Genetic Algorithms for RDF Query Path Optimization

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}@few.eur.nl

Abstract. In this paper we present an approach based on genetic algorithms for determining optimal RDF query paths. The performance of this approach is benchmarked against the performance of a two-phase optimization algorithm. For more complex queries, the genetic algorithm RDFGA generally outperforms two-phase optimization in solution quality, execution time needed, and consistency in performance. Setting a time limit improves the overall performance of RDFGA compared to two-phase optimization even more.

1 Introduction

The potential of the Semantic Web has been demonstrated by different proof-of-concept applications, generally focussing on small domains. This limited focus, however, results in a Semantic Web that seems to be scattered into small pieces. Being available only on a small scale and for very specific domains, the access to the Semantic Web seems rather limited from the perspective of the average user.

Addressing the average user could be achieved by offering something that the current Web cannot offer: 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 [1].

Despite current efforts, a successful implementation of an application that is able to query multiple heterogenous sources still seems far away. 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 query path can thus contribute to quick and efficient querying.

In the context of the Semantic Web, some research in this field has already been done: the 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 [2]. 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 [3]. 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 optimized. Furthermore, genetic algorithms have proven to generate good results in traditional query execution environments [4]. 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 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 and/or queries for which the duration of the optimization process is not that big of an issue.

The main goal we pursue consists of investigating whether an approach based on genetic algorithms performs better than a two-phase optimization algorithm in determining RDF query paths. The current focus is on the performance of such algorithms on a single source, and not in a distributed setting.

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.

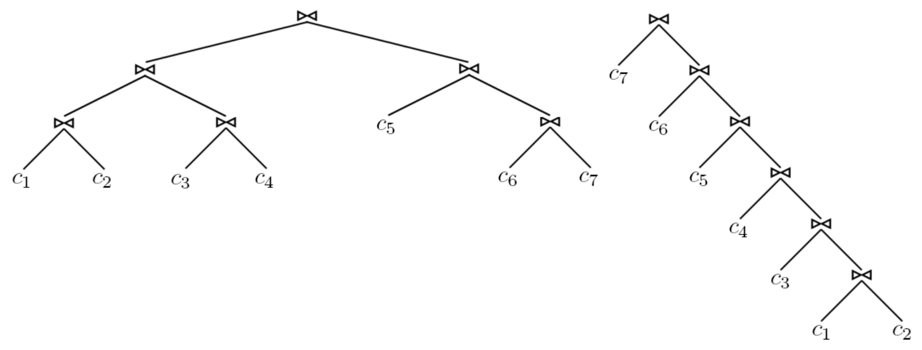
2 RDF and Query Paths

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: *"a node and directed-arc diagram, in which each triple is represented as a node-arc-node link"* [1]. 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 relational algebra operations, which enable a user to specify basic retrieval requests on these sources [5]. 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.

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 [2]. In the context of the Semantic Web, such queries can be expressed as a set of node-arc-node patterns which can be chained (joined). 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 following this node and as an object associated with the predicate preceding this node. The join condition used in joining the node-arc-node patterns is that the object of the former pattern equals the subject of the latter pattern.

In an RDF context, bushy and right-deep query trees can be considered [2]. 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. See Figure 1 for an example of a bushy tree and a right-deep tree, where concepts $(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$ are joined and a $\bowtie$ represents a join.



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

Fig. 1. Examples of possible trees

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. In this process, each join is associated with costs, which are influenced by the number of elements in each operand (their cardinalities) and the method used in the join operation. Several methods can be used for implementing (two-way) joins, as discussed in [5].

The relevance of query path optimization can be demonstrated using a simplified example, in which only the number of results a join yields is considered in determining costs associated with that join. Let us consider an RDF model of the CIA World Factbook [6] containing various data about 250 countries, defined in over 100,000 statements, generated using QMap [7]. 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, an RDF query language, in the following way:

```
PREFIX ont: <http://www.daml.org/2003/09/factbook/factbook-ont#>
SELECT ?partner
WHERE { ?country ont:conventionalShortCountryName ?countryName .
        FILTER regex(?countryName, "^south africa$", "i") .
        ?country ont:importPartner ?impPartner .
        ?impPartner ont:country ?partner .
        ?partner ont:border ?border .
        ?border ont:country ?neighbour .
        ?neighbour ont:internationalDispute ?dispute .
      }
```

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 the specified country, the query for countries actually associated with other countries as import partners, the query for the borders of the latter countries, the query for countries associated with a country border as neighbours, and finally the query for the international disputes the neighbouring countries are involved in. 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 this case, the collection of considered concepts is (?*country*, ?*impPartner*, ?*partner*, ?*border*, ?*neighbour*, ?*dispute*). The model contains 226, 1177, 186, 616, 186, and 548 elements respectively associated with these concepts. However, since the ?*country* concept is constrained to South Africa, the model only contains 1 compliant element.

An example of a query path consisting of joining the concepts in a particular order for this case is shown in Figure 2a. This query path starts with joining the last two concepts, yielding 181 compliant statements. These results are then joined with the ?*border* concept, which yields 2412 compliant statements. Joining these results with the ?*partner* concept yields 156 results. After a consecutive join of these results with the ?*impPartner* concept, 2434 statements are still compliant. A final join with the ?*country* concept yields 7 results. The sum of elements considered in every sub path thus equals 5190.

However, another order of joins is much more efficient. This order is depicted in Figure 2b. A first join of the first two concepts yields 8 results. Joining these results with the ?*partner* concept again yields 8 compliant statements. Joining

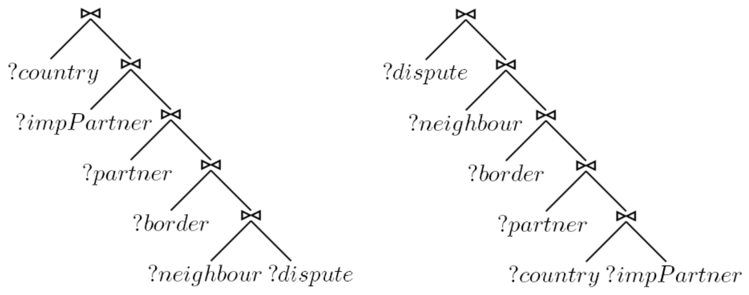


Fig. 2. Possible query paths for the international disputes case

these results with the $?border$ concept results in 38 triples satisfying all conditions. The model contains 33 triples compliant with a join between all previously joined concepts and the $?neighbour$ concept. Finally, a join between these resulting triples and the $?dispute$ concept yields 7 triples. The sum of elements considered in every sub path of this query path equals a mere 94. The order of joins of sub queries can thus make a big difference.

Two solution spaces can be distinguished for the join-order problem in an RDF context: a solution space consisting of bushy trees and a subset of that solution space, containing right-deep trees. The solution space of bushy trees contains $\binom{2n}{n}\frac{n!}{2^n}$ points representing possible permutations of join-orders, for a path length of n. There are 2^{n-1} possible query paths in the subset of right-deep trees [2]. Algorithms for identifying neighbouring solutions in the solution space differ per solution space [4]. If only right-deep query trees are considered, identifying neighbours can be done using the Swap algorithm or the 3Cycle algorithm [8]. However, if the complete solution space (containing bushy query trees) is considered, neighbouring solutions can be found by transforming a solution using transformation rules [9].

Since not every query path is as efficient as others, the challenge in determining which query path should be selected is to optimize query response time and/or execution costs. When utilizing a relational view on RDF sources, queries on these sources could be translated into algebraic expressions. Using some transformation rules for relational algebraic expressions, several heuristics for algebraic query optimization have been developed [5, 10].

However, in complex solution spaces, these simple heuristics are not sufficient; randomized algorithms (e.g. the iterative improvement algorithm and the simulated annealing algorithm) and genetic algorithms have proven to generate better results in traditional query execution environments [4]. Applying these algorithms in determining the order of *select* and *project* operations 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, indicating randomized or genetic algorithms as promising approaches in this area.

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 [2]. The II algorithm randomly generates a set of initial solutions, which are used as starting points for a walk in the solution space. These walks only consist of steps to neighbouring points in the solution space that yield improvement. If no better neighbour can be found in a specified number of tries, the current point 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 the 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 algorithm always 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 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 [8].

4 A Genetic Algorithm for Determining RDF Query Paths

As discussed in Section 1, GAs tend to perform better in query optimization. Based on these results, we propose a GA for determining RDF query paths: RDFGA. A GA is an optimization algorithm which simulates biological evolution according to the principle of survival of the fittest. A population (a set of chromosomes, representing solutions from the solution space) 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). In this process, the fitness of a chromosome (expressing the quality of the solution) determines the chances of survival. Equation 1 depicts that higher the fitness F_s of a chromosome s in relation to the total fitness of n chromosomes, the bigger the probability that this chromosome and/or its offspring will make it to the next generation. Evolution is simulated until either the maximum number of iterations is reached or several generations have not yielded any improvement.

$$\Pr(\texttt{s selected}) = \frac{F_s}{\sum_{c=1}^{n} F_c} \tag{1}$$

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 by 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 [11]. 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. For instance, the initial population could be generated using heuristics for finding (sub-optimal) solutions, heuristics could be embedded in the crossover process or heuristics could (locally) optimize results generated by the crossover process. In these processes, local optimization techniques such as II could be used. However, high quality solutions are not guaranteed to be found within a reasonable running time, since the heuristics implemented in an HGA often are time-consuming [12]. A final strategy to reduce crowding is always selecting the fittest solution at least once (elitist selection) or by applying ranking-based selection [4], in which the probability of a solution s to be selected or used in a cross-over is determined by its rank R_s in relation to the sum of all n ranks (see equation 2). Here, the fittest solution is ranked best, whereas the least fit solution is associated with the worst rank.

$$\Pr(\texttt{s selected}) = \frac{R_s}{\sum_{c=1}^{n} R_c} \tag{2}$$

In order for a GA to be applicable in RDF query path determination, several parameters must be set. General settings, derived from literature, are discussed briefly in Section 4.1 before presenting suggestions for improving the performance of a GA in an RDF query execution environment. Since a GA is based on the principle of survival of the fittest, determining a solution's fitness is a crucial step in a GA. Section 4.2 discusses fitness determination and related issues. Finally, Section 4.3 provides a quick overview of the encoding scheme used for the current purpose to efficiently encode query paths.

4.1 Settings

Due to the time constraint associated with executing queries in an RDF environment, using an HGA is not an option, regardless of its potential of returning even better results than the algorithm used in [4]. This is because solutions of good quality are not guaranteed to be found within a reasonable amount of time, as discussed above. Therefore, it would be best to opt for a basic GA, adopting the settings best performing in [4].

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 has turned out to be needing more execution time than 2PO. On the other hand, research did show that a GA is aware of good solutions faster than 2PO [4]. Hence, the algorithm spends a lot of time optimizing good results before it terminates. The latter property is an interesting property of GAs to exploit in RDFGA for the current purpose. 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, even though this increases the probability of outputting a sub-optimal result. If the model could somehow quickly converge in the final stage of optimization of good results, 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. For instance, the algorithm could be forced to select the best solution for proliferation in the next generation at least once (elitist selection), hereby avoiding the loss of a good solution. Replacing ranking-based selection with fitness-based selection could be a subject of tests too in this case, since 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 state with sufficient certainty that the best known solution is either a very good local optimum or a global optimum, especially in solution spaces with a relatively small number of solutions (which is the case with smaller queries). 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 (especially for smaller queries), hereby also enforcing quicker model convergence.

4.2 Determining a solution's fitness

In the context of RDF query path determination, let the fitness F_s of a solution s depend on its associated costs g_s. When ranking the solutions, the solution associated with the lowest costs should be associated with the highest rank and the solution associated with the highest costs should be associated with the lowest rank. In case of fitness-based selection, the probability of a solution to be selected (as defined in equation 1) must be inverse proportional to its associated costs [4]. This can be accomplished by defining the fitness F_s of solution s as shown in equation 3, hereby assuming that the population contains n solutions.

$$F_s = \frac{1 - \frac{g_s}{\sum_{c=1}^{n} g_c}}{n - 1} \tag{3}$$

For the current goal, only nested-loop joins and hash joins are considered in the calculation of solution costs. No index or hash key exists for the source used here (making single-loop joins impossible) and the source data are unsorted (requiring the sort-merge join algorithm to sort the data first, hereby unnecessarily taking up precious running time).

When joining two operands, say c_1 and c_2, using a nested-loop join, the processing costs are $|c_1| \times |c_2| \times compC$, where $|c_1|$ and $|c_2|$ represent the cardinality of respectively operand c_1 and c_2 and $compC$ denotes the cost of comparing two elements. In case a hash join is used, the processing costs are $(insC \times |c_1|) + (retC \times |c_2| \times avgB)$, where $|c_1|$ and $|c_2|$ again represent the cardinality of respectively operand c_1 and c_2, $insC$ denotes the costs of inserting an element into the hash table, $retC$ represents the cost of retrieving a bucket (which contains elements) from the hash table and $avgB$ stands for the average bucket size [2]. In an RDF environment, cardinalities 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. Hence, we work with estimated cardinalities. These estimations could 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.

4.3 Query path encoding

Encoding of query processing trees is done using an ordinal number encoding scheme for bushy trees, proposed in [4], 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 [4].

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 [6], generated using QMap [7]. The first algorithm to be tested is the 2PO algorithm as proposed in [2]. The performance of the BG algorithm [4] and its improved version (RDFGA) as proposed in Section 4.1 are benchmarked as well. Finally, the performance of time-constrained 2PO and RDFGA (respectively 2POT and RDFGAT, 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 3 for a 6-predicate example). Each experiment is iterated 100 times, in order to increase the accuracy of the results. The parameters in cost determination, *compC*, *insC*, *retC* and *avgB*, are assigned random values of 0.02, 0.05, 0.05 and 5.0 respectively, since these exogenous variables are computer, programming language, and/or implementation dependent and hence would be hard to determine. Since these variables are exogenous, their values will not affect the way the algorithm works, so their exact values are not relevant for the goal pursued here.

The algorithms are configured according to the settings proposed in their sources and thus all consider the entire solution space containing bushy query trees. However, preliminary experimental results on the data set used in this research show that, ranking-based selection perform quicker and yield better results than fitness-based selection. Hence, we have decided to use the ranking based-selection method in this research for RDFGA. Furthermore, the time limit for 2POT and RDFGAT is set to 1000 milliseconds, since this allows the algorithms to perform at least a couple of iterations and since in practice, waiting 1 second in order to have your complex query executed quickly, would probably not be too long.

	2PO	2POT
maxSol	10	10
startTempFactor	0.1	0.1
tempRed	0.05	0.05
frozenTemp	1	1
maxConsRedNoImpr	4	4
neighbourExpFactor	16	16
timeLimit	-	1000

Table 1. Parameters of considered two-phase optimization algorithms

Table 1 presents an overview of the parameters of the 2PO algorithms considered. 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*. Finally, the maximum running time in milliseconds is configured using the *timeLimit* parameter.

	BG	RDFGA	RDFGAT
popSize	128	64	64
crossoverRate	0.65	0.65	0.65
mutationRate	0.05	0.05	0.05
stableFitnessGens	50	30	30
rankingBased	true	true	true
elitist	false	true	true
timeLimit	-	-	1000

Table 2. Parameters of considered genetic algorithms

An overview of the parameters of the GAs is presented in Table 2. The number of chromosomes (solutions) to be subjected to a simulated biological evolution process is defined using the *popSize* parameter. The *crossoverRate* parameter represents which fraction of each new generation is to be filled with offspring resulting from crossover operations between pairs of randomly selected chromosomes. The rest of the new generation is filled with direct selections from the current generation. 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 *ranking-Based* parameter is used to define whether ranking-based selection should be applied rather than fitness-based selection, whereas the *elitist* parameter states whether the best solution should always be selected for the next generation. The maximum running time in milliseconds is defined in *timeLimit*.

5.2 Results

For each algorithm tested, Figure 3 visualizes the average time needed for optimizing chain queries. The chain queries considered in the experiments vary in length from 2 to 20 predicates. The average execution times depicted in Figure 3 are based on 100 iterations of the query optimization process per experiment.

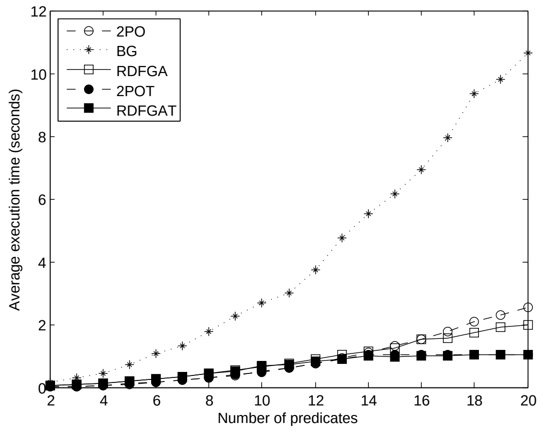


Fig. 3. Average execution times

For all considered query lengths, on average, BG needs the most execution time of all considered algorithms. Furthermore, 2PO turns out to be the fastest performing optimization algorithm for relatively small chain queries containing up to about 10 predicates. For the latter chain queries, on average, RDFGA performs slower than 2PO, but still needs less execution time than BG. For bigger chain queries, RDFGA is the fastest performing algorithm. However, the time-constrained variants of 2PO and RDFGA obviously take the lead for even bigger queries, where RDFGA's execution time exceeds the time limit.

For each algorithm that we consider, the average costs associated with the optimal solutions of chain queries varying in length from 2 to 20 predicates, based on 100 iterations of the query optimization process per experiment, do not appear to differ very much. However, a closer look to the relative deviations from the optimal solutions found by 2PO can reveal more clear indications of differences in performance. Without a time limit, both genetic BG and RDFGA tend to find lower cost solutions, especially for larger queries. When a time limit for query optimization is set, a GA tends to generate even better results compared to 2PO, as shown in Figure 4. The known behaviour of both algorithms supports this observation, since 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 result of a GA will be compared to the solution generated by 2PO.

The consistency in performance is shown in Figures 5 and 6, using coefficients of variation (standard deviation, expressed in relation to the mean) of the execution times and optimal solution costs, respectively, of chain queries of varying

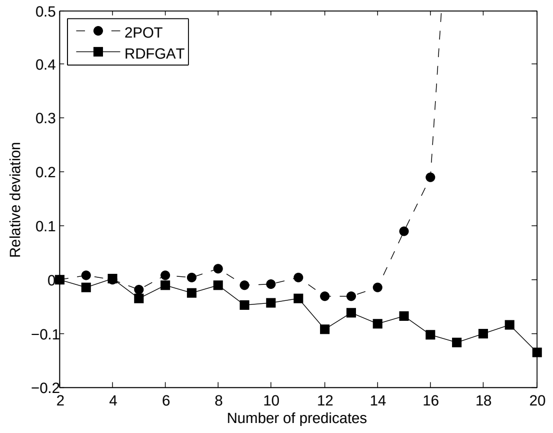


Fig. 4. Relative deviation of average optimal costs from 2PO average

lengths. These statistics are based on 100 iterations of the query optimization process per experiment. A coefficient of variation close to 0 indicates all observed values are closely clustered around the average. Hence, the higher the coefficient of variation, the less consistent the performance of the algorithm.

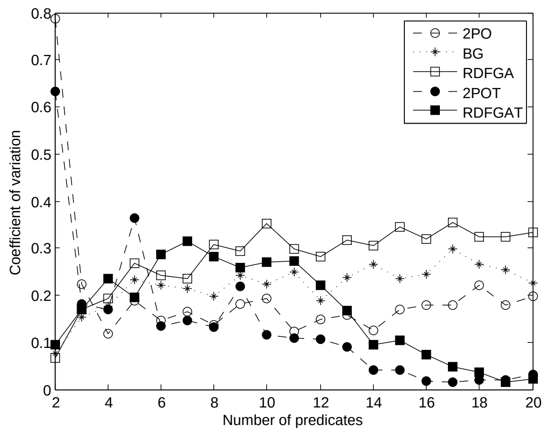


Fig. 5. Coefficients of variation of execution times

The coefficients of variation of the execution times for chain queries of different lengths indicate that time-constrained algorithms tend to perform more and more consistently for bigger chain queries. This observation can be explained by realizing bigger chain queries require longer execution times, which are increasingly likely to exceed the time limit. Hence, increasing parts of iterations of

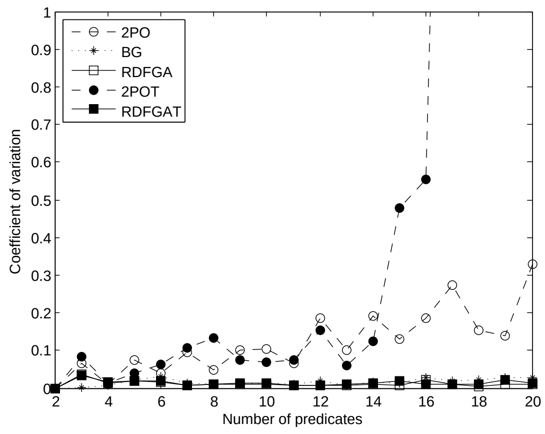


Fig. 6. Coefficients of variation of optimal costs

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.

The 2PO algorithm shows a higher coefficient of variation of optimal costs than BG and RDFGA. Also, the more predicates a chain query consists of, the higher the coefficient of variation of optimal costs. When a time limit is set, the coefficient of the 2PO algorithm increases rapidly with the number of predicates chain queries consist of. GAs on the other hand show a constantly low coefficient of variation of optimal costs. The results of RDFGA are not clearly affected by a time limit.

6 Conclusions

The results detailed in this paper lead to the conclusion that in determining the (optimal) query path in a single-source RDF query execution environment, a correctly configured genetic algorithm can outperform the two-phase optimization algorithm in i) solution quality, ii) execution time needed, and iii) consistency in performance, especially for more complex solution spaces. The superiority of genetic algorithms relative to the two-phase optimization algorithm becomes more clear in positive correlation with the restrictiveness of the environment (e.g. a time limit) and the complexity of the solution space. However, it should be noted that in less complex solution spaces, a genetic algorithm performs worse compared to the two-phase optimization algorithm when it comes to execution time. Furthermore, in some cases, the optimization process could take longer than the actual execution of a query. This falls outside the scope of this paper, but the total query execution process deserves more detailed study and should be considered for further research.

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. Klyne, G., Carroll, J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (2004)
2. Stuckenschmidt, H., Vdovják, R., Broekstra, J., Houben, G.J.: Towards Distributed Processing of RDF Path Queries. International Journal of Web Engineering and Technology 2(2-3), 207–230 (2005)
3. Manikas, T.W., Cain, J.T.: Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem. Technical report, Univerisy of Pittsburgh (1996)
4. Steinbrunn, M., Moerkotte, G., Kemper, A.: Heuristic and Randomized Optimization for the Join Ordering Problem. The VLDB Journal 6(3), 191–208 (1997)
5. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 4th edn. Addison-Wesley (2004)
6. Central Intelligence Agency: The CIA World Factbook (2007) See https://www.cia.gov/cia/publications/factbook/, last visited April 2007.
7. Hogenboom, F., Hogenboom, A., van Gelder, R., Milea, V., Frăsincar, F., Kaymak, U.: QMap: An RDF-Based Queryable World Map. In: Third International Conference on Knowledge Management in Organizations (KMO 2008), pp. 99–110 (2008)
8. Swami, A., Gupta, A.: Optimization of Large Join Queries. In: The 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD 1988), pp. 8–17 ACM Press, New York, NY, USA (1988)
9. 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), pp. 312–321 ACM Press, New York, NY, USA (1990)
10. Frăsincar, F., Houben, G.J., Vdovjak, R., Barna, P.: RAL: An Algebra for Querying RDF. World Wide Web Journal 7(1), 83–109 (2004)
11. Mitchell, T.M.: Machine Learning. McGraw-Hill Series in Computer Science. McGraw-Hill (1997)
12. Misevicius, A.: A Fast Hybrid Genetic Algorithm for the Quadratic Assignment Problem. In: The 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006), pp. 1257–1264 ACM Press, New York, NY, USA (2006)