J Math Model Algor (2007) 6:361–391
DOI 10.1007/s10852-007-9060-y

Finding Edge-disjoint Paths in Networks: An Ant Colony Optimization Algorithm

Maria J. Blesa · Christian Blum

Received: 1 October 2005 / Accepted: 1 December 2006 /
Published online: 16 March 2007

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 in particular 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. This problem can be modeled as a combinatorial optimization problem as follows. Given a graph G representing a network topology, and a collection $T = \{(s_1, t_1) \dots (s_k, t_k)\}$ of pairs of vertices in G representing connection request, the maximum *edge-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 edge-disjoint $s_i - t_i$ paths. We propose an *ant colony optimization* (ACO) algorithm to solve this problem. ACO algorithms are approximate algorithms that are inspired by the foraging behavior of real ants. The decentralized nature of these algorithms makes them suitable for the application to problems arising in large-scale environments. First, we propose a basic version of our algorithm in order to outline its main features. In a subsequent step we propose

Work partially supported by the FET Integrated Project 15964 (AEOLUS), and by the Spanish CICYT projects TIN2005-09198-C02-02 (ASCE), TIN2005-08818-C04-02 (OPLINK) and TIN2005-25859-E. C. Blum also acknowledges support by the RAMÓN Y CAJAL postdoctoral program of the Spanish Ministry of Science and Technology. Preliminary versions of this work were presented at the 1st European Workshop on Evolutionary Computation in Communications, Networks, and Connected Systems, LNCS 3005:160–169, Springer 2004, and in the 9th Intl. Workshop on Nature Inspired Distributed Computing, p. 239, IEEE 2006.

M. J. Blesa · C. Blum (✉)
ALBCOM research group, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Jordi Girona 1–3, Ω building Campus Nord, E-08034 Barcelona, Spain
e-mail: mjblesa@lsi.upc.edu, cblum@lsi.upc.edu

several extensions of the basic algorithm and we conduct an extensive parameter tuning in order to show the usefulness of those extensions. In comparison to a multi-start greedy approach, our algorithm generates in general solutions of higher quality in a shorter amount of time. In particular the run-time behaviour of our algorithm is one of its important advantages.

Keywords Ant colony optimization · Maximum edge-disjoint paths problem

Mathematics Subject Classifications (2000) 90-08 · 68Wxx · 68T20

Abbreviations

EDP	(maximum) edge-disjoint paths problem
SGA	Simple Greedy Algorithm
MSGA	Multi-start Greedy Algorithm
ACS	Ant Colony System
ACO	Ant Colony Optimization

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 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. The weight $w(e) \in R^+$ of an edge $e \in E$ 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 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 $j = 1, \ldots, |T|$. Deciding whether a given set of pairs is realizable in a given graph is one of Karp's original NP-complete problems [25] (other references on the computational complexity of the problem are [34, 43]). The problem remains NP-complete for various graph types such as, for example, two-dimensional meshes.

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. An EDP 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. The objective function value $f(S)$ of a solution S is defined as

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

In general, the "disjointness" of paths may refer to nodes or to edges. We decided to consider the latter case, because it seems of higher importance in practical applications. We henceforth refer to our problem 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, in which demands, profits, and capacities are considered to be one.

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 networks, namely large-scale, high-speed and optical networks [1, 3, 4, 37]. 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, 39]. 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 tackling the EDP problem. Except for greedy approaches (which we will mention in Section 3), our preliminary ant colony optimization (ACO) approach presented in [7] is the only existing method.[1] ACO [16, 18] is a recent metaheuristic for solving hard combinatorial optimization problems. Except for the application to combinatorial optimization problems (see [19] for an extensive overview) the method has also gained recognition for the applications to adaptive routing in static and dynamic communication networks [14, 15]. ACO algorithms are composed by independently operating computational units that generate a global perspective without the necessity of direct interaction. They provide several advantageous features – such as, for example, the usage of only local information – that are useful when applications in large-scale environments are concerned in which the computation of global information is often too costly.

Organization The paper is organized as follows. In Section 2 we deal with the complexity of the EDP problem, including an overview on approximability results. Existing results show that the EDP problem is not only NP-complete, but it is also hard to obtain good approximation schemes for it. In Section 3, we outline a (multi-start) greedy approach for the purpose of benchmarking our ACO algorithm. Section 4 is devoted to the detailed introduction of our ACO approach. The algorithm is developed incrementally; starting from a basic ACO approach, we introduce features that, as we show in the experimental part, help greatly on solving the EDP problem. The experimental evaluation of our approach is presented in Section 5. Besides the creation of a benchmark set of instances, we conducted an extensive tuning of the

[1]In [36], a multi-colony ACO approach for the EDP problem was presented. However, the aim of this paper was not to solve the EDP problem but rather to explore the feasibility of multi-colony systems. Therefore, only toy examples of graphs of up to 20 nodes with two or three commodities were considered.

considerable number of algorithm parameters. The details on the parameter tuning process are reported on in [8]. In Section 6, we conclude and point out possible lines for future research.

2 Understanding the Complexity of the Problem

The decisional version of the EDP problem was early known to be NP-complete [25] in arbitrary graphs. The problem remains NP-complete for specific types of graphs such as planar graphs [34, 43], series-parallel graphs (a.k.a. partial 2-trees) [35], and grid graphs [30, 32]. A more detailed classification of the complexity of the optimization version of the problem can be obtained with the help of *approximation algorithms*. Approximation algorithms tackle optimization problems in polynomial time (w.r.t. the length of the input) and output a solution that is guaranteed to be at some bounded quality difference to the optimal solution. “Close” has some well-defined sense called the *approximation ratio* (or *performance guarantee*).

Definition 1 (ρ-approximation algorithm for a problem P) Given an approximation ratio $\rho > 1$ and any problem instance x of P, a *ρ-approximation algorithm* for an optimization problem P outputs, in polynomial time, a solution to x of cost at most ρ times the optimum.

NP-hard problems vary greatly in their approximability; some can be approximated to arbitrary factors while some can essentially not be approximated at all. The problems which are approximable within a $\rho > 1$ belong to the complexity class APX. The class PTAS is a particular subclass of APX [2] which includes problems that admit a polynomial-time approximation scheme.

Definition 2 (Poly-time approximation scheme for a problem P) Given any fixed $1 > \epsilon > 0$, a *polynomial-time approximation scheme* (PTAS) for an optimization problem P is a $(1 + \epsilon)$-approximation algorithm for P.

These algorithms are desirable because one can get arbitrarily close to an optimal solution. A problem is said to be APX-hard (w.r.t. the PTAS-reducibility) if there exists some constant $\epsilon > 0$ such that it is NP-hard to obtain a $(1 + \epsilon)$-approximation algorithm, i.e., it is NP-hard to obtain a PTAS. This means that those problems are even hard to approximate, since no PTAS can be obtained efficiently unless P = NP [2]. For several types of graphs, the EDP problem belongs to the class of APX-hard problems [20, 22, 23, 31]. This fact explains the notorious hardness of the EDP problem in terms of approximation, despite the attention and effort that researchers have put on it. Interestingly, for the specific case of complete graphs, we are not aware of any inapproximability results. In particular, it is not even known whether the problem in complete graphs is APX-hard.

The approximability of the EDP problem has been tackled by greedy algorithms, LP relaxations, and rounding. In directed arbitrary graphs, the approximability was

[2]Unless P = NP, the approximability class PTAS is strictly contained in the class APX.

settled for some time by the $\Omega(|E|^{1/2-\epsilon})$-hardness results in [23] and the $O(\sqrt{|E|})$ approximation results in [6, 26, 27, 29, 40]. Those approximation bounds were recently applied in [12] to arbitrary dense graphs and improved down to sub-linear. The recent work in [42] trims slightly these bounds via LP rounding and a deeper analysis of a greedy algorithm. However, better approximation ratios have been achieved for some specific types of graphs, such as (sub-)classes of planar graphs (e.g., grid graphs, trees, rings, densely-embedded, plane switch, etc.), complete graphs, random graphs, and expander graphs. Also better performance guarantees are obtained for commodity-restricted versions of the problem, for example for the *edge-disjoint Menger problem* [11, 38, 45]. We address the reader to [13, 27] for recent summaries of the successive achievements concerning the approximation ratios.

3 Greedy Algorithms

A greedy heuristic is a constructive algorithm that builds a solution step-by-step starting from an empty solution. At 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, which lends the name to the algorithm. Advantages of greedy heuristics are that they are usually easy to implement and that they are fast in execution. The disadvantage is that the quality of the solutions provided by greedy algorithms can be quite far from optimal. Due to the fact that the EDP problem usually has to be solved in rather large graphs, research has focused on the development of greedy algorithms. Many of the approximation ratios cited in Section 2 have been calculated when analyzing greedy algorithms. Examples are the *simple greedy algorithm* (SGA) [26] (see Section 3.1), its constrained variant the *bounded-length greedy algorithm* [26, 28], and the *greedy path algorithm* [12, 27]. Due to its lower time complexity when compared to the other greedy approaches we decided to implement SGA and a multi-start version, which we both outline in the following.

3.1 The Simple Greedy Algorithm and its Multi-start Version

The SGA algorithm (see Algorithm 1) is a natural way of approximating the EDP problem that works as follows. It starts with an empty solution S. Then, it proceeds

Algorithm 1 Simple greedy algorithm (SGA) for the EDP problem

```
INPUT: a problem instance $(G, T)$, consisting of a graph $G$ and a commodity list $T$
$S \leftarrow \emptyset$, $\hat{E} \leftarrow E$
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 $S$
```

Algorithm 2 Multi-start simple greedy algorithm (MSGA) for the EDP problem

```
INPUT: a problem instance $(G, T, N_{perm})$, where $N_{perm}$ is the number of restarts
$S_{best} \leftarrow \emptyset$
$T_1 \leftarrow T$
for $i = 1$ to $N_{perm}$ do
  $S_i \leftarrow$ Simple Greedy Algorithm SGA$(G, T_i)$          {See Algorithm 1}
  if $f(S_i) > f(S_{best})$ then
    $S_{best} \leftarrow S_i$
  end if
  if $i < N_{perm}$ then
    $\pi \leftarrow$ random permutation of $|T|$
    $T_{i+1} \leftarrow (\pi(1), \pi(2), \ldots, \pi(|T| - 1), \pi(|T|))$
  end if
end for
OUTPUT: $S_{best}$
```

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 between s_j and t_j (with respect to the edge-weights) is assigned as path for the commodity $T_j = (s_j, t_j)$. Note that the algorithm is deterministic and that the quality of the solutions it provides depends heavily on the order in which the commodities are treated. In addition to its simplicity, the SGA algorithm can be naturally considered an on-line algorithm.

Observe that the SGA 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 SGA by permuting the order of the commodities for each restart. This approach is pseudo-coded in Algorithm 2, in which N_{perm} denotes the number of restarts, S_i denotes the solution under construction in the embedded SGA, and S_{best} denotes the best solution found so far. In the following, we refer to this algorithm as *multi-start greedy algorithm* (MSGA).

3.2 The Greedy Algorithms are Non-optimal

Due to the deterministic decisions that greedy algorithms take during the solution construction, it is sometimes not possible for them to find an optimal solution. This is also the case for the SGA and MSGA greedy algorithms presented here. Consider for example the instance of the EDP problem depicted in Fig. 1, which consists in the depicted graph and the set $T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$ of three commodities to join. The optimal solution in which all three commodities are connected is also shown in bold font in Fig. 1. This solution is found by our ACO algorithm, which is presented next, in a small amount of time (less than 30 ms.). Observe however, that there is no way for any of the greedy algorithms, neither SGA nor MSGA, to find the solution of size greater than two. Since these greedies are based on the shortest paths, the algorithms will tend to connect the commodities through non-consecutively-numbered vertices. For example, when trying to connect first the commodity (v_1, v_7), the SGA algorithm will establish the path $\{v_1, v_9, v_{10}, v_5, v_6, v_7\}$.

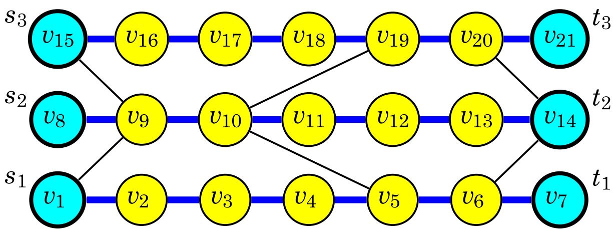


Fig. 1 Instance of the EDP problem with $T = \{(v_1, v_7), (v_8, v_{14}), (v_{15}, v_{21})\}$. Neither SGA nor MSGA can find the optimal solution depicted in *bold font*

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.

4 An Ant Colony Optimization Approach

Ant colony optimization (ACO) [19] is inspired by the foraging behavior of real ants. This behavior enables an ant colony to find shortest paths between food sources and their nest. 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.

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. The exclusive use of local information is certainly a desirable property for algorithms that are aimed for the application to problems for which the computation of global information is costly. This property makes ACO algorithms a natural choice for the application to the EDP problem.

4.1 The Basic ACO 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 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.

4.1.1 Ant Solutions

A solution S constructed by the $|T|$ 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 (see Eq. 1) is characterized by having many plateaus when it is applied to ant solutions. This is because many ant solutions will have the same number of disjoint paths. Thus, a consequence of decomposing the EDP problem is the need to 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 is defined as follows:

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

$$\delta^j(S, e) = \begin{cases} 1 & : \ e \in P_j \in S \\ 0 & : \ \text{otherwise.} \end{cases}$$

This second criterion quantifies 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 fewer edges are shared in a solution, the closer the solution is to disjointness," a comparison operator $f^a(\cdot)$ that differentiates between ant solutions can be defined as follows. For two ant solutions S and S', it holds that $f^a(S) > f^a(S')$ if, and only if,

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

4.1.2 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_e^j for each edge $e \in E$. The set of $|T|$ pheromone models is henceforth denoted by $\tau = \{\tau^1, \ldots, \tau^{|T|}\}$. Our ACO algorithm is implemented in the hyper-cube framework (HCF) [9], which is a way of implementing ACO algorithms such that the pheromone values are bounded between 0 and 1. Furthermore, we borrow an idea from so-called $\mathcal{MAX}$-$\mathcal{MIN}$ Ant Systems (MMASs) [41] and introduce pheromone value limits $\tau_{\min} = 0.001$ and $\tau_{\max} = 0.999$ in order to prevent that the algorithm converges to a solution.

4.1.3 Algorithm Framework and Components

Algorithm 3 is a high level description of our ACO algorithm. Two different ant solutions are kept in the algorithm: S_{ibest} is the *iteration-best* solution, i.e., the best ant solution generated in the current iteration, and S_{gbest} is the *best-so-far* solution, i.e., the best ant solution found since the start of the algorithm.

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 InitializePheromoneValues(τ).

Algorithm 3 Basic ACO algorithm for the EDP problem

```
INPUT: a problem instance $(G, T)$
$S_{gbest} \leftarrow \emptyset$
InitializePheromoneValues($\tau$)
while termination conditions not met do
  $\pi \leftarrow (1, 2, \ldots, |T| - 1, |T|)$
  for $i = 1$ to $N_{sols}$ do
    $S_i \leftarrow \emptyset$
    for $j = 1$ to $|T|$ do
      $P_{\pi(j)} \leftarrow$ ConstructFullPath($s_{\pi(j)}, t_{\pi(j)}$)
      $S_i \leftarrow S_i \cup \{P_{\pi(j)}\}$
    end for
    if $i < N_{sols}$ then $\pi \leftarrow$ GenerateRandomPermutation($|T|$)
  end for
  Choose $S_{ibest} \in \{S_i \mid i = 1, \ldots, N_{sols}\}$ such that
    $f^a(S_{ibest}) \geq f^a(S),\ \forall S \in \{S_i \mid i = 1, \ldots, N_{sols}\}$
  if $f(S_{ibest}) > f(S_{gbest})$ then $S_{gbest} \leftarrow S_{ibest}$
  UpdatePheromoneValues($\tau$,$S_{pbest}$)
end while
OUTPUT: the EDP solution generated from the best solution $S_{gbest}$
```

Second, N_{sols} ant solutions are constructed per iteration. To construct a solution, each ant applies the function ConstructFullPath($s_{\pi(j)}, t_{\pi(j)}$), where π is a permutation of T. 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 π in which the ants construct a path for their commodity is randomly generated by the function GenerateRandomPermutation($|T|$). Third, the value of the variables S_{ibest} and S_{gbest} is updated. Finally, the pheromone values are updated depending on the edges included in S_{gbest}. The algorithm is iterated until some opportunely defined termination conditions are satisfied, and it returns the EDP solution generated from the ant solution S_{gbest}.

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

InitializePheromoneValues(τ) initializes all the pheromone values $\tau_e^j \in \tau^j \in \tau$ to the value $\tau_{\min}$. (see Section 4.1.2.)

ConstructFullPath($s_{\pi(j)}, t_{\pi(j)}$). For constructing a path between the endpoints of the commodity $(s_{\pi(j)}, t_{\pi(j)})$, an ant first chooses randomly to start either from the source $s_{\pi(j)}$ or the target $t_{\pi(j)}$. Then, the ant iteratively moves from node to node using available edges that are not already in the path $P_{\pi(j)}$ 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 and the iterative process is also stopped. 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 incident to v_c which were 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 particularly effective ACO variant called Ant Colony System (ACS) which was proposed by Dorigo and Gambardella in [17]. We draw a random number d_{rate} between 0 and 1. If $d_{rate} \leq 0.75$ (where 0.75 was chosen by parameter tuning as outlined in Section 4 and [8]), the next edge to join path $P_{\pi(j)}$ under construction is chosen deterministically:

$$e^* = \{v_c, u\} \leftarrow \text{argmax}\,\{\tau_e^j \cdot \mathbf{p}(D_e)^\beta \cdot \mathbf{p}(U_e)^\gamma \mid e \in \mathcal{I}^\star_{v_c}\}\ , \tag{3}$$

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 the path of another ant for the same solution. The parameters $\beta > 0$ and $\gamma > 0$ weight the influence of these two terms. The length of the shortest path between two vertices u and v in G is henceforth denoted by $\sigma(u, v)$. The terms $\mathbf{p}(D_e)$ and $\mathbf{p}(U_e)$ are defined as follows:

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

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

$$U(e) = \begin{cases} 2 : e \text{ already used in } S_i \\ 1 : \text{otherwise} \end{cases}$$

If $d_{rate} > 0.75$, the next edge e^* is chosen according to the following transition probabilities:

$$\mathbf{p}(e \mid \mathcal{I}^\star_{v_c}) = \frac{\tau_e^j \cdot \mathbf{p}(D_e)^\beta \cdot \mathbf{p}(U_e)^\gamma}{\sum\limits_{e'\in\mathcal{I}^\star_{v_c}} \tau_{e'}^j \cdot \mathbf{p}(D_{e'})^\beta \cdot \mathbf{p}(U_{e'})^\gamma}\ , \forall\, e \in \mathcal{I}^\star_{v_c} \tag{4}$$

If the probability of doing a deterministic construction step is too high, there is the danger that the algorithm gets stuck in low quality regions of the search space. On the other side, doing deterministic construction steps bears the potential of leading the algorithm quite quickly to good areas of the search space. In our experiments (see Section 4 and [8]) we found 0.75 to be a good trade-off. Concerning the composition of the transition probabilities, 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:

$$\tau_e^j \leftarrow \begin{cases} (1 - e_{rate}) \cdot \tau_e^j & : \quad e \in P_{\pi(j)} \in S, \quad j = 1, \ldots, |T| \\ \tau_e^j & : \quad \text{otherwise.} \end{cases} \tag{5}$$

The reason for this pheromone evaporation is the desire to diversify the search in each iteration. After parameter tuning we chose a setting of $e_{rate} = 0.10$.

UpdatePheromoneValues(τ,S_{update}). As it is usual in ACS algorithms, in our basic ACO algorithm only the ant solution S_{gbest} is used for updating the pheromone values for all $j \in \{1, \ldots, |T|\}$ as follows:

$$\tau_e^j \leftarrow \max\left\{\tau_e^j + \rho \cdot \left(1 - \tau_e^j\right),\ \tau_{\max}\right\} \quad \forall\, e \in P_j, \tag{6}$$

where $\rho \in (0, 1]$ is a constant value which is called *learning rate* in algorithms that are implemented in the hyper-cube framework. For all our experiments we have set ρ to 0.1.

4.2 Motivation for Additional Algorithmic Features

The direct application of a basic ACO scheme to a problem achieves sometimes quite good results. However, the algorithms' performance can often be improved by applying some additional features to the search process, especially when a rather unusual problem such as the EDP is tackled. In this section, we describe how the basic ACO approach introduced in Section 4.1 may be enriched with different strategies that modify the way the algorithm explores the solution space. In the following we propose four additional features, explaining why these features might lead to an improvement, before we outline the implementation of these features more in detail in the subsequent section.

Sequential versus parallel solution construction For constructing a solution, method ConstructFullPath($s_{\pi(j)}$, $t_{\pi(j)}$) as applied in the basic ACO algorithm (see Algorithm 3) considers one commodity after the other, and constructs for each commodity a path between its endpoints before the next commodity is considered. In the following we refer to this way of constructing solutions as the sequential way. As an alternative we propose to construct paths for all the commodities in parallel. Hereby, at each constructing step each ant changes its partial path by either adding exactly one edge, or by doing a backtracking move. Note that there is a considerable difference in the influence of the edge-usage information between sequential and parallel construction. This changes the dynamics of the search process and might lead to different results.

The use of a candidate list strategy A candidate list strategy is a mechanism to restrict the number of available choices to be considered at each construction step. Usually this restriction applies to a number of the best choices with respect to their transition probabilities (see Eq. 4). For example, in the case of the application of ACS to the travelling salesman problem the restriction to the closest cities at each construction step improved the final solution quality as well as it led to a significant speedup of the algorithm (see [21]). The reasons for that are as follows: First, in ACO algorithms each choice has a positive probability to to selected. However, in order to construct high quality solutions it is often enough to consider only the "good" choices at each construction step, and, therefore, to consider choices with a low probability is often a waste of time. Second, to consider less choices at each step speeds up the solution construction.

Different search phases characterized by the pheromone update In general, the pheromone update procedure is an important component of every ACO algorithm. In fact, it determines to a large degree the failure or the success of the algorithm. Most of the existing generic variants of ACO only differ in the pheromone update. In the case of the EDP application, we propose a pheromone updating scheme that is based on the following idea. In our basic algorithm, all the paths of the ant solution S_{gbest} are used for updating the pheromone values (including the non-disjoint paths). However, at the beginning of the search it might be better not to use these non-disjoint paths for updating in order to maintain a higher degree of freedom for finding also edge-disjoint paths for the commodities that initially prove to be problematic. Therefore, we propose a first phase of the algorithm in which only disjoint paths are used for updating the pheromone values, followed by a second phase which is initiated when no improvements can be found over a certain time. In this second phase, all the paths are used for updating the pheromone values. In a way, in the first phase of the algorithm we try to improve the first criterion of the objective function (while disregarding the second one), and in the second phase we try to improve also the second criterion. Once the second phase leads to an improvement also in terms of the first criterion, the algorithm changes back to the first phase.

Partial destruction of solutions (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. Most of the successful applications incorporate algorithm features to escape from these situation once detected. In case of our algorithm for the EDP problem we propose a partial destruction of the disjoint part of the solution which is used for updating the pheromone values. This mechanism is initiated once the algorithm was unable to improve the currently best solution for a number of subsequent applications of first and second phase. Similar ideas are applied in backtracking procedures, or in the perturbation mechanism of local search based methods, such as iterated local search or variable neighborhood search (see [10]).

4.3 The Result: An Extended ACO Algorithm

In the following we outline in more detail our extended ACO algorithm including the additional features motivated in the previous section. The extended algorithm (for the pseudo-code see Algorithm 4) is based on the basic ACO algorithm as described

Algorithm 4 Extended ACO algorithm for the EDP problem

```
INPUT: a problem instance $(G, T)$
$S_{gbest} \leftarrow \emptyset$, $S_{pbest} \leftarrow \emptyset$, $c_{crit1} \leftarrow 0$, $c_{crit2} \leftarrow 0$
all_update $\leftarrow$ FALSE
InitializePheromoneValues($\tau$)
while termination conditions not met do
  $\pi \leftarrow (1, 2, \ldots, |T|-1, |T|)$
  for $i = 1$ to $N_{sols}$ do
    $S_i \leftarrow$ ConstructSolution($G$,$\pi$)  {See Algorithm 5}
    if $i < N_{sols}$ then $\pi \leftarrow$ GenerateRandomPermutation($|T|$)
  end for
  Choose $S_{ibest} \in \{S_i \mid i = 1, \ldots, N_{sols}\}$ s.t. $f^a(S_{ibest}) \geq f^a(S)$, $\forall S \in \{S_i \mid i = 1, \ldots, N_{sols}\}$
  if $f(S_{ibest}) > f(S_{gbest})$ then $S_{gbest} \leftarrow S_{ibest}$
  if $f^a(S_{ibest}) > f^a(S_{pbest})$ then
    $c_{crit2} \leftarrow 0$
    $S_{psave} \leftarrow S_{pbest}$
    $S_{pbest} \leftarrow S_{ibest}$
    if $f(S_{ibest}) > f(S_{psave})$ then
      $S_{update} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)  {First phase}
      $c_{crit1} \leftarrow 0$
      all_update $\leftarrow$ FALSE
    else
      $c_{crit1} \leftarrow c_{crit1} + 1$
    end if
    if all_update then $S_{update} \leftarrow S_{pbest}$  {Second phase}
  else
    $c_{crit2} \leftarrow c_{crit2} + 1$
  end if
  if all_update then
    if $c_{crit2} > c2_{\max}$ then
      $S_{pbest} \leftarrow$ DestroyPartially($S_{pbest}$)  {Escape mechanism}
      $S_{update} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)
      $c_{crit2} \leftarrow 0$, $c_{crit1} \leftarrow 0$
    end if
  else
    all_update $\leftarrow (c_{crit1} > c1_{\max})$
  end if
  UpdatePheromoneValues($\tau$,$S_{update}$)
end while
OUTPUT: the EDP solution generated from the best solution $S_{gbest}$
```

in Section 4.1 (see Algorithm 3). Three different solutions are kept in the algorithm. In addition to the ant solutions S_{ibest} and S_{gbest}, we keep an ant solution S_{pbest}, which is the *currently best* solution, i.e., the best ant solution generated since the last escape action (see below). Note that the solution S_{pbest} takes over the role of solution S_{gbest} when compared to the basic ACO algorithm. S_{gbest} is only used to keep the best solution found. Additionally, two parameters c_{crit1} and c_{crit2} are introduced. When the algorithm is in the first phase (i.e., the phase in which only the disjoint paths of solution S_{pbest} are used for updating), c_{crit1} counts the number of successive iterations without improvement of the first criterion of the objective function. Similarly, when the algorithm is in the second phase (i.e., all paths of S_{pbest} are used for updating) c_{crit2} counts the number of successive iterations without improvement of the second criterion. Limits $c1_{\max}$ (for c_{crit1}) and $c2_{\max}$ (for c_{crit2}) are used to determine when the algorithm should change phases. In the following we explain in more detail the features of the extended algorithm.

Solution construction (including the candidate list strategy) The solution construction of our extended ACO algorithm is performed in method ConstructSolution(G,π), which includes the possibility of a sequential as well as the one of a parallel solution construction. Algorithm 5 shows a high-level description of the extended solution construction mechanism. Two (setup) parameters are needed: the parameter $construction_{type}$, which determines whether the construction is done sequentially (i.e., $construction_{type}$ = SEQUENTIAL) or in parallel (i.e., $construction_{type}$ = PARALLEL), and the parameter $candidatesList_{size}$, which configures the candidate list strategy. This strategy restricts the set of candidate edges $\mathcal{I}^{\star}_{v_c}$ that can be considered at every

Algorithm 5 Method ConstructSolution(G,π) of Algorithm 4.

```
INPUT: a graph $G$ from a problem instance $(G, T)$, and a permutation $\pi$ of $T$.
$S \leftarrow \emptyset$
**if** $construction_{type}$=SEQUENTIAL **then**
   **for** $j = 1$ to $|T|$ **do**
      $P_{\pi(j)} \leftarrow$ ConstructFullPath($s_{\pi(j)}, t_{\pi(j)}$)
      $S \leftarrow S \cup \{P_{\pi(j)}\}$
   **end for**
**else if** $construction_{type}$=PARALLEL **then**
   **for** $j = 1$ to $|T|$ **do**
      $P_{\pi(j)} \leftarrow \emptyset$
   **end for**
   $nb\_paths\_finished \leftarrow 0$
   $j \leftarrow 0$
   **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**
            $nb\_paths\_finished \leftarrow nb\_paths\_finished + 1$
            $S \leftarrow S \cup \{P_{\pi(j+1)}\}$
         **end if**
      **end if**
      $j \leftarrow (j+1) mod |T|$
   **until** $nb\_paths\_finished = |T|$
**end if**
OUTPUT: an ant solution $S$
```

construction step in one of the three following ways: Either (1) the two best choices are considered (i.e., the two choices that have a higher transition probability $\mathbf{p}(\cdot \mid \mathcal{I}^{\star}_{v_c})$ than the others), or (2) the 50% best choices, or (3) all the choices (which is, in fact, the setting of the basic ACO algorithm). This means that $candidatesList_{size} \in \{2, 50\%, \text{all}\}$. The procedures of Algorithm 5 work as follows:

– ConstructFullPath($s_{\pi(j)}, t_{\pi(j)}$). This method is the same as in the basic ACO algorithm just that candidate list strategies might be applied.
– isFinishedPath($P_{\pi(k)}$). This method returns a boolean value indicating whether the path $P_{\pi(k)}$ is finished, i.e., whether a path could be established from $s_{\pi(k)}$ to $t_{\pi(k)}$, or if it was determined that no path exists.
– ExtendOneStepPath($P_{\pi(j+1)}, \tau^{\pi(j+1)}$). This method tries to extend the path $P_{\pi(j+1)}$, i.e., the path under construction by the $(j+1)$-th ant, by adding exactly one edge. Otherwise, it performs a backtracking step. Note that also in this method the use of the candidate list strategies applies.

Implementation of two algorithm phases and an escape mechanism In the following we outline the implementation of the two phases of our algorithm (as motivated in the previous section). The pheromone update performed in function UpdatePheromoneValues(τ,S_{update}) of Algorithm 4 works in the same way as

Algorithm 6 Method DestroyPartially (S_{pbest}) of Algorithm 4. ExtractDisjointPaths(S_{pbest}) implements the process of returning a valid EDP solution from an ant solution as explained in Section 4.1.1. The method Cost(S_{temp}) returns the number of disjoint paths in S_{temp}. The methods ChoosePathAtRandom(S_{temp}) and ChooseLongestPath(S_{temp}) return, respectively, a randomly chosen disjoint path of S_{temp} and the longest disjoint path of S_{temp}. The method ResetPheromoneModel(τ^i) resets to $\tau_{\min}$ all the pheromone values of the pheromone model τ^i, i.e., $\tau^i_e \leftarrow \tau_{\min}$, $\forall e \in E$.

```
INPUT: an ant solution $S_{pbest}$
if $destruction_{rate} > 0$ then
  $S_{pbest} \leftarrow$ ExtractDisjointPaths($S_{pbest}$)
  $nb_{paths} \leftarrow \lceil destruction_{rate} \cdot$ Cost($S_{pbest}$)$\rceil$
  $nb_{removed} \leftarrow 0$
  repeat
    if $destruction_{type}$=RANDOM then
      $P_i \leftarrow$ ChoosePathAtRandom($S_{pbest}$)
    else if $destruction_{type}$=LONGEST then
      $P_i \leftarrow$ ChooseLongestPath($S_{pbest}$)
    end if
    $S_{pbest} \leftarrow S_{pbest} \setminus \{P_i\}$
    ResetPheromoneModel($\tau^i$)
    $nb_{removed} \leftarrow nb_{removed} + 1$
  until $nb_{removed} = nb_{paths}$
end if
OUTPUT: the solution $S_{pbest}$ partially destroyed
```

explained in Section 4.1.3, except for the following difference: The solution S_{update} that is used for updating is obtained in different ways, depending on the search phase in which our algorithm is at the moment of the update.

As mentioned above, our algorithm works in two phases based on the two criteria of function $f^a(\cdot)$ (see Eq. 2). First, it tries to improve only the first criterion. For this purpose, solution S_{update} that is used for updating the pheromone values is obtained by applying function ExtractDisjointPaths(S_{pbest}), which implements the process of returning a valid EDP solution from the ant solution S_{pbest} as explained in Section 4.1.1. If for a number of $c1_{\max}$ iterations the first criterion could not be improved, then the algorithm tries to improve the second criterion. For this purpose, solution S_{update} that is used for updating the pheromone values is a copy of the current solution S_{pbest}, including possibly non-disjoint paths. 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, some of the paths from the EDP solution that can be produced from S_{pbest} are deleted from S_{pbest}. This action, which is performed by function DestroyPartiallyS_{pbest} of Algorithm 4, can be seen as a mechanism to escape from the current area of the search space.

Function DestroyPartially(S_{pbest}), whose pseudo-code is outlined in Algorithm 6, has two different setup parameters: (1) The parameter $destruction_{rate}$ determines how many of the disjoint paths are destroyed, and (2) parameter $destruction_{type}$ indicates which paths to destroy. We chose the following three settings for $destruction_{rate}$: 0, 0.25, and 0.5, where 0 means that none of the paths is destroyed (i.e., the escape mechanism is not applied), 0.25 means that 25% of the paths are destroyed, and similarly for 0.5. Concerning parameter $destruction_{type}$, we propose two different schemes: (1) Setting $destruction_{type}$ = RANDOM causes the destruction of randomly selected paths, whereas (2) setting $destruction_{type}$ = LONGEST initiates the destruction of the longest paths, i.e., those paths with the highest number of edges. The second setting assumes that the longer a path is, the more restrictions it introduces to assure disjointness of the paths that still conflict with others. Thus, by removing the longest disjoint paths, the number of total edges available is maximized.

5 Experimental Evaluation

In the following we first outline the characteristics of our problem instances (see Section 5.1), before we describe in Section 5.2 the parameter setting for our ACO algorithms. The tuning process is documented in detail in [8]. Finally, in Section 5.3 we present the experimental evaluation of our ACO approaches in comparison to the results obtained by the greedy approaches that we outlined in Section 3. All the algorithms in this paper were implemented in C++ 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 3.06 GHz and 900 Mb of memory running a Linux operating system and with 1.98 processor load. Moreover, our algorithms 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 greedy approaches need to partially recompute this information after the routing of each commodity, this is not necessary for our ACO algorithm.

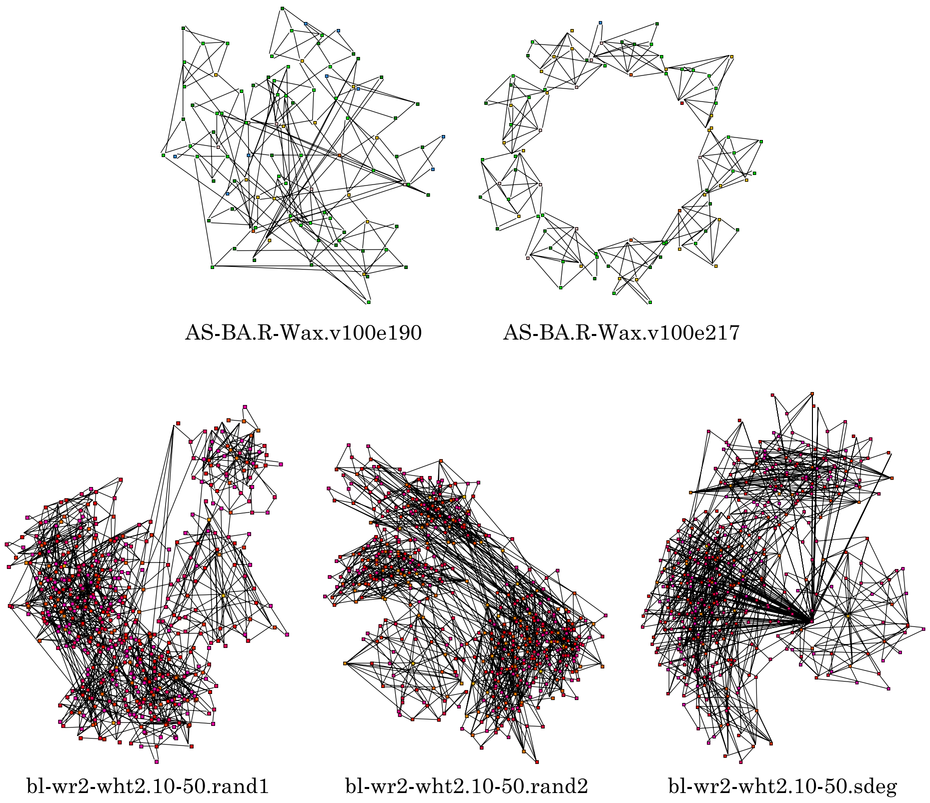


Fig. 2 Graphs generated with BRITE. These graphs consist of a *two-level top–down hierarchical topology* (Autonomous System level plus router level), which are typical for Internet topologies

5.1 Problem Instances

In [7] we proposed a first set of benchmark instances for the EDP problem in order to experimentally evaluate our preliminary ACO approach. This set of instances includes three graphs representing different communication networks. Two of them, namely graph3 and graph4, were created by researchers of the Computational Optimization & Graph Algorithms group of the Technische Universität Berlin. Their structure resembles parts of the communication network of the Deutsche Telekom AG in Germany. The third graph, namely bl-wr2-wht2.10-50.rand1,[3] was created with the network topology generator BRITE [33]. With the same network topology generator, but with different topology properties, we now generated more graphs, namely AS-BA.R-Wax.v100e190, AS-BA.R-Wax.v100e217, bl-wr2-wht2.10-50.rand2, and bl-wr2-wht2.10-50.sdeg (see Fig. 2). More information about the options chosen to generate these topologies with BRITE is provided in Table 2. Additionally to these graphs, we have also considered two mesh graphs , one of size 15×15 (graph

[3]In [7], this graph was just called bl-wr2-wht2.10-50.

Table 1 Main quantitative measures of our graphs

Graph	$\lvert V \rvert$	$\lvert E \rvert$	min.	Degree avg.	max.	Diameter	Clustering coefficient
graph3 [7]	164	370	1	4.51	13	16	0.226
graph4 [7]	434	981	1	4.52	20	22	0.156
AS-BA.R-Wax.v100e190	100	190	2	3.80	7	11	0.379
AS-BA.R-Wax.v100e217	100	217	2	4.34	8	13	0.411
bl-wr2-wht2.10-50.rand1 [7]	500	1020	2	4.08	13	23	0.102
bl-wr2-wht2.10-50.rand2	500	1020	2	4.08	11	27	0.097
bl-wr2-wht2.10-50.sdeg	500	1020	2	4.08	14	28	0.086
mesh15x15	225	420	2	3.73	4	28	0
mesh25x25	625	1200	2	3.84	4	48	0

mesh15x15), and one of size 25×25 (graph mesh25x25). All these new graphs, together with the three graphs used in [7], define the topologies of the instances to be used in our experiments. Table 1 shows their main features and quantitative measures.

An instance of the EDP problem consists of a graph and a set of commodities. For each of the nine graphs we have therefore randomly generated different sets of commodities. Hereby, we made the size of the commodity sets dependent on the number of vertices of the graph.[4] For each graph $G = (V, E)$ we generated 20 different instances with $0.10|V|$, $0.25|V|$ and $0.40|V|$ commodities. This makes a sum of 60 instances for each graph, and 540 instances altogether.

5.2 Tuning of the Algorithm Parameters

Due to the relatively high number of algorithm parameters it was not feasible to tune all the parameter values together. In order to ease the task of parameter tuning we first divided the set of parameters into two subsets. The first subset consists of parameters which are in our opinion not critical, in the sense that appropriate values can easily be found. These parameters are $\tau_{\min}$ and $\tau_{\max}$ for limiting the pheromone values, N_{sols} (the number of solution constructions per iteration), the learning rate ρ, and the maximum number of non-improving iterations for the first and second algorithm phase, namely $c1_{\max}$, and $c2_{\max}$ respectively. For these parameters we chose values that we found to be well-working after preliminary tests.

The remaining parameters, which define our extended ACO approach in terms of its added features, are the following ones: d_{rate} is the proportion of deterministic construction steps; β and γ balance the influence of the distance to the goal vertex and the overall usage of edges, respectively; e_{rate} determines how much pheromone is evaporated from the edges belonging to the current best solution; $construction_{type}$ specifies the strategy followed for constructing the paths of a solution; $candidatesList_{size}$ specifies the size of the candidate lists; the variable $distinguish_{phase1}$ indicates whether to use the two search phases based on

[4]Note that this is different to what we did in [7], where we used fixed commodity set sizes independent of the graph sizes. The drawback of fixed number of commodities is that instances composed by bigger graphs are easier to solve than instances composed by smaller graphs.

Table 2 Parameters used for the generation of network topologies with BRITE (see [33] for details)

Graph	$\|V\|$	Model	Node placement	m	Edge connection model
AS-BA.R-Wax.v100e190	(20, 5)	(Barabási-Albert [5], Waxman)	(random, random)	(2, 2)	random
AS-BA.R-Wax.v100e217	(10,10)	(Barabási-Albert [5], Waxman)	(random, random)	(2, 2)	random
bl-wr2-wht2.10-50.rand1	(10,50)	(Waxman, Waxman)	(random, heavy-tailed)	(2, 2)	random
bl-wr2-wht2.10-50.rand2	(10,50)	(Waxman, Waxman)	(random, heavy-tailed)	(2, 2)	random
bl-wr2-wht2.10-50.sdeg	(10,50)	(Waxman, Waxman)	(random, heavy-tailed)	(2, 2)	smallest degree

In each value tuple (X_{as}, X_{rou}), X_{as} is the value of the parameter at the AS level, and X_{rou} is the value of the parameter at the router level. Parameter m specifies the number of links for each new node that is added while constructing the topology, and the edge connection model specifies under which criteria those edges are added. For all the graphs, the growth type (i.e., how nodes join the topology) is incremental. In graph bl-wr2-wht2.10-50, the edge connections between the AS level and the router level are introduced using the Waxman probability model [44] with parameters $\alpha = 0.15$ and $\beta = 0.20$ (NOTE: this β has no relation with the ones of the same name in Eq. 3 of our algorithm); in graphs AS-BA.R-Wax.v100e190 and AS-BA.R-Wax.v100e217 both levels are interconnected by choosing edges at random the improvement of the two criteria of the objective function; $destruction_{rate}$ and $destruction_{type}$ indicate how much of the current best solution must be destroyed and how this destruction should be performed.

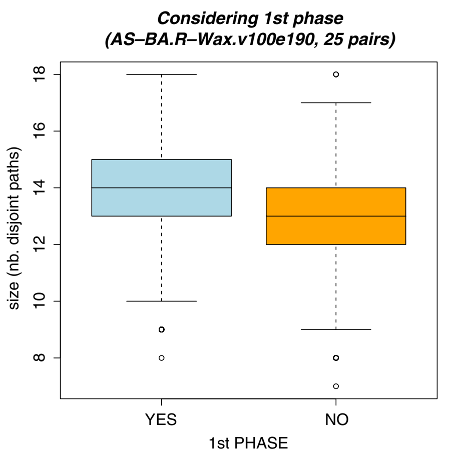
For the purpose of tuning the remaining parameters we used all the instances derived from graph AS-BA.R-Wax.v100e190.[5] First, we applied the extended ACO approach with all possible combinations of parameter value settings to all the 60 instances. Then, we progressively fixed parameter values in the following way. At each step i, we chose the parameter p_i (from the ones that still needed a value) for which we could see a clearer result than for the rest. Note that this process was rather based on personal judgment than on mathematical rigor. Hereby, the i-th parameter p_i is decided according to the experimental results obtained for those experiments in which the previously treated parameters $p_1, \ldots, p_{i-1}$ have their fixed value, and the rest of parameters $p_{i+1}, p_{i+2}, \ldots,$ etc. can have any allowed value. In those cases in which the analysis of only one parameter did not provide enough information to decide for a value, we studied the combined influence of two or more parameters with respect to the rest.

The complete tuning process is documented in [8]. Here we only show some illustrating examples. The first parameter for which we chose a value was parameter $distinguish_{phase1}$, which has settings {YES, NO}. Some results of our experiments are shown in Fig. 3 in the form of box-plots. When analyzing the box-plots it becomes clear that both in terms of solution quality and computation time better results are obtained when setting $distinguish_{phase1}$ to YES. We use this setting for deciding a value for all the remaining parameters. Thus only the results of experiments for which $distinguish_{phase1}$ is set to YES are considered in the following.

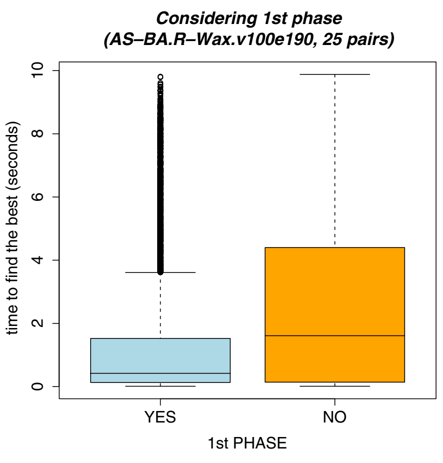
Second, we decided for a value for parameter d_{rate}. This parameter has five possible settings: {0,0.25,0.5,0.75,1}. Figure 4 depicts results of the experiments obtained for the tuning of this parameter. The results show that determinism is needed, although not too much. There is no difference between the quality of the solutions obtained for values 0.25, 0.5, and 0.75. However, the setting of d_{rate} to 0.75 needs less computational time in order to reach the same solution quality (see Fig. 4).

After fixing parameter values for β and γ to 1, it still remained to fix a parameter value for the candidate list size, the destruction rate, the destruction type and the solution construction mechanism. An initial separate study of them was not conclusive, since these parameters seemed to be strongly related. Furthermore, the experiments for the instances derived from graph AS-BA.R-Wax.v100e190 did not provide much information. We used graph4 for continuing the tuning, since this graph proved to be difficult for a preliminary version of our ACO approach [7], which lacked most of the additional features of the extended ACO. Figure 5 shows results (under different destruction rates) concerning the joined analysis of the candidate list size, the destruction type and the solution construction strategies, for the 20 instances composed by graph4 and 173 commodities. First, the results displayed in this figure show that parallel solution construction seems to have advantages over the sequential construction, independently of the destruction rate. It can also be observed that, the higher the number of considered candidates is, the lower is the quality of the solutions obtained. Concerning the destruction type, to destroy the

[5]Note that the computation time limits are the same as described in Table 4.



(a) Cost of the best solutions found

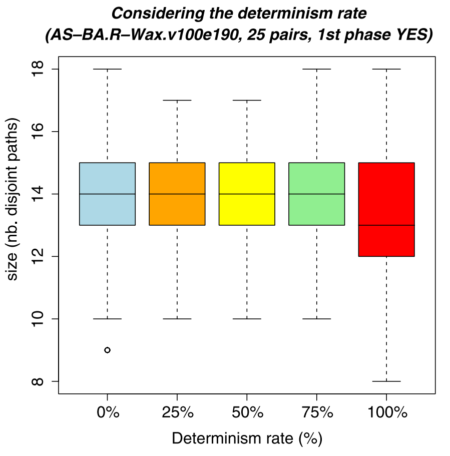


(b) Time to find the best solutions

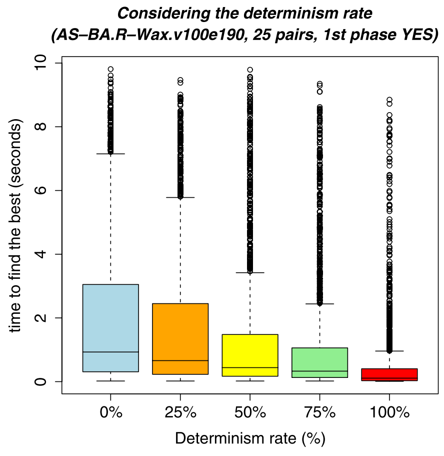
Fig. 3 Results concerning the setting of parameter $distinguish_{phase1}$. The *box-plots* show results of all experiments concerning the instances with 25 commodities derived from graph AS-BA.R-Wax.v100e190 in terms of the best found solutions, respectively the times at which these solutions were found, of the extended ACO algorithm. In each graphic, the *left box-plot* (labelled as YES) shows the case in which only the disjoint paths of the ant solution contribute to the update of the pheromone (1st phase); the *right box-plots* (labelled as NO) show the case in which the whole ant solution is used for updating the pheromone

longest paths in the solution generally provides slightly better results than destroying some paths chosen at random.

The final parameter setting is summarized in Table 3.



(a) Cost of the best solutions found

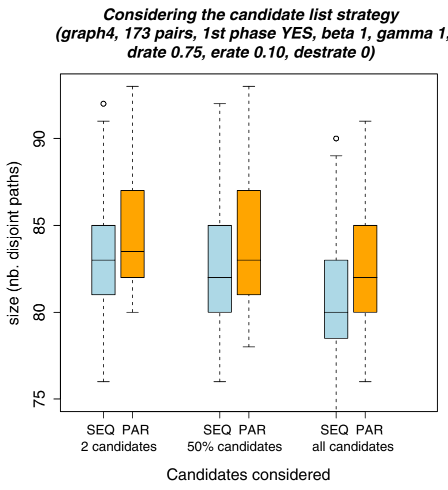


(b) Time to find the best solutions

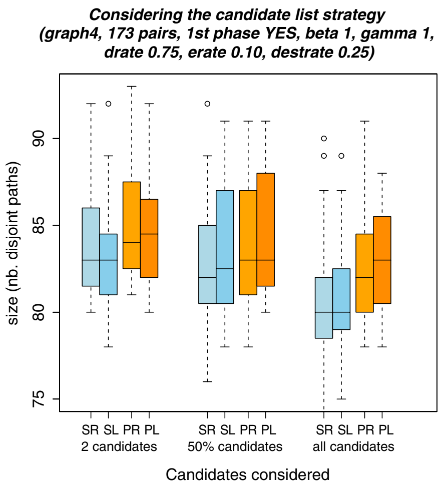
Fig. 4 Results concerning the setting of parameter d_{rate} when having fixed $distinguish_{phase1}$ to YES. The *box-plots* show results of all experiments concerning the instances with 25 commodities derived from graph AS-BA.R-Wax.v100e190 in terms of the best found solutions, respectively the times at which these solutions were found, of the extended ACO algorithm. In each graphic, the *five box-plots* show the results when considering 0%, 25%, 50%, 75% and 100% determinism, respectively

5.3 Experiments and Results

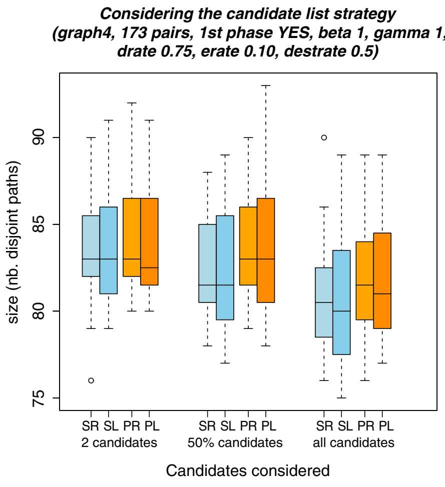
We applied the algorithms presented in this work to all the instances exactly once. First, we applied MSGA with 50 restarts (i.e., $N_{perm} = 50$) to each of the 540 instances. The computation time of MSGA was used as a maximum CPU time limit for both



(a) Destruction rate 0%



(b) Destruction rate 25%



(c) Destruction rate 50%

Fig. 5 Results concerning the joined analysis of the candidate list size, the destruction type and the sequential/parallel solution construction strategies, under different destruction rates. The box-plots show results of all experiments concerning the instances with 173 commodities derived from graph graph4. The x-axis shows different sizes for the candidate lists (i.e., 2 candidates, 50% of the candidates, and all the candidates), together with the different solution construction strategies and the destruction criteria. Hereby we use the abbreviations SR, SL, PR, and PL, where S stands for sequential, P for parallel, L for longest path destruction, and R for random path destruction

Table 3 Final parameter setting for the extended ACO algorithm

Parameter/property	Tuning domain	Chosen value
$\tau_{\min}$	–	0.001
$\tau_{\max}$	–	0.999
N_{sols}	–	10
ρ	–	0.1
$c1_{\max}, c2_{\max}$	–	20 for both
d_{rate}	$\{0, 0.25, 0.50, 0.75, 1\}$	0.75
β, γ	$\{0.1, 1, 10\}$ for both	1 for both
e_{rate}	$\{0.01, 0.05, 0.10\}$	0.10
$construction_{type}$	{SEQUENTIAL, PARALLEL}	PARALLEL
$candidatesList_{size}$	{2, 50%, ALL}	2
$distinguish_{phase1}$	{YES, NO}	YES
$destruction_{rate}$	$\{0, 0.25, 0.50\}$	0.25
$destruction_{type}$	{RANDOM, LONGEST}	LONGEST

Note that for the simple ACO we chose the same values for parameters $\tau_{\min}$, $\tau_{\max}$, N_{sols}, ρ, $c1_{\max}$, $c2_{\max}$, d_{rate}, β, γ, and e_{rate} as for the extended ACO

versions of the ACO algorithm. We present the results as averages over the 20 instances of each combination of graph and commodity number in Table 4 and Table 5. The layout of these tables is explained in their caption. In Table 4, we report the results of our experiments for graphs graph3, graph4, AS-BA.R-Wax.v100e190, AS-BA.R-Wax.v100e217, and bl-wr2-wht2.10-50.rand1. The experiments for the remaining graphs are reported in Table 5, in which we have omitted the results of the experiments with the simple ACO algorithm since, as it can be seen in Table 4, it does never get better results than the extended ACO or the MSGA greedy.

Concerning the comparison between SGA and MSGA, we observe a clear advantage of MSGA. This means that the order in which the commodities are treated is crucial in achieving a good performance. 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 MSGA. The prize we have to pay for exploiting this knowledge is the increased computation time. Concerning the comparison of the two ACO approaches (see Table 4), we can observe that the features that distinguish the extended ACO from the simple ACO approach are of great benefit. The extended ACO approach consistently obtains better solution qualities in less computational time. When comparing the simple ACO approach with the MSGA greedy algorithm, we can observe that the former, although being more sophisticated, does not achieve a better performance. For the graphs representing Internet topologies, both the MSGA and the simple ACO approach perform very similar. For the other graphs, the MSGA is often faster.

More in detail, we can observe that in 21 out of 27 cases the extended ACO approach beats all the other algorithms. The extended ACO approach is on average better than MSGA (0.87% better in the case of 10% of the pairs, 7.98% in the case of 25% of the pairs, and 14.22% in the case of 25% of the pairs). In some cases, the extended ACO approach is even much better than the MSGA, e.g., for graph4 and 173 commodities it is 15.07% better, for mesh15x15 and 90 commodities it is 18.87% better, and for mesh25x25 and 250 commodities it is 29.79% better. Additionally,the

Table 4 Comparison of the results obtained by the SGA, the MSGA, and both versions of the ACO algorithm

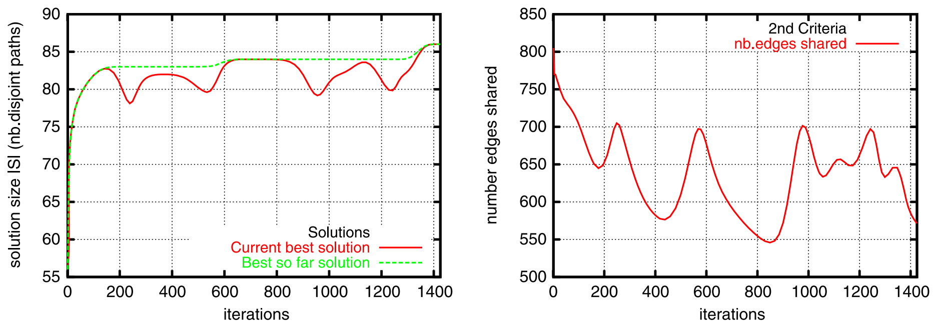
Graph	Number of commodities	SGA			MSGA			Simple ACO			Extended ACO			Avg. CPU time
		$\overline{q}$	σ	$\bar{t}$	$\overline{q}$	σ	$\bar{t}$	$\overline{q}$	σ	$\bar{t}$	$\overline{q}$	σ	$\bar{t}$	
graph3	16	15.30	0.781	0.566	15.70	0.557	0.960	15.65	0.572	1.321	**15.70**	0.557	0.457	30.582
graph3	41	29.00	2.864	1.298	**32.00**	2.302	25.235	30.55	2.109	41.153	31.80	1.990	27.953	79.619
graph3	65	33.70	2.777	2.156	37.60	2.577	49.267	36.80	2.502	82.578	**40.30**	2.571	57.899	126.945
graph4	43	40.50	1.628	12.121	**42.05**	1.024	95.744	40.05	1.396	244.458	41.45	1.284	168.871	237.520
graph4	108	58.10	4.194	31.138	64.10	3.064	697.456	60.85	2.762	1,231.099	**68.15**	2.725	730.436	1,656.475
graph4	173	66.75	4.846	49.281	73.95	3.542	974.350	73.40	4.005	1,940.110	**85.10**	3.534	1,111.982	2,603.872
AS-BA.R-Wax.v100e190	10	8.75	0.942	0.114	**9.10**	0.943	0.579	8.80	0.980	0.549	8.95	0.973	0.611	6.665
AS-BA.R-Wax.v100e190	25	12.30	1.900	0.280	14.25	1.374	4.809	14.05	1.117	4.921	**14.85**	1.195	3.718	16.740
AS-BA.R-Wax.v100e190	40	15.45	2.500	0.443	17.95	1.624	7.796	18.80	1.913	8.608	**19.45**	1.936	4.121	26.850
AS-BA.R-Wax.v100e217	10	7.00	1.225	0.103	**8.05**	0.921	0.427	7.70	1.005	0.480	7.88	0.927	0.164	6.892
AS-BA.R-Wax.v100e217	25	11.40	1.882	0.300	13.60	1.463	4.330	13.70	1.646	3.127	**13.83**	1.579	1.816	17.622
AS-BA.R-Wax.v100e217	40	14.60	1.685	0.497	17.00	1.949	9.833	17.35	1.711	6.433	**17.80**	1.646	2.212	28.318
bl-wr2-wht2.10-50.rand1	50	19.70	2.238	17.926	22.55	2.397	318.518	23.85	2.128	171.857	**24.10**	1.947	155.899	971.488
bl-wr2-wht2.10-50.rand1	125	34.15	4.464	46.387	38.10	4.369	1,004.462	41.80	4.545	594.902	**42.30**	4.540	344.092	2,425.090
bl-wr2-wht2.10-50.rand1	200	46.70	4.961	62.158	50.85	4.892	1,151.197	54.50	4.955	1,101.917	**56.30**	5.245	847.415	3,124.550

The first column gives the name of the graph and the second column the number of the commodities, which are obtained as the 10, 25, and 40% of the number of nodes of the graphs. For each algorithm there are three columns reporting on the average results obtained for the 20 instances of each combination of graph topology and number of commodities. The first of these three columns (headed by $\overline{q}$) shows the average of the values of the best solutions found for the 20 instances. Such an average is in boldface when the result is the best in the comparison. In case of ties the computation time decides. The second column provides the standard deviation of the 20 values used to compute $\overline{q}$, and the third column (headed by $\bar{t}$ reports on the average time (in seconds) needed to find the best solution values for the 20 instances. Finally, the last column shows the average computation time of MSGA (for each instance, the computation time of MSGA on that instance was used as computation time limit for both versions of the ACO approach)

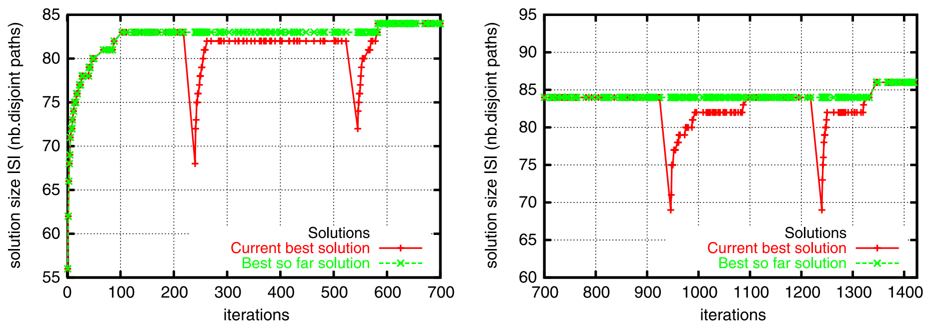
Table 5 Comparison of the results obtained by the SGA, the MSGA, and the extended ACO algorithm

Graph	Number of	SGA			MSGA			Extended ACO			Avg. CPU time
	commodities	$\bar{q}$	σ	$\bar{t}$	$\bar{q}$	σ	$\bar{t}$	$\bar{q}$	σ	$\bar{t}$	
bl-wr2-wht2.10-50.rand2	50	20.70	3.148	12.529	23.85	2.780	187.166	**25.25**	2.700	79.432	665.305
bl-wr2-wht2.10-50.rand2	125	36.00	5.099	31.370	40.10	5.069	674.610	**43.70**	5.197	143.348	1,652.224
bl-wr2-wht2.10-50.rand2	200	50.05	5.045	51.411	54.50	4.685	1,009.073	**59.30**	4.807	720.254	2,650.085
bl-wr2-wht2.10-50.sdeg	50	19.75	2.527	12.380	22.55	2.655	208.489	**24.15**	2.242	48.059	659.053
bl-wr2-wht2.10-50.sdeg	125	35.20	4.697	32.584	38.85	4.704	534.621	**42.25**	4.668	190.057	1,654.224
bl-wr2-wht2.10-50.sdeg	200	46.70	6.900	51.669	51.20	7.075	1,306.132	**55.70**	6.543	798.970	2,653.177
mesh15x15	22	19.65	1.314	1.160	**21.40**	0.663	13.389	21.15	1.424	10.620	57.338
mesh15x15	56	26.80	2.294	2.478	31.45	1.774	54.525	**32.80**	4.005	56.767	144.534
mesh15x15	90	31.70	3.635	4.070	36.30	2.452	60.045	**43.15**	4.840	112.517	249.644
mesh25x25	62	41.00	3.178	24.374	**45.50**	2.907	398.367	43.96	2.747	679.553	1,300.051
mesh25x25	156	52.60	4.259	61.273	59.95	2.819	1,639.323	**69.25**	3.780	1,845.698	3,314.548
mesh25x25	250	56.50	4.213	99.082	67.45	3.138	2,807.358	**87.55**	3.775	3,165.167	5,338.265

The table layout is as described in Table 4, except for the fact that the results for the simple ACO algorithm are omitted here



(a) Example of the evolution of the quality of the current best solution S_{pbest} and the best-so-far solution S_{gbest} during the search (***left***), and the number of shared edges (2nd criterion) of the solution S_{pbest} (***right***). The behavior shown here corresponds to the application to one of the 20 instances composed by graph 4 and a list of 173 commodities. All the curves are smoothed with gnuplots' *sbezier* function.



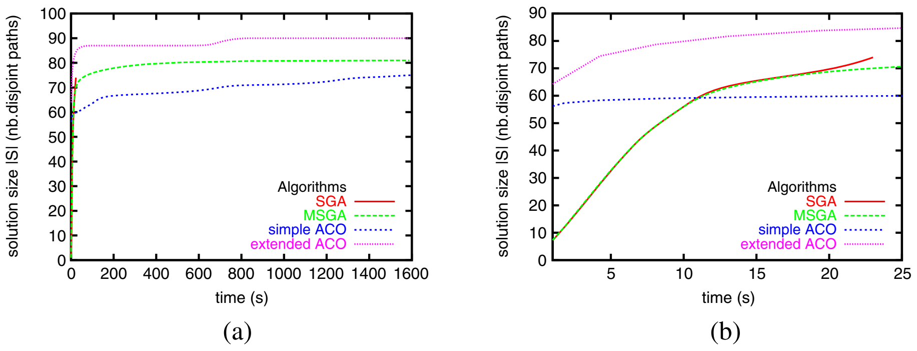
(b) Zoom on the 700 first (***left***) and the 700 last (***right***) iterations of Subfigure a. On the ***left***, the best solution found is quickly improved. At about iteration 250, the algorithm destroys part of the S_{pbest} solution, which produces an instantaneous worsening in the quality (***left***); another solution destruction takes places around iteration 550, which helps in achieving an improvement soon afterwards (***left***). Analogous effects can be observed around iteration 950 and 1250 (***right***). When contrasting with Subfigure a (***right***), we can observe that there exists an (inverted) relation between the number of edges shared and the quality of the solutions obtained. Thus validating our choice of the 2nd criterion as a part of the objective function.

Fig. 6 A representative example of the behavior of the extended 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

extended ACO approach needs in general lesser computation time than the MSGA, except for the mesh graphs. This advantage in computation time increases with increasing number of commodities. Exceptions are some of the results for small number of commodities, namely for 10% of the number of nodes of the graphs. For this combination MSGA has often slight advantages over the extended ACO approach. Therefore, we recommend to use a greedy approach when easy problem instances are concerned. However, the average results for instances with a higher number of commodities show a clear advantage of the extended ACO in contrast to MSGA, both in quality and time.

Mesh graphs deserve a special attention, since the behavior of the algorithms on them is different than on the graphs emulating Internet topologies. For mesh graphs, the extended ACO needs more time but in contrast the quality of the solutions that it obtains is often much better than the ones obtained by the MSGA. In particular this is the case when the number of commodities is high. Although mesh graphs are not representative of large communication networks like the Internet, it is a topology quite used in clusters dedicated to parallel computing.

Another interesting observation concerns the results of the extended ACO algorithm in comparison to the results of our preliminary ACO approach from [7], for which we noticed considerable difficulties when applied to instances derived from graph4. In [7] we conjectured that these difficulties result from the fact that graph4 has some nodes with a very high degree. This is because, first, these nodes are important connectivity points and are often in the shortest paths between two vertices of the graph, and second, because the higher the degree of a vertex, the lesser the probability of choosing a particular outgoing edge in the construction mechanism. More precisely, when constructing a path and being in one of these vertices, in order to choose a good outgoing edge the algorithm has possibly to be lucky, especially in early stages of the search process when the pheromone values are similar. We



(a) Example of the evolution in time of the quality of the solution S_{gbest}. The behavior shown here corresponds to the application to one of the 20 instances composed by graph 4 and a list of 173 commodities.

(b) Zoom on the first 25 seconds of Subfigure a. The time needed to obtain good solutions is clearly smaller for the ACO approaches. Note that in the first seconds the performance of the SGA and the MSGA are identical due to dealing with the same permutation of the commodities.

Fig. 7 A representative example of the run-time behavior of the algorithms presented in this work. All the curves are smoothed with gnuplots' *sbezier* function

can observe that our extended ACO algorithm does not show this behaviour. The additional features added to the simple ACO approach, in particular the usage of candidate lists, have helped on overcoming these difficulties concerning graph4. In general, the additional features of our extended ACO approach proved to be useful for the problem resolution. An example for the usefulness of the solution destruction mechanism is shown in Fig. 6.

Finally, an additional analysis concerns the run-time behaviour of our algorithms. Figure 7 shows that both ACO approaches find 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 (already good) quality of the solutions that are found after a short execution time might be often sufficient in practice.

6 Conclusions and Further Work

In this paper we have proposed an ant colony optimization (ACO) approach to tackle the maximum edge-disjoint paths problem. To the best of our knowledge this is – except for our own preliminary approach from [7] – the first application of ant colony optimization, and more in general of any any metaheuristic, to this version of the problem. Our approach is based on a decomposition of the maximum edge-disjoint paths problem into subproblems. 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, we introduced a basic ACO approach in order to be able to focus on important algorithmic features such as the solution construction and the pheromone update. Then, we proposed several extensions and additional features: concerning the solution construction process, we proposed (1) the parallel construction of all paths (in contrast to their sequential construction), and (2) the use of candidate list strategies for the exploitation of the promising choices at each construction step. Concerning the search dynamics, we proposed (1) the use of two search phases that aim at the improvement of the two different criteria of the objective function, and (2) the partial destruction of the currently best solution as an escape mechanism. We have shown that these features help on improving the performance of the algorithm without spending more computation time. Rather on the contrary, the extended ACO approach needs less computation time than the simple version in order to reach its best solutions. The results showed that our extended 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 also 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. This might prove beneficial for an online-version of our algorithm, in which speed is an issue. The fact that our algorithm only uses local information for building paths, is another advantage, because the computation of shortest path information is costly.

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 [26], length constraints can transform tractable disjoint paths problems into NP-hard variants (e.g., even the source-single sink case is NP-hard.) Furthermore, the relatively high degree of determinism used in our ACO approach might suggest ideas for new greedy approaches. Concerning nature-inspired metaheuristics, it would be of high interest to explore the potential advantages of having multiple ant colonies or particle swarms. A first attempt on using more than one ant colony was recently done in [36]. However, the aim in [36] was not to solve the EDP problem but rather to explore the feasibility of multi-colony systems. Therefore, only toy examples of graphs of up to 20 nodes with two or three commodities were considered there. One of the drawbacks of having multiple colonies is the increase of the computational requirements of the algorithm, which may not be affordable when time matters.

Further improvements should be tried also in order to tackle 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. To be tackled via ant colony optimization, this will require some changes to the ACO 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 nature of ACO makes it a promising candidate for it.

Acknowledgements The authors would like to thank Martin Oellrich from the Combinatorial Optimization & Graph Algorithms group of the Technische Universität Berlin for providing them with graph3 and graph4.

References

1. Aggarwal, A., Bar-Noy, A., Coppersmith, D., Ramaswami, R., Schieber, B., Sudan, M.: Efficient routing and scheduling algorithms for optical networks. In: 5th. ACM-SIAM Symposium on Discrete Algorithms (SODA'94), pp. 412–423. SIAM (1994)
2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. Journal of the ACM **45**(3), 501–555 (1998)
3. Aumann, Y., Rabani, Y.: Improved bounds for all-optical routing. In: 6th. ACM-SIAM Symposium on Discrete Algorithms (SODA'95), pp. 567–576. SIAM, Philadelphia, PA (1995)
4. Awerbuch, B., Gawlick, R., Leighton, F., Rabani, Y.: On-line admission control and circuit routing for high performance computing and communication. In: 35th IEEE Symposium on Foundations of Computer Science (FOCS'94), pp. 412–423. IEEE Computer Society Press, Los Alamitos, CA (1994)
5. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)
6. Baveja, A., Srinivasan, A.: Approximation algorithms for disjoint paths and related routing and packing problems. Math. Oper. Res. **25**(2), 255–280 (2000)
7. Blesa, M., Blum, C.: Ant colony optimization for the maximum edge-disjoint paths problem. In: 1st European Workshop on Evolutionary Computation in Communications, Networks, and Connected Systems (EvoCOMNET'04). Lecture Notes in Computer Science, vol. 3005, pp. 160–169. Springer, Berlin Heidelberg New York (2004)
8. Blesa, M., Blum, C.: Finding edge-disjoint paths with artificial ant colonies. Technical Report LSI-05-13-R, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (2005)
9. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Trans. Syst. Man Cybern., Part B, Cybern. **34**(2), 1161–1172 (2004)
10. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)

11. Brandes, U., Wagner, D.: A linear time algorithm for the arc disjoint menger problem in planar directed graphs. Algorithmica **28**(1), 16–36 (2000)
12. Chekuri, C., Khanna, S.: Edge disjoint paths revisited. In: 14th annual ACM-SIAM Symposium on Discrete algorithms (SODA'03), pp. 628–637. SIAM, Philadelphia, PA (2003)
13. Costa, M.-C., Létocart, L., Roupin, F.: Multicut and integral multiflow: a survey. Eur. J. Oper. Res. **162**(1), 55–69 (2005)
14. Di Caro, G., Dorigo, M.: AntNet: distributed stigmergetic control for communications networks. J. Artif. Intell. Res. **9**, 317–365 (1998)
15. Di Caro, G., Ducatelle, F., Gambardella, L.M.: AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. Eur. Trans. Telecommun. **16**(5), 443–455 (2005)
16. Dorigo, M.: Ottimizzazione, Apprendimento Automatico, ed Algoritmi basati su Metafora Naturale. PhD thesis, DEI, Politecnico di Milano, Milan, Italy (1992)
17. Dorigo, M., Gambardella, L.: Ant Colony System: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
18. Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: Optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cyber., Part B, Cybern. **26**(1), 29–41 (1996)
19. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA (2004)
20. Erlebach, T.: Approximation algorithms and complexity results for path problems in trees of rings. In: 26th International Symposium on Mathematical Foundations of Computer Science (mfcs'01). Lecture Notes in Computer Science, vol. 2136, pp. 351–362. Springer, Berlin Heidelberg New York (2001)
21. Gambardella, L., Dorigo, M.: Solving symmetric and asymmetric TSPs by ant colonies. In: IEEE International Conference on Evolutionary Computation (icec'96), pp. 622–627. IEEE Computer Society Press, Los Alamitos, CA (1996)
22. Garg, N., Vazirani, V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica **18**(1), 3–20 (1997)
23. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J. Comput. Syst. Sci. **67**b(3), 473–496 (2003)
24. Hromkovič, J., Klasing, R., Stöhr, E., Wagener, H.: Gossiping in vertex-disjoing paths mode in d-dimensional grids and planar graphs. In: 1st Annual European Symposium on Algorithms (ESA'93). Lecture Notes in Computer Science, vol. 726, pp. 200–211. Springer, Berlin Heidelberg New York (1993)
25. Karp, R.: Compexity of computer computations, Chapter Reducibility Among Combinatorial Problems, pp. 85–103. Plenum, New York (1972)
26. Kleinberg, J.: Approximation algorithms for disjoint paths problems. PhD thesis, MIT Press, Cambridge, USA (1996)
27. Kolliopoulos, S., Stein, C.: Approximating disjoint-path problems using packing integer programs. Math. Program. **99**(1), 63–87 (2004)
28. Kolman, P., Scheideler, C.: Simple on-line algorithms for the maximum disjoint paths problem. In: 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA'01), pp. 38–47. ACM Press, New York (2001)
29. Kolman, P., Scheideler, C.: Improved bounds for the unsplittable flow problem. In: 13th annual ACM-SIAM symposium on Discrete algorithms (SODA'02), pp. 184–193. SIAM, Philadelphia, PA (2002)
30. Kramer, M., van Leeuwen, J.: Advances in computing research. In: VLSI theory. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits, vol. 2, pp. 129–146. JAI Press, Greenwhich, CT (1984)
31. Ma, B., Wang, L.: On the inapproximability of disjoint paths and minimum steiner forest with bandwidth constraints. J. Comput. Syst. Sci. **60**(1), 1–12 (2000)
32. Marx, D.: Eulerian disjoint paths problem in grid graphs is NP-complete. Discrete Appl. Math. **143**(1–3), 336–341 (2004)
33. Medina, A., Lakhina, A., Matta, I., Byers, J.: BRITE: Boston University Representative Internet Topoloy Generator. http://cs-pub.bu.edu/brite/index.htm (2001)
34. Middendorf, M., Pfeiffer, F.: On the complexity of the disjoint path problem. Combinatorica **13**(1), 97–107 (1993)
35. Nishizeki, T., Vygen, J., Zhou, X.: The edge-disjoint paths problem is np-complete for series-parallel graphs. Discrete Appl. Math. **115**(1–3), 177–186 (2001)
36. Nowé, A., Verbeeck, K., Vrancx, P.: Multi-type ant colony: the edge disjoint paths problem. In: 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS'04).

Lecture Notes in Computer Science, vol. 3172, pp. 202–213. Springer, Berlin Heidelberg New York (2004)
37. Raghavan, P., Upfal, E.: Efficient all-optical routing. In: 26th. Annual ACM Symposium on Theory of Computing (stoc'94), pp. 134–143. ACM Press, New York (1994)
38. Ripphausen-Lipa, H., Wagner, D., Weihe, K.: The vertex-disjoint Menger problem in planar graphs. SIAM J. Comput. **26**(2), 331–349 (1997)
39. Sidhu, D., Nair, R., Abdallah, S.: Finding disjoint paths in networks. ACM SIGCOMM Computer Communication Review **21**(4), 43–51 (1991)
40. Srinivasan, A.: Improved approximations for edge-disjoint paths, unsplittable flow and related routing problems. In: 38th Annual IEEE Symposium on Foundations of Computer Science (focs'97), pp. 416–425. IEEE Computer Society Press, Los Alamitos, CA (1997)
41. Stützle, T., Hoos, H.: $\mathcal{MAX}$-$\mathcal{MIN}$ ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
42. Varadarajan, K., Venkataraman, G.: Graph decomposition and a greedy algorithm for edge-disjoint paths. In: 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04), pp. 379–380. SIAM, Philadelphia, PA (2004)
43. Vygen, J.: NP-completeness of some edge-disjoint paths problems. Discrete Appl. Math. **61**(1), 83–90 (1995)
44. Waxman, B.: Routing of multipoint connections. IEEE J. Sel. Areas Commun. **6**(9), 1671–1622 (1988)
45. Weihe, K.: Edge-disjoint $(s - t)$-paths in undirected planar graphs in linear time. J. Algorithms **23**(1), 121–138 (1997)