

TECHNICAL REPORT

Specification of the SemAnt Algorithm*

Elke Michlmayr

michlmayr@wit.tuwien.ac.at

Women's Postgraduate College for Internet Technologies (WIT)

Institute for Software Technology and Interactive Systems

Vienna University of Technology

Austria

March 2006

*This research has 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.

Chapter 1

Specification of the SemAnt Algorithm

This chapter covers the specification of the proposed ant algorithm SEMANT for metadata-based search in peer-to-peer networks. After defining the problem and stating the assumptions which were made, what follows is a detailed description of all aspects related to the SEMANT algorithm. The remainder of the chapter is devoted the evaluation of algorithm's performance in various scenarios and settings, and to a discussion of related work on ant algorithms in peer-to-peer networks. Note that parts of the work presented in this chapter have been published in [8, 7, 9].

Contents

1.1 Problem description	1
1.1.1 Assumptions	3
1.2 Specification of the algorithm	3
1.2.1 Data structures	3
1.2.2 Query routing	4
1.2.3 Link selection	7
1.2.4 Routing table updates	8
1.2.5 Complex queries	9
1.2.6 Peer activity	11
1.2.7 Bootstrapping	11
1.2.8 Program flow	11

1.1 Problem description

A peer-to-peer network is a network consisting of interconnected nodes in which each node manages an information repository containing a certain number of resources. Every peers offers its resources to the other nodes in the network. The peers issue queries for those resources to the network. All nodes collaborate to answer the queries and – as a whole – implement a distributed search engine. For each query, the shortest path through

the network must be found that leads from the querying peer to one or more answering peers offering one or more results.

Resources can be any kind of files that are annotated with metadata. The metadata are composed of name-value pairs, also called *elements*. These elements are used for specifying additional information about a certain resource. In general, a meta-data schema defines the name and the meaning for each of the elements the schema is comprised of. In the following, a simplified view on the problem is used. In the simplified view, only one element is considered. In addition, there is a restriction on the allowed values. The values that can be used for annotation originate from a controlled vocabulary, e.g, the concepts of a taxonomy or an ontology. The query vocabulary is the same as the metadata vocabulary used for annotating resources. This means that the queries do not consider the actual content of a resource, but rather the metadata that describes this content. A query Q consists of one or more concepts c_1, \dots, c_n that can be connected either with boolean *OR* or boolean *AND*. If the *OR* connector is used, a resource R is appropriate for satisfying query Q if it is classified to be an instance of one of the concepts c_1, \dots, c_n . In case of connector *AND*, resource R must be an instance of all of the concepts c_1, \dots, c_n for satisfying query Q . The concepts c_1, \dots, c_n will be referred to as the keywords of the query hereafter. The resource R will be referred to as the result of the query.

Since the task of connecting keyword with *AND* and or *OR* is present in this scenario, it means that the solution to the simplified problem described in this chapter can be generalized for the general case in which more than one metadata element is used for annotation. The technique to connect the different values for the same name of an element is the same as connection different values for different names.

The challenges for query routing are the following. Each peer is connected via outgoing links to some other peers which are called its neighbor peers. If a peer issues a query or receives a forwarded query from one of its neighbor peers, it has to decide based on its local information which neighbor peer to send the query to. The local information of a peer is gained by continuously observing the queries and answers that pass the local node and recording which kind of queries its neighbor peers were able to answer in the past (cf. *reputation learning* [6]). The recorded data acts must be accumulated and stored in an appropriate way to support the selection process. Based on this information, the peer has to choose the neighbor peer which is (1) most likely to store results itself, or (2) has neighbor peers that are likely to store such resources. Approaches to query routing in which this decision depends on the keywords of the query are called *content-based* approaches. They are sometimes also referred to as *semantic query routing*. The goals for the query routing procedure are to maximize the number of query results and, at the same time, to minimize the usage of network resources. The best metric for measuring if those two goals are reached is to divide the total amount of network resources (measured in number of hops) consumed by a query by the number of query results.

The objective of the thesis is to employ ant-based methods for the query routing procedure. Basically, the ant metaphor can be applied rather straightforward. The pheromone

trails store the recorded data about the successful queries in the past. The queries themselves are represented as ants.

1.1.1 Assumptions

In order to make it possible to concentrate on the problem of query routing, the following assumptions are made about the application scenario.

1. Each peer has an unique address that can be used as an unique identifier.
2. The resources at each peer can be uniquely identified using an existing resource identifier (such as a filename) together with the peer identifier.
3. All links between peers are bi-directional, that is, can be used in both directions.
4. The network topology already exists. Each peer already knows which neighbor peers it is connected to. The problem of peer discovery is not within the scope of this thesis.

An additional assumption for the work described in this chapter is that the network topology and the content distribution in the network is considered static. The extension of the algorithm for a dynamic setting is subject of future work.

1.2 Specification of the algorithm

This section specifies the components of the SEMANT algorithm. Section 1.2.1 documents the data structures that need to be stored at each peer. Section 1.2.2 describes the query routing procedure and all aspects related to facilitating ant algorithms in a peer-to-peer network. In Section 1.2.3, all issues related to the selection of outgoing links for query routing are covered. Section 1.2.4 explicates the basic design issues of the algorithm's pheromone trail management and shows how routing tables are updated after successful queries. Section 1.2.5 defines the format of the queries. Section 1.2.6 describes the activities executed locally at the peers. Section 1.2.7 discusses how to improve the performance of the algorithm in the start-up phase. Finally, in Section 1.2.8 the algorithm flow is specified.

1.2.1 Data structures

The routing information is stored in two tables τ and η which are present at each peer P_i . Table τ maintains the pheromone trails. It is of size $C \times n$, where C is the size of the controlled vocabulary that defines the allowed keywords in a query and n is the number of peer P_i 's outgoing links to neighbor peers. Each τ_{cu} stores the amount of pheromone corresponding to concept c dropped at the link from peer P_i to peer P_u , for each concept c

and each neighbor peer P_u . The left side of Figure 1.1 shows an example. At startup, all entries in table τ are initialized with the same value $\tau_{init} = 0.009$. Initialization with a small value is necessary to prevent divisions by zero in the evaporation feature (see Section 1.2.6) and in link selection (see Section 1.2.3).



Figure 1.1: Routing tables

Next to table τ , each peer manages a table η that stores link costs to its neighbor peers. This table consists of only one column (see right side of Figure 1.1). Each entry η_u is the inverse value $\frac{1}{lc_u}$ of the cost lc_u for sending an ant from peer P_i to peer P_u . The entries in this table are initialized with zero.

Note that these are the same data structures as used in *AntNet*. The difference is that in *AntNet* the columns of table τ refer to destination nodes, while in the SEMANT algorithm the columns of table τ refer to the keywords that can occur in a query. This implies that a different type of pheromone is used for every query keyword that can occur. Allowing free-text search is not feasible in this setting because it would result in an unlimited number of pheromone types, and therefore in an unlimited number of columns in the routing tables.

1.2.2 Query routing

The concept of forward ants and backward ants from the *AntNet* algorithm [1] is employed as the foundation of the query routing procedure. In addition, *AntNet*'s mechanism for preventing cycles is adopted. Whereas the latter can be adopted without any changes, the concept of forward ants and backward ants needs to be adapted to the application purpose of query routing. In the following, the adaptations necessary are discussed.

1.2.2.1 Queries

In the SEMANT algorithm, queries are represented as ants. This approach has two advantages. First, no additional traffic is created in the network. Second, representing queries as ants guarantees that the degree of optimization for certain query keywords directly depends on the popularity of a given keyword. The more often a query keyword is requested, the better its paths will be optimized in terms of indicating the way through the network to the most appropriate peers. Instead of sending out forward ants at regular

intervals from random peers like in *AntNet*, a forward ant is created for each query that occurs in the network. This ant is created at the peer which issued the query, and it is responsible for answering the query.

1.2.2.2 Time-to-live parameter

Next, it is necessary to define at which point the forward ant should stop its travel. In *AntNet*, forward ants terminate their travel through the network when they arrive at their well-defined destination peer. This behavior can not be transferred, since the forward ant's task is to find an unknown destination peer that in the worst case does not even exist. Instead, a stop point for forward ants must be defined to prevent forward ants from running infinitely if no results can be found. There are two possibilities for defining the stop point. The simplest solution is to use a time-to-live (TTL) parameter $t_{ll_{max}}$ like introduced in Gnutella (see [4]). Each time an ant travels one hop to reach another peer, it decrements $t_{ll_{max}}$ by one. The stop point is reached if $t_{ll_{max}} = 0$.

Another possibility is to use a maximum lifetime t_{max} measured in seconds. This option is more complex, since it requires that either the clocks at every peer are synchronized, or that the ants carry a clock themselves. The benefit of this approach is that it creates an upper bound for the time it takes until the result arrives at the querying peer. If a forward ant arrives at a peer that stores results, it creates a backward ant. Consequently, if results were found and – as a consequence – backward ants were created, the upper bound for a backward ant to arrive at the querying peer is the time interval of $2 * t_{max}$. If no backward ants arrived within this time interval, the querying peer can deduce that no result could be found.

1.2.2.3 MinResources variation and maxResults variation

After a forward ant has found a result and creates a backward ant, there are two possibilities for proceeding further. Either the forward ant (1) terminates its travel, or (2) it continues it until the maximum time-to-live parameter is reached. The idea behind the latter approach is that if forward ants are allowed to go on after they found the first peer that stores results, they can increase the absolute number of results found by detecting other appropriate peers. Keeping in mind that query routing in peer-to-peer network is a special kind of optimization problem, the choice between these two options determines the optimization goal of the algorithm.

- **MinResource variation.** If the ants are terminated after they found the first result, the optimization goal is to use the minimum amount of network resources. Therefore, the backward ant strategy of stopping after the first result is found will be referenced to as the *minResource* variation of the SEMANT algorithm.

- **MaxResults variation.** In the other case, where the ants use the maximum time-to-live parameter, the ants use approximately the same amount of network resources for each query and the optimization goal is to maximize the number of results that are found for a query. In the remainder of this thesis, the strategy of using the maximum time-to-live parameter will be referenced to as the *maxResults* variation of the SEMANT algorithm.

In practice, both of these variations are valid because both of the optimization goals are desired at the same time. Using the *minResource* variation is of benefit for the performance of the entire network, since the algorithm saves as much network resources as possible. Using the *maxResults* variation is of benefit for the individual users of the network, since the algorithm tries to find as many results for a single query as possible. It is possible to combine these variations by using a weight that defines the ratio between using the *minResource* variation and the *maxResults* variation.

1.2.2.4 Step-by-step description of the query routing procedure

Now the complete procedure for answering a query is laid out. Consider a query q issued at a peer P_q . The following seven steps are necessary for answering query q .

- Step 1** Check the resource repository of peer P_q . If any results are found, present them to the user. If the number of results found is less than r_{max} , go to step 2. If the number of results found is greater than r_{max} , terminate the algorithm.
- Step 2** Create a forward ant F_q with starting time t_{Fstart} and timeout t_{max} at peer P_q . Add the identifier of peer P_q to F_q 's stack of already visited peers $s(F_q)$. Initialize the list $lc(F_q)$ that stores the link costs of all links used by F_q .
- Step 3** Use the link selection procedure described in Section 1.2.3 for selecting the neighbor peer(s) P_{j_x} the forward ant F_q should choose ($x \in [1..n], n \in \mathbb{N}$). Send ant F_q to peer P_{j_1} . For every peer P_{j_x} where $x \in [2..n]$, create a clone F_q^c of forward ant F_q and send ant F_q^c to peer P_{j_x} .
- Step 4** For every forward ant F_q that arrives at a peer P_j , check if peer P_j was already visited by a clone F_q^c . If so, terminate ant F_q . Otherwise, check the resource repository of peer P_j for resources r that are results for query Q . If there are no results, continue at step 6. Otherwise, add the identifiers of all resources r to the set R .
- Step 5** Generate a backward ant B_q . Pass it R , the identifier of the peer P_j that stores R , the stack of already visited peers $s(F_q)$, and the recorded link costs $lc(F_q)$. Send B_q back to the querying peer P_q using the procedure described in Section 1.2.4. In case the *minResources* variation is used, terminate the forward ant F_q . Otherwise, continue at step 6.

Step 6 Add the identifier of peer P_j to the stack of already visited peers $s(F_q)$. Add the cost of the last used link to list $lc(F_q)$.

Step 7 If $t_{F_{start}} + t_{max} < CurrentTime$, continue at step 3. Otherwise, terminate F_q .

As soon as a backward ant B_q arrives at the querying peer P_q , the results $r \in R$ are presented to the user. In case the user decides to download a resource r , a direct connection between peer P_q and the peer P_j that stores r is established and resource r is retrieved from peer P_j .

1.2.3 Link selection

Now the selection of outgoing links by the forward ants is described. In ant algorithms, this selection is made by applying a so-called transition rule. The transition rule designed for the SEMANT algorithm is based on the transition rule from the *Ant Colony System* algorithm [2]. The *Ant Colony System* transition rule consists of two strategies that supplement each other. In the exploiting strategy, the ant determines the quality of the links depending on the amounts of pheromone and the link costs and always selects the link with the highest quality. The exploring strategy encourages ants to discover new paths. This is achieved by deriving a goodness value p_u for each neighbor peer P_u not already visited, and by applying the *roulette wheel selection technique* [3] to select one of the peers.

In reinforcement learning approaches, the combination of exploiting the best results so far and exploring new paths is also present. The question of which one of the two strategies to select according to its desirability in the current context of the agent is referred to as the *exploration-exploitation-dilemma* [5, 11]. The dilemma occurs not only in ant algorithms, but in reinforcement learning [10] in general. In *Ant Colony System*, the decision for one of the strategies is based on a defined weight $w_e \in [0, 1]$ which stays constant during the execution of the algorithm. Each ant individually decides for a strategy based on weight w_e and the roulette wheel selection technique every time it has to decide for an outgoing link. The SEMANT algorithm also relies on weight w_e . For the static scenario, it is desirable to use a dynamic factor w_e that can change during the execution of the algorithm. This allows to put more weight on exploring in the start-up phase and after that subsequently degrades according to the lifetime of the network.

For simplicity, the remainder of this section assumes that the given query contains is a simple query containing exactly one keyword. The extensions to support link selection for complex queries are described in Section 1.2.5.

1.2.3.1 Exploiting strategy

In case the exploiting strategy is used, a forward ant F_q located at a certain peer P_i selects the best neighbor peer P_j by applying the transition rule shown in Equation 1.1.

$$j = \arg \max_{u \in U \wedge u \notin s(F_q)} \left([\tau_{cu}] \cdot [\eta_u]^\beta \right), \quad (1.1)$$

where β is a constant that defines the influence of link costs, U is the set of neighbor peers of peer P_i , and $s(F_q)$ is the set of peers already visited by F_q .

1.2.3.2 Exploring strategy

In case a forward ant F_q utilizes the exploring strategy, the transition rule shown in Equation 1.2a and Equation 1.2b is applied for *each* neighbor peer P_j in order to decide whether peer P_j should be selected. Note that this is an adaptation of the roulette wheel selection technique: Each p_j is *separately* placed on the continuum between 0 and 1, and for each p_j a random value q is calculated for deciding whether peer P_j should be selected. This mechanism allows more than one peer to be selected in order to account for the fact that there are multiple possible destination peers which contain answers for a query. To ensure that at least one peer will be selected, the algorithm falls back to the exploiting strategy in case applying the exploring strategy does result in not selecting any peer.

$$p_j = \frac{[\tau_{cj}] \cdot [\eta_j]^\beta}{\sum_{u \in U \wedge u \notin s(F_q)} \left([\tau_{cu}] \cdot [\eta_u]^\beta \right)}, \quad (1.2a)$$

where the assumption is that a forward ant F_q is located at a certain peer P_i . Parameter β is a constant that defines the influence of link costs, U is the set of neighbor peers of peer P_i , $s(F_q)$ is the set of peers already visited by F_q , and

$$GO_j = \begin{cases} 1 & \text{if } q \leq p_j \wedge j \in U \wedge j \notin s(F_q) \\ 0 & \text{else} \end{cases} \quad (1.2b)$$

In Equation 1.2b, q is a random value, $q \in [0, 1]$, and the sum of all goodness values $\sum_{j \in U \wedge j \notin s(F_q)} p_j = 1$. If $GO_j = 1$, the forward ant F_q creates a clone F_q^c of itself and sends the clone F_q^c to peer P_j .

1.2.4 Routing table updates

Now the mechanism used by the backward ants for updating the routing tables is described. A backward ant is created at a certain peer P_r storing a set of results R . Each backward ant B_q is in possession of a copy of the stack data recorded by its corresponding forward ant F_q . This stack contains all visited peers $s(F_q)$ and all recorded link costs $lc(F_q)$.

Depending on whether t_{max} or tll_{max} is used for either measuring the time or measuring the number of hops consumed by the forward ant, the backward ant B_q either calculates the sum of all entries in $lc(F_q)$ to get the total link costs t_{qr} for the path from the querying peer P_q to the answering peer P_r , or it calculates the number of hops h_{qr} between peer P_q

and peer P_r . After performing the calculation, the backward ant 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, the backward ant is responsible for two different tasks:

- First, it updates the link cost η_j according to the entry in $lc(F_q)$.
- Second, it drops pheromone by applying the pheromone trail update rule shown in Equation 1.3a to Equation 1.3c. In Equation 1.3a, the amount Z of newly added pheromone depends on the goodness of the found path. If t_{max} is used, amount Z is derived according to Equation 1.3b. If t_{lmax} is used, amount Z is derived according to Equation 1.3c.

The goodness of the found path is determined by comparing the number of resources found and the length of the path to pre-defined optimal values. For the optimal solution, the optimal value for a path's total link costs is set to $\frac{1}{2} \cdot t_{max}$, and the optimal number of resources is set to r_{max} .

$$\tau_{cj} \leftarrow \tau_{cj} + Z, \quad (1.3a)$$

where

$$Z = w_d \cdot \frac{|R|}{r_{max}} + (1 - w_d) \cdot \frac{t_{max}}{2 \cdot t_{qr}} \quad (1.3b)$$

or

$$Z = w_d \cdot \frac{|R|}{r_{max}} + (1 - w_d) \cdot \frac{t_{lmax}}{2 \cdot h_{qr}} \quad (1.3c)$$

In Equation 1.3b and Equation 1.3c, parameter w_d weights the influence of resource quantities and link costs.

1.2.5 Complex queries

The algorithm as described in Section 1.2.2.4 acts on the assumption that simple queries consisting of only one keyword c are issued. Now the extensions to the algorithm to provide for complex queries containing multiple keywords c_1, \dots, c_n are described. Section 1.2.5.1 deals with conjunctive queries. Section 1.2.5.1 addresses disjunctive queries.

1.2.5.1 Conjunctive queries

In case of conjunctive queries (Boolean *AND*) with keywords c_1, \dots, c_n , the pheromone trails corresponding to c_1, \dots, c_n must be incorporated. The following changes are necessary. First, link selection must consider the average amount of pheromone of all keywords c_1, \dots, c_n instead of τ_{cx} for keyword c and all peers P_x . Equation 1.4 shows the changes to the exploiting strategy (cf. Equation 1.1). Equation 1.5 shows the changes to the exploiting strategy (cf. Equation 1.2a).

$$j = \arg \max_{u \in U \wedge u \notin s(F_q)} \left(\left[(\tau_{c_1 u} + \dots + \tau_{c_n u}) \cdot \frac{1}{n} \right] \cdot [\eta_u]^\beta \right), \quad (1.4)$$

$$p_j = \frac{\left[(\tau_{c_1 j} + \dots + \tau_{c_n j}) \cdot \frac{1}{n} \right] \cdot [\eta_j]^\beta}{\sum_{u \in U \wedge u \notin s(F_q)} \left(\left[(\tau_{c_1 u} + \dots + \tau_{c_n u}) \cdot \frac{1}{n} \right] \cdot [\eta_u]^\beta \right)}, \quad (1.5)$$

Second, in case of a successful query, the routing table entries that belong to the pheromone trails $\tau_{c_1 x}, \dots, \tau_{c_n x}$ need to be updated at every peer P_x which is a member of the found path between the querying and the answering peer. Equation 1.6 shows the changes to the pheromone trail update rule (cf. Equation 1.3a).

$$\tau_{c_x j} \leftarrow \tau_{c_x j} + Z, \quad (1.6)$$

where $x \in 1, \dots, n$ and amount Z is derived according to Equation 1.3b and Equation 1.3c, respectively.

1.2.5.2 Disjunctive queries

In case of a disjunctive query (Boolean *OR*) with keywords c_1, \dots, c_n , the best one of all pheromone trails corresponding to c_1, \dots, c_n must be considered in the link selection procedure. Equation 1.7 shows the changes to the exploiting strategy (cf. Equation 1.1). Out of all trails related to keywords c_1, \dots, c_n , the link with the highest product of link costs and amount of pheromone is selected.

$$j = \arg \max_{u \in U \wedge u \notin s(F_q) \wedge x=1, \dots, n} \left([\tau_{c_x u}] \cdot [\eta_u]^\beta \right), \quad (1.7)$$

The exploring strategy (cf. Equation 1.2a) needs to be changed only slightly. It is executed independently for each keyword c_1, \dots, c_n .

$$p_j = \frac{[\tau_{c_x j}] \cdot [\eta_j]^\beta}{\sum_{u \in U \wedge u \notin s(F_q)} \left([\tau_{c_x u}] \cdot [\eta_u]^\beta \right)}, \quad (1.8)$$

where $x \in 1, \dots, n$. Note that the resource usage created by disjunctive queries is proportional to the number of keywords in the query. Hence, malicious users can potentially abuse disjunctive queries by employing a high number of keywords. This can be avoided by defining a maximum number of allowed keywords.

The pheromone update rule remains unchanged as defined in Equation 1.3a. It is applied to the keyword the found resource is annotated with.

1.2.6 Peer activity

Each peer can perform management procedures on its local routing table. It is responsible for periodically applying an evaporation feature to the locally maintained pheromone trails. Evaporation is one of the constituents of pheromone management in ant algorithms. In a static scenario, the evaporation rule is useful for preventing that paths become too dominant. In a dynamic setting, it can also help to remove outdated information.

$$\tau_{cu} \leftarrow (1 - \rho) \cdot \tau_{cu} \quad (1.9)$$

Each peer applies the evaporation rule shown in Equation 1.9 in predefined intervals t_e for each link to neighbor peer P_u and each concept c . The amount of pheromone that evaporates in every interval is controlled by parameter $\rho \in [0, 1]$. This rule is adopted from the evaporation part of *Ant Colony System's* global pheromone update rule [2].

1.2.7 Bootstrapping

In the start-up phase – directly after the initialization of the routing tables – all pheromone trails are of equal strength. Therefore, the ants make their decisions completely randomly. This means that the performance values of the algorithm will be rather bad in the beginning. Informed decisions are made as soon as an appropriate number of ant found results, and updated the pheromone trails accordingly. This also improves the performance values. After a certain time, the pheromone trails will be near to the optimum. After reaching this point, the performance values do not improve significantly anymore. One possibility to speed up the time to reach this point is to send additional ants in the start-up phase. Since the exploring strategy already provides for selecting more than one outgoing link, sending additional ants is most useful in the exploiting strategy.

A simple criteria to detect if the algorithm is in the start-up phase is to check the amounts of pheromone on the outgoing links before link selection is conducted in order to find out if all amounts for the keywords of the query are of equal size. If this is the case, one possibility would be to send an ant to every outgoing link, but this is equal to flooding. Another option would be to randomly choose two outgoing links, but initial experiments show that this still creates a high amount of network traffic.

Instead, a parameter $p_{sendAdditional} \in [0, 1]$ is used which defines the probability that two outgoing links are chosen in case (1) the pheromone trails on all outgoing links are equal and (2) the exploiting strategy is employed.

1.2.8 Program flow

Finally, the program flow of the SEMANT algorithm can be specified in pseudo-code. It is shown in Figure 1.2.

```

1  t = CurrentTime;
2  t_end = EndTimeOfSimulation;
3
4  # Concurrent activity
5  foreach (Peer) {
6      initializePheromoneTables ();
7      initializeLinkCostsToNeighborNodes ();
8      while (t <= t_end) {
9
10         # Concurrent activity at each peer
11         in_parallel {
12             if (Query Q) {
13                 checkLocalDocumentRepository ();
14                 createForwardAnt(query_parameter);
15             }
16
17             foreach (ForwardAnt) {
18                 initializeParameters ();
19                 while (TTL > 0) {
20                     applyTransitionRule ();
21                     t_a = CurrentTime;
22                     foreach (ForwardAnt) {
23                         GoToNode(P_j);
24                         t_b = CurrentTime;
25                         checkLocalDocumentRepository ();
26                         if (DocumentsFound > 0) {
27                             createBackwardAnt(stackData , P_j );
28                             if (minResourceStrategy) {
29                                 terminate ();
30                             }
31                         }
32                         addDataToStack(P_j , t_b-t_a);
33                     }
34                     TTL = TTL - 1;
35                 }
36                 terminate ();
37             }
38
39             foreach (BackwardAnt) {
40                 calculateQuality ();
41                 do {
42                     node = popStackData_Node ();
43                     GoToNode(node);
44                     applyPheromoneTrailUpdateRule ();
45                     updateListCosts(popStackData_Costs ());
46                 } while (node <> source_node);
47             }
48
49             foreach (PeriodicalTimeInterval t_e) {
50                 applyEvaporationRule ();
51             }
52
53         }
54     }
55 }

```

Figure 1.2: The SEMANT algorithm in pseudo-code

Bibliography

- [1] Gianni Di Caro and Marco Dorigo. AntNet: Distributed Stigmergy Control for Communications Networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317–365, July 1998.
- [2] Marco Dorigo and Luca Maria Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.
- [3] David E. Goldberg and Kalyanmoy Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Proceedings of the 1st Workshop on Foundations of Genetic Algorithms*, pages 69–93, July 1990.
- [4] Manfred Hauswirth and Schahram Dustdar. *Software-Architekturen für Verteilte Systeme*, chapter Peer-to-Peer: Grundlagen und Architektur, pages 161–197. Springer, August 2003.
- [5] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
- [6] Sam Joseph and Takashige Hoshiai. Decentralized Meta-Data Strategies: Effective Peer-to-Peer Search. *IEICE Transactions on Communications*, E86-B(6):1740–1753, June 2003.
- [7] Elke Michlmayr. Ant Algorithms for Search in Unstructured Peer-to-Peer Networks. In *Proceedings of the Ph.D. Workshop, 22nd International Conference on Data Engineering (ICDE 2006)*, April 2006.
- [8] Elke Michlmayr, Sabine Graf, Wolf Siberski, and Wolfgang Nejdl. Query Routing with Ants. In *Proceedings of the 1st Workshop on Ontologies in P2P Communities, 2nd European Semantic Web Conference 2005 (ESWC2005)*, Heraklion, Crete, May 2005.
- [9] Elke Michlmayr, Arno Pany, and Sabine Graf. Applying Ant-based Multi-Agent Systems to Query Routing in Distributed Environments. In *Proceedings of the 3rd IEEE Conference On Intelligent Systems (IEEE IS06)*, September 2006.
- [10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Bibliography

- [11] Stewart W. Wilson. Explore/Exploit Strategies in Autonomy. In *From Animals to Animats 4: Proceedings of the 4th International Conference on the Simulation of Adaptive Behavior*, pages 325–332, 1996.