Query Routing with Ants*

Elke Michlmayr[1], Sabine Graf[1], Wolf Siberski[2], and Wolfgang Nejdl[2]

[1] Women's Postgraduate College for Internet Technologies (WIT),
Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
{lastname}@wit.tuwien.ac.at

[2] Learning Lab Lower Saxony (L3S), Hannover, Germany
{lastname}@l3s.de

Abstract. In this paper we propose *SemAnt*, a novel ant-based algorithm designed for query routing in taxonomy-based peer-to-peer environments. We introduce the reader to the pheronome trail-laying-and-following behaviour observed from natural ants and show how it can be applied to query routing in peer-to-peer networks. Our proposed algorithm accounts for network parameters such as bandwidth and latency and optimizes the pheromone trails to results for a given query depending on the query's popularity. If a query is common, its pheromone trails will converge and lead to the nodes that offer the most results for the given query. Pheromone trails that are used rarely will evaporate over time. In addition, the proposed algorithm accounts for the inherent dynamics of peer-to-peer networks by adjusting pheromone trails when peers join or leave the network.

1 Introduction

Different approaches for query routing in peer-to-peer networks exist, ranging from simple broadcasting techniques to sophisticated methods ([7],[3],[15]) that store information about user-generated queries in the past for predicting which peer is capable of answering it based on a given query's keywords, and to route the query accordingly. The aim is to maximize the number and the quality of query results while minimizing the overhead necessary for management of routing tables. Since no central authorities exist in peer-to-peer networks, these methods must rely on each peer's local knowledge only.

Ant-based trail-discovery and -following algorithms have successfully been applied to diverse problems in distributed systems. The Ant Colony Optimization (ACO, [8]) meta-heuristic proposed by Di Caro and Dorigo in 1999 exploits these techniques for solving graph-based optimization problems. A dedicated subset of ant-based algorithms – subsumed under the term Ant Colony Routing (ACR, [4]) – was specifically designed for managing routing tables in

* This research has partly been funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

IP networks. We assume ant-based techniques to be suitable for query routing in peer-to-peer networks for the following reasons:

Decentralization: Ants are autonomous agents that travel through a network and spread pheromones on their paths. At each node, they examine the pheromone amounts already spread on the available outgoing links in order to decide which link to follow. Communication among ants is indirect and exclusively based on modifications of pheronomon trails. Hence, ant-based methods do not require any global knowledge. They need local knowledge only, which makes them suitable for application in distributed environments.

Dynamic behavior: Since strategies for reacting to network changes exist, ACO algorithms are suitable for peer-to-peer networks, which characteristically allow peers to leave or join at any time and where the routing algorithm has to cope with these dynamics. ACO includes methods that adapt the pheromone trails in the network according to these changes.

However, the optimization problems previously solved with ACO differ in some respects from the task of query routing in peer-to-peer networks, which can be seen as a special kind of optimization problem. Moreover, ACR has proven to perform well [5] when routing data packets in packet-switched networks but has not yet been applied to the task of query routing. One basic difference between traditional routing and query routing is that the former does not account for the contents of data packets. Neither ACO nor ACR are applicable *as-is* in peer-to-peer networks, but must be adapted according to the environment and the application scenario.

The paper is organized as follows. In Section 2 we describe Ant Colony Optimization and Ant Colony Routing. We discuss the main building blocks of both ACO and ACR and identify those that can be transferred to the realm of peer-to-peer networks as well as those that can not. In Section 3 we discuss previous research on employing these techniques in peer-to-peer networks. After this discussion which provides its design rationale, in Section 4 we describe *SemAnt*, our adapted ant-based algorithm. Finally, in Section 5, we sum up our findings.

2 Ant-based algorithms

In this section, we provide an introduction to Ant Colony Optimization and Ant Colony Routing.

2.1 Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms [8] are inspired by the collective foraging behavior of specific ant species. When these species of ants are searching for food sources they follow a trail-laying trail-following behavior. Trail-laying means that each ant drops a chemical substance called pheromone on its chosen path. Trail-following means that each ant senses its environment for existing pheronome trails and their strength. This information builds the basis for their decision which path to follow. If there is a high amount of pheromone on a path the probability that the ant will choose it is also high. If there is a low amount of pheromone, the probability is low. The more often a path is chosen, the more pheromones are laid on it which increases the probability that it will be chosen again. Since the decision is based on probabilities, an ant does not always follow the way that has the highest pheromone concentration. Paths which are marked as poor are also chosen, but with lower probability. One of the features of pheromones is that they evaporate over time, leading to the effect that rarely used trails will vanish. These strategies enable ants to build a map of pheromone trails which indicates the best paths to a food source.

Summarized by a meta-heuristic called *Ant Colony Optimization* [8], several algorithms exist that model and exploit this behavior for solving graph-based NP-hard combinatorial optimization problems, e.g., the travelling salesman problem. In these algorithms, after initializing each edge of the problem graph with a very small amount of pheromone and defining each ant's starting node, a small number of ants (e.g., 10) runs for a large number of iterations (e.g., 10.000). For every iteration, each ant determines a path through the graph from its starting to the destination node. It does this by applying a so-called *random proportional transition rule* at each decision point. This rule derives which of all possible next nodes to choose, based on (1) the specific edge's amount of pheromone and (2) its costs. When the ant arrives at the destination node, the total costs of the newly found solution are calculated. In case the newly found solution outperforms the existing solutions, it is saved to memory as the currently best one. After all ants have found a solution, the *pheromone trail update rule* is applied for each edge which is part of the solution. The amount of newly dropped pheromones depends mostly on the quality of the solution. In each iteration, some pheromone evaporates according to a evaporation factor. The different instances of ACO – i.e, *Ant system* [6], *Ant colony system* [9], or *MAX-MIN ant system* [14] – vary mostly in their transition rules and their pheromone trail update rules, but also some other additional constraints or small add-ons.

One specific strength of ACO algorithms is their adaptability to different kinds of problems. For our purposes, we focus on their application to dynamic and to distributed problems. A problem is called dynamic when the problem graph changes during the solution finding process. Guntsch and Middendorf [10] elaborated three strategies for handling dynamic problems with ACO. The main research on distributed problems was performed in the area of routing in telecommunication networks, as presented in the next section.

2.2 Ant Colony Routing

The two most prominent variants of ant-based algorithms for routing are *ant based control (ABC)* [12] by Schoonderwoerd *et al.* and *AntNet* [5] by Di Caro and Dorigo. ABC is designed for circuit-switched networks and its pheromone updating approach is appropriate for symmetric networks only, whereas AntNet is designed for packet-switched networks and its pheromone updating approach is appropriate for both symmetric and asymmetric networks. Since we deal with a packet-switched network in our application scenario, we focus on AntNet.

In AntNet, ants collaborate in building routing tables that adapt to current traffic in the network with the aim of optimizing the performance of the entire network. The network is mapped on a directed weighted graph with N nodes. Each node manages a routing table. The edges of the graph are the links between nodes and are viewed as bit pipes having a certain cost (bandwidth and transmission delay) that depends on the current load of this link. The routing tables are matrices of size $N \times l$, where l is the number of outgoing links. At startup, all routing tables are initialized with a uniform distribution of all reachable nodes. At regular intervals, each node generates a so-called forward ant that builds a path to a randomly selected destination node. Similar to ACO, the decision about which node to choose next is based on (1) link cost and on (2) the amount of pheromones already dropped on this link by ants in the previous iterations. When a forward ant F_{sd} launched at a source node N_s has reached its randomly selected destination node N_d and is now able to calculate the total costs of the solution, it cannot update the pheromone trails directly like in ACO. Since the problem is distributed, it has to generate a backward ant B_{ds} that will return to N_s through the same path that was used by F_{sd}. The backward ant is responsible for updating the pheronome trail according to the information gathered by F_{sd} by altering the routing table of each visited node. The backward ants update all entries corresponding to destination node d.

AntNet includes a strategy for preventing cycles. F_{sd} manages a stack of peers already visited. Each time it has to decide which peer to visit next, it excludes all already visited peers. If F_{sd} detects a cycle because it is forced to go to an already visited peer P_v since all possible next peers were already visited, it calculates the time span t it spent inside the circle. If t is greater than 50% of the ant's total lifetime, F_{sd} is terminated. If it is less, F_{sd} removes all peers that are part of the cycle from its stack and continues travelling at P_v.

ACO's evaporation feature, as described in the last section, is not incorporated into AntNet. One shortcoming of the algorithm is that it does not provide support for link failures, node failures, or new nodes that join the network. The evaluation presented in [5] shows that AntNet outperforms all state-of-the-art routing algorithms existing at the time the paper was published. The two main reasons that make it impossible to apply AntNet to peer-to-peer environments without any modifications are:

- AntNet needs information about all nodes that exist in the network in order to choose destination nodes. This information is not available in peer-to-peer environments.
- When routing data packets, the destination IP address of the packets is known beforehand. Query routing, on the contrary, is the task of finding one or more appropriate destinations for a given query.

In the following we discuss related work on ant-based algorithms in peer-to-peer environments that was evaluated as a starting point for the design of SemAnt.

3 Related work

Anthill [2] is a Java-based open source framework for the design, implementation, and evaluation of ant algorithms in peer-to-peer networks. An Anthill system is an overlay network of interconnected nests (peers). Nests provide services like document storage, routing table management, topology management, and generation of ants upon user requests. In addition, the Anthill API provides a basic set of actions for ants that enables them to travel from nest to nest, and to interact with the services provided by nests. The ant-based algorithm is not specified by Anthill, but must be designed by the user of the framework according to the application scenario. Hence, the main difference between Anthill and SemAnt is that Anthill aims to deliver a test bed for ant algorithms in peer-to-peer networks, but does not focus on the design of the algorithms themselves.

One part of the Anthill project that is similar to SemAnt is *Gnutant* [2], a peer-to-peer file-sharing application that was built by the Anthill developers for demonstration and evaluation purposes. In this application, each file is identified by an unique file identifier and associated with meta-data comprised of textual keywords. Three different types of ants are responsible (1) for constructing a distributed index that contains URLs pointing to shared files and (2) for managing routing tables. If a user adds a new file to a nest, one *InsertAnt* is generated for each keyword of the file. InsertAnts propagate the presence of new files to the network by updating the distributed index. Gnutant utilizes hashed keyword routing based on the Secure Hash Algorithm (SHA). Each index entry contains the hash value of a keyword together with a set of nests that are likely to store files associated with the given keyword. *SearchAnts* are generated upon user queries and exploit the information stored in the routing tables in order to find files that match the queries' keywords. If no index entry that exactly matches the query's hash value exists, the SearchAnt selects the hash value that most closely matches the hash value of the query. If a SearchAnt localizes an appropriate file, it generates a *ReplyAnt* that immediately returns to the source nest and informs the user about the result of his or her query. The SearchAnt itself continues travelling until it reaches a defined time-to-live parameter TTL, that is, the maximum number of hops the ant is allowed to move. Once the TTL is reached, it returns to its source nest via the same path it travelled before and updates both routing tables and distributed index. This behaviour is derived from AntNet.

Schelthout et al. [11] evaluate whether the principle of synthetic pheromones can be employed for coordination in distributed agent-oriented environments. Their framework is different from ours since it is based on the idea of objectspaces known from concurrent computing, but for its evaluation a simple peer-to-peer filesharing application was created. Similar to Anthill, Schelthout et al. create pheronome trails for each query and allow agents to follow trails that represent only one out of the multiple keywords in the query. In addition, they use an evaporation factor of 0.1. The test environment is too small (200 peers, one kind of resource) to be useful, but the experimental results show that the hit rate increases with the number of queries sent to the network (32% hit rate after 400 queries, 53% hit rate after 700 queries). This fact proves that pheromone trails enhance the performance of the network.

4 SemAnt

In this section, we introduce our application scenario. After that, our proposed ant-based algorithm *SemAnt* is described in detail.

4.1 Application scenario

Our application scenario is that of a distributed search engine where each peer manages a repository of documents and offers the content of its repository to other peers. All available documents are classified by content according to a taxonomy or hierarchical classification scheme, such as the ACM Computing Classification System (ACM CSS) [1]. Each document is associated with one or more concepts of the taxonomy. Each peer owns a copy of the taxonomy. In addition to offering documents, peers pose queries to the network. Each query consists of one or more keywords. The set of allowed keywords is limited to the concepts of the chosen taxonomy. Multiple keywords are connected using the boolean operator AND. Hence, a document is an appropriate result for a given query Q if it is classified to be an instance of all concepts that are keywords of the query Q. Peers are identified by their IP addresses. Documents are identified by their filenames. Since we do not focus on the problem of peer discovery, our assumption is that peers already know which neighbouring peers they are able to connect to. In addition, we make the assumption that the clocks of all peers are synchronized and set to the same time.

4.2 The proposed algorithm

First of all, we must define which entity of the system should be represented by ants. For each query, the shortest path through the network must be found that leads from the querying peer to a destination peer offering documents matching this query. The more often a query is requested, the stronger its paths should be optimized. Thus, it is reasonable to represent queries as ants. This guarantees that the degree of optimization for a certain query directly depends on its popularity.

As described in Section 2.1, foraging ants find the shortest path to *one* kind of food. In our scenario, all concepts of the taxonomy are allowed as keywords in queries. Hence, there is more than one kind of "food". Consequently, the network must be independently optimized for all possible keywords, that is, each concept. This goal can only be reached by employing multiple pheromone types as introduced in [13] and representing each concept by a corresponding type of pheromone. At each peer P_i, pheronome trails are maintained in a table τ of size $C \times n$, where C is the number of concepts in the taxonomy and n is the number of P_i's outgoing links to neighbouring peers P_u where $u \in \{1, ..., n\}$.

```
t = CurrentTime;
t_end = EndTimeOfSimulation;
foreach (Peer) {                    # Concurrent activity
   initializePheromoneTables();
   initializeLinkCostsToNeighbourPeers();
   while (t <= t_end) {
      in_parallel {                 # Concurrent activity at each peer
         if (Query Q) {
            checkLocalDocumentRepository();
            createForwardAnt(query_parameter);
         }
         foreach (StartingForwardAnt) {
            while (Timeout_not_reached) {
               applyTransitionRule();
               t_a = CurrentTime;
               foreach (ActiveForwardAnt, ClonedForwardAnt) {
                  GoToPeer(P_j);
                  t_b = CurrentTime;
                  checkLocalDocumentRepository();
                  if (DocumentsFound > 0) {
                     createBackwardAnt(stackData, P_j);
                  }
                  addDataToStack(P_j, t_b-t_a);
               }
            }
         }
         foreach (StartingBackwardAnt) {
            do {
               peer = popStackData_Peer();
               GoToPeer(peer);
               applyPheromoneTrailUpdateRule();
               updateListCosts(popStackData_Costs());
            } while (peer <> source_peer);
         foreach (PeriodicalTimeInterval t_e) {
            applyEvaporationRule();
         }
      }
   }
}
```

Fig. 1. SemAnt

Each τ_{cu} stores the amount of pheronome type c dropped at the link from peer P_i to peer P_u, for each concept c and each neighbouring peer P_u. At startup, all table entries are initialized with the same small value τ_{init}. In addition, each peer manages a table η that stores link costs to neighbouring peers P_u. Each η_u is the inverse value $\frac{1}{lc_u}$ of the cost lc_u for sending an ant from P_i to P_u.

Figure 1 shows the proposed algorihm in pseudo-code. *SemAnt* adopts the AntNet strategy for supporting distributed problems by forward ants F and backward ants B as well as AntNet's strategy for preventing cycles. In AntNet, forward ants terminate their travel through the network when they reach their well-defined destination node. This behaviour can not be transferred to *SemAnt*, since in our scenario the forward ant's task is to find an unknown destination node that in the worst case does not even exist. To specify the point of time at which a forward ant terminates its travel, we define a timeout parameter T_{max} that is the maximum lifetime of F.

The algorithm's routing strategy is exclusively defined in the transition rule. We cannot rely on AntNet's transition rule since it aims to optimize the path to one specific destination node only. Instead, we employ *Ant Colony System*'s transition rule [9] that consists of two strategies. Based on probability w_e, each ant decides whether it applies an exploiting or an exploring strategy:

– In the *exploiting strategy*, the ants determine the quality of the links depending on the amounts of pheromones and the link costs and then always select the link with the highest quality.
– The *exploring strategy* encourages ants to discover new paths. This is achieved by deriving a probabilistic value p_u for each peer P_u in the set of neighbouring peers U and not in the set of peers $S(F)$ which were already visited by F. The *roulette wheel selection* technique is applied for selecting the peer. All p_us are placed on the continuum between 0 and 1 and the sum $\sum_{u \in U \wedge u \notin S(F)} p_u$ is 1. In the next step, a random value q where $q \in [0,1]$ is calculated in order to decide which peer P_u to select.

To facilitate searching in multiple directions, *SemAnt* utilizes an adapted exploring strategy. This adaptation accounts for the fact that there are multiple possible destination peers in our application scenario. After computing p_u for each peer P_u, instead of using roulette wheel selection we *separately* place each p_u on the continuum between 0 and 1 and calculate q to decide whether P_u is selected. This strategy allows more than one peer to be selected. To ensure that at least one peer will be selected, we fall back to the exploiting strategy in the rare cases that the exploring strategy decides not to select any peer.

A step-by-step description of our algorithm follows. Consider a query Q containing a keyword c issued at a peer P^Q. The following seven steps are necessary for answering query Q:

Step 1 Check P^Q's local document repository. If any results are found, present them to the user. If the number of documents found is less than D_{max}, go to step 2. Otherwise, enough documents are found and the algorithm terminates.

Step 2 At P^Q, create a forward ant F^Q with starting time T_{Fstart} and timeout T_{max} responsible for retrieving results for query Q. Add P^Q to F^Q's empty stack of already visited peers $S(F^Q)$ and initialize the list $LC(F^Q)$ that stores the link costs of all links used by F^Q.

Step 3 Apply the transition rule in order to decide which outgoing link(s) F^Q should choose. As described above, F^Q applies the exploiting strategy with probability w_e or the exploring strategy with probability $1 - w_e$.
In case the *exploiting strategy* is used, F^Q applies the following transition rule to select the present best neighbouring peer P_j:

$$j = arg\ max_{u \in U \wedge u \notin S(F^Q)} \left([\tau_{cu}] \cdot [\eta_{cu}]^{\beta} \right) \ ,$$

where β is a constant weighting the influence of link costs, U is the set of neighbouring peers of P_i, and $S(F^Q)$ is the set of peers F^Q already visited.

If F^Q uses the *exploring strategy*, the following transition rule is applied *for each* neighbouring peer P_j in order to decide whether F^Q should be routed to P_j:

$$p_j = \frac{[\tau_{cj}] \cdot [\eta_{cj}]^\beta}{\sum_{u \in U \wedge u \notin S(F^Q)} \left([\tau_{cu}] \cdot [\eta_u]^\beta\right)} \ , \ GO_j = \begin{cases} 1 \text{ if } q \leq p_j \wedge j \in U \wedge j \notin S(F^Q) \\ 0 \text{ else} \end{cases}$$

where q is a random value, $q \in [0, 1]$, and $\sum_{j \in U \wedge j \notin S(F^Q)} p_j = 1$. If $GO_j = 1$, F^Q creates a clone of itself and sends it to P_j.

Step 4 Upon arrival at P_j, check the local document repository for documents d that are results for query Q. All matching documents are contained in the set D.

Step 5 If there are any documents d, where $d \in D$ and $D \neq \{\}$, generate a backward ant B^Q and pass it D and P^D, which is the peer that stores D. In addition, B^Q is given a copy of F^Q's stack data which contains all visited peers $S(F^Q)$ and all recorded link costs $LC(F^Q)$. In the first step, B^Q calculates the sum of all entries in $LC(F^Q)$ to get the total link costs T_D for the path from P^Q to P^D. After that, B^Q travels back hop-by-hop according to the information stored in $S(F^Q)$ until it arrives at querying peer P^Q. At each intermediate peer, B^Q is responsible for (1) updating the link cost η_j to the entry in $LC(F^Q)$ and (2) dropping pheromones by applying the pheromone trail update rule. Our pheromone trail update rule is adopted from [9] and defined as follows:

$$\tau_{cj} \leftarrow \tau_{cj} + Z, \text{ where } Z = w_d \cdot \frac{|D|}{d^*} + (1 - w_d) \cdot \frac{T_{max}}{2 \cdot T_D}$$

The amount Z depends on the number of documents found and the total link costs. w_d weights the influence of documents' quantities and link costs. Z is derived by comparing the goodness of the found solution to an optimal one. Since there is no optimal solution, we set the optimal value for a path's total link costs to $\frac{T_{max}}{2}$ and use a constant d^* for the optimal number of documents. Since both values merely act for assessing the quality of the found solution and each solution is compared to the same values, we can safely rely on estimates. Additionally, 50% of Z are dropped on the pheromone trail corresponding to the superconcept of c, one level higher in the taxonomy. This way, the algorithm exploits the information from the taxonomy by reflecting the hierarchical structure of concepts in the pheromone trails.

Step 6 Add P_i to F^Q's stack of already visited peers $S(F^Q)$ and add the cost of the last used link to $LC(F^Q)$.

Step 7 If $T_{Fstart} + T_{max} < \mathit{CurrentTime}$, continue at step 3. Otherwise, if F^Q's maximum lifetime is over, kill it.

The algorithm as described above acts on the assumption that the query consists of only one keyword c, but *SemAnt* also provides for queries containing multiple keywords $c_1, \ldots, c_n$. In that case, all corresponding pheromone trails must be incorporated. In the transition rules, the average amount of pheromones of all keywords $(\tau_{c_1u} + \ldots + \tau_{c_nu}) \cdot \frac{1}{n}$ instead of τ_{cu} is used for all neighbouring peers P_u.

Additionally, the pheromone trail update rule is applied not only to to τ_{cu} but to τ_{c_1u}, ..., τ_{c_nu} and to all trails for the superconcepts of c_1, ..., c_n.

Please note that unlike in AntNet, a single forward ant can generate multiple backward ants. Since the maximum lifetime of F^Q is T_{max}, all B^Qs will arrive within a time interval of $2 * T_{max}$. As soon as a backward ant arrives at the querying peer P^Q, the result documents D are presented to the user. If the user decides to download a document d, a direct connection between P^Q and the peer P^D that stores d is established and the document is retrieved from P^D.

Each peer applies the following evaporation rule in a predefined interval t_e for each link to neighbouring peer P_u and each concept c, where the amount of evaporating pheromones is controlled by parameter $\rho \in [0, 1]$:

$$\tau_{cu} \leftarrow (1 - \rho) \cdot \tau_{cu}$$

If new peers are joining the network, if peers are leaving, or if a peer adds or removes documents from the repository, it is necessary to adapt the pheromone trails in order to reflect these changes. To correct the amount of pheromones we apply the η-strategy elaborated by Gutsch and Middendorf [10]. According to this strategy, the closer a link is to a joining or leaving node, the more pheromones are removed from it. This helps the ants to find new paths by increasing the influence of new links. In [10], the measurement of closeness is based on the link costs. *SemAnt* uses a simplified version of the η-strategy where closeness is measured in number of hops.

We define a modification rule for calculating the necessary modifications of τ_{cu} which considers the number of added or removed documents $|D_c|$ that are instances of concept c:

$$\tau_{cu} \leftarrow \begin{cases} \tau_{cu} - \tau_{cu} \cdot \lambda_h & \text{if } |D_c| \geq d^* \\ \tau_{cu} - \tau_{cu} \cdot \lambda_h \cdot \frac{|D_c|}{d*} & \text{else} \end{cases} ,$$

where λ is a list of predefined values $\lambda_h \in [0, 1]$ specifying the decrease of τ_{cu}, $h = \{1, 2, 3\}$ represents the number of hops from P_u to the peer where the changes of τ_{cu} occur, and $\lambda_1 < \lambda_2 < \lambda_3$. The constant d^* represents the optimal number of documents and is used as a point of reference that restricts the maximum decrease of pheromones. For the adaptation of trails using the above formula, we have to employ a broadcast strategy. Each message contains an unique key, and all peers maintain a list of already processed messages to prevent processing a message twice.

If a peer P_x leaves in a controlled way, it creates a list C_x that contains the total number of documents $|D_c|$ that are instances of c for each concept c. A message M_{leave} containing the list C_x and $h = 1$ is sent to all neighbouring peers P_u. Upon receipt, every P_u removes all entries τ_{cx} where $0 \leq c < C$ and C is the number of concepts in the taxonomy. After that, it applies the modification rule. In the next step, P_u increases h by one and broadcasts M_{leave} to all its neighbouring peers. These peers apply the modification rule and again (1) increase h by one and (2) send M_{leave} to all their neighbouring peers which are three hops away from P_x. Each receiving peer applies the modification rule.

If a peer fails, it leaves without the possibility of adapting pheromone trails. In this case, we have to rely on evaporation and on the exploring strategy.

If a new peer P_y joins the network, all its table entries τ_{cu} are initialized with τ_{init}. To modify the pheromone amounts of its neighbourhood, a list C_y as described above is created by P_y. In the next step, a message M_{join} containing C_y and $h = 1$ is sent to all neighbouring peers P_u. Each P_u applies the modification rule and adds a new row τ_{cy} where $0 \leq c < C$ to its pheromone trail table τ, and initializes the new pheromone trails with τ_{init}. Finally, the peer increases h by one and sends M_{join} to all its neighbouring peers, which in turn apply the modification rule.

If documents are added to or removed from the repository of a peer P_z, it creates a list C_z as described above and sends a message M_{change} to its neighbouring peers, which in turn apply the modification rule.

4.3 Simulation Environment

To prove that our ideas are feasible, *SemAnt* must be tested using simulation. We planned to build our simulation environment on the Anthill [2] open source framework described in Section 3, but the initial test runs performed with the Gnutant algorithms which are part of Anthill did not produce the expected results. We used the ACM CCS Classification Scheme [1] containing 369 concepts for our tests. Our simulations showed that the Anthill framework is not scalable because Out-of-Memory errors occured after approximately 2000 iterations. These shortcomings lead to the decision to build our own environment. We are currently working on it, but have not finished it yet. Our simulation environment is configurable for number of peers, network topology, network parameters (bandwidth, latency) for each link, peers that join during the simulation, peers that leave during the simulation, documents added or removed during the simulation, and for all configurable parameters of the algorithm. As soon as our prototype is finished, *SemAnt* will be tested in networks with different topologies to determine the one in which it performs best. In addition, appropriate values for the configurable parameters of the algorithm must be figured out. We expect that the optimal parameter values are different for each topology.

5 Conclusion

This paper presents work in progress on our efforts to utilize ant-based algorithms in taxonomy-based peer-to-peer environments. We undertook an exhaustive study of the constituents of ant-based algorithms to identify the ones that are best qualified for deployment in peer-to-peer environments. Those selected were then customized to our application scenario and used in the design of our proposed algorithm *SemAnt*. Our approach exploits the knowledge provided by the underlying taxonomy by reflecting it in the pheromone tables. Hence, it is taken into consideration for all routing decisions.

Acknowledgements

We thank Veronika Stefanov and Alexander Löser for fruitful discussions and Christoph Tempich for providing us with a simulation environment.

References

1. Association for Computing Machinery. ACM Computing Classification System (ACM CCS). `http://www.acm.org/class/1998/`, 1998.
2. O. Babaoglu, H. Meling, and A. Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 02)*. IEEE, July 2002.
3. W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, April 2005.
4. E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for Optimization from Social Insect Behaviour. *Nature*, 406:39–42, 2000.
5. G. D. Caro and M. Dorigo. AntNet: Distributed Stigmergy Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
6. A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In *Proceedings of the 1st European Conference on Artificial Life*, pages 134–142. MIT Press, 1991.
7. B. F. Cooper. Guiding Queries to Information Sources with InfoBeacons. In *Middleware 2004, ACM/IFIP/USENIX International Middleware Conference*, volume 3231 of *Lecture Notes in Computer Science*, pages 59–78. Springer, October 2004.
8. M. Dorigo and G. D. Caro. *New Ideas in Optimization*, chapter The Ant Colony Optimization Meta-Heuristic, pages 11–32. McGraw-Hill, 1999.
9. M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.
10. M. Guntsch and M. Middendorf. Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP. In *Applications of Evolutionary Computing*, Lecture Notes in Computer Science, pages 213–222. Springer, 2001.
11. K. Schelfthout and T. Holvoet. A Pheromone-Based Coordination Mechanism Applied in Peer-to-Peer. In *Agents and Peer-to-Peer Computing, Second International Workshop (AP2PC 2003)*, volume 2872 of *Lecture Notes in Computer Science*, pages 71–76. Springer, July 2003.
12. R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based Load Balancing in Telecommunications Networks. *Journal an Adaptive Behavior*, 5:169–207, 1996.
13. K. M. Sim and W. H. Sun. Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 33(5):560–572, 2003.
14. T. Stützle and H. Hoos. Improvements on the Ant-System: Introducing the MAX-MIN Ant System. In *Proceedings of the International Conference of Artificial Neural Networks and Genetic Algorithms*, pages 245–249. Springer, 1997.
15. C. Tempich, S. Staab, and A. Wranik. REMINDIN': Semantic Query Routing in Peer-to-Peer Networks based on Social Metaphors. In *Proceedings of the 13nd International World Wide Web Conference (WWW2004)*, May 2004.