Reordering Triple Patterns of SPARQL Queries using Ant Colony Optimization

Elem GÜZEL KALAYCI[1] Tahir Emre KALAYCI[2]

[1]Computer Engineering Department
Izmir University of Economics

[2]Computer Engineering Department
Manisa Celal Bayar University

International Conference on Soft Computing (MENDEL'12), 2012

Outline

Optimizing SPARQL Queries using Ant Colony Optimization
- SPARQL Query Optimization
- Ant Colony Optimization
- Selectivity Estimation and Cost Calculation
- Implementation

Experimental Results
- Experimental Setup
- Results

Conclusions and Future Work

References

What are we doing?

- We proposed an Ant Colony Optimization (ACO) approach for optimizing SPARQL queries:
  - SPARQL Basic Graph Pattern (i.e. optimizing order of the triple patterns) is optimized by using ACO
  - It is a real time optimization without requiring any prior knowledge

SPARQL Query Optimization

- SPARQL (SPARQL Protocol and RDF Query Language) is a RDF query language and protocol
- Reordering triple patterns is a significant part of low-level SPARQL query optimization.
- The purpose of reordering triple patterns is to find fastest (optimum - better) query execution plan
  - The plan that returns the result set with minimum execution time compared to other execution plans.

SPARQL Query Optimization - Example

- Basic Graph Pattern of a SPARQL query which queries neighbours of Turkey and also queries import commodities, industry branches and import partners of these neighbours; is listed at *a*.
- Executing the query in *a* takes **762 ms**.
- The reordered triple patterns (*b*) query takes **163 ms**.

(a) Before Optimization

1. ?border o:importsCommodity ?iCommodity.
2. ?border o:industry ?industry.
3. c:TU o:border ?tuBorder.
4. ?tuBorder o:country ?border.
5. ?border o:importPartner ?impPartner.
6. ?impPartner o:country ?iPartner.

(b) After Optimization

1. c:TU o:border ?tuBorder.
2. ?tuBorder o:country ?border.
3. ?border o:importsCommodity ?iCommodity.
4. ?border o:industry ?industry.
5. ?border o:importPartner ?impPartner.
6. ?impPartner o:country ?iPartner.

Ant Colony Optimization

- Ant Colony Optimization (ACO) is a paradigm for designing meta-heuristic algorithms for combinatorial optimization problems
- It is inspired from the foraging behaviour of ant colonies

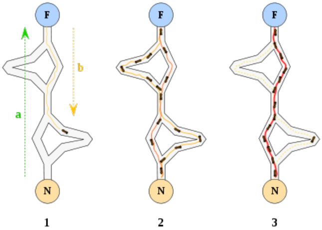


Figure source : http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

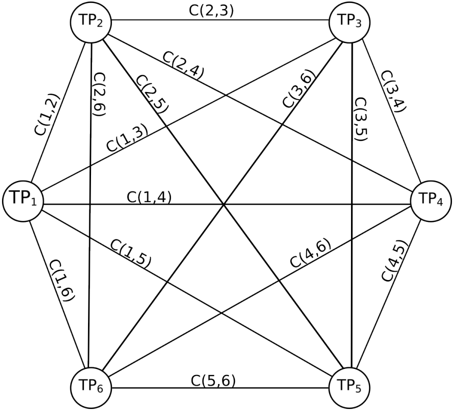
Ant Colony Optimization

- The original ant colony optimization algorithm, Ant System (AS) is used for solving the problem.

1. Set parameters and initialize ants
2. Iterate until reaching tour count
   (a) iterate until *tabu* list is full
      i. calculate probability of nodes
      ii. move the ant to the most possible node
   (b) calculate tour length for every ant and find best solution
   (c) evaporate
   (d) calculate ant travel cost, deposit pheromone and reset ant
3. return best solution path

Abstraction

- We abstract the Basic Graph Pattern as a complete graph.
- Each node represents a triple pattern and each edge represents estimated join cost of connected nodes.
- This graph is input of Ant System, which tries to find optimum triple pattern order.



Calculation of Costs

- Join cost of triple patterns is required by the ACO.
- It is based on selectivity (triple_pattern_cardinality/ontology_triple_count) of triple patterns.
- Calculation of costs (i.e. weights) of complete graph consists of two steps:
    1. Selectivity estimation of triple patterns.
    2. Cost calculation of join process.

Selectivity estimation of triple patterns

- Two techniques for selectivity estimation of triple patterns are used:
  1. Variable Counting for Selectivity Estimation: Based on ranking components of triple patterns as $sel(Subject) < sel(Object) < sel(Predicate)$ and classifying them as bound or unbound.
  2. Graph Statistics Handler: GSH provides the most accurate estimations, but it does not support estimation for triple patterns that has more than one bound component.

Cost Calculation of Join Process

- Two different weight finding (i.e. cost calculation) approaches have been implemented and experimented:
  1. Variable Counting for Cost Calculation: Based on ranking join types, e.g., Subject-Subject, Subject-Object joins.
  2. Modified Variable Counting (VC-M) for Cost Calculation: VC is modified with the aim of meeting the requirements of chain and chain-star queries. This modification consists of increasing ranking of Object-Subject joins by doubling its rank.
- To be able to calculate costs for edges, techniques that discussed above are combined.

Implementation: GSH and VC

- Selectivity of triple pattern which has only one bound component is estimated with GSH.
- In other cases VC is used for selectivity estimation.
- After selectivity estimation process, for cost calculation of join operation VC is used.

Implementation:GSH for Selectivity Estimation and VC-M for Cost Calculation

- Selectivity of triple patterns is estimated by using GSH technique.
- If triple pattern has more than one bound component, each bound component selectivity is calculated with GSH and product of these selectivities is returned as selectivity of triple pattern.
- For example triple pattern TP1 (*c:TU o:border* ?tuBorder) has two bound component.
- To calculate estimated selectivity of this triple pattern;
  - selectivity of subject $sel(S_1)$ and predicate $sel(P_1)$ - bound components - are obtained from GSH.
  - Then selectivity of TP_1 is calculated as $sel(TP_1) = sel(S_1) * sel(P_1)$.
- Afterwards VC-M is applied for cost calculation, to find the weights of the edges.

Implementation: Ant System

- After estimating the selectivity of triple patterns and calculating costs, the cost matrix is composed from obtained values and fed on to AS.
- At the start, ants are put on randomly chosen nodes.
- Ants decide for the next node using transition formula (eq. 1).

Implementation: Ant System

Transition formula

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} \qquad if \quad j \in N_i^k \tag{1}$$

Pheromone deposition formula

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij}^{k} \qquad \forall (i,j) \in L \tag{2}$$

Value of deposited pheromone formula

$$\triangle\tau_{ij}^{k} = \begin{cases} 1/C^k, if\ edge(i,j)\ belongs\ to\ T^k; \\ 0,\ otherwise; \end{cases} \tag{3}$$

Implementation: Ant System

- During the algorithm run, the pheromone trails of all edges are updated:
  - after every ant have constructed its tour (local update)
  - at the end of the every iteration (global update) when all ants are constructed their tour.
- This update mechanism is done in two steps:
  - First, pheromone values on all edges are decreased by pheromone evaporation rate ($0 < \rho \leq 1$) based on $\tau_{ij} \leftarrow \rho \times \tau_{ij}$ formula
  - Second, every ant deposits pheromone using formula 2 to the edges it has visited.

Experimental Setup

- All experiments are conducted using Java programming language
- Apache Jena framework with ARQ engine is used for queries
- All experimented queries use target ontologies that are extracted from CIA The World Factbook Web page.
- There are two ontologies that are stored in memory with total triple count of 95812.
- Queries have different triple pattern counts: 4, 6, 8, 10, 12 and 14.

Experimental Setup

- There are 4 different execution types:
  - Normal execution (NE) without any optimization
  - Algorithm proposed by Stocker et al. (2008) which uses Variable Counting estimation method (STO-VC)
  - Reordered execution which is an optimization included in Jena (RE)
  - AS Execution with variable counting (AS-VC) and AS execution which uses VC-M method (AS-VC-M) which are developed for this study

Experimental Setup - Parameters

- Parameters for AS algorithm are chosen by a preliminary parameter analysis which is performed by running algorithm for a fixed query for different values of parameters.
- Every query was run 10 times for every different execution types and average of timings are calculated and used for comparison.

Parameter	Value
graph size	triple pattern count
population size	50
iteration	100
α	2
β	1
evaporation rate (ρ)	0.5

Results

- All values in tables are in terms of milliseconds.
- These values include optimization, execution and population (retrieving data from the ontology) time except for NE. Values for NE include execution and population time.
- Executing some queries that contain less than 4 triple patterns takes less time than optimizing it.
- Importance of the proposed method can be seen in situations where optimized query can be saved and used later for several times.

Results

(a) Queries with 4 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	13435	6490	88	67	**63**
Q2	2828	2369	101	97	**74**
Q3	676	65	69	313	**46**
Q4	11371	11252	150	480	**127**

(b) Queries with 6 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	310	291	67	241	**56**
Q2	140	126	176	156	**116**
Q3	258	163	230	300	**212**
Q4	**13**	**13**	79	107	66

(c) Queries with 8 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	51873	7609	292	325	**277**
Q2	862	862	200	242	**192**
Q3	794	795	73	4056	**62**
Q4	49804	40972	**209**	279	230

(d) Queries with 10 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	4328	4251	3780	3760	**617**
Q2	126559	5289	4176	4277	**2241**
Q3	4657	4738	1824	4714	**1441**
Q4	313	293	**70**	188	137

(e) Queries with 12 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	24796	26410	**158**	502	378
Q2	5968	85	**72**	400	97

(f) Queries with 14 triple patterns

	NE	RE	STO-VC	AS-VC	AS-VC-M
Q1	19307	19219	2986	**1441**	2685
Q2	148241	153994	**216**	79389	247

Conclusions

- Proposed approach
  - Optimizes SPARQL triple patterns order using ant colony optimization for better execution
  - Does not require any prior knowledge
  - Reduces execution time considerably for the majority of the queries as shown in the experiments.

Future Work

- Improving heuristics - selectivity estimation and cost calculation - to optimize **all** the queries without requiring any prior graph information.
- Experimenting this method for different benchmark queries and different ontologies
- Development of different optimization techniques for the problem
- Integration of local search techniques to ACO and employing better parameter tuning techniques

- JENA - http://incubator.apache.org/jena/
- Java - http://www.oracle.com/tr/technologies/java/
- CIA Factbook
- ARQ-2.6.0
- Abdel Kader, R. and van Keulen, M. Overview of query optimization in xml database systems. Technical Report TR-CTI, EEMCS, University of Twente, Enschede, 2007.
- Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. Sci. Am., 284(5):34–43, 2001.
- Dorigo, M. and Stützle, T. Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.
- Harris, S. and Seaborne, A. SPARQL 1.1 Query Language - W3C Working Draft 05 Jan. 2012. 2012.

- Hartig, O. and Heese, R. The sparql query graph model for query optimization. In Proc. of the 4th European Conf. on The Semantic Web: Research and Applications, ESWC'07, pages 564–578, 2007.
- Hogenboom, A., Milea, V., Frasincar, F., and Kaymak, U. Rcq-ga: Rdf chain query optimization using genetic algorithms. In Proc. of the 10th Int. Conf. on EC-Web, pages 181–192, 2009.
- Hogenboom, F., Hogenboom, A., van Gelder, R., Milea, V., Frasincar, F., and Kaymak, U. Qmap: An RDF-based queryable world map. In 3rd Int. KMO Conf., pages 99–110, 2008.
- Ioannidis, Y. E. Query optimization. ACM Comput. Surv., 28(1):121–123, 1996.

- Maduko, A., Anyanwu, K., Sheth, A., and Schliekelman, P. Estimating the cardinality of rdf graph patterns. In Proc. of the 16th Int. Conf. on World Wide Web, pages 1233–1234. ACM, 2007.
- Maniezzo V, Gambardella L.M., D. L. F. New Optimization Techniques in Engineering, chapter Ant Colony Optimization, pages 101–117. Springer-Verlag, 2004.
- Neumann, T. and Weikum, G. Rdf-3x: a risc-style engine for rdf. Proc. VLDB Endow., 1(1):647–659, 2008.
- Ozsu, M. T. and Blakeley, J. A. Query processing in object-oriented database systems. In Modern Database Systems, pages 146– 174. ACM Press and Addison-Wesley, 1995.

- Ruckhaus, E., Ruiz, E., and Vidal, M. Query evaluation and optimization in the semantic web. Theory Pract. Log. Program., 8(3):393–409, 2008.
- Shironoshita, E. P., Ryan, M. T., and Kabuka, M. R. Cardinality estimation for the optimization of queries on ontologies. SIGMOD Rec., 36(2):13–18, 2007.
- Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., and Reynolds, D. Sparql basic graph pattern optimization using selectivity estimation. In Proc. of the 17th Int. Conf. on WWW, pages 595–604. ACM, 2008.
- Stuckenschmidt, H., Vdovjak, R., Broekstra, J., and Houben, G. Towards distributed processing of rdf path queries. Int. J. Web Eng. Technol., 2(2/3):207–230, 2005.