I. J. Computer Network and Information Security, **2018, 3, 52-59**
Published Online March 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2018.03.06

Optimization of Different Queries using Optimization Algorithm (DE)

Sahil Saharan
Department of Computer Applications, National Institute of Technology, Kurukshetra, India
E-mail: sahil.saharan_1376@nitkkr.ac.in

J.S. Lather
Department of Electrical Engineering, National Institute of Technology, Kurukshetra, India
E-mail: jslather@nitkkr.ac.in

R. Radhakrishnan
Department of Computer Science and Engineering, ABES Ghaziabad (Uttar Pradesh), India
E-mail: ramaswamiradhakrishnan@gmail.com

Received: 30 November 2017; Accepted: 16 January 2018; Published: 08 March 2018

***Abstract*—**The biggest challenge in modern web is to tackle tremendous growth of data, scattered and continuously updating in nature. Processing of such unscattered data by human or machine remains a tedious task. Semantic Web; as a solution has already been invented. But, still there are some other challenges, like as optimization of the query. We introduce a new approach for real–time SPARQL query optimization with different forms and different triple patterns. The strategy introduces rearrangement of order of triple pattern using Differential Evolution(DE). The experimental study focus on main-memory model of RDF data and ARQ query engine of Jena. We compare the result of proposed approach with the Ant Colony Optimization(ACO) different versions and some other approaches. Results shows that proposed approach provides better execution time as compare to the other approaches.

***Index Terms*—**RDF query optimization, Differential Evolution(DE), SPARQL, Reordering triple patterns, Semantic Web.

I. Introduction

The biggest challenge in modern web is to tackle tremendous growth of data, scattered and continuously updating in nature. Processing of such unscattered data by human or machine remains a tedious task. This demands for new ways to represent and query data, such as semantic web which otherwise is not simple using optimized RDBMs representation.

Semantic web is W3C recommendation for sharing and integration of data across heterogeneous but interconnected web resources [1]. It uses RDF as its data model which provides meta-data for machine-interpretability. RDF[2] encodes data in the form of triples structure <s, p, o> forming a RDF graph. The triple structure is particularly suited for connecting different heterogeneous resources, making it feasible for user to issue structure queries and fetch relevant answers.

Semantic Web offers many challenges like processing queries over the interconnected heterogeneous web resources, acquiring knowledge by applying reasoning, ontology aligning, query optimization over billions of triples and performance enhancement.

Different research has already been done in different context like Relational DBMS [3-5], Object-Oriented DBMS [6-7], and XML [8-9] etc regarding the query optimization problem. But in Semantic Web context, query optimization is not fairly matured yet[5][10]. So, our main concern for optimizing an execution plan which will result in a lower execution time. Thus, there is a need for better optimization algorithm which will provide a better execution plan.

Different soft computing algorithms have already been evolved for query optimization in context of RDBMS [11-16], and in context of Semantic Web[10][17-19]. The current state of art techniques need for better optimization algorithm.

As better optimization algorithm, we choose Differential evolution (DE)[20] to provide better or optimal query path to run the query in lesser time. Also, DE requires less control mechanism. DE has been applied to different optimization problems [21-22] etc. This paper, introduced DE algorithm applicability to the RDF query optimization on SPARQL different queries having several triple pattern over a single data source.

The paper structure is organized in the following manner. Section II introduces prior related work. Section III includes different phase of query processing and optimization in ARQ engine. Section IV introduces RDF, SPARQL queries, Jena[23] and the ARQ query engine. We also introduce the BGP construction from where clause predicates through SPARQL query garph. Section V describes the actual problem with the solution and DE

algorithm working. Section VI presents the observations of experiment of the proposed algorithm and its comparison with other algorithms. Finally, in later section the conclusion which is followed by references.

II. Related Work

Prior related work regarding query optimization in RDF database using rearrangement of order of triple pattern, and other important factors for join ordering are discussed here:

The first solution in the RDF databases context was given by Stuckenschmidt et al. [10]. They introduced a hybrid algorithm named 2PO(two phase optimization which is a combination of iterative improvement and simulation annealing) over chain query and implemented the working over the Sesame system[24]. Maduko et al.[25] introduced a method for estimation of cardinality using pattern-based summarization. The method used two main functionality i.e. pattern and their sub-pattern can have almost equal frequencies, and prior knowledge of patter's importance. Also, they used Dynamic programming with two greedy solutions. Stocker et al.[26] presented a technique for the static query optimization for reordering of triple pattern of BGP. Additionally, they also defined heuristic for selectivity estimation using and without using pre-computed statistics. Then, Hogenboom et al.[27] presented an optimization algorithm named GA(genetic algorithm) and through tested queries, they shown that GA performed better for large chain query when compared with 2PO. But for small queries with 10 predicates, 2PO performed better. Neumann and Weikum [5] invented a RDF-3X engine with dynamic programming as an algorithm for query optimization. For cost estimation of joins between triple patterns, selectivity histogram was used. Neumann and Weikum [28] improved their prior research in [5] and presented a method for sideways in-formation passing between separate joins, and at compile time, they used aggregated statistics to provide exact cardinality of triple pattern. Kaoudi et al. [29] implemented their strategy in Atlas system. They used 3 greedy optimization algorithm to minimize the size of intermediate facts which are generated by query processing algorithm using selectivity based heuristic. Neumann and Moerkotte [30] presented a method for the estimation of cardinality based on Characteristic set. Ouyang et al. [31] presented a method using genetic algorithm(GA) to optimize the SPARQL query and also for the plan generation bushy tree was used. Hogenboom et al. [18] studied the query optimization problem using the ACO algorithm over the chain query. They experiment the proposed approach in comparison with [17] and [10] and proved the superiority of the results over other algorithms. Gubichev and Neumann [32] extend Characteristic Sets to provide a new RDF statistical synopsis that accurately estimates cardinalities. Gomathi et al.[19] presented an optimization algorithm named Adaptive Cuckoo search(ACS) to optimize the SPARQL query. Kalayci et al. [33] presented a new solution for the optimization using Ant Colony Optimization (ACO) and reordering triple patterns and selectivity estimation introduced by [26] with some modification. Meimaris and Papastefanatos [34] presented a new approach of join reordering that converts a query into a multidimensional vector space and performs distance-based optimisation.

After reviewing related work in the field of query optimization, the literature suggested that there is requirement for better heuristic and so we proposed a novel solution for this problem using an algorithm named: Differential Evolution. We are using Differential Evolution[35] algorithm that work over discrete problem. Result proves its effectiveness over other algorithm.

III. Different Phase of the Query Processing Optimizationand Execution

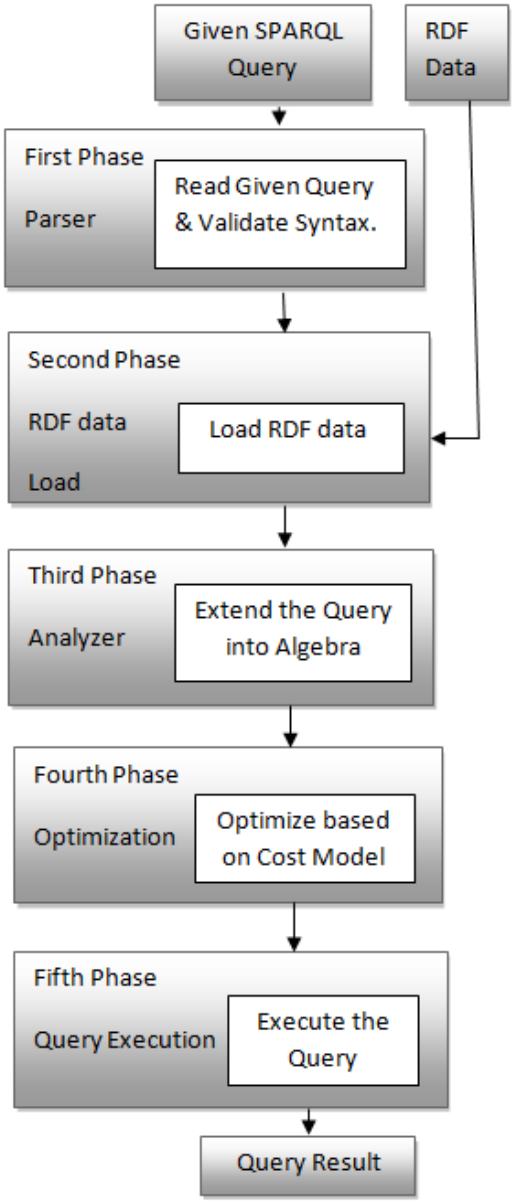

Fig.1. Different phase for SPARQL Query Processing, Optimization and Execution

To process the query, different steps are to be followed in such a way: firstly enter the query which is to be processed and then optimize the query internal structure and then finally execute the optimized query through engine and fetch the result.

Step by step processing of the query through different phase is shown here.

Step 1: Firstly, write the query for execution and pass it to the query engine for the evaluation.
Step 2: Query execution engine then parse the query into the parser for the SPARQL syntax matching and order of the keywords such as SELECT, WHERE etc. and then validate for the attribute of the RDF.
Step 3: Load the RDF repositories side by side and these RDF repositories offer the querying capabilities.
Step 4: Next, analyzer transform the query into its equivalent form(SPARQL algebra) which have same order of the triple pattern as given in the original query.
Step 5: Based on the main-memory model of RDF repositories, query can fetch result without considering any optimization but when the size of RDF repositories increases, there is a need to execute the query with some optimization technique to ensure that query result in a reasonable time. The optimization technique populates the best execution plan from all the plans.
Step 6: After all the steps, if there is no error upto now then the query execution step comes where optimized query execute through query engine and fetch the results of the SPARQL query.

IV. Preliminaries: SPARQL and SPARQL Graph Query

Resource Description Framework (RDF) works as a metadata data model for data interchange over the web. Due to the flexibility of schema-free nature, RDF can merge data even if they differ in their underlying schema. RDF data model is similar to entity-relationship approach. The linking structure can be represented using node-arc-node link[2].

SPARQL[36] is a language for querying RDF data and generally used for expressing query over different data sources using triple patterns named BGPs(Basic Graph Patterns)[37]. The formation of triple patterns is like as < subject, predicate, object > where the different positions can be concrete or variable[26].

Jena[23] is store and manipulates main-memory RDF data. ARQ[38] query engine used by Jena. For querying, we are using ARQ engine.

The following example (Fig 2.) shows representation of SPARQL star query having four triple patterns(can be represented by sequence number {0, 1, 2, 3}) constructed over the The World Factbook[39] (in short Factbook) dataset. Our main focus is to optimise the order of BGP so that they produce the optimal execution plan.

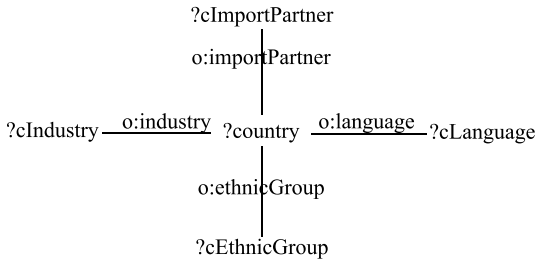

Fig.2.

V. Problem And Proposed Solution

1. Cost Model Used

Given a SPARQL query graph as shown in Fig 2, the optimizer returns the query plan that is defined by ordering of joins between triple patterns. The optimizer provide a search space of different equivalent query plans, and also searches optimal plan from all the different query plans based on the cost function.

As the sequence number can be created using Fig 2, the rearrangement of order of triple pattern(or sequence numbers) is an important query optimization approach. Hence, we are considering DE optimization algorithm as a proposed approach to rearrange the order of triple pattern.

Following are the steps for the calculation of cost between the two triple patterns[33]:

Step 1: Calculate the cardinality of triple patterns using GSH[26]. If more than one element is bound then find the cardinality of each different concrete element at position(s, p, o) using GSH and finally conclude the lowest value of the cardinality by comparing.
Step 2: Find the selectivity of the triple patterns using division of the cardinality of the triple pattern by total no of triple patterns.
Step 3: Use the two triple patterns of the given BGP and collect all different joins betweens the triple patterns. Subtract the different join ranks from the assumed cost value(32) and then divide the generated value by the assumed cost to evaluate the factor for the joined triple patterns.
Step 4: This step computes the joined triple patterns cost values by multiplying the selectivity of the joined triple patterns with the factor of joined triple patterns.
Step 5: Finally, compute a new cost using addition of the selectivity of the first triple pattern out of the two triple patterns and output of step 4.
Step 6: Whenever no join found (Cartesian product possibility in complete digraph construction) in between two triple patterns, then use the Step 1 to Step 5 and change in Step 4 with output value as 1(it is the upper limit of selectivity[0 1])

2. Second Step: Differential Evolution

For the given Ɓ, our aim is to search for the better

execution plan using rearrangement of order of triple pattern of B. The first step to reach our aim is to find the fitness function using cost and then apply the DE algorithm.

The DE algorithm[19][35] briefly described as follows:

The algorithm starts with a randomly generated. Then for evolution of next generation, three evolutionary operators namely mutation, crossover and selection using special form of DE operator are used to evolve the population.

(1) Initialization

The DE problem is to minimize a function $f(x):\Re^D \to \Re$, where x is a D-dimensional candidate solution. The algorithm is initialized with an initial population of n_p candidate solutions. The D-dimensional candidate solutions at every generation can be represented as: $x^g_{n,i} = [x^g_{n,1}, x^g_{n,2}, x^g_{n,3}, ..., x^g_{n,D}]$ where g is the generation and $n = 1, 2, 3, ..., n_p$. The initial population of candidate solutions with $g = 0$ is chosen to cover the entire solution space. Thus, the population of individuals is randomly selected uniformly in the interval (x^L_i, x^U_i), where i indicate the population member with lower bound x^L_i and upper bound x^U_i. Therefore, the initial value of jth parameter of the ith population member may be set as:

$$X_{n,j,i,0} = x^L_{n,i} + \text{rand}(0,1) * (x^U_{n,i} - x^L_{n,i}) \quad (1)$$

(2) Mutation

Let $x^g_{n,i}$, $n = 1, 2, 3, ...n_p$ be the target vector, then for each individual, two vectors $(x^g_{r2,n} \text{and } x^g_{r3,n})$ are randomly selected and select $x^g_{r1,n}$ either randomly or as one of the best members of the population. F is real and constant factor generally taken between $[0,1]$ .For the next generation, create mutant vector or donor vector, v^g_n equal to the randomly generated population and can be defined as follows:

$$v^g_n = x^g_{r1,n} + F(x^g_{r2,n} - x^g_{r3,n}) \quad (2)$$

For our problem, vector addition and subtraction in equation (2) was made to utilize the definition of permutation matrix as explained in [40].

(3) Crossover

The crossover uses crossover operators to generate a trial vector $u^g_{n,i}$ by mixing target vector $x^g_{n,i}$ to the current generation with mutant vector $v^g_{n,i}$ given as:

$$u^g_{n,i} = \begin{cases} v^g_{n,i} & \text{if rand}(0,1) \le CR \text{ or } i = I_{rand} \\ x^g_{n,i} & \text{else} \end{cases} \quad (3)$$

Here, I_{rand} is a integer uniformly distributed random number between $[1, D]$, and CR crossover factor belongs to $[1, D]$; both usually defined by user.

For our experiment, to generate a trial vector, a crossover operator was used named PMX [41].

(4) Selection

The selection is defined in terms of objective function, and the vector with lower value of objective function implies higher fitness. In selection process, the objective function of the trail vector is compared with that of the target vector, and includes better fitness value in next generation. The operation can be defined as:

$$x^{g+1}_n = \begin{cases} u^g_{n,i} & \text{if } f(u^g_n) < f(x^g_n) \\ x^g_n & \text{else} \end{cases} \quad (4)$$

The DE algorithm involves the iterative operation of the three mutation, crossover and selection phases till the stopping criteria i.e. vector with best objective value is obtained.

Algorithm: DE Psuedocode
Initialization: Generate the initial population of agent with random
positions
Evaluate the fitness of the initial population.
Repeat
For each agent j in the population
Choose three numbers $m1, m2 \text{ and } m3$ from the population at
random,
i.e., $1 \le m1, m2, m3 \le N$ with $m1 \ne m2 \ne m3 \ne j$
Create a random integer $j_{rand} \in (1, N)$
For each i

$$y^{i,g} = x^{m1,g} + F(x^{m2,g} - x^{m3,g})$$

$$z^{i,g}_j = \begin{cases} y^{i,g}_j & \text{if rand}() \le CR \text{ or } j = j_{rand} \\ x^{i,g}_j & \text{else} \end{cases}$$

end for
Change $x^{i,g}$ with $z^{i,g}$ if $z^{i,g}$ is better
end for
Until (stopping criteria met)

VI. Experiment

1. Setting of the Experiment

The experiment have been conducted over 64-bit Oracle JDK virtual machine processing on Intel(R) Core(TM) i5 -2410M CPU @ 2.30 GHz, 64-bit with 4 GB RAM and Window 8 OS with Java as programming language. All the results are evaluated over a data set named Factbook[39] which provide summary of the demographic, communication, economy, geography, government, and military data. The total numbers of RDF triples are 141644. For different join and for particular query forms(chain, star, cyclic, chain-star), we used 3

queries with 4, 6, 8 triples pattern for all forms of queries except chain-star. As chain-star query is a combination of 2 query form, it is complex to design so we consider 6, 8 and 10 triple patterns for this query form. The reason for this choice of these triple patterns is that small triple pattern sometimes shows no optimization at all. For result calculation, we consider only single query of every different triple patterns.

For the DE algorithm, we are using starting node as random, population size as 100, iteration size 100, CR value as 0.8 and F value as 0.2. These parameters are selected after making experimental study using all the queries and by taking a number of test over the population size from 10 to 250 and iteration size from 50 to 250, with corresponding change in F and CR values. We 10 times run each algorithm for each query to evaluate the result and finally considered the average.

For the result calculation, we are considering the execution time of the query in millisecond and consider the execution time as a sum of time taken in the optimization, time taken in the execution and time taken in iteration of result set for all the algorithms except WO.

2. Results of Comparisons

Jena Reorder Weight class(JW): Reorder Weight class is the class of Jena query engine[23]. This class work for the processing of join based on statistics generated from the repositories of RDF to change the arrangement of the triple patterns. This class works on the basis of number of variables in a triple pattern.

Pre-computed statistics (PCS): Probabilistic framework(mainly PFJ) given by [26] which uses pre-computed statistics that must be computed before using it. This framework provides the accurate selectivity estimation for triple pattern and joined triple pattern using the customized statistics.

Without Pre-computed statistics (WPCS): This heuristic is a combination of the Graph Statistic Handler [26] and Variable Counting Predicate [26], and given by [26].

WO provides higher execution time for all the queries, so we are considering it as highest value for all the queries, as shown in the following figures. And all the optimization algorithms are considered proportional to WO execution time. So the results of all optimization algorithms in comparison to WO in terms of percentage can be seen from Fig 3 to Fig 6. These figures can easily show that our proposed solution results in comparison to the other approaches.

Finding of Table 1 shows that DE performs better in terms of execution time when we compare the results with WO, JW, WPCS, PCS, AS, EAS, MMAS. AS, EAS, MMAS are different versions of ACO[33].

Table 1. Results of Chain Query with different Triple Patterns

	Four	Six	Eight
WO	1264	2093	727835
JW	307	2065	763108
WPCS	203	222	948
PCS	62	863	5414
AS	31	126	5533
EAS	29	122	5424
MMAS	31	121	5419
DE	18	85	5370

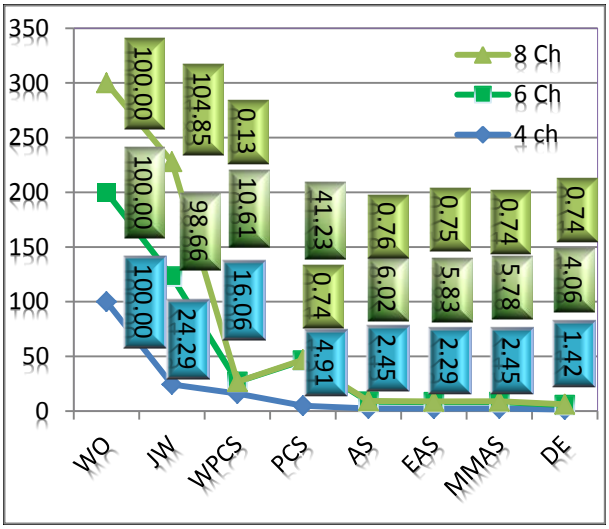

Fig.3. Chain Query with 4, 6, 8 Triple Pattern Execution Time in Percentage

In case of PCS, there is a need to load the index that is created over the RDF data and it takes around 13.5 second to load it for all the chain queries. Approximately same time is taken by other forms of queries also. And after loading this index, execution time taken by the PCS is given in the row of PCS. In case of WPCS, it does not include bound-predicate-predicate join between two triple patterns and always use default selectivity as 1.0. That's the reason why this heuristic does not include other joins between two triple patterns where bound-predicate-predicate occur. But the heuristic [33] provides a new solution where it considers other joins those occur at different position with bound-predicate-predicate instead of considering bound-predicate-predicate. Also, this heuristic gives chance to Cartesian product between two triple patterns. Thus, by considering these particular run times, DE shows best results in all the triple patterns of chain queries.

Table 2. Results of Star Query with different Triple Patterns

	Four	Six	Eight
WO	1513	6337	4297
JW	1412	6441	4130
WPCS	428	4503	2193
PCS	193	713	146
AS	575	1505	307
EAS	572	1409	306
MMAS	573	1464	316
DE	553	1310	218

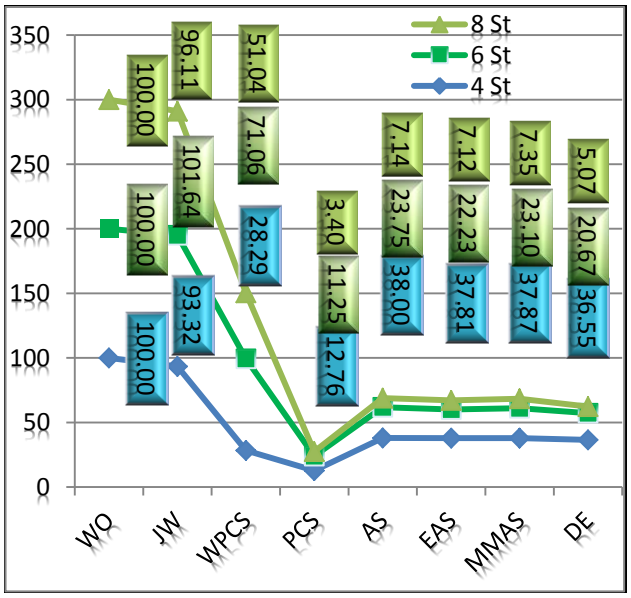

Fig.4. Star Query with 4, 6, 8 Triple Pattern Execution Time in Percentage

Table 3. Results of Cyclic Query with Different Triple Patterns

	Four	Six	Eight
WO	421	46974	23224
JW	414	46837	23200
WPCS	209	213	181
PCS	21	9	9
AS	43	103	734
EAS	36	103	487
MMAS	42	115	503
DE	28	59	412

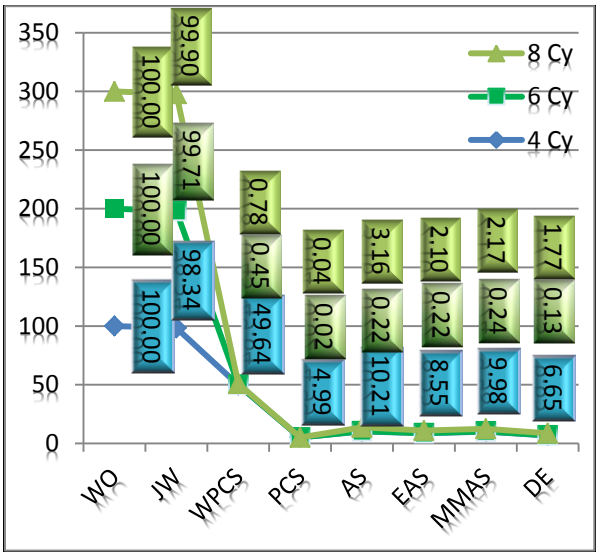

Fig.5. Cyclic Query with Different Triple Pattern Execution Time in Percentage

Table 4. Execution time of Chain-star Query with different Triple Patterns

	Six	Eight	Ten
WO	5862	7513	62973
JW	5271	7159	61450
WPCS	17844	3481	11740
PCS	3144	3988	6650
AS	3958	4587	9428
EAS	4053	4553	9265
MMAS	3903	4746	9323
DE	3783	4454	7992

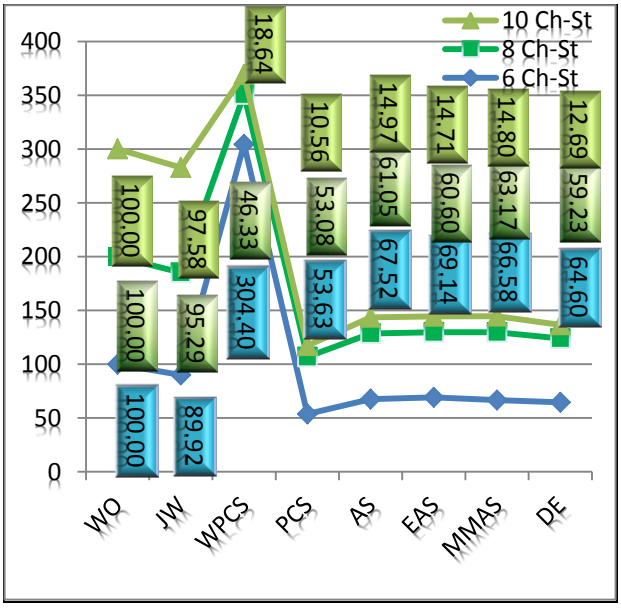

Fig.6. Chain-star Query with Different Triple Pattern Execution time in Percentage

Table 1 to Table 4 shows the result of different forms of queries consist of 4 triple patterns, 6 triple patterns, 8 triple patterns and for chain–star query, it is made up of 6, 8, 10 triple patterns respectively. All the results shows that DE performs better in terms of execution time for all the queries when compare with the WO, JW, WPCS, PCS, AS, EAS, MMAS.

VII. Conclusion

This study contributes to SPARQL SELECT query optimization of different forms of queries having different triple patterns using an optimization algorithm named Differential Evolution. Through the result observation in comparison to other approach, we can conclude that the proposed strategy decrease the execution time of the different form of the query and it works in real time for ARQ engine. Additionally, our proposed approach optimize the SPARQL query on main-memory RDF data model and experiment proves its success. Also, our approach presents outcomes in good and consistent form when compared with other algorithms and heuristics. Results can further be mended by searching better cost computation strategy. This solution for rearrangement of order of triple pattern can be implemented over different query engines. Also, our approach can be further expanded to different optimization new algorithms (Artificial Bee Colony, Particle Swarm Optimization).

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, "The semantic web", Sci. Am. vol 284, no.5, pp: 34–43, 2001.

[2] J.J. Carroll, G. Klyne, "Resource description framework (RDF): Concepts and abstract syntax"– W3C recommendation, 2004.

[3] Y.E. Ioannidis, "Query optimization," *ACM Computing Surveys*, vol/issue:28(1), pp.121–123, 1996.

[4] S. Chaudhuri, "An overview of query optimization in relational systems," in *Proceedings of the 17th Symposium on Principles of Database Systems*, PODS'98, ACM, Seattle, Washington, pp.34– 43, 1998.

[5] T. Neumann, G. Weikum, "RDF-3X: a RISC- style engine for RDF," *Proc. VLDB Endow*, vol/issue:1(1), pp.647–659, 2008.

[6] G. Mitchell, S.B. Zdonik, U. Dayal, "Object-oriented Query Optimization: What's the Problem?", Technical Report, Brown University, Providence, RI, USA, 1991.

[7] M.T. Özsu, J.A. Blakeley, "Query processing in object-oriented database systems," *Modern Database Systems*, ACM Press, Addison-Wesley, New York, NY, USA, pp.146–174, 1995.

[8] D. Che, K. Aberer, T. Ozsu, "Query optimization in XML structured- document databases," *VLDB* J,vol/issue:15(3), pp.263–289, 2006.

[9] R. Abdel Kader, M. van Keulen, "Overview of query optimization in XML database systems," URL: ⟨http://doc.utwente.nl/64449/⟩, 2007.

[10] H. Stuckenschmidt, R. Vdovjak, J. Broekstra, G. Houben, "Towards distributed processing of RDF path queries," *Int. J. Web Eng. Technol*.vol/issue:2(2/3), pp.207–230, 2005.

[11] M.M. Astrahan, M.W. Blasgen, D.D. Chamberlin, K.P. Eswaran, J.N. Gray, P.P. Griffiths, W.F. King, R.A. Lorie, P.R. McJones, J.W. Mehl, G.R. Putzolu, I.L. Traiger, B.W. Wade, V. Watson, "System R: relational approach to database management," *ACM Trans. Database Syst*.vol/issue:1(2) pp.97–137, 1976.

[12] N. Li, Y. Liu, Y. Dong, J. Gu, "Application of ant colony optimization algorithm to multi-join query optimization," in *Proceedings of the 3rd International Symposium on Advances in Computation and Intelligence*, ISICA'08, Springer-Verlag, Berlin, Heidelberg, pp.189–197, 2008.

[13] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price, "Access path selection in a relational database management system," in *Proceedings of the International Conference on Management of Data*, SIGMOD'79, ACM, New York, NY, USA, pp.23–34, 1979.

[14] M. Steinbrunn, G. Moerkotte, & A. Kemper, "Heuristic and randomized optimization for the join ordering problem", *The VLDB Journal*, vol:6, pp.191–208, 1997.

[15] T. Ibaraki, & T. Kameda, "On the optimal nesting order for computing N relational joins". *ACM Transactions on Database Systems*, vol:9, pp.482–502, 1984.

[16] Y. E. Ioannidis, & E. Wong, "Query optimization by simulated annealing," *SIGMOD Rec*.16, pp.9–22, 1987.

[17] A. Hogenboom, V. Milea, F. Frasincar, U. Kaymak, "RCQ-GA: RDF chain query optimization using genetic algorithms," in *Proceedings of the 10th International Conference on EC-Web*, pp.181–192, 2009.

[18] A. Hogenboom, F. Frasincar, U. Kaymak, "Ant colony optimization for RDF chain queries for decision support," *Expert Syst. Appl*,vol/issue:40(5),2013.

[19] R. Gomathi, D. Sharmila, "A novel adaptive cuckoo search for optimal query plan generation," *The Scientific World Journal*, 2014.

[20] R. Storn, K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *J Glob Optim*, vol/issue: 11(4), pp.341–359, 1997.

[21] V. Plagianakos, D. Tasoulis, M. Vrahatis, "A Review of Major Application Areas of Differential Evolution," Advances in Differential Evolution, Springer, Berlin, vol:143 pp.19- 238, 2008.

[22] J. Ilonen, J.K. Kamarainen, J. Lampinen, "Differential Evolution Training algorithm for Feed-Forward Neural Networks," *Neural Process Lett*, vol/issue: 17(1), pp.93–105, 2003.

[23] http://jena.apache.org.

[24] J. Broekstra, A. Kampman, F. Van Harmelen, "Sesame: A generic architecture for storing and querying rdf and rdf schema," in: *International semantic web conference*, Springer Berlin Heidelberg, pp. 54-68, 2002.

[25] A. Maduko, K. Anyanwu, A. Sheth, P. Schliekelman, "Estimating the cardinality of RDF graph patterns," in *Proceedings of the 16th International Conference on World Wide Web*, ACM, Banff, AB, Canada, pp.1233–1234, 2007.

[26] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, D. Reynolds, "SPARQL basic graph pattern optimization using selectivity estimation," in *Proceedings of the 17th International Conference on WWW*, ACM, Beijing, China, pp:595–604, 2008.

[27] A. Hogenboom, V. Milea, F. Frasincar, U. Kaymak, "RCQ-GA: RDF chain query optimization using genetic algorithms," in *Proceedings of the 10th International Conference on EC-Web*, pp:181–192, 2009.

[28] T. Neumann, G. Weikum, "Scalable join processing on

very large RDF graphs," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'09, ACM, New York, NY, USA, pp:627–640, 2009.
[29] Z. Kaoudi, K. Kyzirakos, M. Koubarakis, "SPARQL query optimization on top of DHTs," in *Proceedings of the 9th International Conference on the Semantic Web*, ISWC'10, Springer-Verlag, Berlin, Heidelberg, pp:418–435, 2010.
[30] T. Neumann and G. Moerkotte, "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins," in *ICDE*, Hannover, Germany, pp:984-994, 2011.
[31] D. Ouyang, X. Wang, Y. Ye, and X. Cui, "A GA-based SPARQL BGP reordering optimization method," *Advances in Information Sciences and Service Sciences*, vol/issue:4(9), pp:139–147, 2012.
[32] A. Gubichev and T. Neumann, Exploiting the query structure for efficient join ordering in SPARQL queries. in: EDBT, 2014, pp. 439–450.
[33] E. Guzel Kalayci, T.E. Kalaycı, D. Birant, "An ant colony optimization approach for optimising SPARQL queries by reordering triple patterns," Information Systems, vol:50 pp:51–68, 2015.
[34] Meimaris M, Papastefanatos G. Distance-Based Triple Reordering for SPARQL Query Optimization. in: Proceedings of the 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 1559-1562.
[35] J.G. Sauer, L dos Santos Coelho, V.C. Mariani, L. de Macedo Mourelle, N. Nedjah, "A discrete differential evolution approach with local search for traveling salesman problems," *in Innovative Computing Methods and Their Applications to Engineering Problems, Springer Berlin Heidelberg, pp. 1-12, 2011.*
[36] S. Harris, A. Seaborne, "SPARQL1.1querylanguage" – W3C working draft 05 January 2012.
[37] G.H.L. Fletcher, "An algebra for basic graph patterns," in: *Proceedings of the Workshop on Logic in Databases*, 2008.
[38] http://jena.apache.org/documentation/query.
[39] http://www.cia.gov/library/publications/download/
[40] B Hegerty, CC Hung, K Kasprak, "A comparative study on differential evolution and genetic algorithms for some combinatorial problems," in Proceedings of 8th Mexican International Conference on Artificial Intelligence, pp. 9-13, 2009.
[41] S. N. Sivanandam, S.N. Deepa, "Introduction to Genetic Algorithm," ISBN 978-3-540-73189-4 Springer Berlin Heidelberg New York, Springer ñ Verlag Berlin Heidelberg 2008.

Authors' Profiles

Sahil Saharan has done her MCA degree from NIT Kuruksherta and is pursuing Ph.D from the Department of Computer Applications, NIT, Kurukshetra. Her research interest is focused on Semantic Web, Query Optimization, Database and Data Analytics, Soft- Computing.

Dr. J.S. Lather has received B.E, M. Tech and Ph.D. from REC Kurukshetra. He has more than 23 year experience and presently working as Professor in Electrical Engineering Department, NIT Kurukshetra. His area of interest Wireless Communication, Robust Control of Time Delay Systems, Networked Control Systems, Consensus in WSN, Coop Control in Multi Agent Sys, Control of FACTs incorporating renewable energy. He has published more than 50 papers in various International National conferences and Journals.

Dr. R. Radhakrishnana has received B.E. and M.E. from NIT Trichy and Ph.D. from Jamia Milia Islamia in Handover Management in MIPv6. He has more than 17 years Industry and more than 10 years academic experience. His area of interest is Mobile and Wireless Communication. He has published more than 22 papers in various International National conferences and Journal.

How to cite this paper: Sahil Saharan, J.S. Lather, R. Radhakrishnan,"Optimization of Different Queries using Optimization Algorithm (DE)", International Journal of Computer Network and Information Security(IJCNIS), Vol.10, No.3, pp.52-59, 2018.DOI: 10.5815/ijcnis.2018.03.06

