RCQ-GA: RDF Chain Query Optimization using Genetic Algorithms

Alexander Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak

Erasmus School of Economics, Erasmus University Rotterdam
P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
alexander.hogenboom@gmail.com
{milea, frasincar, kaymak}@ese.eur.nl

Abstract. The application of Semantic Web technologies in an Electronic Commerce environment implies a need for good support tools. Fast query engines are needed for efficient querying of large amounts of data, usually represented using RDF. We focus on optimizing a special class of SPARQL queries, the so-called RDF chain queries. For this purpose, we devise a genetic algorithm called RCQ-GA that determines the order in which joins need to be performed for an efficient evaluation of RDF chain queries. The approach is benchmarked against a two-phase optimization algorithm, previously proposed in literature. The more complex a query is, the more RCQ-GA outperforms the benchmark in solution quality, execution time needed, and consistency of solution quality. When the algorithms are constrained by a time limit, the overall performance of RCQ-GA compared to the benchmark further improves.

1 Introduction

Semantic Web [1] technologies have more and more viable applications in today's Electronic Commerce environments. Compared to the current Web, the Semantic Web offers the possibility to query significant heaps of data from multiple heterogeneous sources more efficiently, returning more relevant results. In the context of the Semantic Web, the keyword is meta-data: describing the context of data and enabling a machine to interpret it. Semantic data is commonly represented using the Resource Description Framework (RDF), a World Wide Web Consortium (W3C) framework for describing and interchanging meta-data [2]. RDF sources can be queried using SPARQL [3]. When implemented in an Electronic Commerce environment, Semantic Web technologies can facilitate a personalized shopping experience for customers through, e.g., recommender systems. For instance, when product data in a webshop is represented using RDF, complex queries can be executed in order to find products of interest to a customer. In a real-time environment with many products per website and many products that need to be compared from different sources, fast RDF query engines are needed.

A successful implementation of an application that is able to query multiple heterogenous sources still seems far away, as several aspects of such an application are subject to ongoing research. An interesting research field in this context

is the determination of query paths: the order in which the different parts of a specified query are evaluated. The execution time of a query depends on this order. A good algorithm for determining the optimal query path can thus contribute to efficient querying. In the context of the Semantic Web, some research has already been done: an iterative improvement (II) algorithm followed by simulated annealing (SA), also referred to as the two-phase optimization (2PO) algorithm, addresses the optimal determination of query paths [4]. This implementation aims at optimizing the query path in an RDF query engine.

However, other algorithms have not yet been used for RDF query path determination, while genetic algorithms (GA) have proven to be more effective than SA in cases with some similar characteristics. For example, a GA performed better than SA in solving the circuit partitioning problem, where components have to be placed on a chip in such a way, that the number of interconnections is optimized [5]. The query path determination problem is somewhat similar to this problem, since the distinctive parts of the query have to be ordered in such a way, that the execution time is minimized. Furthermore, genetic algorithms have proven to generate good results in traditional query execution environments [6].

Therefore, we seek to apply this knowledge from traditional fields to an RDF query execution environment, which differs from traditional ones in that the RDF environment is generally more demanding when it comes to response time; entirely new queries should be optimized and resolved in real-time. In the traditional field of query optimization for relational databases, queries considered for optimization tend to be queries which are used (more) frequently. Such queries can hence be optimized and cached a priori, implying that the duration of the optimization process of such queries is less important. The main goal we pursue consists of investigating whether an approach based on GAs performs better than a 2PO algorithm in determining RDF query paths. As a first step, the current focus is on the performance of such algorithms when optimizing a special class of SPARQL queries, the so-called RDF chain queries, on a single source.

The outline of this paper is as follows. In Section 2 we provide a discussion on RDF and query paths, the optimization of which is discussed in Section 3. Section 4 introduces the genetic algorithm employed for the current purpose. The experimental setup and obtained results are detailed in Section 5. Finally, we conclude in Section 6.

2 RDF and Query Paths

Essentially, an RDF model is a collection of facts declared using RDF. The underlying structure of these facts is a collection of triples, each of which consists of a subject, a predicate and an object. These triples can be visualized using an RDF graph, which can be described as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link [2]. The relationship between a subject node and an object node in an RDF graph is defined using an arc which denotes a predicate. This predicate indicates that the subject has got a certain property, which refers to the object.

An RDF query can be visualized using a tree. The leaf nodes of such a query tree represent inputs (sources), whereas the internal nodes represent algebra operations, which enable a user to specify basic retrieval requests on these sources [7]. The nodes in a query tree can be ordered in many different ways, which all produce the same result. These solutions all depict an order in which operations are executed in order to retrieve the requested data and are referred to as query plans or query paths.

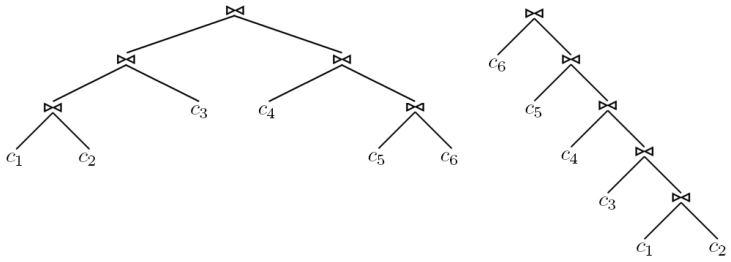
The RDF queries considered in this paper are a specific subset of SPARQL queries, where the WHERE statement only contains a set of node-arc-node patterns, which are chained together. Each arc is to be interpreted as a predicate. Each node represents a concept and is to be interpreted as a subject associated with the predicate leaving this node and as an object associated with the predicate entering this node. When querying RDF sources is regarded as querying relational databases, computing results for paths from partial results resembles computing the results of a chain query. In a chain query, a path is followed by performing joins between its sub paths of length 1 [4]. The join condition used in joining the node-arc-node patterns considered here is that the object of the former pattern equals the subject of the latter pattern.

In order to illustrate chain queries in an RDF environment, let us consider an RDF model of the CIA World Factbook [8] containing data about 250 countries, defined in over $100,000$ statements, generated using QMap [9]. Suppose a company, currently located in South Africa, wants to expand its activities to a country already in a trading relationship (in this example an import partnership) with South Africa. In order to assess the risks involved, the board wants to identify the candidates that have one or more neighbours involved in an international dispute. This query can be expressed in SPARQL as follows:

```
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:SouthAfrica o:importPartner ?impPartner .
5.         ?impPartner o:country ?partner .
6.         ?partner o:border ?border .
7.         ?border o:country ?neighbour .
8.         ?neighbour o:internationalDispute ?dispute .
9.       }
```

This query is a simple example of a chain query and can be subdivided into five parts: the query for information on the import partners of South Africa (line 4), the query for countries actually associated with other countries as import partners (line 5), the query for the borders of the latter countries (line 6), the query for countries associated with a country border as neighbours (line 7), and finally the query for the international disputes the neighbouring countries are involved in (line 8). The results of these sub queries can be joined in order to resolve the complete query. Here, the number of statements resulting from a join is equal to the number of statements compliant with both operands' constraints.

In an RDF context, bushy and right-deep query trees can be considered [4]. In bushy trees, base relations (containing information from one source) 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 1 depicts examples of a bushy tree and a right-deep tree, where concepts $(c_1, c_2, c_3, c_4, c_5, c_6)$ are joined and a $\bowtie$ represents a join. These concepts represent (c:SouthAfrica, ?impPartner, ?partner, ?border, ?neighbour, ?dispute).



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

Fig. 1. Examples of possible query trees for a chain query with six concepts.

3 RDF Query Path Optimization

The order of joins of sub paths 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 right order in which the joins should be computed, hereby optimizing the overall response time. A solution space can be considered, in which each solution s represents a query path and is associated with execution costs C_s, which are mainly realized by the cost of data transmission from the source to the processor and the cost of processing these data [4]. As we focus on a single source, we omit data transmission costs for now and only consider data processing costs: the sum of costs associated with all joins within solution s.

Join costs are influenced by the cardinalities of each operand and the join method used. Several methods can be used for implementing (two-way) joins, as discussed in [10]. We consider only nested-loop joins, as no index or hash key exists a priori for the source used here (making single-loop and hash joins impossible) and the source data are unsorted (requiring the sort-merge join algorithm to sort the data first, which would take up precious running time). When joining operands c_1 and c_2 using a nested-loop join, the processing costs C_j^{nest} associated with this join j are

$$C_j^{\text{nest}} = |c_1| \cdot |c_2|, \tag{1}$$

where the cardinalities of operand c_1 and c_2 are represented by $|c_1|$ and $|c_2|$, respectively.

In an RDF environment, cardinalities of results from joins could be estimated, as actually performing the joins in order to retrieve the number of elements resulting from each join of sub paths would imply the execution time of the optimization process to be very likely to exceed the execution time of a random query path. These estimations could be initialized as the number of elements in the Cartesian product of the operands and be updated after a query has been evaluated; computed join costs can be saved for possible re-use in order to reduce the time needed for evaluating joins.

In the solution space containing all possible query paths for an arbitrary query to be optimized, similar solutions are regarded as neighbours. Solutions are considered to be neighbouring solutions of a solution if they can be transformed into the latter solution by applying one of the following transformation rules once to one part of the query tree they represent [11]:

- Join commutativity: $c_1 \bowtie c_2 \Rightarrow c_2 \bowtie c_1$.
- Join associativity: $(c_1 \bowtie c_2) \bowtie c_3 \Leftrightarrow c_1 \bowtie (c_2 \bowtie c_3)$.
- Left join exchange: $(c_1 \bowtie c_2) \bowtie c_3 \Rightarrow (c_1 \bowtie c_3) \bowtie c_2$.
- Right join exchange: $c_1 \bowtie (c_2 \bowtie c_3) \Rightarrow c_2 \bowtie (c_1 \bowtie c_3)$.

Since not every query path is as efficient as others, the challenge in query path determination is to optimize query execution costs. When utilizing a relational view on RDF sources, queries on these sources could be translated into algebraic expressions. Using transformation rules for relational algebraic expressions, several algebraic query optimization heuristics have been developed [10, 7]. However, in complex solution spaces, these heuristics are not sufficient; randomized algorithms (e.g., II and SA) and GAs generate better results in traditional query execution environments [6]. Applying these algorithms in determining the order of SELECT operations in RDF chain queries would not be very interesting due to the lack of complexity in the associated solution spaces and due to the sufficiency of the heuristics mentioned above. The real challenge lies in optimizing the order and nature of the joins specified in the WHERE statement, whereby randomized algorithms and/or GAs are identified as promising approaches.

In the context of the Semantic Web, the query path determination problem has already been addressed using an II algorithm followed by SA, also referred to as the two-phase optimization (2PO) algorithm [4]. The II algorithm randomly generates a set of initial solutions, which are used as starting points for a walk in the solution space. Each step in such a walk is a step to a neighbouring solution in the solution space that yields improvement. At some point in a walk, a solution is reached for which no better neighbour can be found in a specified number of tries, in which case the current solution is assumed to be a local optimum. The number of times the algorithm tries to find a better neighbour (i.e., randomly selects a neighbour) is limited to the number of neighbours of that solution. The described process is repeated for all starting points.

The best local optimum thus found is subsequently used as a starting point for a SA algorithm, which tends to accept (with a declining probability) moves not yielding improvement. The latter algorithm thus searches the proximity of possibly sub-optimal solutions, hereby reducing the risk for a local optimum.

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.

Just like II, the SA algorithm accepts moves in the solution space yielding lower costs. However, SA can also accept moves leading to higher costs, hereby reducing the chances for the algorithm to get stuck in a local optimum. The probability for accepting such moves depends on the system's temperature: the higher the temperature, the more likely the system is to accept moves leading to higher costs. However, for every state of the algorithm it applies that the more the costs associated with a solution exceed the current costs, the less likely the system is to accept such a move [12].

4 A Genetic RDF Query Path Determination Algorithm

As discussed in Section 1, GAs tend to perform better in query optimization, but have not been assessed in an RDF environment yet. In this paper, we propose to optimize RDF Chain Queries using a Genetic Algorithm: RCQ-GA.

A GA is an optimization algorithm simulating biological evolution according to the principle of survival of the fittest. A population (a set of chromosomes, representing solutions) is exposed to evolution, 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 F_s of a chromosome (expressing the quality of solution s) determines the chances of survival and depends on the associated solution costs C_s. The probability of a solution to be selected must be inversely proportional to its associated costs [6]. This can be accomplished by defining the fitness F_s of solution s as shown in (2), hereby assuming that the population contains n solutions. The fitness-based selection probability can then be defined as shown in (3).

$$F_s = 1 - \frac{C_s}{\sum_{i=1}^{n} C_i}, \tag{2}$$

$$\Pr(s) = \frac{F_s}{\sum_{i=1}^{n} F_i}. \tag{3}$$

Since a GA utilizes a randomized search method rather than moving smoothly from one solution to another, a GA can move through the solution space more abruptly than for example II or SA, by replacing parent solutions with offsprings that may be radically different from their parents. Therefore, a GA is less likely to get stuck in local optima than for example II or SA. However, a GA can experience another problem: crowding [13]. An individual with a relatively high fitness compared to others could reproduce quickly due to its relatively high selection probability, hereby taking over a large part of the population. This reduces the population's diversity, which slows further progress of the GA.

Crowding can be reduced by using different selection criteria, sharing a solution's fitness amongst similar solutions or controlling the generation of offspring. Another option is using a hybrid GA (HGA), which essentially is a GA with some additional, embedded heuristics. However, high quality solutions are not guaranteed to be found within a reasonable running time, as the implemented heuristics often are time-consuming [14]. Ranking-based selection [6] can also reduce crowding. Here, the selection probability of a solution s depends on its rank R_s (the fittest solution is ranked best) in relation to the sum of all n ranks:

$$\Pr(s) = \frac{R_s}{\sum_{c=1}^{n} R_c}. \tag{4}$$

4.1 Settings

In order for a GA to be applicable in RDF query path determination, several parameters must be set. Due to the time constraint associated with executing queries in an RDF environment, using an HGA is not an option. It would be best to opt for a basic GA, adopting the settings best performing in [6]. The algorithm, BushyGenetic (BG), considers a solution space containing bushy query processing trees. A crowding prevention attempt is made by implementing ranking-based selection. Furthermore, the population consists of 128 chromosomes. The crossover rate is 65%, while the mutation rate equals 5%. The stopping condition is 50 generations without improvement. However, long executing times are not desirable for a GA in an RDF query execution environment. Therefore, the stopping condition is complemented with a time limit.

In literature, a GA has been proven to generate better results than a 2PO algorithm in many cases. However, in order to accomplish these results, a GA needs more execution time than 2PO to accomplish this. On the other hand, a GA is aware of good solutions faster than 2PO [6]. Hence, the algorithm spends a lot of time optimizing good results before it terminates. This is an interesting property to exploit in RCQ-GA; since in a real-time environment like the Semantic Web queries need to be resolved as quickly as possible, preliminary and/or quicker convergence of the model might not be such a bad idea after all. If the algorithm could be configured such that it converges quickly when optimizing relatively good solutions, the execution time could be reduced remarkably and the sub-optimal result would not be too far from the global optimum. The challenge is to find a balance between execution time and solution quality.

The BG algorithm could be adapted in order to improve its performance in an RDF query execution environment. The algorithm could be forced to select the best solution for proliferation in the next generation at least once (elitist selection), hereby avoiding losing a good solution. Ranking-based selection could also be replaced with fitness-based selection, as this increases the probability of relatively fit solutions to be selected, which could result in quicker convergence of the model due to increased crowding. Furthermore, evolution could be considered to have stopped after, e.g., 30 generations without improvement instead of 50; long enough in order for the algorithm to be able to determine with sufficient

certainty that the best known solution is either a very good local optimum or a global optimum. Finally, the population size could be reduced to for example 64 solutions, which would noticeably reduce the time needed for computing the costs of all solutions in the population and would provide just enough room for diversity in the population, hereby also enforcing quicker model convergence.

4.2 Query Path Encoding

Encoding of query processing trees is done using an ordinal number encoding scheme for bushy trees, proposed in [6], which not only efficiently represents bushy trees (including the subset of right-deep trees), but enables relatively easy and efficient crossover operations as well. This encoding algorithm iteratively joins two concepts in an ordered list of concepts, the result of which is saved in the position of the first appearing concept. In each iteration, the positions of the selected concepts are saved into the encoded solution.

For example, consider the following ordered list of concepts: (c_1, c_2, c_3, c_4). An initial join between the third and fourth concept yields the list (c_1, c_2, c_3c_4). Another join between the first and second concept in this new list yields (c_1c_2, c_3c_4). A final join between the first and second concept in this list results in $(c_1c_2c_3c_4)$. A possible encoded notation of these joins is $((3,4),(1,2),\ (1,2))$. Additional information, such as the applied join method, can also be stored in this encoded notation. For details on the crossover and mutation methodology applied for the current goal, we refer to [6].

5 Experimental Setup & Results

5.1 Experimental Setup

All experiments performed for the current purpose are run in a Microsoft Windows XP environment, on a $2,400$ MHz Intel Pentium 4 system with $1,534$ MB physical memory (DDR SDRAM). Tests are conducted on a single source: an RDF version of the CIA World Factbook [8], generated using QMap [9]. The first algorithm to be tested is the 2PO algorithm as proposed in [4]. The performance of the BG algorithm [6] and its improved version (RCQ-GA) as proposed in Section 4.1 are benchmarked as well. Finally, the performance of time-constrained 2PO and RCQ-GA (respectively 2POT and RCQ-GAT, in which the T denotes the time-constrained nature of these algorithms) is evaluated.

Several experiments are conducted in order to determine the performance of the considered algorithms; each algorithm is tested on chain queries varying in length from 2 to 20 predicates (see Section 2 for a 6-predicate example). Each experiment is iterated 100 times. The algorithms are configured according to the settings proposed in their sources and thus all consider the entire solution space containing bushy query trees. The time limit for 2POT and RCQ-GAT is set to 1000 milliseconds, as this allows the algorithms to perform at least a couple of iterations and we assume that a maximum waiting time of 1 second for efficiently executing a complex query, would be acceptable in a real-time environment.

Parameter	2PO	2POT	Parameter	BG	RCQ-GA	RCQ-GAT
maxSol	10	10	popSize	128	64	64
startTempFactor	0.1	0.1	crossoverRate	0.65	0.65	0.65
tempRed	0.05	0.05	mutationRate	0.05	0.05	0.05
frozenTemp	1	1	stableFitnessGens	50	30	30
maxConsRedNoImpr	4	4	rankingBased	true	false	false
neighbourExpFactor	16	16	elitist	false	true	true
timeLimit (ms)	-	1000	timeLimit (ms)	-	-	1000

Table 1. Parameters of considered optimization algorithms.

Table 1 presents an overview of the algorithms' configurations. For the considered 2PO algorithms, the *maxSol* parameter sets the maximum number of starting solutions analyzed in the II part of 2PO. The fraction of the optimal cost resulting from II to be used as starting temperature in SA is specified in *startTempFactor*, whereas *tempRed* is the factor with which the temperature of the system is to be reduced every iteration of SA. The *frozenTemp* parameter defines the temperature below which the system is considered to be frozen. The maximum number of consecutive temperature reductions not yielding improvement is defined in *maxConsRedNoImpr*. For each visited solution, SA tries to move to neighbouring solutions for a limited number of times, which equals the number of joins in the query, multiplied by *neighbourExpFactor* [12]. The maximum running time in milliseconds is configured using the *timeLimit* parameter.

As for the considered GAs, the number of chromosomes is defined using the *popSize* parameter. The *crossoverRate* parameter represents the fraction of each new generation to be filled with offspring resulting from crossover operations between pairs of selected chromosomes. The rest of the new generation is filled with direct selections from the current generation; the fitter the chromosome, the higher the selection probability. The fraction of the new population to be mutated is defined using the *mutationRate* parameter. Furthermore, *stableFitnessGens* is the number of consecutive generations not showing improvement in optimal fitness needed for the fitness of the population to be considered stable. The *rankingBased* parameter is used to define whether ranking-based selection should be applied rather than fitness-based selection. The *elitist* parameter states whether the best chromosome should always be selected for the next generation. The time limit in milliseconds is defined in *timeLimit*.

5.2 Results

For each algorithm tested, Fig. 2(a) visualizes the extent to which the average time needed for optimizing chain queries deviates from the average time the 2PO algorithm needs for this optimization, divided by the latter average. This is done in order to directly provide insight into the performance of the tested approaches, with respect to the 2PO benchmark. The results are based on 100 iterations of the query optimization process per experiment.

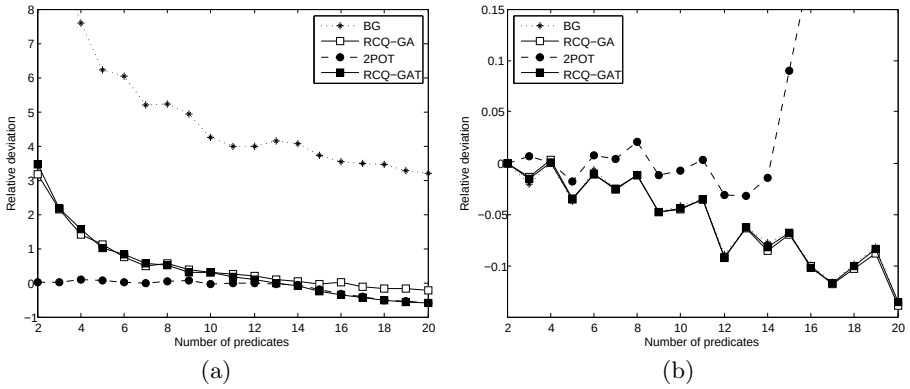


Fig. 2. Relative deviation of average execution times and optimal costs from 2PO average, depicted in (a) and (b), respectively.

For all considered query lengths, on average, the BG algorithm needs the most execution time of all algorithms considered as alternative to 2PO. Furthermore, 2PO turns out to be the fastest performing optimization algorithm for relatively small chain queries containing up to about 10 predicates. For bigger chain queries, RCQ-GA is the fastest performing algorithm. The time-constrained variants of the 2PO and RCQ-GA algorithms obviously take the lead over the RCQ-GA algorithm for even bigger queries, where the execution time needed by the RCQ-GA algorithm exceeds the time limit.

Without a time limit, BG and RCQ-GA tend to find better solutions than 2PO with respect to the average costs associated with optimized chain query paths, especially for larger queries. When a time limit is set, a GA tends to generate even better results compared to 2PO (see Fig. 2(b)). The known behaviour of both algorithms supports this observation, as a GA tends to generate better results in less time, although it needs more time to converge than a 2PO algorithm (as discussed in Section 4.1). Therefore, the earlier in the optimization process both algorithms are forced to stop, the better the optimal solution found by a GA will be compared to the optimal solution generated by 2PO.

The consistency in performance is shown in Fig. 3, using coefficients of variation (standard deviation, divided by the mean) of the execution times and optimal solution costs, respectively, of chain queries of varying lengths. These statistics are based on 100 iterations of the query optimization process per experiment. For each considered algorithm, the deviation of these coefficients of the coefficient of variation of the 2PO algorithm is divided by the latter coefficient. A coefficient of variation close to 0 indicates all observed values are closely clustered around the average. Hence, a positive relative coefficient of variation indicates less consistency of performance of the algorithm, compared to 2PO.

The relative coefficients of variation indicate that compared to 2PO, time-constrained algorithms tend to perform more and more consistently in execution time needed for bigger chain queries. The latter observation can be explained

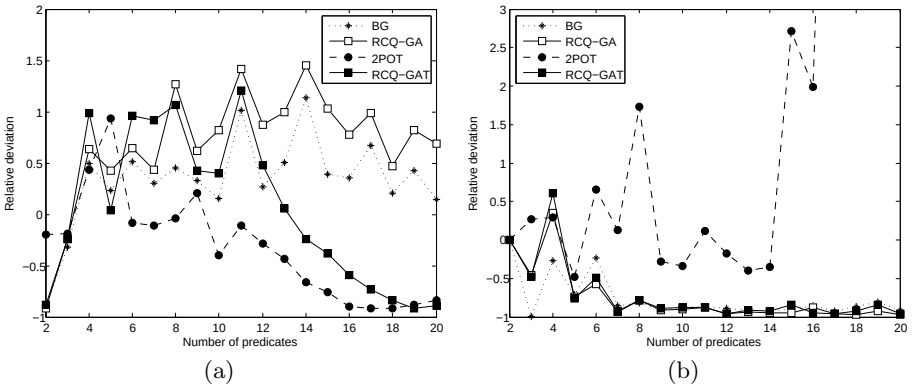


Fig. 3. Relative deviation of coefficients of variation of execution times and optimal costs from 2PO average, depicted in (a) and (b), respectively.

by realizing that bigger chain queries require longer execution times, which are increasingly likely to exceed the time limit. Hence, increasing parts of iterations of bigger queries execute exactly as long as allowed, hereby reducing the variance in execution times. As for the algorithms not constrained by a time limit, the GAs appear to be less consistent in execution time needed than 2PO, especially for more complex queries.

Regarding the costs associated with found optimal solutions, BG and RCQ-GA tend to perform more consistently than 2PO. Also, the more predicates a chain query consists of, the more GAs outperform 2PO when it comes to consistency in solution quality. When a time limit is set, the relative coefficient of variation of the 2PO algorithm increases rapidly with the number of predicates in the chain queries. The consistency in solution quality of RCQ-GA on the other hand is not clearly affected by a time limit.

6 Conclusions

Semantic Web technologies are promising enablers for large-scale knowledge-based systems in an Electronic Commerce environment; they facilitate machine-interpretability of data through effective data representation. Fast query engines are needed in order to efficiently query large amounts of data, usually represented using RDF. The results detailed in this paper lead to the conclusion that in determining the (optimal) query path for chain queries in a single-source RDF query execution environment, the performance of a genetic algorithm compared to two-phase optimization is positively correlated with the complexity of the solution space and the restrictiveness of the environment (in this case a time limit). An appropriately configured genetic algorithm can outperform the two-phase optimization algorithm in solution quality, execution time needed, and consistency of solution quality. As future work, we would like to optimize the

parameters of our algorithm, for instance using meta-algorithms [15] and try our algorithm in a distributed setting. Also, we plan to experiment with other algorithms, such as ant colony optimization or particle swarm optimization.

Acknowledgement

The authors are partially supported by the EU funded IST STREP Project FP6 - 26896: Time-determined ontology-based information system for realtime stock market analysis. More information is available on http://www.towl.org.

References

[1] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5) (2001) 34–43
[2] Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax – W3C Recommendation 10 February 2004 (2004)
[3] Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF – W3C Recommendation 15 January 2008 (2008)
[4] Stuckenschmidt, H., Vdovjak, R., Broekstra, J., Houben, G.J.: Towards Distributed Processing of RDF Path Queries. International Journal of Web Engineering and Technology (IJWET) **2**(2-3) (2005) 207–230
[5] Manikas, T.W., Cain, J.T.: Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem. Technical report, University of Pittsburgh (1996)
[6] Steinbrunn, M., Moerkotte, G., Kemper, A.: Heuristic and Randomized Optimization for the Join Ordering Problem. The VLDB Journal **6**(3) (1997) 191–208
[7] Frasincar, F., Houben, G.J., Vdovjak, R., Barna, P.: RAL: An Algebra for Querying RDF. World Wide Web Journal **7**(1) (2004) 83–109
[8] Central Intelligence Agency: The CIA World Factbook (2008) See https://www.cia.gov/cia/publications/factbook/, last visited April 2008.
[9] Hogenboom, F., Hogenboom, A., van Gelder, R., Milea, V., Frasincar, F., Kaymak, U.: QMap: An RDF-Based Queryable World Map. In: Third International Conference on Knowledge Management in Organizations (KMO 2008), Vaasa, Finland (2008) 99–110
[10] Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 4th edn. Addison-Wesley (2004)
[11] Ioannidis, Y.E., Kang, Y.C.: Randomized Algorithms for Optimizing Large Join Queries. In: The 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD 1990), New York, NY, USA, ACM Press (1990) 312–321
[12] Swami, A., Gupta, A.: Optimization of Large Join Queries. In: The 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD 1988), New York, NY, USA, ACM Press (1988) 8–17
[13] Mitchell, T.M.: Machine Learning. McGraw-Hill Series in Computer Science. McGraw-Hill (1997)
[14] Misevicius, A.: A Fast Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In: The 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006), New York, NY, USA, ACM Press (2006) 1257–1264
[15] de Landgraaf, W.A., Eiben, A.E., Nannen, V.: Parameter Calibration using Meta-Algorithms. In: IEEE Congress on Evolutionary Computation. (2007) 71–78