighton, and Y. Rabani. Online admission control and circuit routing for high performance computing and communication. In *35th. IEEE Symposium on Foundations of Computer Science*, pages 412–423. IEEE Computer Society Press, 1994.

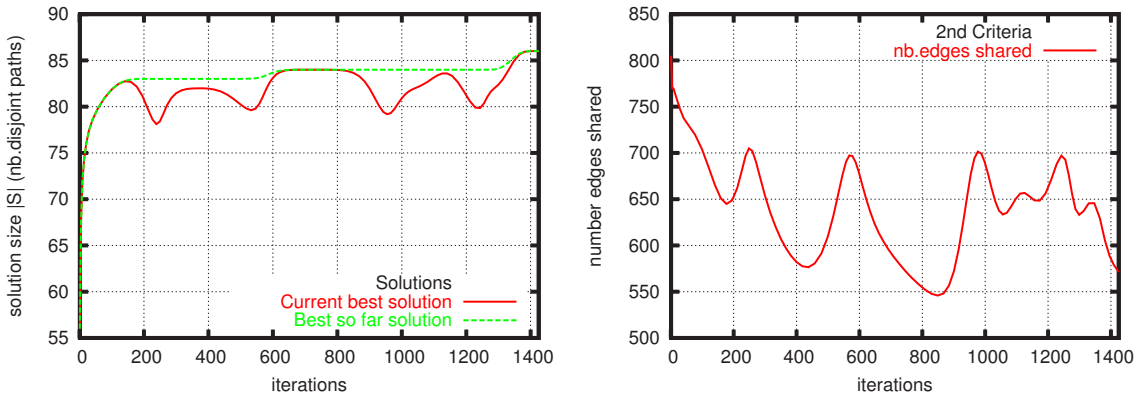
[4] M. Blesa and C. Blum. Ant colony optimization for the maximum edge-disjoint paths problem. In *1st European Workshop on Evolutionary Computation in Communications, Networks, and Connected Systems*, volume 3005 of *LNCS*, pages 160–169. Springer-Verlag, 2004.

[5] M. Blesa and C. Blum. Finding edge-disjoint paths with artificial ant colonies. Technical Report LSI-05-13-R, ALBCOM research group, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2005.

[6] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics–B*, 34:1161–1172, 2004.

[7] P. Carmi, T. Erlebach, and Y. Okamoto. *Graph-Theoretic Concepts in Computer Science*, volume 2880 of *LNCS*, chapter Greedy Edge-Disjoint Paths in Complete Graphs, pages 143–155. Springer-Verlag, 2003.

[8] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *14th annual ACM-SIAM Symposium on Discrete algorithms*,



(a) Example of the evolution of the quality of the current best solution S_{pb} and the best-so-far solution S_{gb} during the search (**left**), and the number of shared edges (2nd criterion) of the solution S_{pb} (**right**). The behavior shown here corresponds to the application to one of the 20 instances composed by graph4 and a list of 173 commodities. All the curves are smoothed with gnuplots' *sbezier* function.

Figure 3. A representative example of the behavior of the ACO algorithm. The effect of the mechanism for the partial destruction of the current best solution can be clearly observed. It is also interesting to observe the evolution of the second criterion as a measure for disjointness.

pages 628–637. Society for Industrial and Applied Mathematics, 2003.

[9] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[10] M. Dorigo. *Ottimizzazione, Apprendimento Automatico, ed Algoritmi basati su Metafora Naturale*. PhD thesis, DEI, Politecnico di Milano, Milan, Italy, 1992.

[11] M. Dorigo and L. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions Evolutionary Computation*, 1:53–66, 1997.

[12] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics–B*, 26:29–41, 1996.

[13] J. Hromkovič, R. Klasing, E. Stöhr, and H. Wagener. Gossiping in vertex-disjoing paths mode in d-dimensional grids and planar graphs. In *1st Annual European Symposium on Algorithms*, volume 726 of *LNCS*, pages 200–211. Springer-Verlag, 1993.

[14] R. Karp. *Compexity of Computer Computations*, chapter Reducibility among combinatorial problems, pages 85–103. Plenum Press, 1972.

[15] J. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, MIT, Cambridge, USA, 1996.

[16] S. Kolliopoulos and C. Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99:63–87, 2004.

[17] P. Kolman. A note on the greedy algorithm for the unsplittable flow problem. *Information Processing Letters*, 88:101–105, 2003.

[18] M. Kramer and J. van Leeuwen. *Advances in computing research*, volume 2: VLSI theory, chapter The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits, pages 129–146. JAI Press, 1984.

[19] D. Marx. Eulerian disjoint paths problem in grid graphs is NP-complete. *Discrete Applied Mathematics*, 143:336–341, 2004.

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Boston University Representative Internet Topoloy Generator, 2001.

[21] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint path problem. *Combinatorica*, 13:97–107, 1993.

[22] T. Nishizeki, J. Vygen, and X. Zhou. The edge-disjoint paths problem is np-complete for series-parallel graphs. *Discrete Applied Mathematics*, 115:177–186, 2001.

[23] P. Raghavan and E. Upfal. Efficient all-optical routing. In *26th. Annual ACM Symposium on Theory of Computing*, pages 134–143. ACM Press, 1994.

[24] D. Sidhu, R. Nair, and S. Abdallah. Finding disjoint paths in networks. *ACM SIGCOMM Computer Communication Review*, 21:43–51, 1991.

[25] T. Stützle and H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16:889–914, 2000.

[26] J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61:83–90, 1995.