Ant Colony Optimization for RDF Chain Queries for Decision Support

Alexander Hogenboom[a,*], Flavius Frasincar[a], Uzay Kaymak[b]

[a]*Erasmus University Rotterdam, P.O. Box 1738, 3000 DR, Rotterdam, The Netherlands*
[b]*Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Abstract

Semantic Web technologies can be utilized in expert systems for decision support, allowing a user to explore in the decision making process numerous interconnected sources of data, commonly represented by means of the Resource Description Framework (RDF). In order to disclose the ever-growing amount of widely distributed RDF data to demanding users in real-time environments, fast RDF query engines are of paramount importance. A crucial task of such engines is to optimize the order in which partial results of a query are joined. Several soft computing techniques have already been proposed to address this problem, i.e., two-phase optimization (2PO) and a genetic algorithm (GA). We propose an alternative approach – an ant colony optimization (ACO) algorithm, which may be more suitable for a Semantic Web environment. Experimental results with respect to the optimization of RDF chain queries on a large RDF data source demonstrate that our approach outperforms both 2PO and a GA in terms of execution time and solution quality for queries consisting of up to 15 joins. For larger queries, both ACO and a GA may be preferable over 2PO, subject to a trade-off between execution time and solution quality. The GA yields relatively good solutions in a comparably short time frame, whereas ACO needs more time to converge to high-quality solutions.

Keywords: RDF Chain Query Optimization, Ant Colony Optimization, Genetic Algorithm, Iterative Improvement, Simulated Annealing

*Corresponding author; tel: +31 (0)10 408 2264; fax: +31 (0)10 408 9031

Email addresses: hogenboom@ese.eur.nl (Alexander Hogenboom), frasincar@ese.eur.nl (Flavius Frasincar), u.kaymak@ieee.org (Uzay Kaymak)

1. Introduction

In today's information-driven society, decision makers need to process a continuous flow of data through various input channels, by extracting information, understanding its meaning, and acquiring knowledge by applying reasoning to the gathered information (Hogenboom et al., 2012). Yet, an overwhelming amount of data is available at any given time, whereas decision makers – businesses and consumers alike – need a complete overview of their environment in order to enable effective, well-informed decision making. In order to address this issue, Semantic Web (Berners-Lee et al., 2001) technologies can be utilized in expert systems for decision support, as has been demonstrated in recent research on knowledge management (Joo and Lee, 2009), annotation (Aksac et al., 2012) and recommendation of data sources (Hsu, 2009), intelligent search (Batzios et al., 2008; Lupiani-Ruiz et al., 2011; Vandic et al., 2012), and personalized user experiences in, e.g., tourism (Garcia-Crespo et al., 2011) or e-commerce (Blanco-Fernandez et al., 2010).

The rise of the Semantic Web facilitates an ever-growing amount of data to be stored in many heterogenous, yet interconnected sources. This data is commonly represented by means of the Resource Description Framework (RDF), a framework for describing and interchanging meta-data (Klyne and Carroll, 2004), which describes the context of data and thus enables machine-interpretability. Due to the interconnectivity of data rather than pages, the Semantic Web has the potential of addressing today's typical users' complex information needs in a more effective and efficient way than the current Web can.

Semantic Web technologies allow a user to explore many different data sources in order to address very specific information needs. A typical scenario here may be a query for features and reviews of a holiday destination and comparable alternatives, as well as prices and details of trips to any of these locations. Such queries can be executed on multiple RDF sources by means of the SPARQL Protocol and RDF Query Language (SPARQL) (Prud'hommeaux and Seaborne, 2008). In order for SPARQL queries to disclose the ever-growing amount of widely distributed RDF data to demanding users in real-time environments, fast RDF query engines are crucial.

Recent expert systems for querying distributed environments with multiple heterogenous information sources such as databases or repositories focus on the problem of identifying the information sources that are most relevant with respect to queries (Jung, 2010) or on the problem of combining the query results from multiple sources while optimizing coverage and search effectiveness (Amin and Emrouznejad, 2011; Batzios and Mitkas, 2012). However, to the best of our knowledge, expert systems dealing with optimizing the execution efficiency of a given query in a distributed environment have been relatively unexplored.

One of the problems today's RDF query engines face is the optimization of the order in which the distinct parts of a query are executed. The total execution times of queries depend on these query paths. A good algorithm for optimizing the execution order in a query path can thus contribute to efficient querying. As the number of possible query paths grows exponentially with the query size, the optimization of RDF query paths is far from trivial. Therefore, several soft computing techniques have been proposed to address this problem. For instance, Stuckenschmidt et al. (2005) present a two-phase optimization (2PO) algorithm, consisting of an iterative improvement (II) method followed by simulated annealing (SA). More recently, a genetic algorithm (GA) has been shown to be a promising alternative when optimizing RDF queries (Hogenboom et al., 2009).

As their design has been inspired by methods for optimization of query paths in traditional databases, existing soft computing approaches to RDF query path optimization have essentially been designed for more or less static environments – changes in the environment typically require the optimization to be run all over again. However, the Semantic Web is a complex and dynamic environment. Data changes, sources come and go, and latency between sources may be volatile. In this light, ant colony optimization (ACO) (Dorigo et al., 1996, 2006) is a good alternative to the existing soft computing approaches in an RDF environment. ACO is a soft computing technique inspired by the foraging behavior of ant colonies. Its nature allows the algorithm to be run continuously and to adapt to changes in the environment in real time. Moreover, ACO has been shown to outperform GAs in solving complex problems such as scheduling (Merkle et al., 2002) and sequential ordering (Gambardella and Dorigo, 2000).

As its characteristics render ACO an attractive alternative to existing soft computing techniques for RDF query optimization, we explore the applicability of ACO to query path optimization in an RDF environment. Furthermore, we compare the performance of existing soft computing techniques for RDF query path optimization with our novel ACO algorithm. We focus on the performance of the considered algorithms when optimizing a special class of SPARQL queries, the so-called RDF chain queries, on a single source.

The remainder of this paper is structured as follows. First, Section 2 provides a short introduction to RDF and chain queries. We then discuss the two existing soft computing techniques as well as our novel ACO algorithm for RDF chain query optimization in Sections 3 and 4. In Section 5, we evaluate the performance of our considered methods in terms of execution time and solution quality. Finally, we draw conclusions and propose directions for future work in Section 6.

2. RDF and Chain Queries

In an RDF model, facts are declared as a collection of triples, each consisting of a subject, a predicate, and an object. RDF triples can be visualized in a graph, which can be described as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (Klyne and Carroll, 2004). The relationship between a subject node and an object node in an RDF graph is defined using an arc which denotes a predicate.

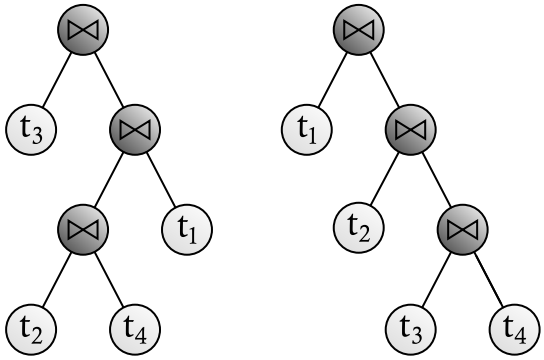
When querying RDF sources, RDF triples are essentially matched with a series of patterns specified in a SPARQL query. In the specific subset of SPARQL queries we consider in our current endeavors, the `WHERE` statement only contains a set of node-arc-node patterns which are chained together such that the object of one predicate is the subject of the next predicate. Such RDF chain queries resemble chain queries in traditional relational databases, where a query path is followed by performing joins between its subpaths of length 1 (Stuckenschmidt et al., 2005). Randomized and genetic algorithms have proven to outperform heuristic methods in optimizing these types of queries in relational databases (Steinbrunn et al., 1997).

A typical example of a chain query in an RDF environment is the following. Let us consider an RDF model of the CIA World Factbook generated by using QMap (Hogenboom et al., 2008). This model contains data about 250 countries, defined in over 100,000 triples. Suppose a financial risk analyst wants to identify the export partners of The Netherlands that have dependent areas (i.e., areas that do not possess full political independence or sovereignty) that are involved in an international conflict. This query can be expressed in SPARQL as demonstrated in Figure 1.

```
1. PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
2. PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
3. SELECT ?partner
4. WHERE { c:NL o:exportPartner ?expPartner .
5.         ?expPartner o:country ?partner .
6.         ?partner o:dependentArea ?area .
7.         ?area o:internationalDispute ?conflict .
8.       }
```

Figure 1: Example RDF chain query in SPARQL.

This query can be subdivided into four subqueries: a query for information on the export partners of The Netherlands (line 4), a query for countries associated with other countries as export partners (line 5), a query for dependent areas (line 6), and a query for international disputes (line 7). In order to resolve the complete query, the results of the individual subqueries can be joined in any order. Here, the number of statements resulting from a join is equal to the number of statements compliant with both operands' constraints.



(a) Bushy tree. (b) Right-deep tree.

Figure 2: Examples of query trees for an RDF chain query with three joins. Light gray nodes represent matches with triple patterns, whereas dark gray nodes with a ⋈ represent joins.

A sequence of joins in such a query can be visualized as a tree. The leaf nodes of an RDF query tree typically represent inputs, whereas the internal nodes represent algebra operations, enabling one to specify basic retrieval requests on the inputs (Frasincar et al., 2004). We consider the leaf nodes of a query tree to be matches with the individual patterns of triples constituting an RDF chain query. The internal nodes represent join operations.

The nodes in a query tree can be ordered in many different ways, which are referred to as query paths. In an RDF context, bushy and right-deep query trees can be considered (Stuckenschmidt et al., 2005). In bushy trees, base relations (containing matches with one triple pattern) as well as results of earlier joins can be joined. Right-deep trees, which are a subset of bushy trees, require the left-hand join operands to be base relations. Figure 2 depicts a bushy tree and a right-deep tree for our example query, where matches with triple patterns $\{t_1, t_2, t_3, t_4\}$ – corresponding with lines 4 through 7 of our example query in Figure 1, respectively – are joined, with ⋈ representing a join. Irrespective of the order of these joins, the result of a query will always be the same, ceteris paribus. However, the total execution time of a query does depend on the order of joins. Therefore, query path optimization is crucial in today's real-time RDF environments.

3. Soft Computing Techniques for RDF Chain Query Optimization

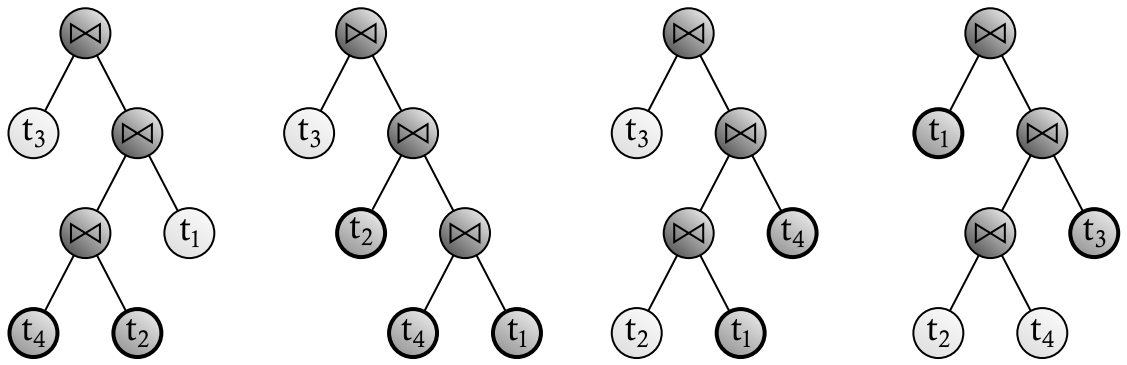
The order of joins of subpaths in an RDF chain query path is variable and affects the time needed for executing the query. In this context, the join-order problem arises. The challenge is to determine the right order in which the joins should be computed, hereby optimizing the overall response time. Typical approaches to address this problem involve exploring a solution space in an attempt to find low-cost solutions.

3.1. Solution Space

In the solution space, each solution represents a query path. The size of this solution space depends on the type of query trees. For n base relations, and hence $n - 1$ joins, there are $n!$ different right-deep query trees, whereas there are $\binom{2(n-1)}{(n-1)}(n-1)!$ possible bushy query trees (Steinbrunn et al., 1997). Solutions are located in this solution space in such a way that their neighbors are similar solutions. We consider solutions to be neighbors of a solution if they can be obtained by transforming the latter solution by applying one of four transformation rules once to a part of the solution query tree, i.e., join commutativity, join associativity, left join exchange, or right join exchange (Ioannidis and Kang, 1990). Figure 3 demonstrates these rules for our example bushy query tree presented in Figure 2(a).

3.2. Solution Costs

Each solution is associated with execution costs, which are mainly realized by the costs of data transmission from the source to the processor and the costs of processing this data (Stuckenschmidt et al., 2005). As our current research focuses on a single source, we omit the (constant) data transmission costs and only consider data processing costs, i.e., the sum of costs associated with all joins within a solution. Join costs are typically influenced by the cardinalities of the operands and the join method used. Several methods can be used for implementing (two-way) joins (Elmasri and Navathe, 2004). We consider only nested-loop joins, as we assume that no index or hash key exists a priori for sources used in a dynamic Semantic Web environment (making single-loop and hash joins unfeasible) and that the source data is unsorted (requiring the sort-merge join algorithm to sort the data first, which would take up precious running time). When



(a) Join commutativity. (b) Join associativity. (c) Left join exchange. (d) Right join exchange.

Figure 3: Examples of neighboring solutions of the bushy query tree presented in Figure 2(a). Light gray nodes represent matches with triple patterns, whereas dark gray nodes with a ⋈ represent joins. Gray nodes with thick borders mark changes with respect to the original bushy query tree, which result from applying the considered transformation methods.

performing a nested-loop join, all elements in both join operands need to be compared with one another. In this light, when joining operands o_{js1} and o_{js2} for each join j in a solution s with n base relations, we define the total execution costs c_s as

$$c_s = \sum_{j=1}^{n-1} \left|o_{js1}\right| \left|o_{js2}\right|, \tag{1}$$

with $\left|o_{js1}\right|$ and $\left|o_{js2}\right|$ being the cardinalities of the first and second join operands of join j for solution s, respectively.

For base relations, cardinalities can be derived from the data, i.e., by counting the number of triples corresponding to a predicate. Conversely, the cardinality of the result of an arbitrary join is a function of the cardinalities of its operands. In the worst-case scenario, the result of a two-way join equals the Cartesian product of the two operands. However, some join selectivity may take place. Therefore, we define the cardinality $\left|o_{js}\right|$ of join j for solution s as

$$\left|o_{js}\right| = \left|o_{js1}\right| \left|o_{js2}\right| \sigma_{js}, \tag{2}$$

where σ_{js} represents the selectivity of join j for solution s. The join selectivity of partial results may not be known in real-time without actually performing the join. Therefore, we need to approximate this selectivity. To this end, following (Steinbrunn et al., 1997), we propose to use an established selectivity heuristic (Selinger et al., 1979), i.e.,

$$\sigma_{js} = \frac{1}{\max\left(\left|o_{js1}\right|, \left|o_{js2}\right|\right)}, \tag{3}$$

an estimation which could of course be updated over time in a real-time environment.

As not every query path is as efficient as others, the challenge in query path determination is to minimize query execution costs. Queries on RDF sources could be translated into algebraic expressions (Hogenboom et al., 2009), which can then be optimized by applying simple transformation rules for relational algebraic expressions (Frasincar et al., 2004; Elmasri and Navathe, 2004). However, in complex solution spaces, these heuristics are not sufficient; several soft computing techniques have been shown to yield better results in traditional query execution environments (Steinbrunn et al., 1997). Inspired by these results, soft computing techniques have been applied to query optimization in the context of the Semantic Web.

3.3. Two-Phase Optimization

One of the first soft computing methods for exploring the solution space of RDF query paths is the 2PO algorithm (Stuckenschmidt et al., 2005). This optimization method consists of two phases. In the first phase, an II algorithm randomly generates a set of initial solutions, each of which is used as a starting point for a walk in the solution space. In each walk, every step is a move towards a better neighbor. At some point in an arbitrary walk, a solution is reached for which no better neighbor can be found in a limited number of tries, in which case the current solution is considered to be a local optimum.

The best local optimum thus found is subsequently used as a starting point for the second phase – a SA algorithm. Inspired by the natural process of annealing of crystals from liquid solutions, SA simulates a continuous temperature reduction, enabling the system to cool down completely from a specified starting temperature to a state in which the system is considered to be frozen. The probability of the algorithm to accept moves not yielding improvement is proportional to the system's temperature and is inversely proportional to the difference in costs between a current solution and an arbitrary neighboring solution. The algorithm thus searches the proximity of possibly suboptimal solutions, hereby reducing the risk for a local optimum.

3.4. Genetic Algorithm

More recently, a GA has been proposed as a promising alternative to 2PO for RDF chain query optimization (Hogenboom et al., 2009). A GA is an optimization algorithm simulating biological evolution according to the principle of survival of the fittest (Holland, 1975). A population of chromosomes – representing solutions – is exposed to evolutionary operations, consisting of selection (where individual chromosomes are chosen to be part of the next generation), crossovers (creating offspring by combining some chromosomes), and mutations (randomly altering some chromosomes). Evolution is simulated until the maximum number of iterations is reached or several generations have not yielded any improvement. The fitness of a chromosome determines the probability of its survival and is inversely proportional to the associated solution's execution costs.

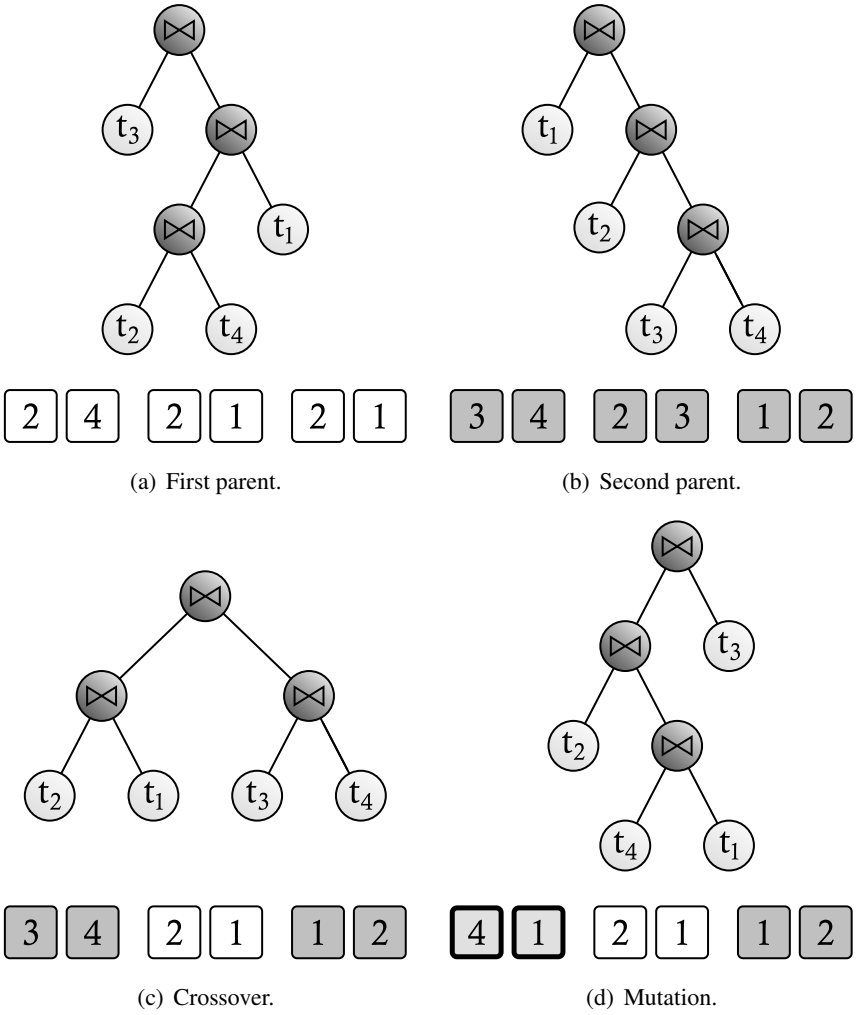
The GA tha has been proposed for RDF chain query optimization (Hogenboom et al., 2009) makes use of an efficient ordinal number encoding scheme (Steinbrunn et al., 1997). This encoding scheme iteratively joins two operands in an ordered list of operands, the result of which is saved in the position of the first appearing operand. The sequence of pairs of indices of operands thus obtained is used to encode the solution. Because of this, for n base relations, the first two numbers are two unique integers ranging from

1 to n (first pair), the third and fourth numbers are two unique integers ranging from 1 to $n-1$ (second pair), etc. This facilitates easy crossover operations, as every pair of join operands in one encoded solution can be interchanged with another pair of join operands on the same position in another encoded solution. Additionally, mutation operations can be implemented in a straightforward way, involving generating new join operands that comply with the restrictions of their respective positions in the encoded solutions.

This encoding scheme can be nicely illustrated by means of the bushy query tree for our example chain query, presented in Figure 2(a). First, consider the ordered list of matches with triple patterns $\{t_1, t_2, t_3, t_4\}$. An initial join between the second and fourth triple pattern yields the list $\{t_1, t_2t_4, t_3\}$. A subsequent join between the second and first operand in this new list yields $\{t_2t_4t_1, t_3\}$. A final join between the second and first operand in the latter list results in $\{t_3t_2t_4t_1\}$. This join order would be encoded as $((2,4),(2,1),(2,1))$.

The solution thus encoded can be subject to evolutionary operations, as demonstrated in Figure 4. In this example, the bushy query tree presented in Figure 2(a) and the right-deep query tree presented in Figure 2(b) are selected to serve as parent solutions in a crossover operation. For each set of two positions in the encoded solutions, information is copied from a randomly selected parent into an offspring solution, as shown in Figure 4(c). This procedure ensures the validity of the resulting offspring solution, as each set of two positions in the encoded versions of both parent solutions, and thus in the encoded offspring solution, adhere to the same constraints. Figure 4(d) demonstrates that the resulting solution can subsequently be mutated by replacing a randomly selected set of two positions in the encoded solution by a new encoded set of two join operands, compliant with the restrictions of the position in the encoded solution.

Inspired by the observation of GAs being aware of good solutions faster than 2PO, but spending more time on optimizing these already good results in traditional query execution environments (Steinbrunn et al., 1997), Hogenboom et al. (2009) attempt to find a balance between execution time and solution quality. To this end, they propose a GA for RDF chain query optimization with a relatively small population, high crossover rate, low mutation rate, and a low threshold for convergence. Moreover, their algorithm always selects the best solution for proliferation in the next generation at least once (elitist selection) in order to stimulate relatively fast convergence. Initial results demonstrate a promising performance compared to 2PO in terms of both execution time and solution costs, especially for larger queries.



(a) First parent. (b) Second parent.

(c) Crossover. (d) Mutation.

Figure 4: Evolutionary operations on (encoded) solutions. Light gray nodes represent matches with triple patterns, whereas dark gray nodes with a ⋈ represent joins. The encodings are depicted in boxes – white boxes for information from the first parent, and gray boxes for information from the second parent. The mutated part of a solution is marked in the encoding with thick borders of the associated boxes.

4. Ant Colony Optimization for RDF Chain Query Optimization

As the existing 2PO and GA approaches to RDF chain query path optimization have been inspired by methods for optimization of query paths in traditional databases, their nature renders these approaches most applicable to more or less static environments. When something in the environment changes during or after optimization, existing optimization techniques typically need to be run all over again, possibly bootstrapped with the best-so-far solution. However, the Semantic Web is a complex and dynamic environment in which data continuously changes, data sources come and go, and transmission costs between sources may vary over time. ACO algorithms can be run continuously and can inherently adapt to changes in the

environment in real time, irrespective of whether such changes occur during or after (initial) optimization. Moreover, ACO approaches have been shown to outperform GAs in solving complex problems such as scheduling (Merkle et al., 2002) and sequential ordering (Gambardella and Dorigo, 2000). Therefore, we propose a method of optimizing RDF Chain Queries using Ant Colony Optimization (RCQ-ACO).

4.1. Ant Colony Optimization

ACO is a soft computing technique inspired by the foraging behavior of ant colonies (Dorigo et al., 1996, 2006). In such colonies, ants walking between their nest and a food source mark their paths with pheromone traces. Foraging ants make use of these traces, as they tend to follow paths where the pheromone concentration is highest. Over time, shorter paths attract more and more pheromones as they are traversed with increasing frequency because of their length as well as the pheromone traces on these paths. The colony thus converges to using a short path.

ACO is essentially a populated meta-heuristic where each encountered solution is represented by a path of an ant in a solution space. Prototypical applications for ACO are problems of which the solution space can be represented as a graph, with artificial ants finding their way through the graph. Typically, each iteration of the algorithm consists of two steps. First, every ant in the colony constructs a path from a start vertex to an end vertex in the graph. Second, when all ants have reached the end vertex, the edges of each path are marked with a pheromone quantity proportional to the path's quality.

In the classic Ant System (Dorigo et al., 1996), an ant k constructs a path by iteratively moving from its most recently visited vertex x to another, not yet visited vertex y. The probability $p_{xy}^{k}(i)$ for an ant k to move from vertex x to vertex y at iteration i of the algorithm is defined as

$$p_{xy}^{k}(i) = \begin{cases} \frac{\tau_{xy}^{\alpha}(i-1)\eta_{xy}^{\beta}}{\sum_{z \in V_k(x)} \tau_{xz}^{\alpha}(i-1)\eta_{xz}^{\beta}}, & z \in V_k(x), \\ 0, & z \notin V_k(x), \end{cases} \tag{4}$$

where $\tau_{xy}(i-1)$ represents the global pheromone quantity on the edge joining vertices x and y at iteration $i-1$, η_{xy} is a local heuristic measure capturing the inverse of the (estimated) distance between vertices x and y, and $V_k(x)$ represents the unvisited vertices of ant k after visiting vertex x. The α and β parameters control the relative importance of the information conveyed by global pheromone traces and local heuristics, respectively. Using the newly deposited pheromone traces of all m ants, the pheromone quantity $\tau_{xy}(i)$

associated with an edge joining vertices x and y is updated at iteration i as

$$\tau_{xy}(i) = (1-\rho)\,\tau_{xy}(i-1) + \sum_{k=1}^{m} \Delta\tau_{xy}^{k}(i)\,, \tag{5}$$

with ρ being an evaporation rate helping the colony not to converge to a local optimum and $\Delta\tau_{xy}^{k}(i)$ being the pheromone quantity deposited at iteration i by ant k on an edge w_{xy} joining vertices x and y. This quantity is defined as a function of a constant Q and the length L_k of path $T_k(i)$, i.e.,

$$\Delta\tau_{xy}^{k}(i) = \begin{cases} \frac{Q}{L_k}, & w_{xy} \in T_k(i)\,, \\ 0, & w_{xy} \notin T_k(i)\,. \end{cases} \tag{6}$$

4.2. Problem Representation

Because this classic design enables ants to find their way around changes in the graph, the application of ACO to RDF chain query optimization could be an attractive alternative to existing soft computing approaches. Therefore, we propose an ACO-based algorithm that utilizes a graph representation of an ordinal number encoding scheme (Steinbrunn et al., 1997), enabling ants to explore the solution space by iteratively constructing solutions, partly guided by global pheromone traces signaling good solutions and partly guided by their own local cost estimations of every next join.

As Figure 5 shows for the bushy chain query path example presented in Section 2, the ants find their way through a directed graph from a start vertex, representing a situation in which no matches with triple patterns have been joined, to an end vertex, representing a situation in which all joins have been made. An ant's path represents a join sequence, where each step represents a join. For an arbitrary join, the vertices in the graph represent all valid combinations of indices of join operands, in accordance with the ordinal encoding scheme presented in Section 3.4. An ant's observed distance associated with an edge connecting a vertex from one join with a vertex from the next join depends on the path taken up until the former join and represents the estimated additional costs of performing the next join – i.e., as detailed in (1), the product of the join operands' cardinalities, which are in turn estimated by applying (2) and (3). In accordance with (4), these observed distances associated with edges, as well as the pheromone traces on these respective edges guide the ants' decisions. Each iteration, ants deposit new pheromone traces which are inversely proportional to their paths' associated total execution costs, in accordance with (5) and (6). When the ants have not encountered a better path in a number of iterations, the algorithm is considered to have been converged.

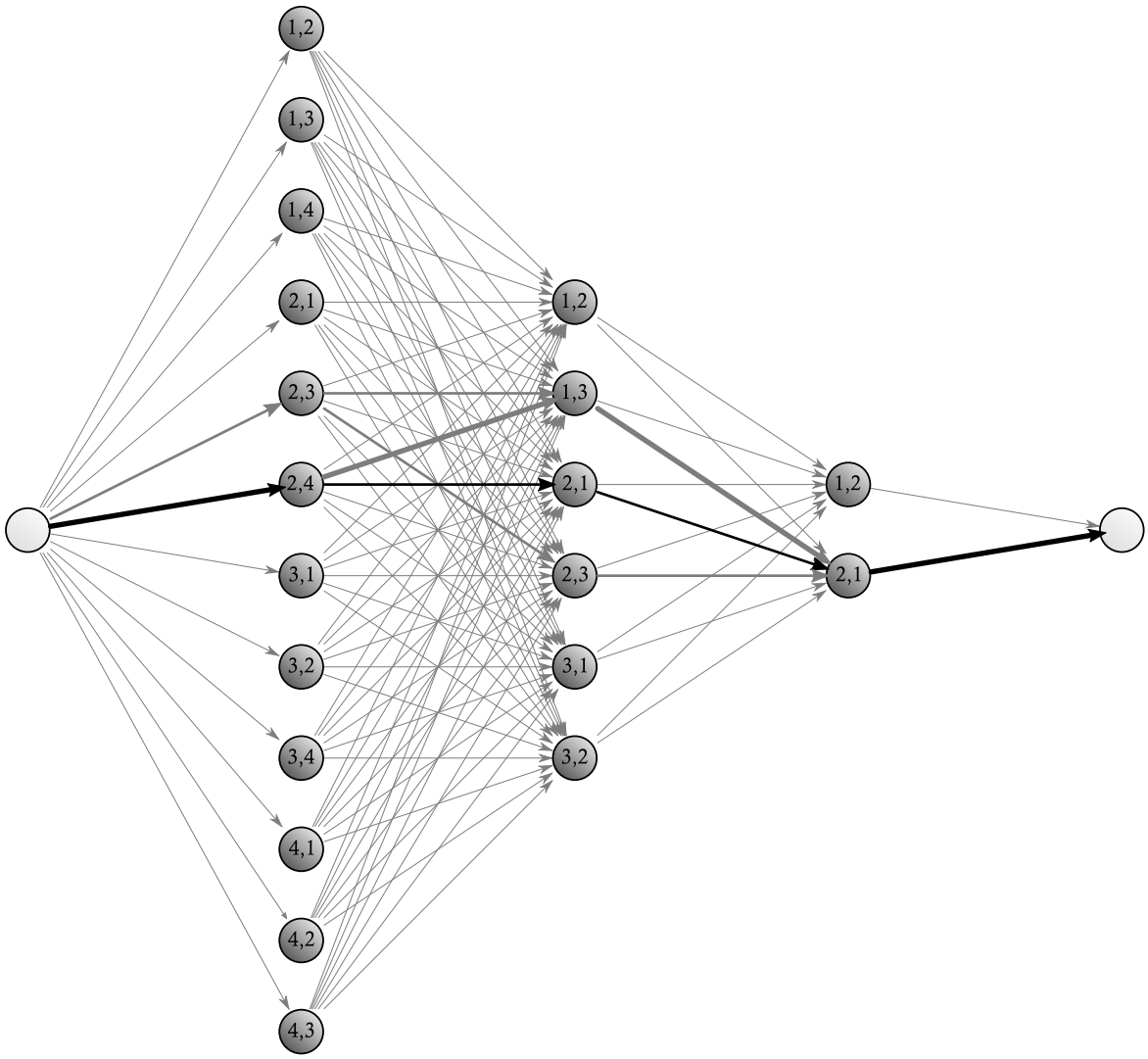


Figure 5: Graph representation of the join ordering problem, as used in our proposed ant colony optimization approach to RDF chain query optimization. A colony of artificial ants iteratively traverses this graph from a starting vertex to an end vertex, both marked light gray. At each step, an ant selects one join from all possible alternatives, represented as columns of dark gray vertices, in accordance with the ordinal number encoding scheme presented in Section 3.4. The width of an edge reflects the pheromone quantities on this edge. This graph is a representation for an RDF chain query with 3 joins. The path representing the bushy query tree example presented in Figure 2(a) is marked by means of black edges.

The benefit of this design is that it allows for an easy representation of bushy query trees as well as for an easy distinction between using intermediate results as left or right join operands. However, the size of the graph grows exponentially with the number of base relations. For n base relations, this representation requires $2 + \sum_{j=1}^{n-1} (j\,(j+1))$ vertices, i.e., a start and an end vertex, as well as $n\,(n-1)$ vertices for the first join, $(n-1)\,(n-2)$ vertices for the second join, etc. These vertices are connected by means of $n\,(n-1)+2+\sum_{j=1}^{n-2} (j\,(j+1))\,((j+1)\,(j+2))$ edges, i.e., $n\,(n-1)$ edges connecting the start vertex to the vertices for the

first join, 2 edges connecting the vertices of the last join to the end vertex, $n\,(n-1)\,((n-1)\,(n-2))$ edges connecting the vertices for the first join to those for the second join, etc. For, e.g., 21 base relations (20 joins), the graph would thus consist of 3,082 vertices and 720,218 edges. Graphs of such sizes can however be easily handled by today's typical desktop computers.

We envisage our approach to be a promising way of guiding the optimization process by means of iteratively constructing solutions, partly guided by previously encountered low-cost solutions and partly guided by local heuristics. One of the strengths of our approach is in that – when run continuously – the ants can compensate for changes in the configuration of the graph as well as for changes in the costs associated with making an arbitrary join, possibly caused by updated data sources (yielding changes in cardinalities) or latency differences (affecting transmission costs in a distributed setting). Moreover, our approach has an attractive advantage over GAs in that it retains the memory of the entire colony over time. Depending on the evaporation rate, our algorithm remembers all traversed paths, whereas GAs only retain the memory of the last generation, in which typically only the best (partial) solutions have left their traces.

5. Evaluation

Because of its characteristics, our proposed ACO algorithm appears to be an attractive alternative to existing soft computing techniques for RDF chain query optimization. In order to assess its potential, we evaluate our approach and compare its performance to the performance of existing soft computing techniques in a Semantic Web environment.

5.1. Experimental Setup

The performance of our considered approaches is assessed on a 32-bit 2.66 GHz Intel Core 2 Duo desktop computer with 2 GB physical memory. On this machine, we evaluate the performance of RDF chain query optimization by means of 2PO (RCQ-2PO), a GA (RCQ-GA), and our novel ACO (RCQ-ACO) on a single source, i.e., an RDF version of the CIA World Factbook, generated by using QMap (Hogenboom et al., 2008). We use this source to evaluate our considered approaches on RDF chain queries varying in length from 3 to 21 predicates, thus covering problem sizes of 2 to 20 joins. For each query length, we evaluate each method's execution times until convergence and costs of found solutions over 100 chain queries on our RDF data. In our experiments, we consider the complete solution space with bushy query trees. We assess statistical significance of observed differences by means of a paired, two-sided Wilcoxon

signed-rank test, evaluating the hypothesis that these differences are symmetrically distributed around a median of 0.

The considered algorithms need to be configured for our current purposes. For RCQ-2PO, we adopt the settings proposed in Steinbrunn et al. (1997). As such, we start the II phase of the process with 10 random starting points for random walks in the solution space. Each step in this walk is a step from a solution to one of its better neighbors. The number of times the algorithm tries to find a better neighbor for a solution during such a walk is limited to that solution's number of neighbors. The best local optimum thus obtained is taken as starting point for another random walk in the SA phase, where the system's temperature is initialized at 10% of the starting solution's associated costs. For each solution on the path traversed through the solution space by means of SA, the algorithm tries to move to neighboring solutions for a limited number of times, which equals 16 times the number of joins in the query. After 16 tries, the system's temperature is reduced with 5%. The system is considered to be frozen when the temperature drops below 1 or when the best solution so far has not been improved in four consecutive temperature reductions.

The RCQ-GA algorithm is configured in accordance with the settings suggested in Hogenboom et al. (2009). A set of 64 chromosomes is exposed to a process of simulated evolution with a crossover rate of 0.65 and a mutation rate of 0.05. Fitness-based selection is applied and in each generation, the best chromosome is always selected for proliferation in the next generation. The GA is considered to have been converged after 30 consecutive generations without any improvement in terms of fitness of the best solution.

Our proposed RCQ-ACO algorithm has its roots in the classic Ant System (Dorigo et al., 1996), which has a prototypical application in constructing a solution to the Traveling Salesman Problem (TSP). In a TSP, the goal is to find the shortest closed tour between a set of cities, such that each city is visited exactly once. The number of ants in the classic Ant System typically equals the number of cities in the TSP to be solved. The α parameter representing the importance of the information conveyed by global pheromone traces equals 1, whereas the importance of local heuristics β equals 5. The evaporation rate ρ equals 0.5 and the constant Q is set to 100. In our application, we have to deal with relatively large graphs. Therefore, we propose to use four times as many ants as the number of joins. Additionally, we propose to stimulate quicker convergence to relatively good solutions by relying more on the information conveyed by global pheromone traces than the classic Ant System does. In this light, we propose to increase α to 2 as well as to reduce ρ to 0.25. Finally, we consider the colony to have been converged after five consecutive iterations without any improvement in solution quality.

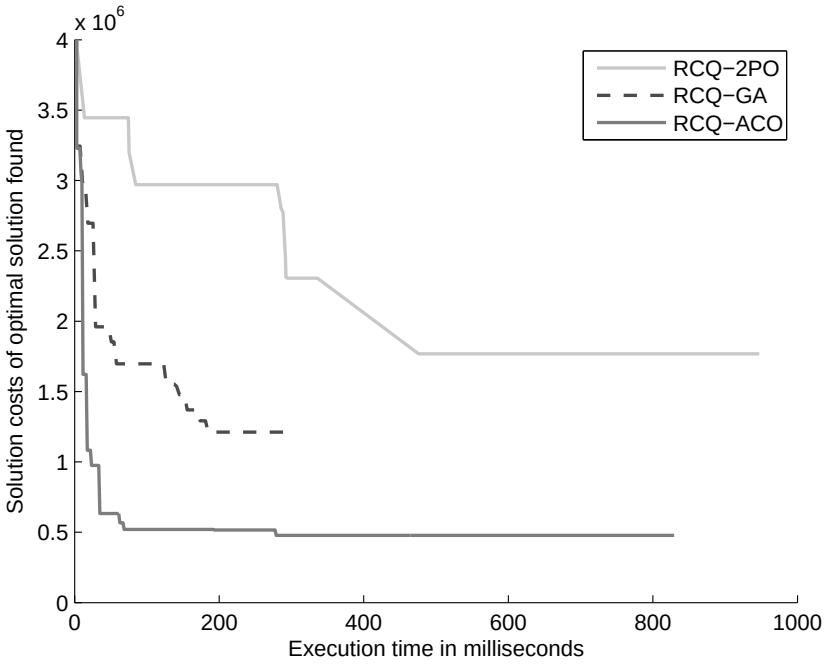


Figure 6: Performance over time when optimizing a typical 20-join RDF chain query.

5.2. Experimental Results

Each of the considered soft computing techniques for RDF chain query optimization has its own typical way of moving through the solution space. As Figure 6 demonstrates for the optimization of an arbitrary 20-join RDF chain query on our data set, RCQ-2PO tends to steadily move through the search space until a local optimum is reached. RCQ-GA on the other hand quickly moves to more fruitful parts of the solution space. Guided by the perceived quality of encountered solutions, the algorithm does this by means of a more randomized walk through the solution space, facilitated by genetic operations such as mutations and crossovers. RCQ-ACO exhibits a swift move to a fruitful part of the solution space as well. The artificial ants use their local heuristics to collectively construct a rather good initial solution and then predominantly search the proximity of this solution in a randomized way, while being increasingly more attracted to the collective solution thus emerging. By doing so, the ants collectively find a comparably good local optimum in a small amount of time, but spend most of their time optimizing this result until convergence.

Table 1: Relative performance differences in terms of mean costs of optimal solutions found (columns 2 to 4) and mean execution times of the optimization process (columns 5 to 7) for RDF chain queries of various lengths (column 1). Columns 2 and 5 represent performance differences between RCQ-ACO and RCQ-GA in terms of RCQ-GA's performance, columns 3 and 6 represent performance differences between RCQ-ACO and RCQ-2PO relative to the performance of RCQ-2PO, and columns 4 and 7 express the differences between RCQ-GA and RCQ-2PO in terms of RCQ-2PO's performance, where *, **, and *** denote respective significance levels of 0.01, 0.001, and 0.0001.

	Differences in solution costs			Differences in execution times		
Joins	ACO/GA	ACO/2PO	GA/2PO	ACO/GA	ACO/2PO	GA/2PO
2	0.0%	0.0%	0.0%	-80.5%***	-63.7%***	86.1%***
3	0.0%	0.0%	0.0%	-77.5%***	-74.6%***	12.8%***
4	0.0%	-2.1%	-2.1%**	-71.9%***	-77.6%***	-20.4%***
5	0.0%	-4.6%***	-4.6%***	-64.4%***	-78.7%***	-40.1%***
6	-0.4%	-7.9%***	-7.6%***	-57.5%***	-79.3%***	-51.2%***
7	-0.6%**	-17.6%***	-17.1%***	-53.5%***	-78.0%***	-52.8%***
8	-1.3%***	-20.2%***	-19.2%***	-50.7%***	-78.5%***	-56.4%***
9	-1.8%***	-27.0%***	-25.7%***	-46.3%***	-76.1%***	-55.5%***
10	-5.8%***	-30.4%***	-26.2%***	-44.1%***	-75.1%***	-55.4%***
11	-7.5%***	-36.8%***	-31.6%***	-37.7%***	-73.2%***	-57.0%***
12	-9.1%***	-37.2%***	-30.9%***	-29.0%***	-70.2%***	-58.0%***
13	-12.3%***	-39.1%***	-30.5%***	-16.4%*	-66.5%***	-60.0%***
14	-16.3%***	-43.3%***	-32.3%***	-19.4%*	-63.3%***	-54.5%***
15	-19.7%***	-46.0%***	-32.7%***	9.4%*	-58.8%***	-62.4%***
16	-22.6%***	-46.3%***	-30.6%***	16.8%**	-54.6%***	-61.1%***
17	-27.8%***	-47.1%***	-26.7%***	50.7%***	-50.6%***	-67.2%***
18	-30.5%***	-49.8%***	-27.7%***	50.0%***	-45.3%***	-63.6%***
19	-32.2%***	-51.3%***	-28.1%***	66.8%***	-45.5%***	-67.4%***
20	-34.9%***	-53.4%***	-28.5%***	90.1%***	-34.2%***	-65.4%***

Their nature renders some of the considered soft computing techniques more suitable for RDF chain query optimization than others. Supported by Table 1, Figures 7 and 8 clearly demonstrate that on average, RCQ-2PO is typically the slowest algorithm, yielding the worst results on our data set for nearly all considered query lengths. However, for RDF chain queries consisting of two or three joins, all considered algorithms yield solutions of comparable quality.

For our considered RDF chain queries consisting of up to about 15 joins, RCQ-ACO is typically the fastest performing algorithm, yielding the best solutions. Compared to both RCQ-2PO and RCQ-GA, our novel ACO approach needs up to approximately 80% less time to converge. As the query size increases, the differences in mean execution times tend to decrease. For queries consisting of 15 joins, RCQ-ACO needs about 60% less time to converge than RCQ-2PO, whereas the advantage of RCQ-ACO over RCQ-GA in terms of execution time is even nullified for queries consisting of 15 joins or more. Besides being faster than RCQ-2PO and RCQ-GA, our novel ACO outperforms these existing soft computing techniques for RDF

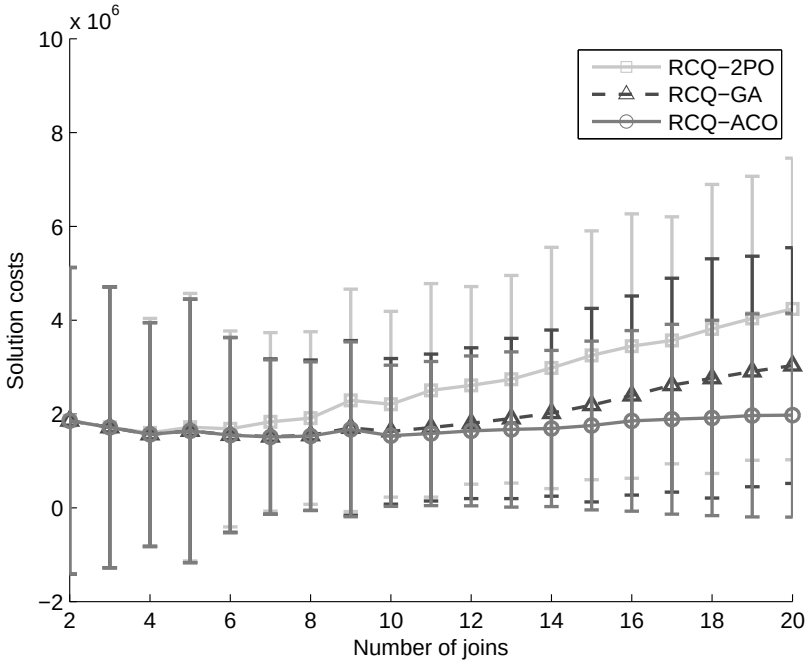


Figure 7: Mean and standard deviations of costs of optimal solutions found for queries of various lengths.

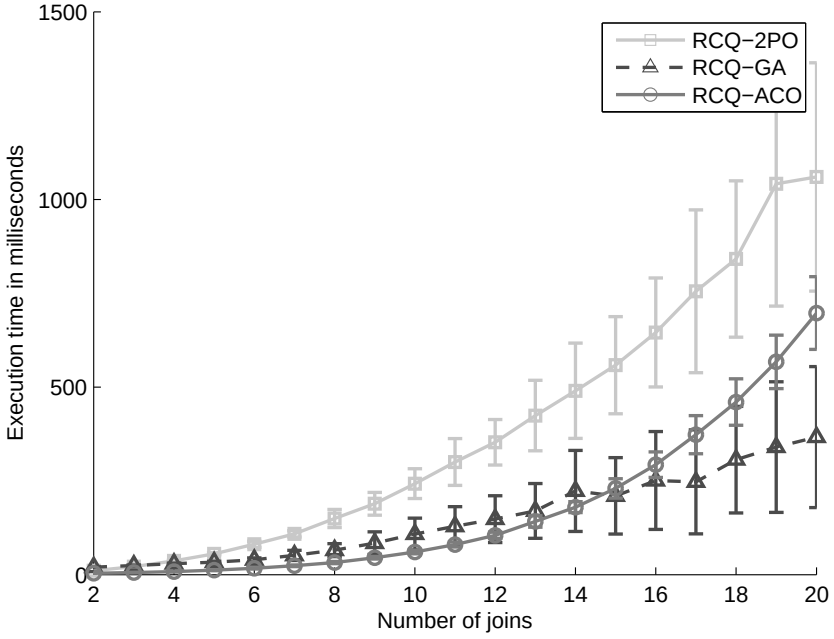


Figure 8: Mean and standard deviations of execution times until convergence for queries of various lengths.

chain query optimization in terms of solution quality as well. The improvement of the quality of the best solution found by RCQ-ACO over the quality RCQ-2PO's solutions increases from approximately 5% for queries of length 5 to 45% for queries of length 15. The extent to which RCQ-ACO outperforms RCQ-GA in terms of solution quality increases from about 5% for queries of length 10 to 20% for queries consisting of 15 joins.

Yet, for larger RDF chain queries, our results indicate that one may want to use either RCQ-GA or RCQ-ACO rather than RCQ-2PO for RDF chain query optimization, subject to a trade-off between execution time and solution quality. As the query size increases, using RCQ-ACO can typically lead to increasingly large solution cost reductions up to over 50% compared to RCQ-2PO, and increasingly large solution cost reductions up to over 30% compared to RCQ-GA. However, these improvements come at a cost of less favorable execution times. For larger queries, RCQ-ACO tends to be increasingly slower than RCQ-GA, while still being approximately 55% to 35% faster than RCQ-2PO for queries consisting of 16 to 20 joins, respectively. RCQ-GA on the other hand yields solutions with costs that are consistently about 30% lower than those associated with solutions suggested by RCQ-2PO. In addition, RCQ-GA consistently needs 65% less execution time than RCQ-2PO in order to reach convergence.

These results suggest that our novel RCQ-ACO algorithm is a promising alternative to both RCQ-2PO and RCQ-GA, as it typically outperforms both approaches in terms of execution time and solution quality for RDF chain queries consisting of up to about 15 joins. For larger queries, RCQ-GA has the attractive advantage of delivering relatively good solutions in relatively little time, whereas RCQ-ACO needs more time than RCQ-GA, but less time than RCQ-2PO, in order to deliver the best solutions.

6. Conclusions and Future Work

In this paper, we have demonstrated how ACO can facilitate effective and efficient chain querying in expert systems for decision support in a Semantic Web environment, by having artificial ants iteratively construct RDF chain query paths, partly guided by the perceived quality of previously encountered solutions and partly guided by local heuristics. On a large, single RDF source, our experimental results demonstrate the potential of our novel ACO approach to RDF chain query optimization in comparison to previously proposed soft computing techniques, i.e., 2PO and a GA. When exploring the large solution spaces associated with the join ordering problem arising when optimizing RDF chain queries, our ACO approach significantly outperforms the existing 2PO and GA approaches in terms of solution quality and execution time for RDF

chain queries consisting of up to 15 joins. For larger queries, consisting of up to 20 joins, the existing GA delivers relatively good solutions in a comparably short time frame. However, our novel ACO approach delivers by far the best solutions in a shorter time frame than 2PO, albeit while needing more time than the recently proposed GA in order to reach convergence.

As its design allows our proposed ACO for RDF chain query optimization to adapt to dynamic environments, our next step will be assessing the performance of our novel algorithm in a dynamic RDF environment with multiple sources. In such an environment, we can explore how our algorithm can best adapt to changes in the environment, caused by, e.g., latency differences or updated data sources. In this light, we also consider real-time updating of our join selectivity estimation, which in our current set-up has a fixed value, as it depends on the cardinalities of a join's operands. Another direction for future work lies in optimizing the parameters of our ACO algorithm. Finally, we also aim to investigate how to devise a more scalable graph representation of the join ordering problem in RDF chain query optimization, as we envisage a more scalable graph representation to improve the performance of our ACO approach even further.

References

Aksac, A., Ozturk, O., Dogdu, E., 2012. A Novel Semantic Web Browser for User Centric Information Retrieval: PERSON. Expert Systems with Applications 39 (15), 12001–12013.

Amin, G., Emrouznejad, A., 2011. Optimizing Search Engines Results Using Linear Programming. Expert Systems with Applications 38 (9), 11534–11537.

Batzios, A., Dimou, C., Symeonidis, A., Mitkas, P., 2008. BioCrawler: An Intelligent Crawler for the Semantic Web. Expert Systems with Applications 35 (1), 524–530.

Batzios, A., Mitkas, P., 2012. WebOWL: A Semantic Web Search Engine Development Experiment. Expert Systems with Applications 39 (5), 5052–5060.

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. Scientific American 284 (5), 34–43.

Blanco-Fernandez, Y., Pazos-Arias, J., Lopez-Nores, M., Gil-Solla, A., Ramos-Cabrer, M., Garcia-Duque, J., Fernandez-Vilas, A., Diaz-Redondo, R., 2010. Incentivized Provision of Metadata, Semantic Reasoning and Time-Driven Filtering: Making a Puzzle of Personalized E-Commerce. Expert Systems with Applications 37 (1), 61–69.

Dorigo, M., Birattari, M., Stutzle, T., 2006. Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique. IEEE Computational Intelligence Magazine 1 (4), 28–39.

Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B 26 (1), 29–41.

Elmasri, R., Navathe, S., 2004. Fundamentals of Database Systems, 4th Edition. Addison-Wesley.

Frasincar, F., Houben, G., Vdovjak, R., Barna, P., 2004. RAL: An Algebra for Querying RDF. World Wide Web Journal 7 (1), 83–109.

Gambardella, L., Dorigo, M., 2000. Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. INFORMS Journal on Computing 12 (3), 237–255.

Garcia-Crespo, A., Lopez-Cuadrado, J., Colomo-Palacios, R., Gonzalez-Carrasco, I., Ruiz-Mezcua, B., 2011. Sem-Fit: A Semantic Based Expert System to Provide Recommendations in the Tourism Domain. Expert Systems with Applications 38 (10), 13310–13319.

Hogenboom, A., Hogenboom, F., Frasincar, F., Schouten, K., van der Meer, O., 2012. Semantics-Based Information Extraction for Detecting Economic Events. Multimedia Tools and Applications, Online First (DOI: 10.1007/s11042-012-1122-0).

Hogenboom, A., Milea, V., Frasincar, F., Kaymak, U., 2009. RCQ-GA: RDF Chain Query Optimization Using Genetic Algorithms. In: Di Noia, T., Buccafurri, F. (Eds.), Tenth International Conference on Electronic Commerce and Web Technologies (EC-Web 2009). Vol. 5692 of Lecture Notes in Computer Science. Springer, pp. 181–192.

Hogenboom, F., Hogenboom, A., van Gelder, R., Milea, V., Frasincar, F., Kaymak, U., 2008. QMap: An RDF-Based Queryable World Map. In: Naaranoja, M. (Ed.), Third International Conference on Knowledge Management in Organizations (KMO 2008). Vaasan Yliopiston Julkaisuja, pp. 99–110.

Holland, J., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press.

Hsu, I., 2009. SXRS: An XLink-Based Recommender System using Semantic Web Technologies. Expert Systems with Applications 36 (2), 3795–3804.

Ioannidis, Y., Kang, Y., 1990. Randomized Algorithms for Optimizing Large Join Queries. In: 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD 1990). ACM, pp. 312–321.

Joo, J., Lee, S., 2009. Adoption of the Semantic Web for Overcoming Technical Limitations of Knowledge Management Systems. Expert Systems with Applications 36 (3), 7318–7327.

Jung, J., 2010. An Evolutionary Approach to Query-Sampling for Heterogeneous Systems. Expert Systems with Applications 37 (1), 226–232.

Klyne, G., Carroll, J., 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax – W3C Recommendation 10 February 2004.

Lupiani-Ruiz, E., Garcia-Manotas, I., Valencia-Garcia, R., Garcia-Sanchez, F., Castellanos-Nieves, D., Fernandez-Breis, J., Camon-Herrero, J., 2011. Financial News Semantic Search Engine. Expert Systems with Applications 38 (12), 15565–15572.

Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant Colony Optimization for Resource-Constrained Project Scheduling. IEEE Transactions on Evolutionary Computation 6 (4), 333–346.

Prud'hommeaux, E., Seaborne, A., 2008. SPARQL Query Language for RDF – W3C Recommendation 15 January 2008.

Selinger, P., Astrahan, M., Chamberli, D., Lorie, R., Price, T., 1979. Access Path Selection in a Relational Database Management System. In: 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD 1979). ACM, pp. 23–34.

Steinbrunn, M., Moerkotte, G., Kemper, A., 1997. Heuristic and Randomized Optimization for the Join Ordering Problem. The VLDB Journal 6 (3), 191–208.

Stuckenschmidt, H., Vdovjak, R., Broekstra, J., Houben, G., 2005. Towards Distributed Processing of RDF Path Queries. International Journal of Web Engineering and Technology 2 (2-3), 207–230.

Vandic, D., van Dam, J., Frasincar, F., 2012. Faceted Product Search Powered by the Semantic Web. Decision Support Systems 53 (3), 425–437.