RCQ-ACS: RDF Chain Query Optimization Using an Ant Colony System

Alexander Hogenboom, Ewout Niewenhuijse, Frederik Hogenboom, and Flavius Frasincar

Erasmus University Rotterdam
Rotterdam, The Netherlands
hogenboom@ese.eur.nl, ewoutnwn@gmail.com, fhogenboom@ese.eur.nl, frasincar@ese.eur.nl

***Abstract*—In order to effectively and efficiently disclose the ever-growing amount of widely distributed RDF data to demanding users in real-time environments, RDF query engines need to optimize the join order of partial query results. For this, a two-phase optimization (2PO) algorithm and a genetic algorithm (GA) have already been proposed. We propose an alternative approach – an ant colony system (ACS). On a large RDF data source, our approach significantly outperforms both 2PO and the GA in terms of execution time and solution quality for RDF chain queries consisting of up to about ten joins. For larger queries, our novel ACS delivers solutions of better quality than 2PO does, while realizing a solution quality that is comparable to the solution quality of the GA method. However, the GA approach offers the best trade-off between execution time and solution quality for such larger queries.**

***Index Terms*—RDF chain query optimization, ant colony system.**

I. Introduction

With the rise of the Semantic Web [1], an ever-growing amount of data is stored in many heterogenous, yet interconnected sources. This data is typically represented by means of the Resource Description Framework (RDF), a World Wide Web Consortium (W3C) framework for describing and interchanging meta-data [2], which enable machine-interpretability by describing the context of data. 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.

Semantic Web technologies allow exploration of many different RDF data sources in order to address very specific information needs. Queries can be executed by means of W3C's SPARQL Protocol and RDF Query Language (SPARQL) [3]. Fast RDF query engines are crucial in order for SPARQL queries to disclose the ever-growing amount of widely distributed RDF data to demanding users in real-time environments. One of the problems here is the optimization of the join order of partial query results, i.e., query paths. The total query execution time depends on such query paths. A good algorithm for join order optimization in a query path can hence contribute to efficient querying. As the number of possible query paths grows exponentially with the query size, the optimization of RDF query paths is challenging. Several methods have already been proposed to address this problem. One of these methods is a two-phase optimization (2PO) algorithm [4], 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 [5].

As their design has been inspired by methods for query path optimization in traditional databases, existing 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, in the dynamic Semantic Web, data changes, sources come and go, and latency between sources may be volatile, which has inspired recent research into adding robustness to the Semantic Web [6]. In this light, an ant colony optimization (ACO) [7] approach in the form of an ant colony system (ACS) [8] is a good alternative to existing approaches in an RDF environment. ACO is a 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 [9] and sequential ordering [10].

As its characteristics make an ACS an attractive alternative to existing RDF query optimization approaches, we explore its applicability to query path optimization in an RDF environment and we compare its performance with the performance of existing RDF query optimization methods. In this work, we focus on the performance of the considered algorithms when optimizing a specific type of SPARQL queries, the so-called RDF chain queries, on a single source.

The remainder of this paper is organized as follows. First, Section II provides a short introduction to RDF, chain queries, and chain query optimization. We then discuss and evaluate our novel ACS approach to RDF chain query optimization in Sections III and IV, respectively. Last, we draw conclusions and propose directions for future work in Section V.

II. RDF and Chain Query Optimization

In an RDF model, facts are declared as a collection of triples, each of which consists 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, where each triple is represented as a node-arc-node link [2]. An arc, denoting a predicate, is used to define the relationship between a subject node and an object node.

When querying RDF sources, RDF triples are essentially matched with a series of patterns specified in a SPARQL query. A specific type of patterns is a so-called chain query, where 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, which are the focus of our current endeavors, resemble chain queries in traditional relational databases, where a query path is followed by performing joins between its subpaths of length 1 [4]. Randomized and genetic algorithms have proven to outperform heuristic methods in optimizing these types of queries in relational databases [11].

For an example of an RDF chain query, let us consider an RDF model of the CIA World Factbook [12], containing over 100,000 triples capturing data about approximately 250 countries. Suppose a financial risk analyst wants to identify the export partners of the Netherlands that have dependent areas (i.e., areas not possessing full political independence or sovereignty) that are involved in an international conflict. This can be expressed in a SPARQL chain query, as shown in Fig. 1.

```
1 PREFIX c: <http://daml.org/2001/09/countries/fips#>
2 PREFIX o: <http://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       }
```

Fig. 1. Example RDF chain query in SPARQL.

This SPARQL chain query can be subdivided into four subqueries. The first subquery is a query for information on the export partners of the Netherlands (line 4). Other subqueries are a query for countries actually 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 subqueries can be joined in any order. In this process, the number of statements resulting from a join (between two triple patterns) equals the number of statements compliant with the constraints of both operands. Irrespective of the order of the joins of partial query results, the result of a query will always be the same. However, the total execution time of a query depends on the order of joins, i.e., the query path. Therefore, query path optimization is crucial in today's real-time RDF environments.

A. Solution Space

The order of joins in a 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 order in which the joins between partial query results should be made, while minimizing the overall execution time of the query. Typical methods for this problem involve exploring a solution space in an attempt to find low-cost solutions.

A sequence of joins in an RDF chain 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 a user to specify basic retrieval requests on these inputs [13]. In our current endeavors, 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 left-deep query trees can be considered. In bushy trees, base relations (containing matches with one triple pattern) as well as results of earlier joins can be joined. Left-deep trees require the right-hand join operands to be base relations. Figure 2 depicts a bushy tree and a left-deep tree for our example query. In these trees, matches with triple patterns $\{t_1, t_2, t_3, t_4\}$ – corresponding with lines 4 through 7 of our query, respectively – are joined, with $\bowtie$ representing a join.

Each solution in the solution space represents a query path. For n base relations, and hence $n-1$ joins, $n!$ different left-deep query tree solutions exist. However, when also considering bushy query trees, $\binom{2(n-1)}{(n-1)}(n-1)!$ solutions exist, which may outperform left-deep solutions in quality for seven in ten chain queries, as shown in a collection of 100 five-join chain queries [11]. Solutions are located in the solution space such 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 into an equivalent solution by applying one out 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 [14].

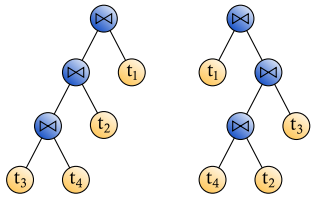
B. Solution Costs

In the solution space, each solution is associated with execution costs, typically constituted by the costs of data transmission from the source to the processor and the costs of processing this data [4]. As in this paper, we focus 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.

The costs of a join are typically influenced by the cardinalities of the operands and the join method used. Several methods can be used for implementing (two-way) joins [15]. 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). Therefore, we consider only nested-loop joins in this paper. When performing a nested-loop join, all elements in both join operands need to be compared with one another. In this light, when joining the first operand o_{s_j1} and the second operand o_{s_j2} for each join j in a solution s with n base relations, we define the total execution costs c_s as

$$c_s = \sum_{j=1}^{n-1} \left|o_{s_j1}\right| \left|o_{s_j2}\right|, \tag{1}$$

where $\left|o_{s_j1}\right|$ and $\left|o_{s_j2}\right|$ represent the cardinalities of the first and second join operands of join j for solution s, respectively.



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

Fig. 2. Examples of query trees for an RDF chain query with three joins. Yellow (light gray) nodes represent matches with triple patterns, whereas blue (dark gray) nodes represent joins.

For base relations, cardinalities can be derived from the data, i.e., by counting the number of triples associated with a predicate. 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, as some join selectivity may take place, we define the cardinality $\left|o_{s_j}\right|$ of join j for solution s as

$$\left|o_{s_j}\right| = \left|o_{s_j 1}\right| \left|o_{s_j 2}\right| \sigma_{s_j}, \tag{2}$$

with σ_{s_j} representing 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, we use a rule of thumb stemming from the field of traditional databases by assuming the join selectivity to be 10% [16]. In a real-time environment, such an estimation could of course be updated over time.

C. Existing Approaches to RDF Chain Query Optimization

The challenge in RDF query path determination lies in minimizing query execution costs in often large solution spaces. Queries on RDF sources could be translated into algebraic expressions [5], which can subsequently be optimized by applying transformation rules for relational algebraic expressions [13], [15]. Yet, such heuristics are not sufficient in complex solution spaces. Several methods have been shown to yield better results in traditional query execution environments [11]. Inspired by these results, a small body of work has been dedicated to applying some of these methods to query optimization in the context of the Semantic Web [4], [5].

One of the first methods for exploring the solution space of RDF query paths is the 2PO algorithm [4]. In the first phase of this approach, 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. When in a walk a solution is reached for which no better neighbor can be found in a limited number of tries, the current solution is considered to be a local optimum. In the second phase of the 2PO approach, a SA algorithm takes the best local optimum thus found as a starting point for another walk, yet of a different nature, in the solution space, while simulating a continuous temperature reduction of a system until the system is considered to be frozen. The probability of the algorithm to accept moves not yielding improvement is direct proportional to the system's temperature and is inversely proportional to the absolute 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 [5].

A GA has recently been proposed as an alternative to 2PO for optimizing (large) RDF chain queries [5]. A GA is an optimization approach in which biological evolution according to the principle of survival of the fittest is simulated. A population of chromosomes – representing solutions – is exposed to evolutionary operations, consisting of selection (where individual chromosomes are selected for proliferation in the next generation), crossovers (creating offspring by combining chromosomes), and mutations (randomly altering 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 chances of its survival and is inversely proportional to the execution cost of the associated solution. Initial results demonstrate a promising performance compared to 2PO in terms of both execution time and solution costs, in particular for larger queries [5].

III. Optimizing RDF Chain Queries using Ant Colonies

As existing RDF chain query path optimization approaches 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, existing optimization techniques typically need to be run all over again. Yet, in a dynamic Semantic Web environment, data continuously changes, data sources come and go, and transmission costs between sources may vary over time. Therefore, we propose to optimize RDF chain queries by using ant colonies, as ACO-based approaches can be run continuously and can adapt to changes in the environment in real time. Moreover, such approaches typically outperform GAs in solving complex problems such as scheduling [9] and sequential ordering [10].

A. Ant Colony Optimization

ACO is an optimization method inspired by ant colonies' foraging behavior [7], i.e., ants walk between their nest and a food source marking 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 an increasing number of pheromone deposits, as they are traversed with increasing frequency because of their length as well as the pheromone traces on these paths. The ant colony thus converges to using a relatively short path.

ACO is essentially a population-based 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 trying to find a shortest path through the graph. A classic example of such a problem is the Traveling Salesman Problem (TSP), where the goal is to find the shortest closed tour between a set of cities, such that each city is visited exactly once. In a graph representation of TSP, vertices typically represent cities, whereas edges connecting two vertices are associated with between-city distances. ACO can be applied in order to let some artificial ants construct tours. After completing a tour, each ant deposits a pheromone trace on the edges of its tour, proportional to the length of the solution found.

Typically, each iteration of the ACO 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 observed quality of the path.

In the classic Ant System [7], an ant k constructs a path by iteratively moving from its most recently visited vertex x to another vertex y that the ant has not visited yet. The probability $p_{xy}^{k}(i)$ for an ant k to move from vertex x to an unvisited vertex y at iteration i of the algorithm is defined as

$$p_{xy}^{k}(i) = \begin{cases} \frac{[\tau_{xy}(i-1)]^{\alpha}[\eta_{xy}]^{\beta}}{\sum_{z \in V_k(x)}[\tau_{xz}(i-1)]^{\alpha}[\eta_{xz}]^{\beta}}, & z \in V_k(x), \\ 0, & z \notin V_k(x), \end{cases} \tag{3}$$

where $\tau_{xy}(i-1)$ quantifies 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 vertex x and vertex 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.

Based on the latest experience 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

$$\tau_{xy}(i) = (1-\rho)\,\tau_{xy}(i-1) + \sum_{k=1}^{m} \Delta\tau_{xy}^{k}(i), \tag{4}$$

where ρ represents an evaporation rate helping the colony not to converge to a local optimum and $\Delta\tau_{xy}^{k}(i)$ captures 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 $T_k(i)$ taken by ant k at iteration i, i.e.,

$$\Delta\tau_{xy}^{k}(i) = \begin{cases} \frac{Q}{L_k}, & w_{xy} \in T_k(i), \\ 0, & w_{xy} \notin T_k(i). \end{cases} \tag{5}$$

B. Ant Colony System

The classic Ant System has known several successors, each with its specific focus. One of these variants is the Ant Colony System (ACS) [8], which is typically faster converging and more efficient than the classic Ant System. The ACS algorithm differs from the Ant System in three ways.

First, the probability distributions used by ants when walking through the graph are changed with a probability $0 \leq q \leq 1$ in such a way that ants occasionally simply select the edge with the highest probability, i.e.,

$$p_{xy}^{k}(i) = \begin{cases} \text{as in (3)}, & r_s^k(i) > q, \\ 1, & r_s^k(i) \leq q, y = y^*, \\ 0, & r_s^k(i) \leq q, y \neq y^*, \end{cases} \tag{6}$$

where $r_s^k(i)$ is a random number between 0 and 1 for step s of ant k at iteration i, and where y^* is the vertex connected with the edge with the highest probability, i.e.,

$$y^* = \underset{z \in V_k(x)}{\arg\max} \left([\tau_{xz}(i-1)]^{\alpha} [\eta_{xz}]^{\beta} \right). \tag{7}$$

This pushes the algorithm to faster convergence. In order to find a balance between fast convergence to known solutions on the one hand, and exploration of other parts of the solution space on the other hand, the parameter q representing the probability of an ant selecting the best edge can be adjusted.

Additionally, ACS differs from the Ant System in that ants only deposit traces of pheromone on the edges that constitute the best-so-far solution, rather than on all edges visited by ants in the latest iteration. This also stimulates faster convergence of the ant colony to the best solution encountered so far.

A final difference between ACS and the Ant System lies in the pheromone evaporation process. In ACS, pheromone evaporation only takes place on those edges that have been visited by ants, rather than on all edges. This stimulates ants to find their way into unexplored parts of the solution space.

C. Problem Representation

Because the classic design of the Ant System enables ants to find their way around changes in the graph, and the ACS variant has been designed as a faster converging and thus more efficient variant, the application of ACS to RDF chain query optimization is an attractive alternative to the existing 2PO and GA approaches. Therefore, we propose an ACS-based algorithm utilizing a graph representation of an ordinal number encoding scheme [11], 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.

The 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 a chain query with n base relations, the first pair of numbers consists of two unique integers ranging from 1 to n, the second pair consists of two unique integers ranging from 1 to $n-1$, etc.

This encoding scheme can be illustrated by means of the bushy query tree for our example chain query, presented in Fig. 2(b). First, consider the ordered list of matches with triple patterns $\{\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3, \mathrm{t}_4\}$. An initial join between the fourth and second triple pattern yields the list $\{\mathrm{t}_1, \mathrm{t}_4\mathrm{t}_2, \mathrm{t}_3\}$. A subsequent join between the second and third operand in this new list

yields $\{t_1, t_4t_2t_3\}$. A final join between the first and second operand in the latter list results in $\{t_1t_4t_2t_3\}$. This join order would be encoded as $((4,2),(2,3),(1,2))$.

As Fig. 3 shows for the bushy query tree presented in Section II, the ants find their way through a directed graph. They walk 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. Each path represents a join sequence, with each step representing a join.

For an arbitrary join (visualized as a node column in Fig. 3), the vertices in the graph represent all valid combinations of indices of join operands, in accordance with our applied ordinal encoding scheme [11]. An ant's observed distance associated with an arbitrary 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., by applying (1) and (2). The observed distances associated with edges, as well as the pheromone traces on these respective edges guide the ants' decisions in accordance with (6).

At every iteration, new pheromone traces, which are inversely proportional to the associated paths' total execution costs, are deposited on the best-so-far path and pheromone traces on all visited paths evaporate to a certain extent. When the ants have not encountered a better path in some number of iterations, the algorithm is considered to have been converged.

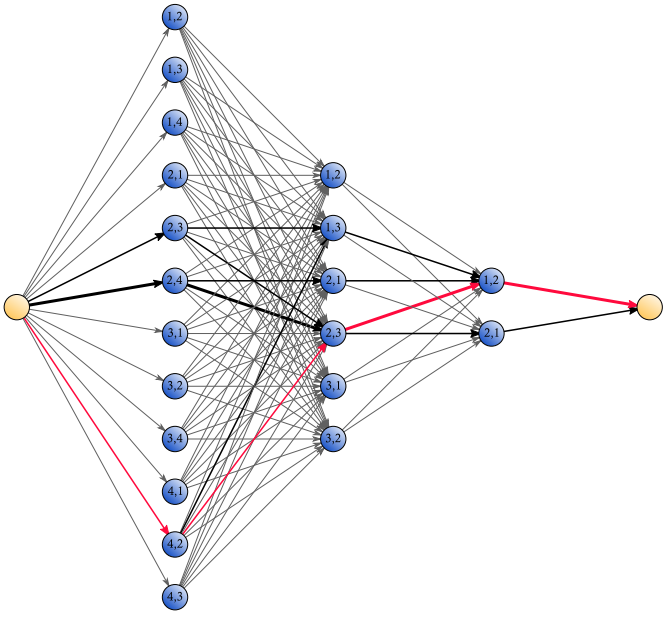


Fig. 3. Graph representation of the join ordering problem, as used in our proposed ACS approach to RDF chain query optimization. A colony of artificial ants iteratively traverses this graph from a starting vertex to an end vertex, both marked yellow (light gray). At each step, an ant selects one join from all possible alternatives, represented as columns of blue (dark gray) vertices, in accordance with an ordinal number encoding scheme. 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 Fig. 2(b) is marked red (dark gray).

Our approach can aid RDF query path optimization by means of iteratively constructing solutions, partly guided by encountered low-cost solutions and partly guided by local heuristics. A strength of our approach is in that – when run continuously – ants could theoretically compensate on the fly for changes in the graph's configuration 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), which are not uncommon in a Semantic Web setting [6]. Yet, assessing this type of robustness is not the focus of our current work, as we aim to compare the performance of our novel method with existing RDF chain query optimization methods in a setting that allows for easy comparison of the considered approaches in terms of execution time and solution quality, without noise introduced by (different reactions to) changes in the environment.

IV. Evaluation

Because of its characteristics described in Section III, our ACS algorithm appears to be an attractive alternative to the existing 2PO and GA methods for RDF chain query optimization. In order to assess its potential, we evaluate our approach and compare it to the performance of existing techniques in a Semantic Web environment.

A. Experimental Setup

The performance of our considered approaches is assessed on a 32-bit 2.66 GHz Intel Core 2 Duo machine with 2 GB physical memory. We evaluate the performance of RDF chain query optimization by means of 2PO (RCQ-2PO), a GA (RCQ-GA), and our novel ACS (RCQ-ACS) on an RDF version of the CIA World Factbook [12]. We evaluate our considered methods on RDF chain queries varying in length from 3 to 20 predicates, i.e., 2 to 19 joins. For each query length, we evaluate each method's execution times until convergence and costs of found solutions over 100 random chain queries on our RDF data. We consider the complete solution space with bushy query trees. Statistical significance of observed differences is assessed by means of a paired, two-sided Wilcoxon signed-rank test, evaluating the hypothesis that these differences are symmetrically distributed around a median equal to 0.

For RCQ-2PO, we adopt the settings proposed in [11]. We hence start the II phase of the process with 10 random starting points for random walks in the solution space. 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. In the SA phase, the system's temperature is initialized at 10% of the costs associated with the best local optimum obtained in the II phase. The number of times the algorithm tries to move to a neighboring solution is limited to 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.

RCQ-GA is configured in accordance with the settings suggested in [5]. As such, solutions are encoded into chromosomes by means of an efficient ordinal encoding scheme which facilitates easy evolutionary operations [11]. 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, while applying fitness-based selection. 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-ACS algorithm has its roots in the classic Ant System [7] and the related ACS [8]. Following these classic approaches, we propose to use a number of ants equal to the problem size, i.e., the number of joins. Additionally, we propose to stimulate quicker convergence to relatively good solutions by relying as heavily on the information conveyed by global pheromone traces as on local heuristics. In this light, we propose α and β to equal 1, with an evaporation rate ρ equal to 0.25 and the probability q of an ant selecting the best edge equal to 0.7. Additionally, the constant Q is set to 10. Finally, we consider the colony to have been converged after 30 consecutive iterations without any improvement in solution quality.

B. Experimental Results

Their nature renders some of the considered methods more suitable for RDF chain query optimization than others. Supported by Table I, Figures 4 and 5 clearly demonstrate that on average, some methods are suitable for a distinct range of queries. Additionally, observed relative differences in solution costs appear to be rather small, which is caused by our considered cost function. Apparently, our cost function yields a solution space in which the considered approaches find local optima which are relatively close together with respect to solution costs, even though absolute differences may be vast.

For RDF chain queries consisting of up to about ten joins, RCQ-2PO yields solutions of a similar quality as RCQ-GA. RCQ-ACS on the other hand significantly outperforms both RCQ-2PO and RCQ-GA in terms of solution quality, albeit while obtaining relatively small improvements in solution costs. For RDF chain queries consisting of two and three joins, all considered algorithms produce solutions of similar quality.

In terms of execution time, RCQ-GA is typically significantly outperformed by both RCQ-2PO and RCQ-ACS for the smaller RDF chain queries consisting of up to about ten joins. RCQ-GA needs up to approximately 120% more time to converge than RCQ-2PO. In turn, compared to RCQ-2PO, our novel ACS-based RDF chain query optimization approach needs up to about 80% less time to converge. Moreover, RCQ-ACS converges in up to approximately 90% less time than RCQ-GA. As the query size increases, the differences in mean execution times tend to decrease. All in all, for smaller RDF chain queries consisting of up to approximately ten joins, RCQ-ACS is typically the fastest performing algorithm, yielding the best solutions.

Yet, for RDF chain queries consisting of more than approximately ten joins, our results exhibit a different pattern. On our considered set of larger RDF chain queries, RCQ-2PO typically yields solutions of a significantly inferior quality, when compared to the solutions produced by both RCQ-GA and RCQ-ACS. Furthermore, for increasing query sizes, the solution quality of RCQ-GA moves towards the quality of solutions found by our novel ACS, leaving no significant difference for queries constituted by more than 12 joins.

However, for RCQ-ACS, the performance improvements compared to RCQ-2PO in terms of solution quality come at a cost of significantly less favorable execution times for larger queries. Compared to RCQ-2PO, the relative difference in execution time needed in order to reach convergence amounts to approximately 55% for queries consisting of 19 joins. In turn, RCQ-2PO needs on average up to over 80% more time

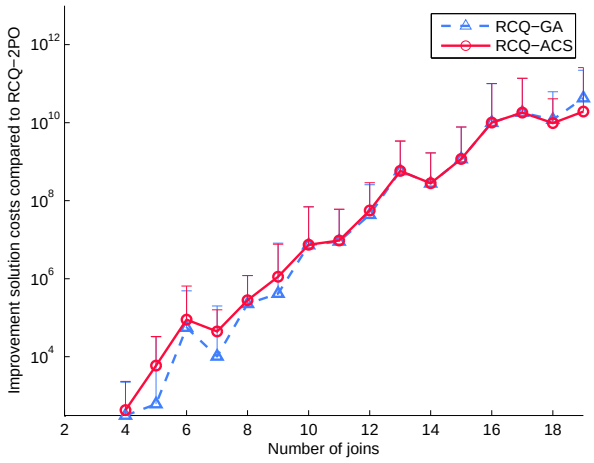


Fig. 4. Mean and standard deviations of costs of optimized solutions for queries of various lengths, in terms of improvement with respect to RCQ-2PO.

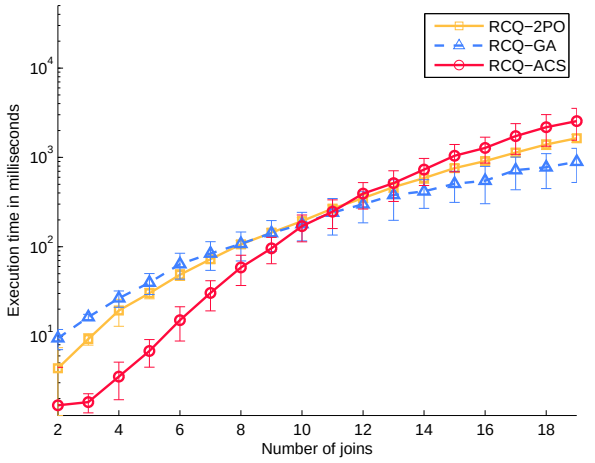


Fig. 5. Mean and standard deviations of execution times until convergence of the optimization algorithms for queries of various lengths.

TABLE I

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-ACS AND RCQ-GA RELATIVE TO THE PERFORMANCE OF RCQ-GA, COLUMNS 3 AND 6 REPRESENT PERFORMANCE DIFFERENCES BETWEEN RCQ-ACS AND RCQ-2PO RELATIVE TO THE PERFORMANCE OF RCQ-2PO, AND COLUMNS 4 AND 7 EXPRESS THE DIFFERENCES BETWEEN RCQ-GA AND RCQ-2PO RELATIVE TO THE PERFORMANCE OF RCQ-2PO, WHERE *, **, AND *** DENOTE RESPECTIVE SIGNIFICANCE LEVELS OF 0.01, 0.001, AND 0.0001.

	Relative differences in solution costs			Relative differences in execution times		
Joins	ACS/GA	ACS/2PO	GA/2PO	ACS/GA	ACS/2PO	GA/2PO
2	0.0E+00	0.0E+00	0.0E+00	-0.823***	-0.615***	1.169***
3	0.0E+00	0.0E+00	0.0E+00	-0.888***	-0.803***	0.760***
4	-3.5E−06*	-1.3E−05*	-9.4E−06	-0.867***	-0.818***	0.373***
5	-4.9E−06**	-5.5E−06**	-5.6E−07	-0.828***	-0.774***	0.312***
6	-5.8E−06***	-1.6E−05***	-9.7E−06	-0.765***	-0.691***	0.315***
7	-1.9E−07***	-2.4E−07***	-5.5E−08	-0.639***	-0.580***	0.161*
8	-1.4E−08***	-7.2E−08***	-5.8E−08	-0.456***	-0.450***	0.011
9	-5.5E−09***	-8.7E−09***	-3.2E−09	-0.325***	-0.331***	-0.009
10	-1.9E−10**	-7.4E−09***	-7.3E−09*	-0.052	-0.122***	-0.074
11	-4.2E−09*	-8.2E−08***	-7.8E−08***	0.026	-0.076*	-0.099**
12	-3.7E−10***	-1.7E−09***	-1.3E−09***	0.317***	0.123*	-0.147***
13	-2.6E−12	-1.4E−09***	-1.4E−09***	0.357***	0.107	-0.184***
14	-6.8E−12	-1.7E−09***	-1.7E−09***	0.753***	0.244***	-0.291***
15	-9.2E−12	-1.4E−09***	-1.4E−09***	1.057***	0.373***	-0.333***
16	-8.8E−13	-2.2E−10***	-2.2E−10***	1.325***	0.399***	-0.398***
17	6.3E−13	-1.4E−10***	-1.5E−10***	1.404***	0.529***	-0.364***
18	8.6E−12	-3.8E−11***	-4.6E−11***	1.811***	0.561***	-0.445***
19	3.0E−12	-2.5E−12***	-5.5E−12***	1.852***	0.557***	-0.454***

to converge for large RDF chain queries than RCQ-GA, thus resulting in RCQ-ACS being up to approximately 180% slower than RCQ-GA for larger queries. As such, even though starting out with comparably high execution times for smaller queries, RCQ-GA exhibits almost linear time complexity for increasing query sizes, whereas RCQ-ACS and, to a lesser extent, RCQ-2PO exhibit more of a polynomial time complexity.

Hence, for RDF chain queries constituted by more than approximately ten joins, RCQ-GA offers the best trade-off between execution time and solution quality. The larger a query, the more the solution quality of RCQ-GA approaches the superior solution quality of RCQ-ACS. While doing so, RCQ-GA needs significantly less time than any other of the considered approaches. This confirms the observations in existing literature of RCQ-GA being particularly useful for optimizing larger RDF chain queries [5]. As such, our results contribute to existing work by exhibiting the potential of our novel ACS of delivering significantly better solutions in significantly less execution time than existing approaches for smaller RDF chain queries consisting of up to approximately ten joins, the optimization of which is not trivial due to the complexity of the solution space.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have demonstrated how an ACS approach can facilitate effective and efficient querying in a Semantic Web environment, by having artificial ants iteratively construct RDF chain query paths, partly guided by encountered low-cost solutions, and partly guided by local heuristics. On a large RDF source, our ACS for RDF chain query optimization significantly outperforms both the existing 2PO and GA approaches in terms of execution time and solution quality for RDF chain queries consisting of up to approximately ten joins. For larger queries, our ACS delivers solutions of better quality than the 2PO method, but needs comparably much time to converge, whereas the GA is the fastest algorithm yielding solutions of a quality comparable to the solutions produced by our ACS. Hence, for larger queries, a GA offers the best trade-off between execution time and solution quality, thus rendering our novel ACS particularly useful for optimizing smaller RDF chain queries.

As future work, we aim to explore how our algorithm can best adapt to changes in the environment. In this light, we also consider real-time updating of our join selectivity estimation. Another direction for future work lies in optimizing the parameters of our ACS approach to RDF chain query optimization and performing a parameter sensitivity analysis. 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 ACS approach even further.

REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

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

[3] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF – W3C Recommendation 15 January 2008," 2008.

[4] H. Stuckenschmidt, R. Vdovjak, J. Broekstra, and G. Houben, "Towards Distributed Processing of RDF Path Queries," *International Journal of Web Engineering and Technology*, vol. 2, no. 2-3, pp. 207–230, 2005.

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

[6] C. Gueret, P. Groth, F. van Harmelen, and S. Schlobach, "Finding the Achilles Heel of the Web of Data: Using Network Analysis for Link-Recommendation," in *Ninth International Semantic Web Conference (ISWC 2010)*, ser. Lecture Notes in Computer Science, P. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Pan, I. Horrocks, and B. Glimm, Eds., vol. 6496. Springer, 2010, pp. 289–304.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[8] M. Dorigo and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[9] D. Merkle, M. Middendorf, and H. Schmeck, "Ant Colony Optimization for Resource-Constrained Project Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333–346, 2002.

[10] L. Gambardella and M. Dorigo, "Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem," *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 237–255, 2000.

[11] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and Randomized Optimization for the Join Ordering Problem," *The VLDB Journal*, vol. 6, no. 3, pp. 191–208, 1997.

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

[13] F. Frasincar, G. Houben, R. Vdovjak, and P. Barna, "RAL: An Algebra for Querying RDF," *World Wide Web Journal*, vol. 7, no. 1, pp. 83–109, 2004.

[14] Y. Ioannidis and Y. Kang, "Randomized Algorithms for Optimizing Large Join Queries," in *1990 ACM SIGMOD International Conference on Management of Data (SIGMOD 1990)*. ACM, 1990, pp. 312–321.

[15] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 4th ed. Addison-Wesley, 2004.

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