DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OPTIMIZATION OF SPARQL QUERIES USING ARTIFICIAL INTELLIGENCE TECHNIQUES

by
Elem GÜZEL KALAYCI

July, 2012
İZMİR

OPTIMIZATION OF SPARQL QUERIES USING ARTIFICIAL INTELLIGENCE TECHNIQUES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of Science in
Computer Engineering

by

Elem GÜZEL KALAYCI

July, 2012
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"OPTIMIZATION OF SPARQL QUERIES USING ARTIFICIAL INTELLIGENCE TECHNIQUES"** completed by **ELEM GÜZEL KALAYCI** under supervision of **ASST. PROF. DR. DERYA BİRANT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Asst. Prof. Dr. Derya BİRANT

Supervisor

.....

Asst. Prof. Dr. Özge ŞAHİN

(Jury Member)

.....

Prof. Dr. Alp KUT

(Jury Member)

Prof.Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank to my supervisor, Asst. Prof. Dr. Derya Birant, for her supervision and suggestions. Also I would like to thank to my family for tirelessly supporting me through all the life.

I would like to mention the support of Tahir Emre and thank to him for his endless patience, for motivating me when I was about to give up, for his helpfulness and for all other things he did for me which are too many to mention.

Also I would like to thank to Alexander Hogenboom for sharing Factbook ontology.

Elem GÜZEL KALAYCI

OPTIMIZATION OF SPARQL QUERIES USING ARTIFICIAL INTELLIGENCE TECHNIQUES

ABSTRACT

Today, configuring and controlling the overwhelming volumes of information on the web is an important problem. Semantic web is a paradigm that is proposed for solving this important problem. Still, semantic web can't be counted as a mature paradigm and it contains some issues that must be dealt. One important challenge in semantic web is decreasing execution times of queries. An approach for decreasing execution times of queries is reordering triple patterns.

In this study, an Ant Colony Optimization approach for optimizing SPARQL queries is proposed. Different Ant Colony Optimization Meta-heuristic algorithms - Ant System, Elitist Ant system and Max-Min Ant System - are implemented based on this approach. This proposed novel optimization method is implemented using ARQ query engine and it optimizes the queries for in-memory models of ontologies.

Queries are abstracted as a complete graph whose nodes represent triple patterns and whose edges represent join costs. Artifical ants that are used in ACO algorithms traverse this graph. Transition rule which effects the decision of ants for choosing the next node is provided by considering selectivity of triple patterns (candidates of the next node). In order to estimate selectivity of triple patterns, GSH which provides accurate size information, Variable Counting which is based on ranking triple pattern components and Modified Variable Counting which modified to improve the performance of chain and chain-star queries, are used.

Proposed approach is examined by querying two different ontologies LUBM (includes 162.871 triples) and Factbook (includes 95.813 triples) with various structures of queries like chain, star, cyclic, chain-star, chain-cyclic, etc.

Contributions of the proposed approach are optimizing order of triple patterns in SPARQL queries using ant colony optimization for lesser and nearly optimal execution time and real time optimization without requiring any prior domain knowledge. Experiments show that proposed methods reduce execution time of queries considerable.

Keywords: SPARQL, query optimization, ant colony optimization, ant system, semantic web, artificial intelligence

YAPAY ZEKA TEKNİKLERİ KULLANILARAK SPARQL SORGULARININ OPTİMİZASYONU

ÖZ

Bugün internetteki oldukça yüksek miktardaki veriyi yönetmek ve yapılandırmak önemli bir problemdir. Anlamsal Ağ yoğun miktarda veriyi yapılandırmak ve yönetmek için önerilmiş bir paradigmadır. Bununla beraber Anlamsal Ağ olgun bir paradigma değildir ve çözülmesi gereken sorunları vardır. Anlamsal Ağ'ın önemli zorluklarından biri sorguların çalıştırılma zamanını azaltmaktır. Sorguların çalışma zamanını azaltmaya yönelik bir yaklaşım üçlü desenlerini yeniden sıralamaktır.

Bu çalışmada SPARQL sorgularını üçlü desenleri yeniden sıralayarak iyileştirmek için bir Karınca Kolonisi Eniyilemesi yaklaşımı sunulmuştur. Karınca Kolonisi Eniyilemesi algoritmaları olan Karınca Sistemi, Elitist Karınca Sistemi ve Max-Min Karınca Sistemi algoritmaları gerçekleştirilmiştir. Bu önerilen yeni yaklaşım ARQ sorgu motoru kullanılarak gerçekleştirilmiştir ve bellekteki ontoloji modellerini sorgulayan sorguları iyileştirmektedir.

Sorgular, düğümleri üçlü desenleri temsil eden, kenarları ise birleştirme (join) maliyetini temsil eden tam çizge şeklinde soyutlanmıştır. KKE için kullanılan yapay karıncalar bu tam çizgeyi dolaşmaktadır. Karıncaların sonraki düğümü seçmesinde etkili olan geçiş kuralı, üçlü desenlerinin (sonraki düğüm adayları) seçiciliği göz önünde bulundurularak oluşturulmuştur. Üçlü desenlerinin seçiciliğini tahminlemek amacıyla; kesin boyut bilgisi sağlayan GSH, üçlü desen bileşenlerini derecelendirmeye dayanan "Variable Counting" ve zincir, zincir-yıldız sorgularının başarımını arttırmak için değiştirilen "Modified Variable Counting" kullanılmıştır.

Önerilen yaklaşımlar LUBM (162.871 üçlü içerir.) ve Factbook (95.813 üçlü içerir.) olmak üzere iki farklı ontolojinin zincir, yıldız, döngüsel, zincir-döngüsel, vb. gibi çeşitli yapıdaki sorgularla sorgulanmasıyla sınanmıştır.

Bu çalışmanın katkıları daha düşük bir çalışma zamanı için Karınca Kolonisi Eniyilemesi algoritmaları kullanılarak SPARQL sorgularındaki üçlü desenlerinin sıralamasının iyileştirilmesi ve önceden herhangi bir alan bilgisine ihtiyaç duymadan gerçek zamanlı eniyileştirmedir. Deneyler önerilen yaklaşımın eniyilenmiş sorguların çalışma zamanını önemli ölçüde azalttığını göstermektedir.

Anahtar sözcükler: SPARQL, sorgu eniyileme, karınca kolonisi eniyilemesi, karınca sistemi, anlamsal ağ, yapay zeka

CONTENTS

Page

M.Sc THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ vi

CHAPTER ONE - INTRODUCTION 1

1.1 General 1
1.2 Purpose 2
1.3 Organization of the Thesis 2

CHAPTER TWO – SEMANTIC WEB 4

2.1 Overview 4
2.2 SPARQL 7
2.3 Jena and ARQ 9

CHAPTER THREE – ANT COLONY OPTIMIZATION META-HEURISTIC ... 11

3.1 Ant System 13
3.1.1 Tour construction 14
3.1.2 Pheromone update 15
3.2 Elitist Ant System 15
3.3 MAX-MIN Ant System 16

CHAPTER FOUR – QUERY OPTIMIZATION 18

4.1 SPARQL Query Optimization in ARQ 20
4.2 Selectivity of Triple Patterns 23

CHAPTER FIVE – RELATED WORK 25

5.1 Join Ordering in Relational Databases using Ant Colony Optimization 26
5.2 Reordering Triple Patterns in RDF Databases 26

CHAPTER SIX – PROPOSED APPROACH 30

6.1 Selectivity Estimation of Triple Patterns 31
6.1.1 Variable Counting for Selectivity Estimation 31
6.1.2 Graph Statistics Handler 32
6.2 Cost Calculation of Join Process 32
6.2.1 Simple Cost - Cartesian Product of Cardinalities 32
6.2.2 Variable Counting for Cost Calculation 33
6.2.3 Modified Variable Counting for Cost Calculation 37

CHAPTER SEVEN - IMPLEMENTATION 39

7.1 Selectivity Estimation and Cost Calculation 39
7.1.1 GSH and Simple Cost 39
7.1.2 GSH and VC 40
7.1.3 GSH for Selectivity Estimation and Modified VC for Cost Calculation 40
7.2 ACO Implementations 40
7.2.1 Ant System Implementation 40
7.2.2 Elitist Ant System Implementation 43
7.2.3 MAX-MIN Ant System Implementation 43

CHAPTER EIGHT – EXPERIMENTAL WORK 45

8.1 Methodology 45
8.2 CIA Factbook Ontology Experiments 45
8.2.1 Results and Discussion 48
8.3 LUBM Ontology Experiments 54
8.3.1 Results and Discussion 55

CHAPTER NINE - CONCLUSION & FUTURE WORK.....66

9.1 Conclusion.....66
9.2 Future Work 67

REFERENCES.....68

APPENDICES 74

A. LIST OF ABBREVIATIONS 75
B. LIST OF SPARQL QUERIES 77
a. CIA Factbook Queries 77
b. LUBM Queries 90

CHAPTER ONE
INTRODUCTION

1.1 General

Today information on the web is getting increased enormously and it is the main environment for universal information share and exchange. But an important problem with that overwhelming volume of information is configuring and controlling it. For this problem “semantic web” has been introduced by the scientists and developers. Semantic web is the paradigm for the understanding of and responding to users requests and understanding, interpreting and deducting web content by the computers (Berners-Lee & others, 2001). Although semantic web is a popular technology with all benefits and dense usage, developers and users encounter with some problems when working on topics like ontology matching, ontology aligning, query optimization and reasoning, improving performance, etc. There are different solutions for these topics in the literature.

Query optimization is a mature and comprehensive research area. Various deterministic and probabilistic techniques have beeen applied to query optimization problem in various domains (e.g., XML (Che, 2006; Kader, 2007), Relational DBMS (Ioannidis, 1996; Chaudhuri, 1998; Neumann, 2008), Object-Oriented DBMS (Mitchell, 1991; Özsu, 1995)) so far. Significant parts of query optimization studies draw attention to enormous effect of join ordering to query execution time. Although there have been remarkable studies about query optimization in RDF databases, query optimization area has not yet reach to sufficient maturity at RDF domain. So this study makes contribution as implementing a swarm intelligence technique ACO to query optimization problem in RDF domain which are not attempted before.

1.2 Purpose

In this work we are proposing ant colony optimization algorithms like Ant System, Elitist Ant System and Max-Min Ant System for optimizing SPARQL queries which are used for semantic web ontology querying. Ant Colony Optimization is a meta-heuristic optimization technique that is inspired from the real ant colonies behaviour and method for finding food. Contributions of this study are optimizing SPARQL Basic Graph Pattern (i.e., optimizing order of the triple patterns) using ant colony optimization algorithms and real time optimization without requiring any prior knowledge.

1.3 Organization of the Thesis

This thesis includes nine chapters and the remaining of this thesis is organized as follows.

In Chapter 2, there is a general information about semantic web paradigm, ontology notion, ontology querying language SPARQL and finally about SPARQL query engine ARQ.

In Chapter 3, fundamentals of Ant Colony Optimization Meta-heuristic and ACO algorithms Ant System, Elitist Ant System and Max-Min Ant System algorithms are explained.

In Chapter 4, an overview of query optimization on Database Management Systems is presented in the point of view of SPARQL query optimization on ARQ.

In Chapter 5, the related work of join ordering using Ant Colony Optimization Meta-heuristic in DBMS and reordering triple patterns with different algorithms is discussed.

In Chapter 6, the proposed and improved selectivity estimation techniques are explained.

In Chapter 7, the determined selectivity estimation and cost calculation combinations and subsequently implemented ACO algorithms are explained in detail.

In Chapter 8, the experimental setup and the methodology for CIA Factbook and LUBM ontologies are explained. Then results of experiments are introduced and discussed.

Finally in Chapter 9, conclusions of this study are presented. Also shortcomings and future directions are discussed.

CHAPTER TWO
SEMANTIC WEB

2.1 Overview

Today, “most of the Web's content is designed for humans to read, not for computer programs to manipulate them meaningfully” (Berners-Lee, Hendler & Lassila, 2001). Although computers can sweep millions of web pages in a really small time, they don't have the ability to understand the content. Content understanding, interpreting and deducting functions are done by the humans. However a human can't do all these work as fast as a computer. Thus, for the computers to understand meaning of information and to build semantic relations, a need for a new paradigm arises. Here semantic web comes to rescue, and meets this need.

Semantic Web is layered and standardized as depicted at Figure 2.1. XML which is located at the bottom of Semantic Web layer cake allows users to generate user-defined vocabulary and to structure web documents using this vocabulary.

RDF which based on XML syntax is data model of Semantic Web to define statements in the form of subject-predicate-object triples.

RDF Schema is used to generate schema of ontology (i.e., to define classes, properties, relations between this concepts and domain, range restrictions).

Proof layer includes representation and validation of proof processes and the top of the cake, trust layer appears for usage of digital signatures.

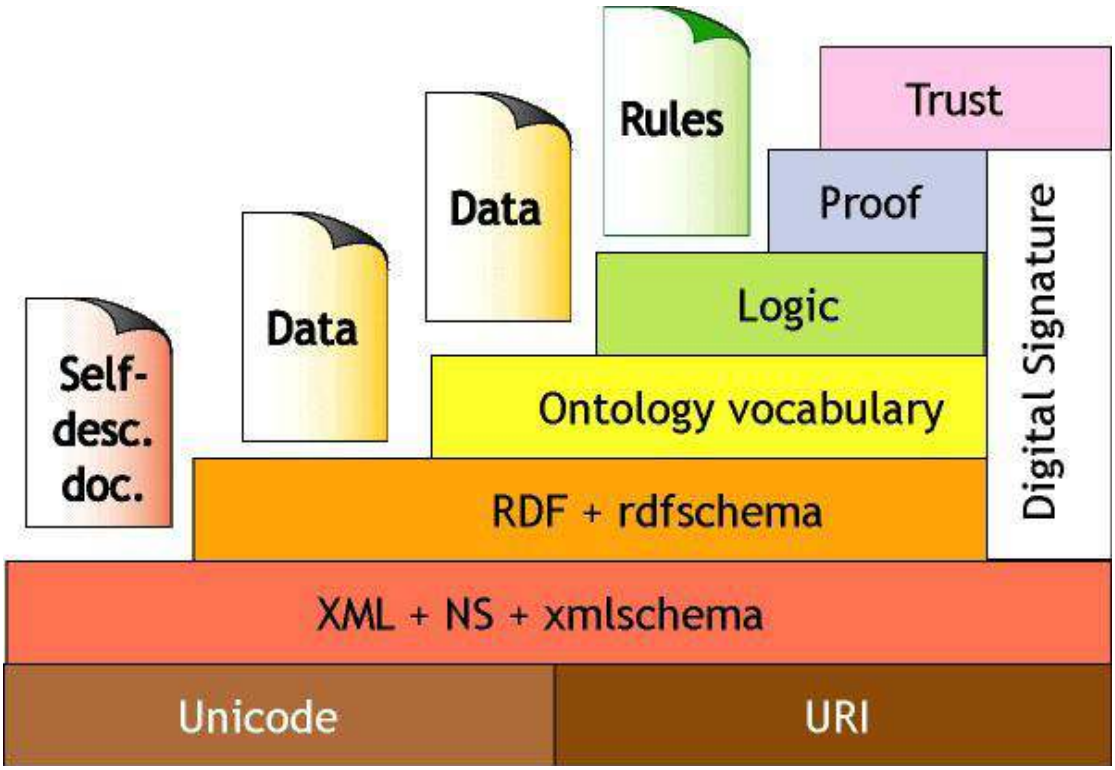

Figure 2.1 Layers of Semantic Web (Antoniou & van Harmelen, 2003)

Tim Berners-Lee, who developed WWW, is started this paradigm development attempt. Currently The "Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation" (Berners-Lee, Hendler & Lassila, 2001). The purpose of the Semantic Web is "to introduce semantic content in the huge amount of unstructured or semi-structured information sources available on the web by using ontologies" (Caliusco & Stegmayer, 2010).

Ontology is theory about "the nature of existence, of what types of things exist in philosophy; ontology as a discipline studies such theories" (Berners-Lee, Hendler & Lassila, 2001). "Artificial Intelligence and Web researchers have co-opted the term for their own jargon, and for them ontology is a document or file that formally defines the relations among terms" (Berners-Lee, Hendler & Lassila, 2001). As Caliusco & Stegmayer (2010) points out "ontology provides a vocabulary about concepts and their

relationships within a domain, the activities taking place in that domain, and the theories and elementary principles governing that domain". Ontologies usually are referred as a graph structure which consists of (Davies, 2006; Caliusco, 2010):

1. a set of concepts (vertices in a graph),
2. a set of relationships connecting concepts (directed edges in a graph), and
3. a set of instances assigned to a particular concept (data records assigned to concepts or relations).

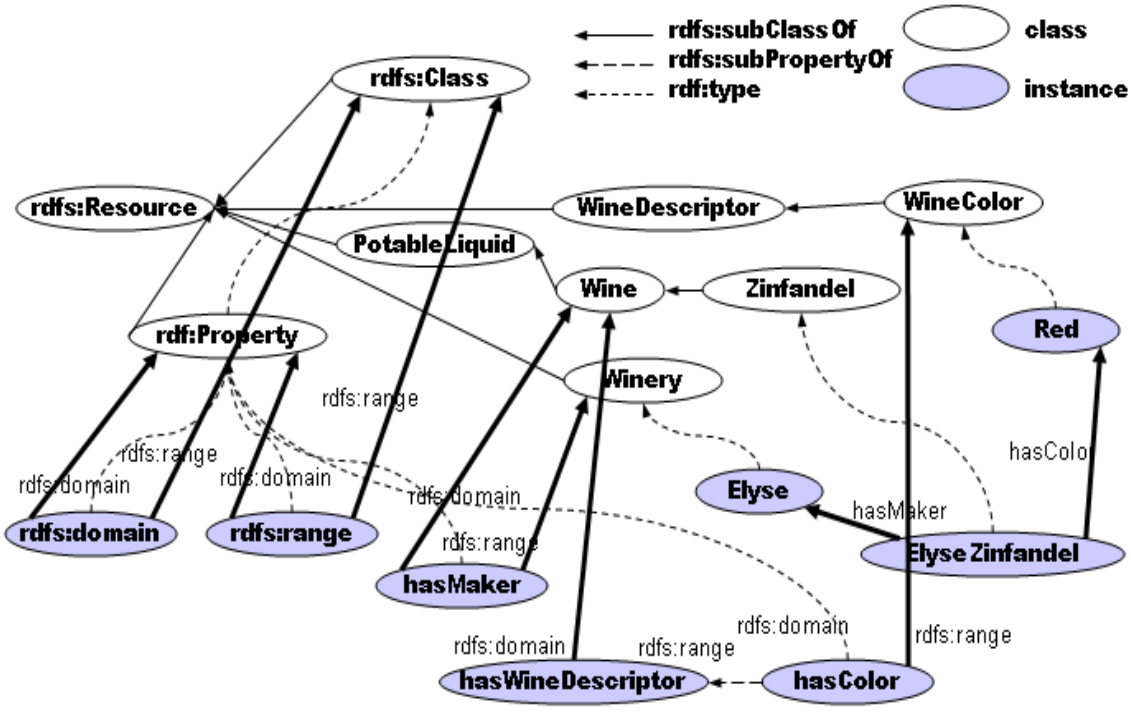

Figure 2.2 Example wine ontology RDF graph (Koide & Kawamura, 2004)

There is a RDF graph at figure 2.2 for an example wine ontology which describes wine kinds, producers, colours, etc.

Currently XML is used for development of semantic web and ontologies. Also for defining the semantics for digital content, it is necessary to formalize the ontologies by

using specific languages as Resource Description Framework (RDF) and Web Ontology Language (OWL) (Caliusco & Stegmayer, 2010). RDF is a "directed, labelled graph data format for representing information in the Web" (W3C, 2012). RDF is a general-purpose language for representing information about resources in the Web (Caliusco & Stegmayer, 2010). It is particularly intended for representing meta-data about web resources, but it can also be used to represent information about objects that can be identified on the Web (Caliusco & Stegmayer, 2010). On the other hand, OWL describes classes, properties, and relations among these conceptual objects in a way that facilitates machine interoperability of web content (Breitmann & others, 2007).

2.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a RDF query language and protocol which is used to express queries across diverse data sources (W3C, 2012). The data can be stored natively as RDF or viewed as RDF via middle-ware (W3C, 2012). SPARQL has the capability to query required and optional graph patterns along with their conjunctions and disjunctions (W3C, 2012). The results of SPARQL queries can be result sets or RDF graphs (W3C, 2012). SPARQL also supports following features (W3C, 2012):

- aggregation,
- sub-queries,
- negation,
- creating values by expressions,
- extensible value testing,
- constraining queries by source RDF graph.

There are essential keywords at a SPARQL query. The keyword *prefix* is used to relate label with IRIs. Select keyword is needed to specify variables that will be projected and where keyword is used for defining triple patterns. Conditions are described in SPARQL with triple patterns. Triple patterns have three components

(subject, predicate and object) like RDF triples and these components may be bound (concrete) or unbound (variable). Basic Graph Pattern represents the set of triple patterns. SPARQL, like most RDF query languages, allows for graph structure search through a conjunction of triples typically processed using joins (Maduko, 2007; Haase, 2004).

There is an example query below which queries agricultural products of Turkey on CIA Factbook ontology.

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
SELECT ?countries ?aProduct
WHERE {
    ?country c:name "TURKEY".
    ?country o:agricultureProduct ?aProduct.
}
```

Table 2.1 Components of triple patterns

Subject	Predicate	Object
?country	c:name	"TURKEY"
?country	o:agricultureProduct	?aProduct

Table 2.1 shows components of triple patterns which indicated at example query. There are two type components: concrete (bound) and variable (unbound). "?" is used for defining variables. Also "c" and "o" are the labels defined by prefixes which are used instead of IRIs and "TURKEY" is a string literal. It means "TURKEY" is a value of "name" property of "http://www.daml.org/2001/09/countries/fips#" IRI.

2.3 Jena and ARQ

Jena is a RDF model API for Semantic Web programmers to manipulate and store RDF graphs in memory or in persistent storage (Carroll & others, 2004). Jena is an open-source project which was developed by researchers of HP Labs (Apache Jena, 2012a). Jena supports N-Triples, Turtle and XML formats of RDF data and includes a RDF API, an Inference API and a storage API (See Figure 2.3 for architecture of Jena). Also presents a rule based inference engine for reasoning.

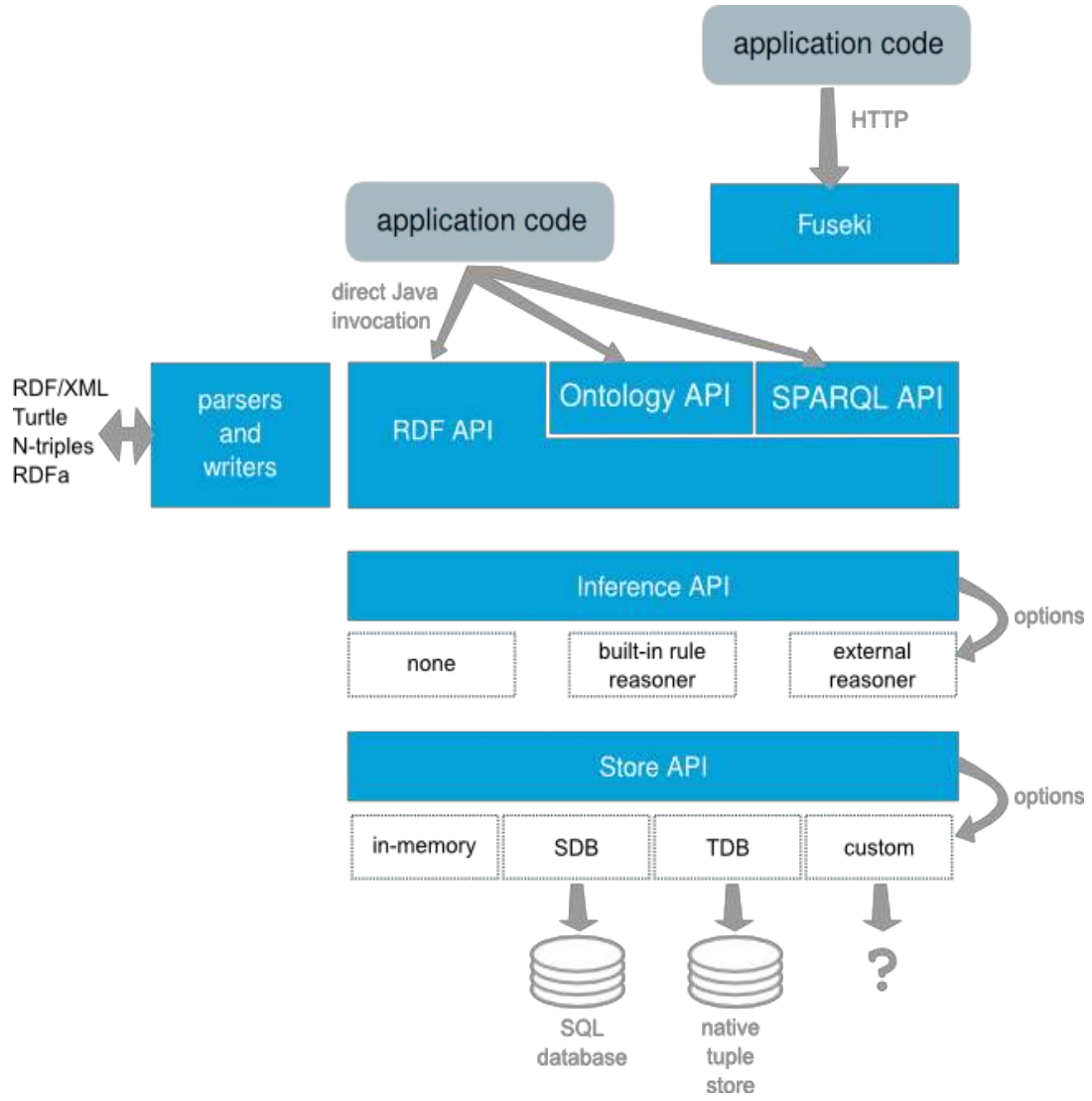

Figure 2.3 Jena Architecture (Apache Jena, 2012b)

In this study most focused part of the Jena is ARQ (Apache Jena, 2012c). ARQ is an extendible SPARQL query engine of Jena RDF toolkit. SPARQL query processing steps of ARQ are listed below:

1. Parsing String to Query
2. Translation
3. Optimization of the Algebra Expression
4. Query Plan Determination and Low Level Optimization
5. Evaluation of the Query Plan

Parsing process means transforming the query string to query object that is represented with abstract syntax tree data structure. *Translation* means transforming the query object to SPARQL algebra expression and *optimization* means applying some kind of transformations to SPARQL algebra expression (e.g., replacing equality filters with a more efficient graph pattern (Apache Jena, 2012d)).

Low level optimization is about reordering triple patterns, and *query plan determination* means choosing the cheapest query execution plan and finally *evaluation* means processing the selected query execution plan.

ARQ can be extended and modified for customizing query execution process by ARQ developers by implementing, custom filters and property functions, query optimizers, graph interfaces, etc., in order to meet particular requirements.

Due to customize query optimizer of ARQ, Ant Colony Optimization Meta-heuristic algorithms are implemented. In the next chapter ACO algorithms are explained.

CHAPTER THREE
ANT COLONY OPTIMIZATION METAHEURISTIC

Swarm intelligence is the collective behaviour that emerges from a group of social individuals (Bonabeau, 2008). One of the early studies of swarm intelligence investigated the foraging behaviour of ants (Bonabeau, 2008). The first algorithm which can be classified as swarm intelligence was presented in 1991 (Dorigo, 1991; Colorni, 1991) and, since these examples, many different examples of the basic principle have been reported in the literature (Maniezzo, 2004).

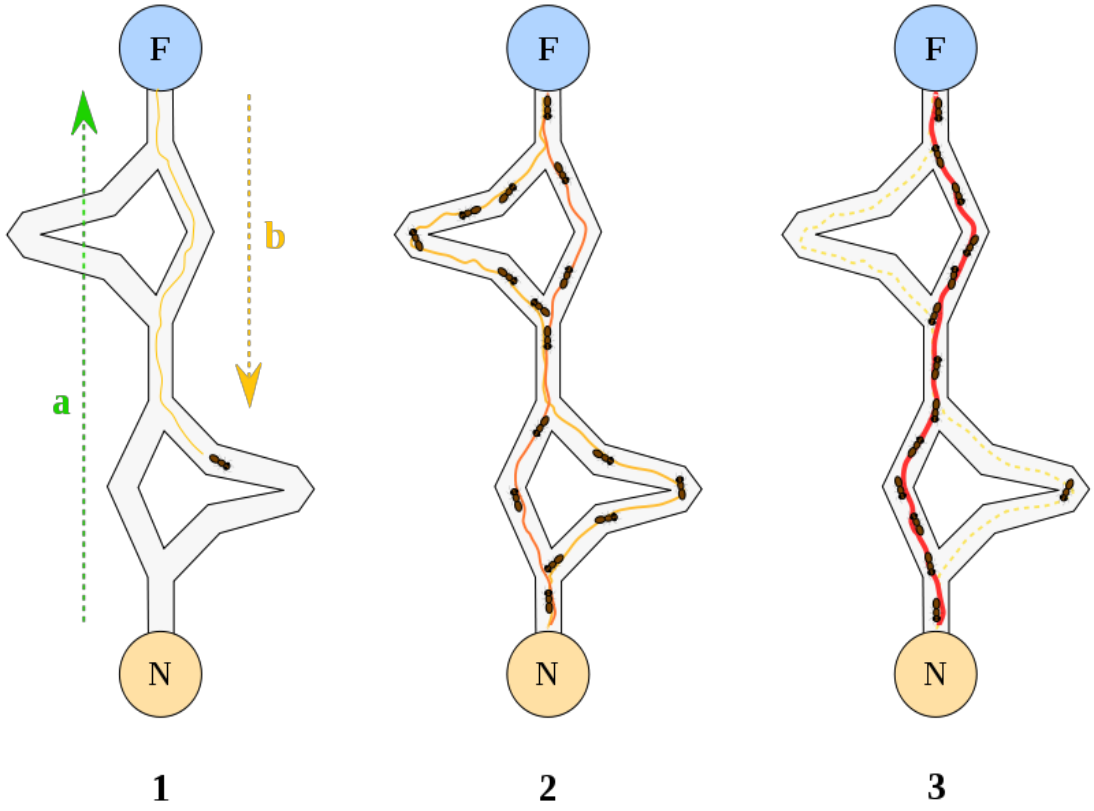

Figure 3.1 Finding shortest path by ants (Dréo, 2006)

Several different aspects of the behaviour of ant colonies have inspired different kinds of ant algorithms (Dorigo & Stützle, 2004). Foraging, division of labour, brood

sorting, and cooperative transport can be given as examples to these different behaviours (Dorigo & Stützle, 2004). All these examples show ants coordination of their activities via “a form of indirect communication mediated by modifications of the environment” which is called *stigmergy* (Dorigo & Stützle, 2004). Figure 3.1 illustrates the indirect communication by pheromone deposition between ants.

Ant Colony Optimization (ACO) is a meta-heuristic in which artificial ants in a colony cooperate to find good solutions to difficult discrete optimization problems (Dorigo & Stützle, 2004). It is also a paradigm for designing meta-heuristic algorithms for combinatorial optimization problems (Dorigo & Stützle, 2004). ACO is inspired by the foraging behaviour of ant colonies and cooperation is a key component of ACO algorithms (Dorigo & Stützle, 2004). Computational resources are allocated to a set of simple agents (artificial ants) which communicate indirectly by stigmergy (Dorigo & Stützle, 2004). This communication is mediated by the environment (Dorigo & Stützle, 2004) by the help of pheromone intensity.

The main underlying idea of ACO is that of “a parallel search over several constructive computational threads based on local problem data and on a dynamic memory structure containing information on the quality of previously obtained result”, and this idea is “loosely inspired by the behaviour of real ants” (Maniezzo, 2004). The collective behaviour emerging from the interaction of the different search threads has proved effective in solving combinatorial optimization (CO) problems (Maniezzo, 2004).

Both static and dynamic combinatorial optimization problems can be solved by the ACO algorithms (Dorigo & Stützle, 2004). In static problems “characteristics of the problem are given once when the problem defined, and doesn't change during the problem solved” (Dorigo & Stützle, 2004). Dynamic problems are defined as “a function of some quantities whose value is set by the dynamics of an underlying system” (Dorigo & Stützle, 2004). A well known example of static problems is Travelling Salesman

Problem. In this study, optimizing SPARQL queries by reordering triple patterns is also an example of static problems. So, ACO algorithms that are developed for this study are based on algorithmic skeleton of ACO meta-heuristic algorithms which is applied to "static" combinatorial optimization. This skeleton is listed below:

- Set parameters, initialize pheromone trails
- While termination condition not met do
  - Construct ant solutions
  - Apply local search (optional)
  - Update pheromones

The original ant colony optimization algorithm is known as Ant System (Dorigo, 1991; Dorigo, 1992; Dorigo, 1996) and was proposed in the early nineties, since then, a number of other ACO algorithms were introduced (Dorigo, 2006). In this study Ant System (AS), Elitist Ant System (EAS) and MAX-MIN Ant System (MMAS) are developed and used to solve the problem discussed. The problem discussed in this study looks like a Travelling Salesman Problem (TSP). Also it can be told that it is a adaptation of TSP. Although this problem seems more similar to the sequential ordering problem (SOP) – "finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and the nodes, subject constraints between nodes" (Dorigo & Stützle, 2004) - , these different ACO algorithms (AS, EAS and MMAS) are going to explained in the bearing TSP in mind for a more easy understanding. Also it must be noted that SOP can be converted to an asymmetric travelling salesman problem without closed path, i.e. final path is not a tour, by removing weight from nodes and adding them to arcs (Dorigo & Stützle, 2004).

3.1 Ant System

Ant System was the first ACO algorithm that is proposed, which is used to solve TSP (Dorigo, 1991a; Dorigo & Stützle, 2004). There were three different versions of AS,

which are called *ant-density*, *ant-quantity* and *ant-cycle* (Dorigo & Stützle, 2004). Difference between these versions is the about when ant pheromone update is done (Dorigo & Stützle, 2004):

- Pheromone update is done after a move from one city to an adjacent city in the ant-density and ant-quantity.
- Pheromone is updated only after all the ants had constructed their tours in the ant-cycle version. The amount of pheromone deposited by each ant is decided by a function of the quality of the tour.

In this study ant-cycle version of the Ant System is used to solve the problem. Solution construction of ants and pheromone update are the two main phases of the AS algorithm (Dorigo & Stützle, 2004). Pheromone trails initialization is an important heuristic in AS, this trails is set to a value slightly higher than the expected amount of pheromone deposited by the ants in one iteration (Dorigo & Stützle, 2004) to act as a good heuristic. Generally this value is obtained from the formula $\forall(i,j), \tau_{ij} = \tau_0 = \frac{m}{C^{nn}}$ where m is ant count and C^{nn} is the length of a tour generated by the nearest-neighbour heuristic – any other reasonable tour construction procedure can be used - (Dorigo & Stützle, 2004). This choice is based on the fact of quickly biased search which is a result of too low initial pheromone values. Also very high pheromone values could result in waiting for reasonable values which is achieved by many iterations evaporation. Formulas for tour construction, pheromone update and importance of parameters for the problem studied are explained in detail in the *Implementation* chapter, so it won't be explained here again to avoid from repetition. Readers who are interested more in AS – and similar algorithms - for TSP can look to the Dorigo & Stützle's book (2004).

3.1.1 Tour construction

In AS, m ants construct the tour concurrently, initally ants are put on random cities (Dorigo & Stützle, 2004). At each construction step a *probabilistic action choice rule*

(*transition rule*) which is called *random proportional rule* is applied to an ant and with this rule this ant decides which city to visit next (Dorigo & Stützle, 2004).

3.1.2 Pheromone update

Updating pheromone trails in AS is done after the all ants construct their tour. It is done in two steps: pheromone evaporation – lowering the pheromone value on all arcs by a constant factor which is called evaporation rate - and pheromone deposition – adding pheromone on the arcs the ants have crossed in their tours by pheromone deposition formula (Dorigo & Stützle, 2004).

Elitist Ants, Elitist Ant System and MAX-MIN Ant System only differ from Ant System in pheromone deposition strategy. Tour construction and pheromone evaporation is implemented just same as the Ant System.

3.2 Elitist Ant System

Elitist Ant System (EAS) was the first improvement to the AS. It was introduced by Dorigo (1992) and Dorigo & others (1991a, 1996). In this *elitist strategy* for ant system the idea is "to provide strong additional reinforcement to the arcs belonging to the best tour found since the start of the algorithms" (Dorigo & Stützle, 2004). This tour is denoted by T^{bs} (best-so-far) and it is used in pheromone update step by adding a quantity e/C^{bs} to its edges (e is a parameter that defines the weight of best-so-far tour) (Dorigo & Stützle, 2004). Computational studies show that elitist strategy with an appropriate e parameter value results in better tours in a lower number of iterations (Dorigo & Stützle, 2004).

Elitist Ants (ES) is another implementation of elitism for ant colony optimization. In elitist ants the only difference resides in pheromone deposition of the ants. In this algorithm, some of the ants are randomly chosen and marked as elitist ants. When these

ants finish constructing their tour, pheromone deposited on the tour they constructed based on their tour cost ($1/C^{elitist}$). All other mechanisms are same as Ant System.

3.3 MAX-MIN Ant System

MAX-MIN Ant System (MMAS) (Stützle & Hoos, 1997, 1999, 2000) vary from AS by four main modifications (Dorigo & Stützle, 2004):

1. Strongly exploits the best tours found: only either the iteration-best ant (the ant that produced best tour in the current iteration), or the best-so-far ant is allowed to deposit pheromone. After all ants have constructed a tour, pheromones are updated first by applying evaporation as in AS, then depositing new pheromone as in formula $\tau(n+1)_{ij} \leftarrow \tau(n)_{ij} + \Delta\tau_{ij}^{best}$. In this formula the ant that allowed to add pheromone may be either best-so-far ($\Delta\tau_{ij}^{best} = \frac{1}{C^{bs}}$) ant, or iteration-best ant ($\Delta\tau_{ij}^{best} = \frac{1}{C^{ib}}$). Decision which ant to choose is changes the greedy behaviour of the MMAS: when updates are always performed by best-so-far ant the search focuses quickly around T^{bs}, when updates are updated by iteration-best ant the number of arcs that receive pheromone is larger, so the search is less directed.

2. To counteract against excessive grow of pheromone trails on arcs of a good (but potential suboptimal) tour, MMAS limits possible range of pheromone trail values to the $[\tau_{min}, \tau_{max}]$. This limit helps to avoid search stagnation situations. Also, each time a new best-so-far tour is found, the value of τ_{max} is updated. And τ_{min} is set to $\frac{\tau_{max}}{a}$, where a is a parameter. Experiments show that the lower pheromone limit τ_{min} plays more important role than τ_{max}, still τ_{max}

remains useful setting for setting the pheromone values during the occasional trail reinitalizations.

3. At the start of the algorithm, the pheromone trails are initialized to the (or to an estimate) upper pheromone trail limit (τ_{max}). This, together with small pheromone evaporation rate increases the exploration of the tours at the start of the search, because of slow increase in the relative difference in the pheromone trail levels.

4. Pheromone trails are reinitalized each time the system reachs a stagnation or when no improvement in the tour has been seen for a certain number of consecutive iterations. This increases the exploration of paths that have only a small probability to being chosen.

CHAPTER FOUR
QUERY OPTIMIZATION

Query optimization that has been dealt for decades, is a critical issue for DBMS's. There is rich knowledge in literature about it. However query optimization is nearly new for RDF databases. Query optimization can be defined as searching for optimal query execution strategy. Due to importance of execution time for queries, query optimization is a crucial step on query processing. There are various query execution plans for a single query that are performed by query engines with different execution times and returns same result set. Performance diversity between query execution plans of same query may be enormous and so that finding optimal or nearly optimal execution plans, is a very important task for query engines on query processing. For this reason query optimizer is the one of the four essential part of query engines. Modules of query processing for DBMS's are (Ioannidis, 1996):

- Query Parser
- Query Optimizer
- Code Generator (Interpreter)
- Query Processor

Pairing query processing parts on DBMS to SPARQL query engines may be possible (See Table 2.1 for query processing steps of ARQ SPARQL query engine). As mentioned by Ioannidis (1996) query optimizer has some essential parts (Figure 4.1).

As depicted in Figure 4.1, query optimizer has two stages: rewriting and planning. The goal at rewriting stage is transforming poorly expressed procedural queries that causes to inefficient execution plans, into more declarative queries that returns same result set with former one (Pirahesh & others, 1992).

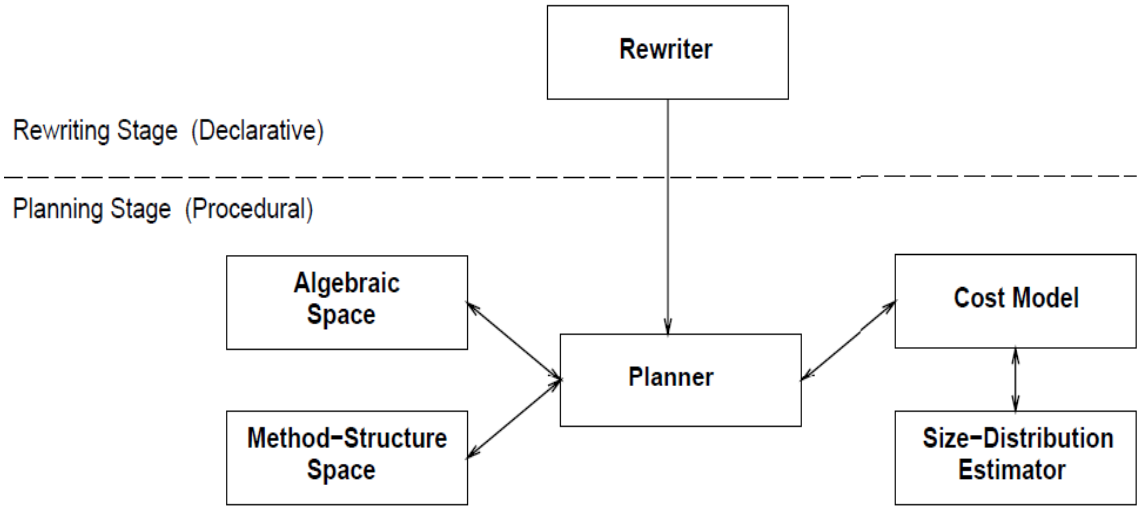

Figure 4.1 Essential parts of query optimizer (Ioannidis, 1996)

At planning stage simply all possible execution plans are determined with respect to algebraic space and method-structure space and the cheapest one is chosen with respect to search strategy and cost model. Functions of planning stage parts can be expressed shortly as below (Ioannidis, 1996):

- *Planner* is the fundamental part that works with other planning stage parts and chooses the optimal (with cheapest cost) query execution plans by a search strategy.

- *Algebraic space* transforms queries to relational algebra and generates query trees considering some restrictions. These restrictions decrease the algebraic space size and distinguish expensive query trees from cheaper query trees.

- *Method-structure space* determines suitable implementation methods for each query tree. To put it clearly; this part determines join methods (nested-loop join, merge scan, hash join, etc.), data structures and index types for each query tree that generated by algebraic space.

- *Cost model* calculates cost of each execution plan considering join methods, index types and estimated selectivity of predicates (i.e., division of predicate cardinality to size of data). Selectivity estimation issue is handled at Size-distribution estimator part of planner stage.

- *Size-distribution estimator* is used by cost model part and also estimates selectivity of predicates, size of relations and size of intermediate results. Different estimation techniques have been used so far, like various heuristics and mathematical formulas and additionally statistical synopses have been utilized.

Things that have been mentioned so far in this chapter, is related with query optimization of SQL queries at DBMSs and with regard to domain of this study (i.e., RDF ontologies) the necessity of handling SPARQL queries as well, occurs.

4.1 SPARQL Query Optimization in ARQ

There is quite a few SPARQL query optimization study either about rewriting stage or about planning stage. However, due to scope of this study, this study is focued to reordering triple patterns at SPARQL queries which corresponds to join ordering process at SQL queries. Reordering triple patterns is a significant part of SPARQL query optimization. The purpose of reordering triple patterns is to find fastest (optimum) query execution plan, i.e., the plan that returns the result set with minimum execution time compared to other execution plans.

Determining the order of triple patterns is a key factor in optimizing joins (Maduko & others, 2007), consequently it is a key factor for decreasing execution time of queries. In SPARQL, order of triple patterns may substantially effects execution time. For example, the 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 in Table 4.1. Executing the query takes 762 ms. If the triple patterns is reordered as in Table 4.2, the query execution takes 163 ms (nearly 4.5 times faster).

Table 4.1 Triple pattern order of basic graph pattern before optimization

	Triple Pattern
1	?border o:importsCommodity ?impCommodity.
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.

Table 4.2 Triple pattern order of basic graph pattern after optimization

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

Another example BGP is analyzed in order to explain the importance of triple patterns order. This BGP is a part of chain query and it is shown in Table 4.3. Graph of this chain query is depicted at Figure 4.2.

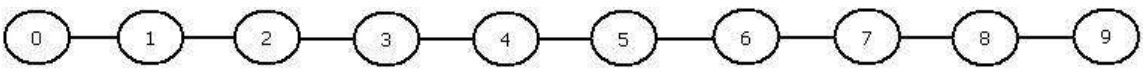

Figure 4.2 Graph of example chain query

Table 4.3 Basic graph pattern of example chain query

	Triple Pattern
0	?faculty rdfs:subClassOf ub:Faculty.
1	?fac rdfs:subClassOf ?faculty.
2	?teacher rdf:type ?fac.
3	?teacher ub:worksFor ?dept.
4	?student ub:memberOf ?dept.
5	?student ub:takesCourse ?course.
6	?tAsst ub:teachingAssistantOf ?course.
7	?tAsst ub:advisor ?adv.
8	?adv ub:doctoralDegreeFrom ?unv.
9	?unv ub:name ?uName.

Execution of the example chain query with the order which is shown in Table 4.3, takes 38.2 sn; but executing the optimized query (Table 4.4) which is obtained by the steps that is shown at Figure 4.3 takes only 0.7 sn.

Table 4.3 Optimized basic graph pattern of example chain query

	Triple Pattern
6	?tAsst ub:teachingAssistantOf ?course.
7	?tAsst ub:advisor ?adv.
5	?student ub:takesCourse ?course.
8	?adv ub:doctoralDegreeFrom ?unv.
9	?unv ub:name ?uName.
4	?student ub:memberOf ?dept.
3	?teacher ub:worksFor ?dept.
2	?teacher rdf:type ?fac.
1	?fac rdfs:subClassOf ?faculty.
0	?faculty rdfs:subClassOf ub:Faculty.

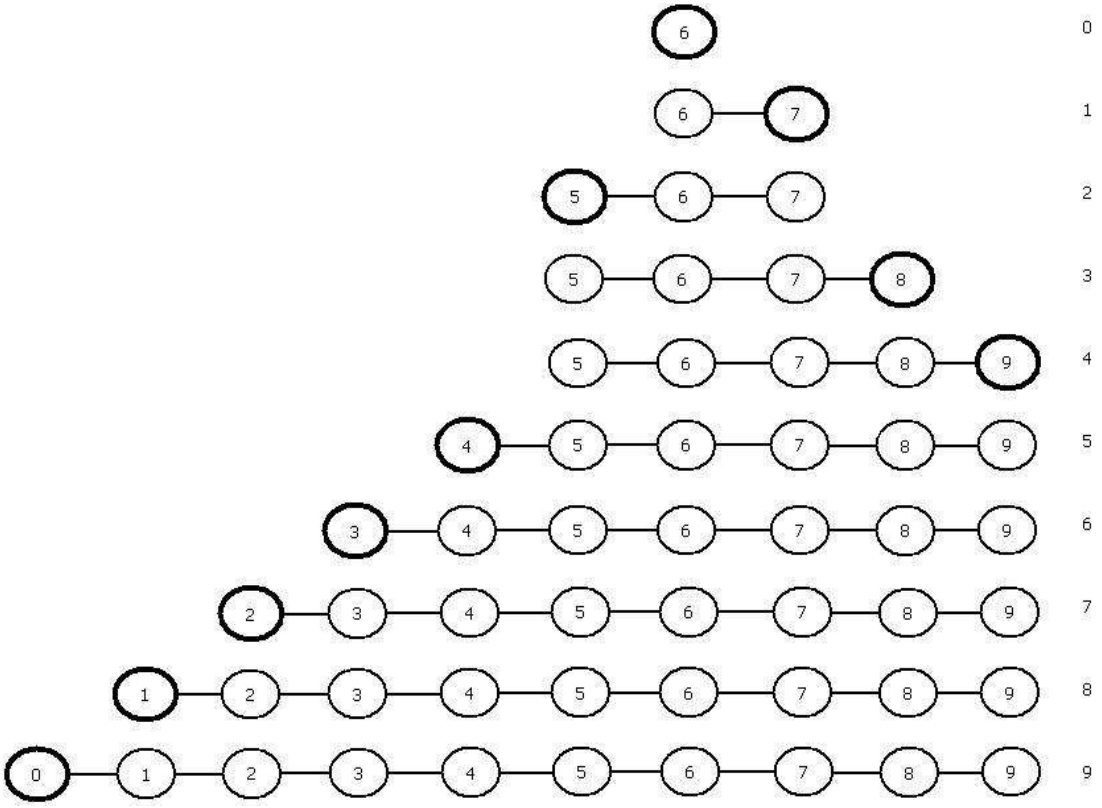

Figure 4.3 Preparation steps of optimal order of example chain query

Hartig and Heese (2007), point out to a posting (Wolf, 2006) in a newsgroup which mentions long execution time of a simple SPARQL query. Solution (Dollin, 2006) that offered to this problem was that "user should put the more specific part of the query first; it makes a significant difference" (Hartig & Heese, 2007).

This case shows that in ARQ, unoptimized order of triple patterns may increase execution time. Although Stocker & others (2008) deals with this problem, it is still open for new solution proposals.

4.2 Selectivity of Triple Patterns

Selectivity of triple patterns is the key factor for reordering triple patterns. As mentioned above, reducing intermediate result sets that return from join operations,

provides lower execution time for query execution plan. So estimating selectivities of triple patterns provides advantage to obtain better triple patterns order. Selectivity of each triple pattern is used for estimating size of intermediate results and intermediate results are used to calculate cost of query execution plan.

Definition of selectivity as general is; selectivity of the condition X is the fraction of tuples satisfying this condition X (Piatetsky-Shapiro & Connell, 1984). It is possible adapting the definition for selectivity of triple patterns as, selectivity of the triple pattern TP corresponds to fraction of triples satisfying the triple pattern TP (Bernstein, Kiefer & Stocker, 2007). We can formulate it as

$$sel(TP) = \frac{|TP|}{|R|} \qquad (4.1)$$

where $|TP|$ is number of triples satisfying TP and $|R|$ is total number of resources in queried ontology. There are wide variety selectivity estimation techniques from simple mathematical formulas to complex statistical techniques and heuristic approaches. Performance of techniques is measured with accuracy of estimation.

CHAPTER FIVE
RELATED WORK

The pioneer and seminal study about query optimization was proposed by Selinger & others at 1979. The proposed algorithm for that problem was named as dynamic programming and examined in the context of System R (Astrahan & others, 1976) experimental relational database management system. The algorithm roughly based on searching entire solution space and dynamically pruning the suboptimal query processing trees.

Dynamic programming algorithm has been used in various areas; inspite of its accuracy, the problems like long execution time and high memory allocation requirement occurred. The problems like above that is needed to overcome, guided the researchers to improve the algorithm and to apply another deterministic, randomized, evolutionary and other various algorithms to the problem. Nowadays state of the art algorithms are applied to various type databases like relational, object-oriented, XML, RDF, etc.

There is an extensive research in the literature for solution of join ordering problem at relational databases (Ioannidis, 1996; Chaudhuri, 1998; Neumann, 2008), object-oriented databases (Mitchell, 1991; Özsu, 1995), and XML databases (Che, 2006; Kader, 2007). Various types of probabilistic and deterministic algorithms has been proposed so far for the solution of the problem. However, some characteristics of RDF and SPARQL, e.g., necessity of additional triple patterns for querying attribute - like properties of entities, differ SPARQL from SQL (Neumann & Weikum, 2008). Earlier works may not entirely solve the problem for the SPARQL, hence there is still a need for new approaches specific to SPARQL query optimization.

Due to domain and scope of this study; the studies which deal with join ordering problem in relational and distributed databases using ant colony optimization and also

which deal with reordering triple patterns in RDF databases using any other algorithms, is considered. Related work which discussed below is classified according to that perspective.

5.1 Join Ordering in Relational Databases using Ant Colony Optimization

Li & others (2008) proposed an ant colony optimization based algorithm to solve join ordering problem at relational database management systems. They constructed a constrained graph without Cartesian product for traversing of ants. The nodes of graph represent the relations that will be joined and edges of graph, represents the estimated cardinality of intermediate result returned from join process of connected nodes. Furthermore the cost of query execution plan is consist summation of edge weights. They use very simple technique for estimation of intermediate result cardinality, and this estimation technique causes to lack of accuracy. Proportion of Cartesian product of cardinalities to maximum number of distinct values for join attribute between joined relations, establishes the cardinality estimation for intermediate results.

Dökeroğlu & Coşar (2011) combined Dynamic Programming algorithm and Ant Colony Optimization meta-heuristic with the aim of solving join ordering problem at distributed databases that viable for multi-way join queries. This DP-ACO algorithm provides nearly optimal solutions in polynomial time up to four relation small queries and generates reasonable solutions in polynomial time for between 5 - 15 relations.

5.2 Reordering Triple Patterns in RDF Databases

The drawback about querying distributed RDF repositories with existing Semantic Web systems, inspired the authors Stuckenschmidt & others (2005) extending the Sesame system to fill this gap. The authors presented an index structure with a query processing algorithm and also they applied 2PO (SA following II) hybrid algorithm for reordering RDF chain query paths.

Shironoshita & others (2007) proposed an effective, accurate algorithm for cardinality estimation of queries on ontology models of data. The proposed algorithm relies on the decomposition of queries into query pattern paths. Each pattern produces a set of values for each variable within the result form of the query. For each path, a set of statistics is compiled on the properties of the ontology. They claim that the algorithm they proposed is an important component for the construction of an efficient querying engine over ontology-modelled, distributed, heterogeneous data sources. Their experimental analysis has shown that the algorithm produces estimates with high accuracy and with high correlation to actual values.

Hartig & Heese (2007) proposed a SPARQL query graph model named SQGM which supports all phases of query processing. On top of SQGM they also defined transformations rules to simplify and to rewrite a query. Finally based on these rules they developed heuristics to achieve an efficient query execution plan. Their experimental work has demonstrated of their approach to reduce query execution time.

Maduko & others (2007) proposed a complex pattern based summarization framework for cardinality estimation. They present real world and synthetic datasets experiments to confirm the feasibility of their approach. This work has been classified as a sophisticated method by Neumann & Weikum (2008) because it is building statistics over a selected of arbitrarily shaped graph patterns. This proposed method cast the selection of patterns into an optimization problem and uses greedy heuristics (Neumann & Weikum, 2008).

Stocker & others (2008) proposed static optimization techniques for reordering triple patterns of Basic Graph Pattern. They defined several heuristics for selectivity estimation that uses and not uses pre-computed statistics. Also they used random sampling due to summarizing RDF data and utilized from that summarized data to generate some statistical synopses. Their algorithm constructs directed acyclic graphs

with consideration of unbound components of triple patterns. Experimental studies showed optimized query plans with heuristics that uses pre-computed statistics have less normalized average distance to optimum query plan.

Neumann & Weikum (2008) presented a RISC style query engine for querying RDF data with SPARQL query language. They used dynamic programming algorithm for reordering triple patterns. Also they developed a cost model that utilizes from statistical synopses and explores frequent join paths for selectivity estimation. Through frequent join paths, more accurate estimations can come up.

Ruckhaus & others (2008) proposed a hybrid cost model for estimation and used dynamic programming algorithm to optimize queries of ontologies formalized as deductive ontology base (DOB). They utilized from adaptive sampling technique for estimation cardinality of intermediate results.

Hogenboom & others (2009) proposed Genetic Algorithm based RCQ-GA (RDF Chain Query Optimization using a Genetic Algorithm) algorithm to optimize RDF chain queries. They benchmarked their RCQ-GA algorithm with 2PO. They consider nested-loop join and bushy trees and also apply rank-based selection for dealing with crowding problem at GA. Instead of to estimate cardinality or selectivity of joined triple patterns, the authors preferred to initialize the estimations as Cartesian product. Then these estimations are updated when the query has been evaluated. They experimented queries that consist up to 20 predicates. The experiment results showed that 2PO faster than RCQ - GA for up to 10 - way joins, on the other hand RCQ-GA is faster for 10 to 20-way joins. Also RCQ-GA generates closer solutions to optimum solution than 2PO.

In order to meet scalability requirements about querying billions of RDF triples, Neumann and Weikum (2009) improved their previous work (Neumann & Weikum, 2008). They proposed more accurate selectivity estimation technique for join-ordering. Difficulties of sampling statistics that used for selectivity estimation, data caching

requirements and high memory allocation forced the authors to leave using aggregated statistics. They choosed calculating exact cardinalities for triple patterns at query compile-time.

CHAPTER SIX
PROPOSED APPROACH

The Basic Graph Pattern - set of triple patterns – has been abstracted as a complete graph. For example, BGP in Table 4.1 is abstracted as a complete graph in Figure 6.1. Each node represents a triple pattern and each edge represents estimated join cost of connected nodes. This complete graph is input of ACO algorithms which try to find optimum triple pattern order.

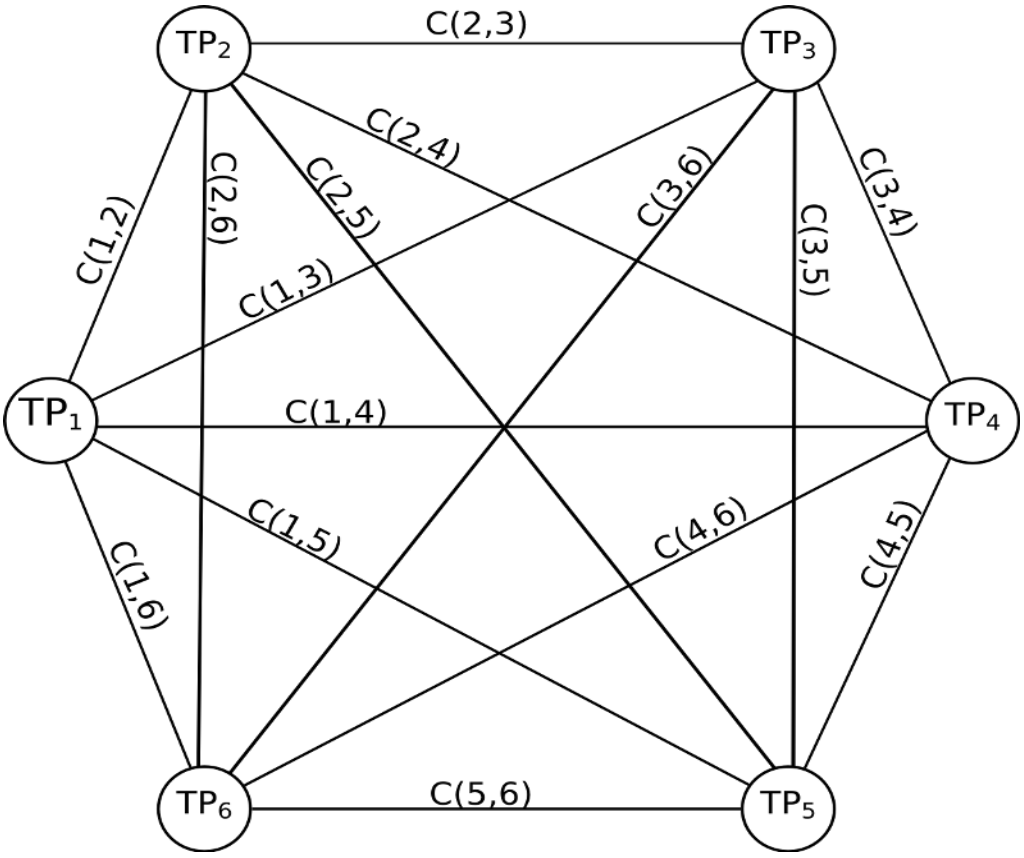

Figure 6.1 Complete graph of BGP in Table 4.1

Proposed AS approach requires cost for every join between triple patterns to reorder the Basic Graph Pattern based on these costs; and join cost of triple patterns is based on

selectivity of these triple patterns. Calculation of costs (weights) of complete graph consists of two steps:

1. Selectivity estimation of triple patterns.
2. Cost calculation of join process.

These steps are explained detailed at following sections.

6.1 Selectivity Estimation of Triple Patterns

Two techniques for selectivity estimation of triple patterns are used in this study: Variable Counting and Graph Statistics Handler (GSH). These techniques are explained below.

6.1.1 Variable Counting for Selectivity Estimation

Variable Counting (VC) heuristic was proposed by Stocker & others (2008). Estimating selectivity of triple patterns with VC is based on ranking components of triple patterns based on ***sel(Subject) < sel(Object) < sel(Predicate)*** ordering and classifying them with number of bound components (concrete thing) and number of unbound components (variable). In other words; this ranking formula means subject components are the most selective, object components are less selective than subjects and more selective than predicates, predicate components are the least selective ones. Selectivity of each unbound component assumed as 1.

For example 1_{st} and 2_{nd} triple patterns in Table 4.1 are analyzed in order to explain clearly. The 1_{st} triple pattern (c:TU o:border ?tuBorder.) includes bound (concrete) subject (c:TU) and predicate (o:border), and unbound (variable) object (?tuBorder). On the other hand, the 2_{nd} triple pattern (?tuBorder o:country ?border.) includes bound predicate, unbound subject and unbound object. So, having bound subject (the most

selective component) and two bound components makes the 1_{st} triple pattern more selective than the other one. However deciding which one is the most selective (object component or subject component), is a quite bit difficult and depends on the RDF data which is queried. This case emerges as a problem that needs to be resolved.

6.1.2 Graph Statistics Handler (GSH)

Jena provides exact size information about triple pattern components for in-memory graph models by GSH. GSH makes the most accurate estimations, but it does not support estimation for triple patterns that has more than one bound component (Stocker & others, 2008).

For 2_{nd} triple pattern (?tuBorder o:country ?border.) in Table 4.1, because it has only one bound component (predicate), GSH can provide accurate size information. On the contrary, because it has more than one bound component (subject and object), GSH can not provide accurate size information for the 1_{st} triple pattern in Table 4.1.

6.2 Cost Calculation of Join Process

Different cost calculation methods are performed to find weights of complete graph as a matrix that is given to AS to find optimized triple pattern order. Two different weight finding (cost calculation) approaches have been implemented and experimented for proposed method. These approaches are explained below.

6.2.1 Simple Cost - Cartesian Product of Cardinalities

Simple Cost is just summation of Cartesian product of triple pattern cardinalities. Each edge in complete graph is weighted with Cartesian product of triple patterns – nodes connected to that edge - cardinalities. For example we assume that estimated cardinalities of triple patterns TP_i and TP_j are $|TP_i|$ and $|TP_j|$ respectively. Thus the

estimated cardinality of result set that returns from join operation of these triple patterns is calculated as $C(i,j) = |TP_i| \times |TP_j|$. Cardinality of triple pattern is calculated using GetAccurateCardinality method which pseudo code is listed below:

```
GetAccurateCardinality(TriplePattern, OntologySize)
flag ← -1
selectivity ← 1
cardinality ← get TriplePattern cardinality from GSH
If cardinality = -1 // TriplePattern includes more than one bound component
    If subject of TriplePattern is concrete
        cardinality ← get subject cardinality from GSH
        selectivity ← selectivity * (cardinality / OntologySize)
        flag ← 0
    If predicate of TriplePattern is concrete
        cardinality ← get predicate cardinality from GSH
        selectivity ← selectivity * (cardinality / OntologySize)
        flag ← 0
    If object of TriplePattern is concrete
        cardinality ← get object cardinality from GSH
        selectivity ← selectivity * (cardinality / OntologySize)
        flag ← 0
    If flag = -1
        cardinality ← OntologySize
    Else
        If selectivity < (1 / OntologySize)
            cardinality ← 1
        Else
            cardinality ← selectivity * OntologySize
Return cardinality
```

6.2.2 Variable Counting for Cost Calculation

VC is based on ranking join types, e.g., Subject-Subject, Subject-Object joins. Join of bound components is more selective than join of unbound components and unusual joins like Subject-Predicate joins are most selective than the others (Stocker & others, 2008).

For the VC *getCost* method pseudo code for two triple patterns is listed below. Value that this method returns added to the cost matrix.

```
getCost(TriplePattern1, TriplePattern2)
    p1 ← 0
    p2 ← 0
    if  GSH is not available
        p1 = GetCost(TriplePattern1)
        p2 = GetCost(TriplePattern2)
    else
        p1 = GSH.getCost(TriplePattern1)
        p2 = GSH.getCost(TriplePattern2)
    if p1 < p2
        Return p1 + getTripleCosts(TriplePattern1, TriplePattern2)
    Return p2 + getTripleCosts(TriplePattern1, TriplePattern2)
```

Flowchart for calculating cost based on variable counting method (*getTripleCosts*) can be seen in Figure 6.2. *Joined* method checks if given triple pattern parameters contains any join between them. *GetJoins* method returns the possible joins between triple patterns that are given as parameters. *SpecificJoinCost* method returns the values for different join types. These values are shown in Table 6.1. *GetCost* method gives the estimated cardinality of triple pattern by ranking components of triple pattern. Pseudo code of GetCost method is listed below.

GetCost(TriplePattern)

cost ← 1

maxCost ← 32

If subject of *TriplePattern* is variable

cost ← *cost* + 4

If predicate of *TriplePattern* is variable

cost ← cost +1

If object of *TriplePatterns* is variable

cost ← *cost* + 2

Return *cost* / *maxCost*

Table 6.1 Specific join costs of various join types for Variable Counting

Join Type	Unbound Join Cost Value	Bound Join Cost Value
Subject – Subject	2	4
Subject – Predicate	3	6
Subject – Object	1	2
Predicate – Subject	3	6
Predicate – Predicate	3	-
Predicate – Object	3	6
Object – Subject	1	2
Object – Predicate	3	6
Object – Object	1	2

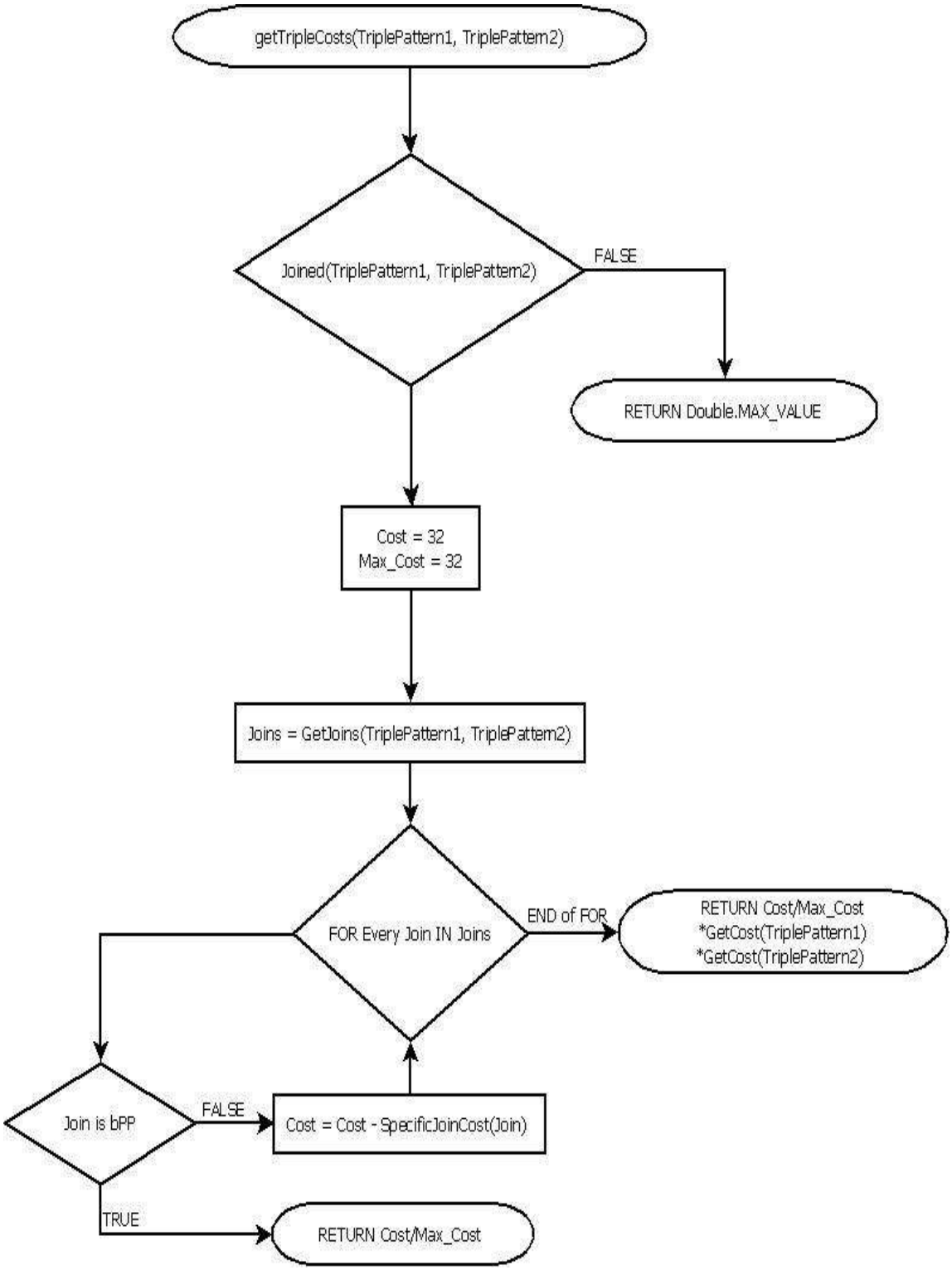

Figure 6.2 Flowchart of Variable Counting method

6.2.3 Modified Variable Counting 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. The action which is implemented when bound predicate – predicate join occurs makes difference between Variable Counting and Modified Variable Counting methods in *getTripleCosts* methods. Instead of to return Cost / Max_Cost, in Modified Variable Counting method bound predicate – predicate join is ranked with 0. Another difference between two methods is the return value at *false* case of first condition. In Modified Variable Counting method, returning multiplication of triple pattern selectivities increases the possibility of Cartesian product selection in small quantities. So it makes the Cartesian product possible – even if it is small possibility- in convenient cases. Flowchart for *getTripleCosts* method is listed in figure 6.3.

Table 6.2 Specific join costs of various join types for Modified Varible Counting

Join Type	**Unbound Join Cost Value**	**Bound Join Cost Value**
Subject – Subject	2	4
Subject – Predicate	3	6
Subject – Object	1	2
Predicate – Subject	3	6
Predicate – Predicate	3	**0**
Predicate – Object	3	6
Object – Subject	**2**	**4**
Object – Predicate	3	6
Object – Object	1	2

The *getCost* method (that method is used for populating cost matrix) pseudo code for modified VC is listed below:

GetCost(TriplePattern1, TriplePattern2)

p1 ← getAccurateCardinality(TriplePattern1) / OntologySize

p2 ← getAccurateCardinality(TriplePattern2) / OntologySize

if *p1* < *p2*

Return *p1* + getTripleCosts(TriplePattern1, TriplePattern2)

Return *p2* + getTripleCosts(TriplePattern1, TriplePattern2)

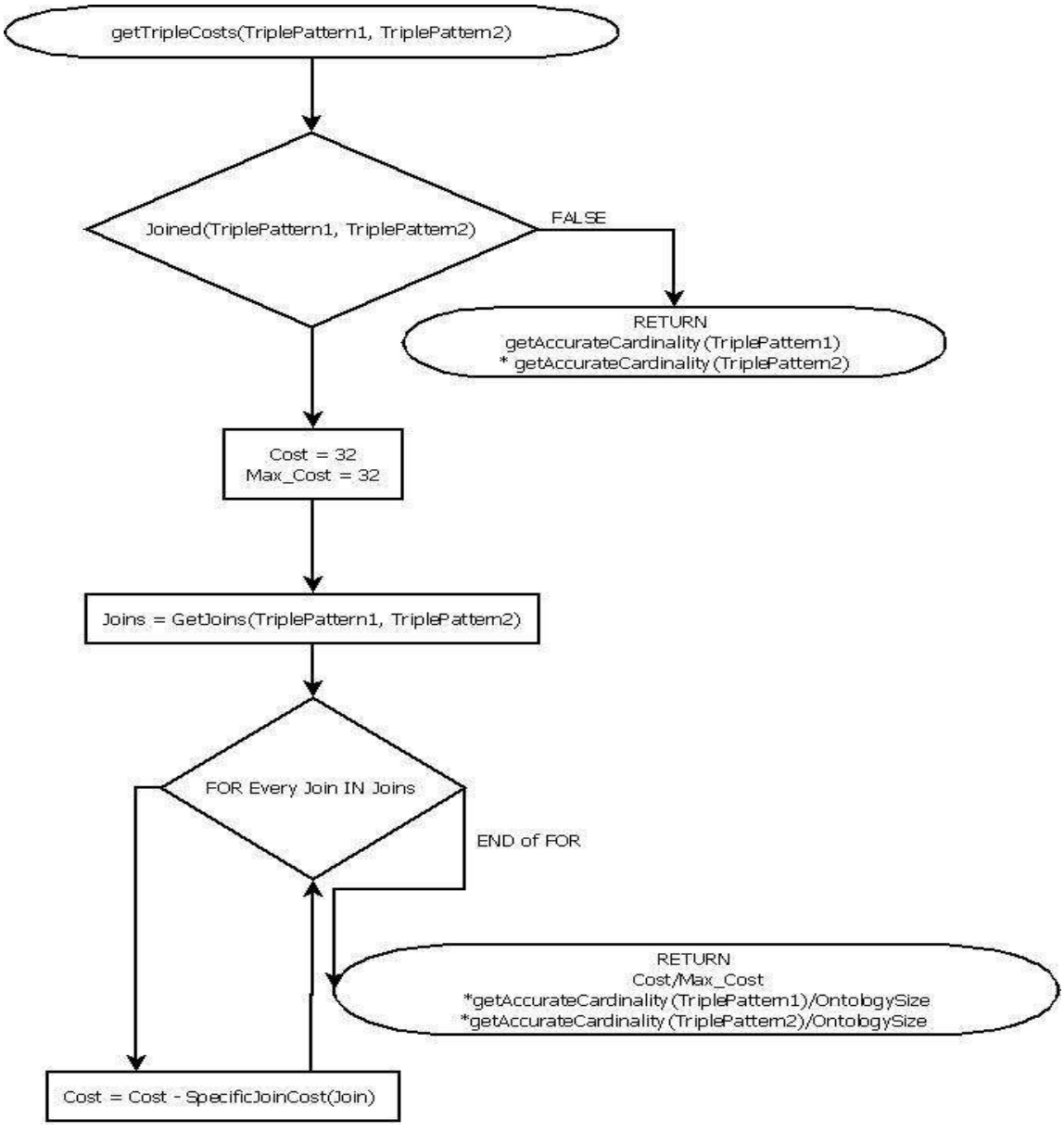

Figure 6.3 Flowchart of Modified Variable Counting method

CHAPTER SEVEN
IMPLEMENTATION

In subsection 7.1 combination of selectivity estimation and cost calculation techniques are explained and in subsection 7.2 implementation of ACO Meta-heuristic algorithms are explained.

7.1 Selectivity Estimation and Cost Calculation

To be able to calculate costs for edges, techniques that discussed in previous chapter are combined. The first part which is pointed at subsection title indicates the algorithm for selectivity estimation and the second part indicates the algorithm for cost calculation.These combinations are explained in the following subsections.

7.1.1 GSH and Simple Cost

In this combination *GetAccurateCardinality* function is used for selectivity estimation and simple cost calculation is used for edge cost calculation. If triple pattern has more than one bound component, selectivity of each bound component is calculated with GSH and product of these selectivities is returned as selectivity of triple pattern. The first triple pattern TP_1 (c:TU o:border ?tuBorder) in Table 4.1 has two bound component. To be able 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 shown in formula 7.1.

$$sel(TP_1) = sel(S_1) * sel(P_1) \qquad (7.1)$$

After estimating cardinality, estimated join cost is available as Cartesian product of estimated cardinalities. For this cost calculation worst case is the Cartesian product of joined triple patterns.

7.1.2 GSH and VC

In this combination 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. This technique is used by Stocker & others (2008) at their work. For more information about their implementation, please refer to the corresponding study of Stocker & others (2008).

7.1.3 GSH and Modified VC

In this combination, cardinality of triple patterns is estimated by using GSH technique as discussed with pseudo code *getAccurateCardinality* method. Afterwards Modified VC is applied for cost calculation, to find the weights of the edges.

7.2 ACO Implementations

In this section, implementations of ACO Meta-heuristic algorithms are explained.

7.2.1 Ant System Implementation

After estimating the selectivity of triple patterns and calculating costs, the cost matrix is composed from obtained values. This cost matrix is fed on ant colony optimization. In this work, ant-cycle version of the Ant System implementation has been used for solving the problem. In ant-cycle AS, the pheromone update was only done after all the ants had constructed their tours and the amount of pheromone deposited by each ant was set to be

a function of the tour quality (Dorigo & Stützle, 2004). Virtual ants that are used in the implementation collect visited nodes in a tabu list.

Here are the steps for algorithm proposed:

- set parameters and initialize ants
- iterate until reaching tour count
  - iterate until tabu list is full
    - calculate probability of nodes
    - move the ant to the most possible node
  - calculate tour length for every ant and find best solution length
  - evaporate
  - calculate ant travel cost, deposit pheromone and reset ant
- return best solution path

At the start ants are put on randomly chosen nodes. Ants decide for the next node using transition formula. There are some minor changes in formulas according to generally used formulas (Dorigo & Stützle, 2004). In transition formula (eq. 7.2) ρ_{ij}^{k} is the probability of choosing node j for ant k which is currently at node i and η_{ij} is a special heuristic value which is obtained from the calculated cost ($\eta_{ij} = \frac{1}{C(i,j)}$) that are derived by the techniques explained above. N_i^k is the feasible neighbourhood of ant k when being at node i, that is, the set of nodes that ant k has not visited yet (the probability of choosing a node outside N_i^k is 0) (Dorigo & Stützle, 2004).

τ_{ij} is the pheromone trail of edge from node i to node j. Before algorithm runs (at the initialization) the minimum pheromone value (which is a parameter taken from user) is deposited on all of the edges (arcs) ($\tau(0)_{ij} = \min Ph$). During the algorithm run, the

pheromone trails of all edges are updated after every ant have constructed its tour (local update) and 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 (if ρ is convergent to 1 then only small amounts of pheromone are evaporated between iterations resulting in slower convergence rate, if ρ is convergent to 0 then more pheromone is evaporated resulting in faster convergence).

- Second, every ant deposits pheromone using formula 7.3 to the edges it has visited. $\Delta\tau_{ij}^{k}$ is the amount of the pheromone ant k deposits. Its value is obtained from formula 7.4 (C^k is the total cost of tour T^k built by the k-th ant.).

$$P_{ij}^{k} = \frac{\left[\tau_{ij}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l} \in N_i^k \left[\tau_{il}\right]^{\alpha}\left[\eta_{il}\right]^{\beta}} \qquad if\ j \in N_i^k \tag{7.2}$$

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \qquad \forall (i,j) \in L \tag{7.3}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^k, if\ edge(i,j)\ \ belongs\ to\ \ T^k; \\ 0, otherwise; \end{cases} \tag{7.4}$$

α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information (Dorigo & Stützle, 2004). When β = 0 algorithm chooses edges based on learned behaviour of ants, this learning is influenced from pheromone intensity, without any heuristic bias. For values of α > 1 this leads “to the rapid emergence of a stagnation situation” (Dorigo & Stützle, 2004). On the contrary α =

0 makes algorithm to choose edges with lower cost. This corresponds to “a classic stochastic greedy algorithm” (Dorigo & Stützle, 2004).

At every iteration these formulas are used by the AS algorithm to construct best triple pattern order. As explained above α, β and ρ are important factors for exploration and exploitation of the proposed approach. Trade off between exploration and exploitation is provided by choosing appropriate values for these parameters and this decision is left to the user in the proposed approach. Selecting right parameter values required for better convergence of the results. This selection can be achieved by a preliminary parameter analysis.

7.2.2 Elitist Ant System Implementation

Main difference of Elitist Ant System (EAS) is resides on the pheromone deposition formula. In EAS all AS formulas are same, except pheromone deposition formula (7.3) is changed as follows:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \tau_{ij}^{best} \forall (i, j) \in L \qquad (7.5)$$

In EAS by using this formula, on best tour path T^{bs} additional pheromone is deposited. This best tour is the recent iteration's best tour. Pheromone quantity is same with other ants deposition: $\frac{1}{C^{bs}}$.

7.2.3 MAX-MIN Ant System Implementation

MAX-MIN Ant System (MMAS) is required a bit more work than EAS to be implemented. Three modifications of the four modifications (Dorigo & Stützle, 2004) which are discussed in section 3.3 have been implemented in this study:

1. Pheromone deposition formula changed as follows:

$$\tau_{ij}(t+1) \leftarrow \tau_{ij}(t) + \Delta\tau_{ij}^{best} \qquad (7.6)$$

In our implementation, only the iteration-best ant ($\Delta\tau_{ij}^{best} = 1/C^{ib}$) is allowed to add pheromone. Motivation to select iteration-best over best-so-far is that our problems that are trying to solve can be thought as a similar to TSP with low nodes.

2. At the start of the algorithm, the pheromone trails on every edge is set to τ_{max} value which is a parameter taken from the user.

3. Two new parameters has been introduced to limit the pheromone trail values on the edges as $[\tau_{min}, \tau_{max}]$: minimum pheromone value (τ_{min}) and maximum pheromone value (τ_{max}). Each time a new best-so-far tour is found, the value of τ_{max} is updated as $\tau_{max} = \frac{1}{(\rho C^{ib})}$ (ρ is pheromone evaporation rate and C^{best} is cost of best-so-far tour). τ_{min} is set to τ_{max}/a, where a is a parameter also ($a > 1$).

CHAPTER EIGHT
EXPERIMENTAL WORK

In this section experimental set-up and results of experiments have been presented.

8.1 Methodology

All experiments are conducted using Java programming language with the help of Apache Jena framework (Apache Jena, 2012a).

Configuration of computer that is used for experiments is Intel(R) 4x Core(TM) i5-2400 3.10Ghz CPU, 4 GB RAM, Debian GNU/Linux Operating System with 64 bit Linux 3.2.0 kernel. Java version is 64 bit OpenJDK Runtime Environment (6b24-1.11.1-3).

Parameters for the ACO algorithms are chosen by a preliminary parameter analysis which is performed by running algorithm for a fixed query for different values of parameters. In this analysis for each experiment only one single parameter is varied and others are fixed, hence only effect of this varied parameter has been investigated.

8.2 CIA Factbook Ontology Experiments

All the SPARQL queries that are used in the experiments are written by authors. All these developed queries use target ontologies that are extracted from CIA The World Factbook Web page (CIA, 2012; Hogenboom & others, 2008). There are two ontologies that are queried in the experiments and total triple count of these ontologies is 95812. These ontologies are stored in memory. For different effects of the developed method, queries with different triple pattern counts, has been developed: 4, 6, 8, 10, 12 and 14 triple patterns queries. When developing these queries no tool is used, all of them are written by hand.

Intrinsic characteristics of RDF, reveals the necessity of star-shaped sub queries for querying attribute-like properties of the same entity (Neumann & Weikum, 2008). The test queries have been prepared considering this case and all of them have been constructed as chain or chain-star queries. Star-shaped sub queries may occur at any place of chain-star queries. There is an example chain-star query and its' graph (Figure 8.1) below:

PREFIX *c: <http://www.daml.org/2001/09/countries/fips#>*
PREFIX *o: <http://www.daml.org/2003/09/factbook/factbook-ont#>*
PREFIX *rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>*
SELECT *?bord ?imp ?exp ?ind*
WHERE *{*

1. *c:AE o:border ?border.*
2. *?border o:country ?bord.*
3. *?bord o:industry ?ind.*
4. *?bord o:imports ?imp.*
5. *?bord o:exports ?exp.*
6. *?bord o:budgetExpenditures ?budget.*
7. *?bord o:importsCommodity ?impCommodity.*
8. *?impCommodity o:commodity ?iCommodity.*

}

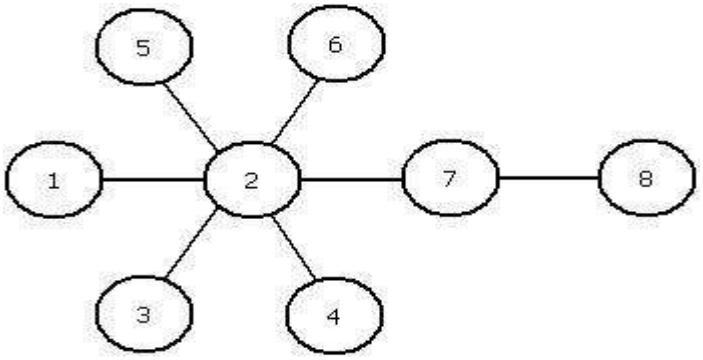

Figure 8.1 Chain-star graph of example query

Each query is run 10 times for every different execution types and average of timings for every execution type are calculated and used for comparison. There are 4 different execution types which are abbreviated in the tables: Normal execution (NE) without any optimization, algorithm proposed by Stocker & others (2008) which uses Variable Counting estimation method (STO-VC) (The algorithm's source code is obtained from ARQ-2.6.0 package which is accessible at http://sourceforge.net/projects/jena/files/ARQ/ARQ-2.6.0/), reordered execution which is an optimization included in Jena (RE) (By using ReorderedWeighted class contained in Jena), Ant System Execution with variable counting (AS-VC) and Ant System execution which uses modified Variable Counting method (AS-VC-M) which are developed by authors of this study.

Table 8.1 Ant system parameters for experiments

Parameter	**Value**
Graph Size	Triple Pattern Count
Start Node	Random
Population Size	50
Max. Pheromone Deposited	25
Min. Pheromone Deposited	0.001
Iteration	100
Alpha (α)	2
Beta (β)	1
Evaporation (ρ)	0.5

Parameters that are used for the Ant System are presented in Table 8.1 and all values in results tables are in terms of **milliseconds**. All values in tables, except values for NE, include optimization, execution and population time. Values for NE include execution and population time.

8.2.1 Results and Discussion

Main problem for queries with less than 4 triple patterns is that executing these queries takes less time than optimizing it. Because optimization time is also added to the execution time, proposed method seems to give bad results for queries such that. Importance of the proposed method can be seen in situations where optimized query can be saved and used later for several times. For that reason this method is experimented and compared with other techniques for queries that contain 4 and more triple patterns up to 14 triple patterns.

When we examine the results for queries with four triple patterns (Table 8.3 and Figure 8.1) we can see that our technique with modified variable counting cost outstanding with smaller run times.

Table 8.3 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**

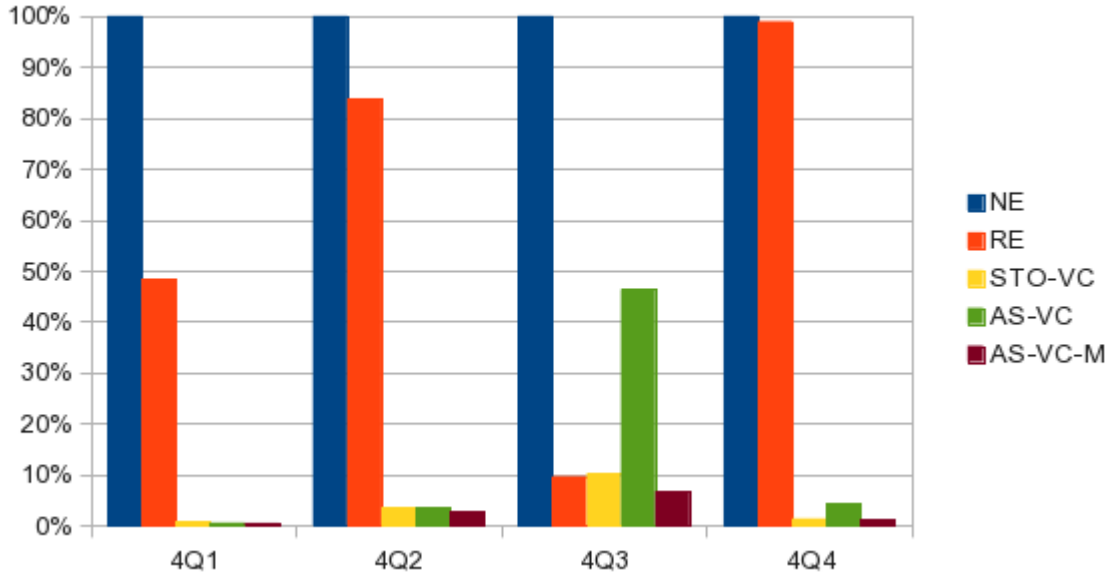

Figure 8.1 Performances on 4 triple pattern queries

For the queries with six triple patterns (Table 8.4 and Figure 8.2) we notice that proposed technique with modified variable counting cost again outstanding with smaller run times, except for the fourth query. Characteristic of fourth query is that it has an optimal order for the triple patterns. Hence, our techniques are unable to optimize it.

Table 8.4 Queries for 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**	179	107	66

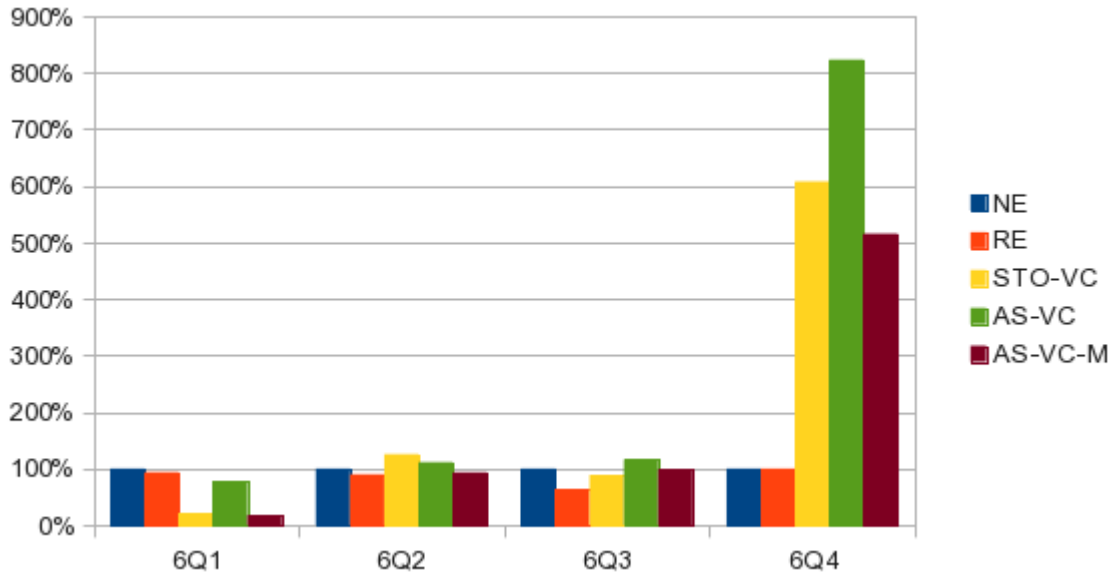

Figure 8.2 Performances on 6 triple pattern queries

For the queries with eight triple patterns (Table 8.5 and Figure 8.3) we notice that proposed technique with modified variable counting cost again outstanding with smaller run times, except for the fourth query, even so it still has better run times in corresponding to normal and ordered executions. It only can't produce better results than Stocker's VC technique.

Table 8.5 Queries for 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

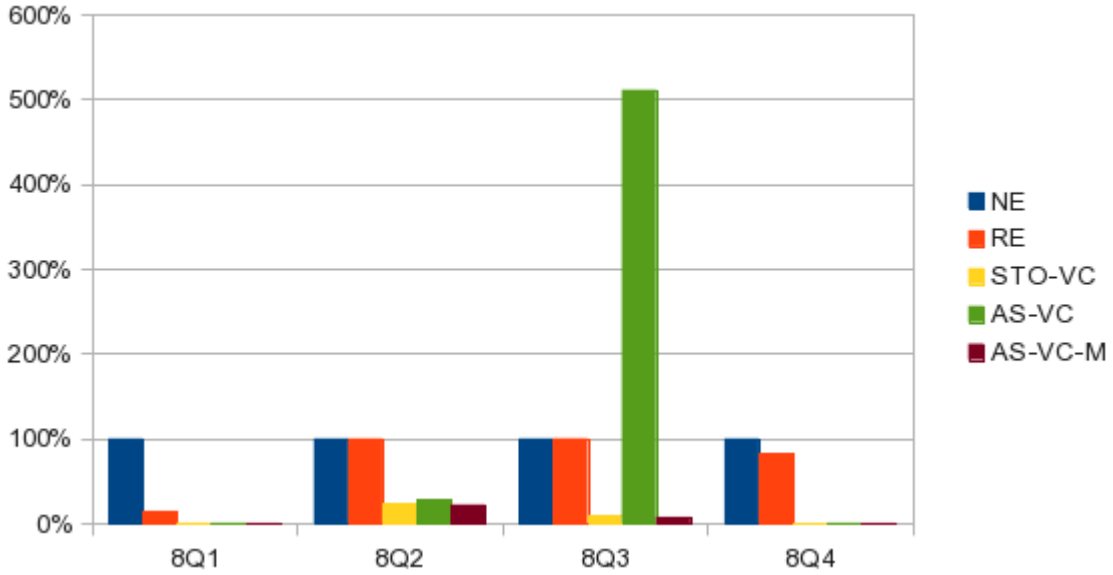

Figure 8.3 Performances on 3 triple pattern queries

For the queries with 10 triple patterns (Table 8.6 and Figure 8.4) we notice that proposed technique with modified variable counting cost again outstanding with smaller run times, except for the fourth query again same like 8 triple patterns fourth query.

Table 8.6 Queries for 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

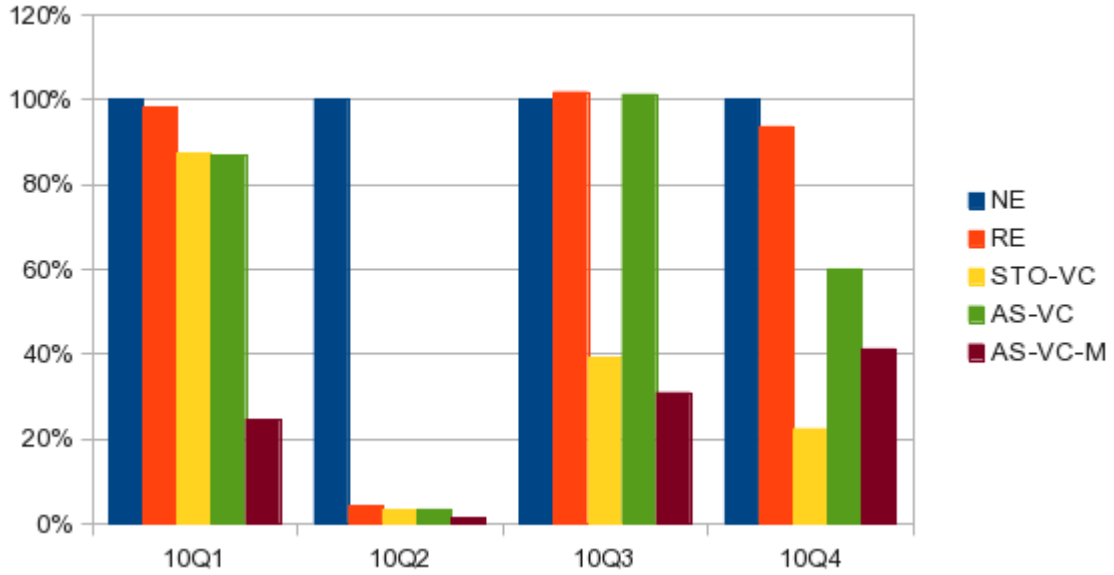

Figure 8.4 Performances on 10 triple pattern queries

For the queries with 12 (Table 8.7 and Figure 8.5) and 14 triple patterns (Table 8.8 and Figure 8.6) we notice that proposed technique with modified variable counting cost outstanding normal and ordered execution time with smaller run times. But Stocker's technique is better for these queries. For these queries main problem was the applicability of the variable counting cost and we are going to try to improve it as being graph independent and applicable for all queries.

Table 8.7 Queries for 12 triple patterns

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

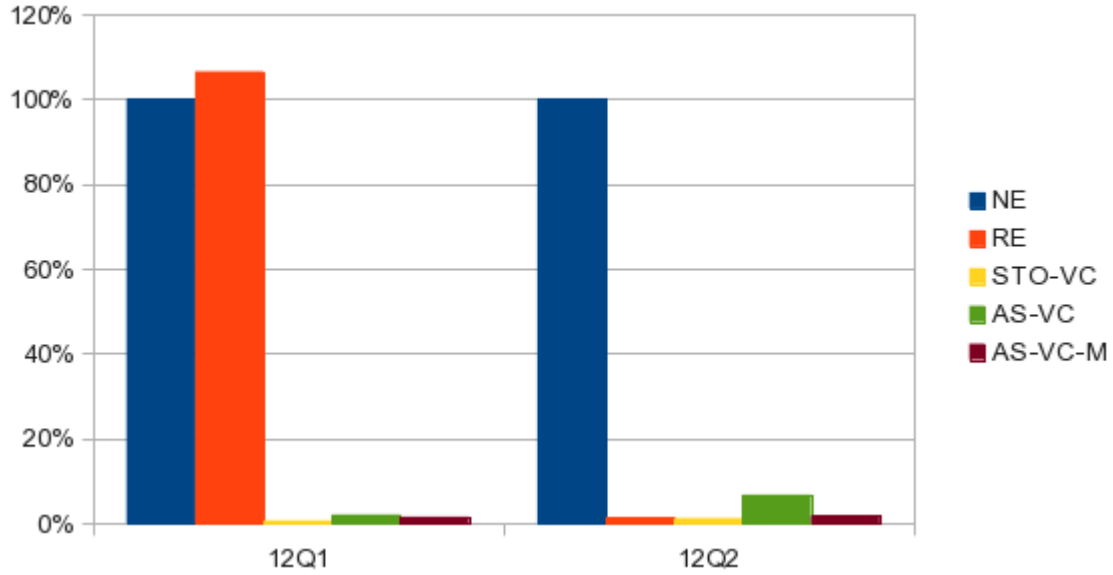

Figure 8.5 Performances on 12 triple pattern queries

Table 8.8 Queries for 14 triple patterns

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

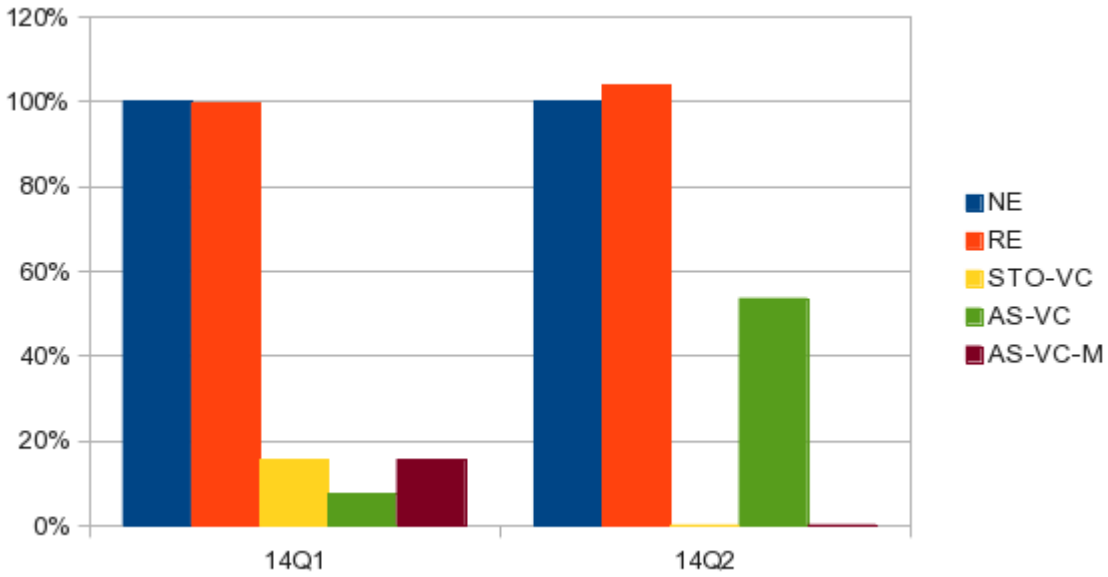

Figure 8.6 Performances on 14 triple pattern queries

The experiment results show that AS-VC-M often produces the best result. Even if it can't produce the best result, it can achieve remarkable enhancement in comparison to Normal Execution.

8.3 LUBM Ontology Experiments

In order to observe scalability of proposed Ant System approach, the experiments are implemented for LUBM (Lehigh University, 2012a) (Lehigh University Benchmark) ontology which contains 162871 triples. Also differently from CIA Factbook ontology experiments; other proposed algorithms (Elitist Ants, Elitist Ant System and MAX-MIN Ant System) are implemented and experimented by querying LUBM ontology with 4, 6, 8, 10, 12 triple pattern queries. Like CIA Factbook queries, LUBM queries are prepared by hand without tools. Additionally certain LUBM test queries (Lehigh University, 2012b) (2, 8, 9) which have more than 5 triple patterns, are also experimented.

LUBM queries are not limited as chain and chain-star queries and prepared as chain, star, cyclic, chain-star, chain-cyclic and cyclic-star graphs. With these different query shapes, optimizing various and complex graph queries has been intended. Each query is run 10 times for each algorithm. There are 7 different execution types:

1. NE – Normal execution without optimization
2. RE – Jena Reordered algorithm
3. STO – Stocker and others (2008) algorithm with Variable Counting
4. AS – Ant System algorithm with Modified Variable Counting
5. EA – Elitist Ants algorithm with Modified Variable Counting
6. EAS – Elitist Ant System algorithm with Modified Variable Counting
7. MMAS – MAX-MIN Ant System algortihm with Modified Variable Counting

All the parameters that are used for the ACO algorithms (AS, EA, EAS, MMAS) are presented in Table 8.2. Values in tables are in terms of **milliseconds** and also all values

in result tables, except values for NE, include optimization, execution and population time. Values for NE include execution and population time.

Table 8.2 Parameters of ACO Meta-heuristic algorithms for experiments

Parameter	Value
Graph Size	Triple Pattern Count
Start Node	Random
Population Size	100
Max. Pheromone Deposited	10
Min. Pheromone Deposited	0.001
Iteration	250
Alpha (α)	1
Beta (β)	2
Evaporation (ρ)	0.5

8.3.1 Results and Discussion

In tables execution times of queries are given in milliseconds. In figures *Normal Execution* time is assumed to be 100% for every query execution in LUBM queries and execution times of other algorithms are proportioned to NE execution time.

Table 8.9 LUBM queries for 4 triple patterns

	NE	RE	STO	AS	EA	EAS	MMAS
Q1	87138	86263	323	310	297	**296**	344
Q2	**40**	41	203	50	51	46	103
Q3	704	509	604	444	442	**428**	466

The execution times in Table 8.9 show that EAS provides the fastest execution plan with 0.34% (Figure 8.7) performance for Q1 that is a 4 triple pattern chain query. It is

also seen execution plans that are optimized by ACO algorithms and STO are convergent to EAS.

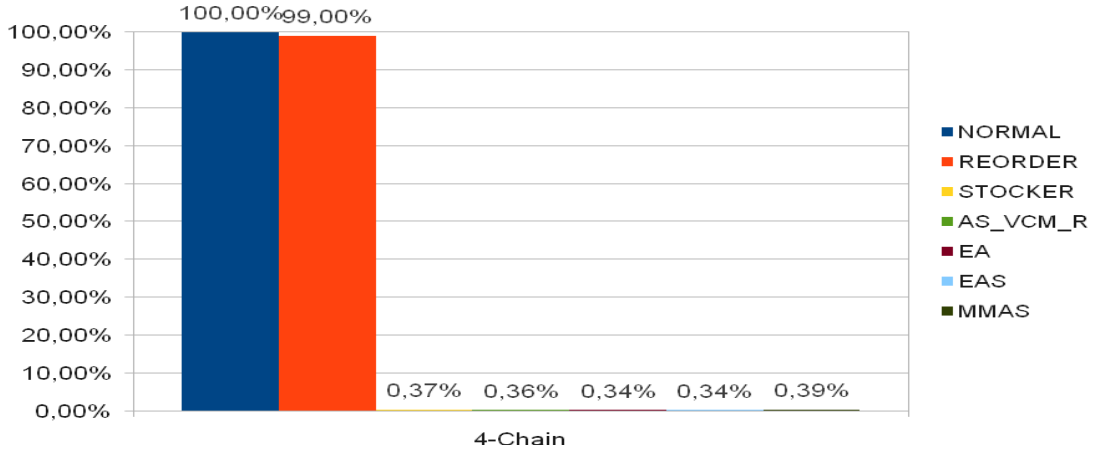

Figure 8.7 Performances for Q1 chain query with 4 triple pattern

The unoptimized order of Q2 (cyclic query with 4 triple patterns) is nearly optimal by default and optimization time for other algorithms causes to exceed normal execution time (Figure 8.8). Even optimization time is added to execution time, ACO algorithms still approximate to NE.

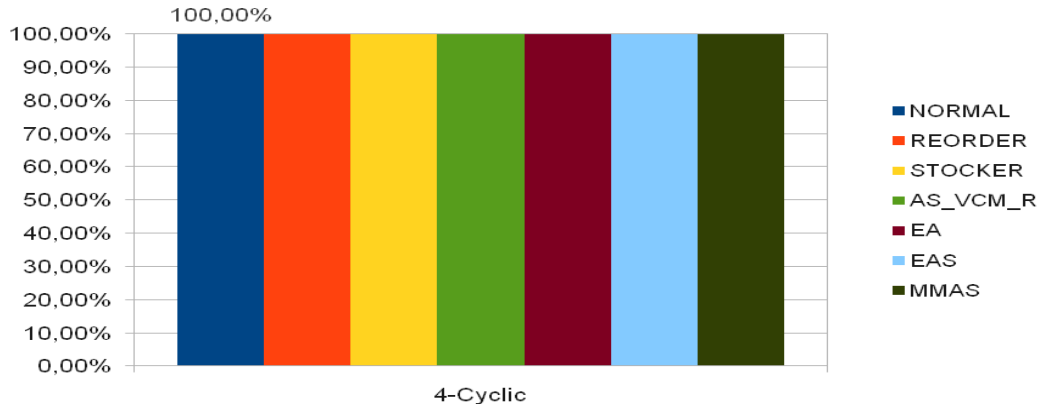

Figure 8.8 Performances for Q2 cyclic query with 4 triple patterns

For Q3 (star query with 4 triple patterns) EAS is the fastest algorithm and other ACO algorithms are close to EAS (Figure 8.9). Execution times of MMAS algorithm for 4 triple pattern queries are bigger than other ACO algorithms simply because of computational load.

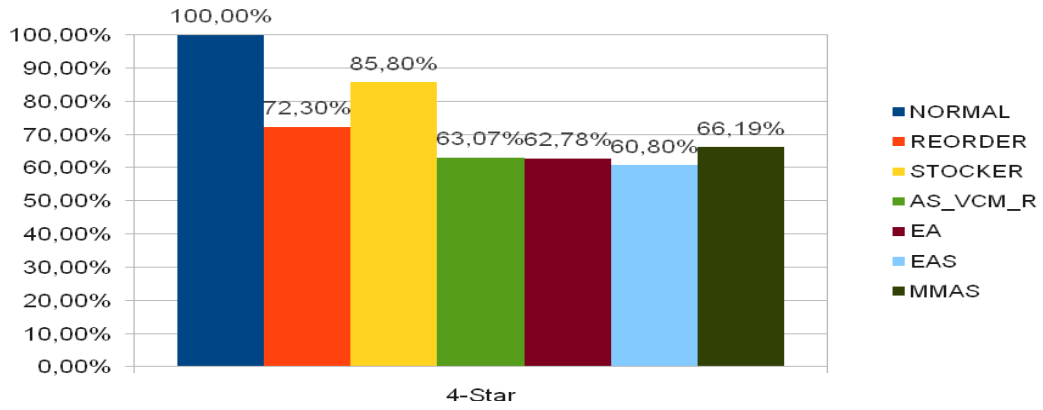

Figure 8.9 Performances for Q3 star query with 4 triple patterns

Table 8.10 LUBM queries for 6 triple patterns

	NE	RE	STO	AS	EA	EAS	MMAS
Q1	31039	31004	342	**239**	242	247	297
Q2	307372	12322	260	88	**87**	**87**	142
Q3	296	**135**	190	176	175	178	233

As indicated in Table 8.10, for Q1 (chain query with 6 triple patterns) ACO algorithms and STO makes substantial decrease at query execution time in comparison to NE and this time AS is the fastest one (Figure 8.10).

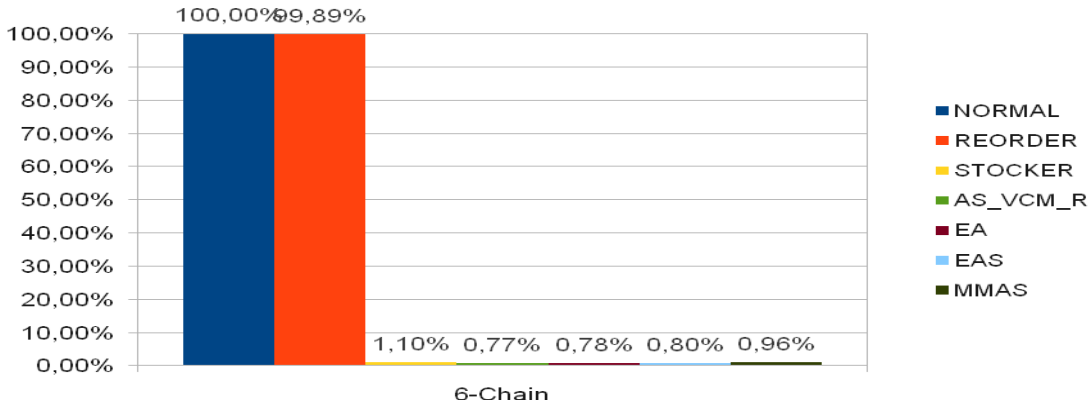

Figure 8.10 Performances for Q1 (chain query with 6 triple patterns)

EA and EAS algorithms provide the fastest execution plans (Figure 8.11) for Q2 (cyclic query with 6 triple patterns).

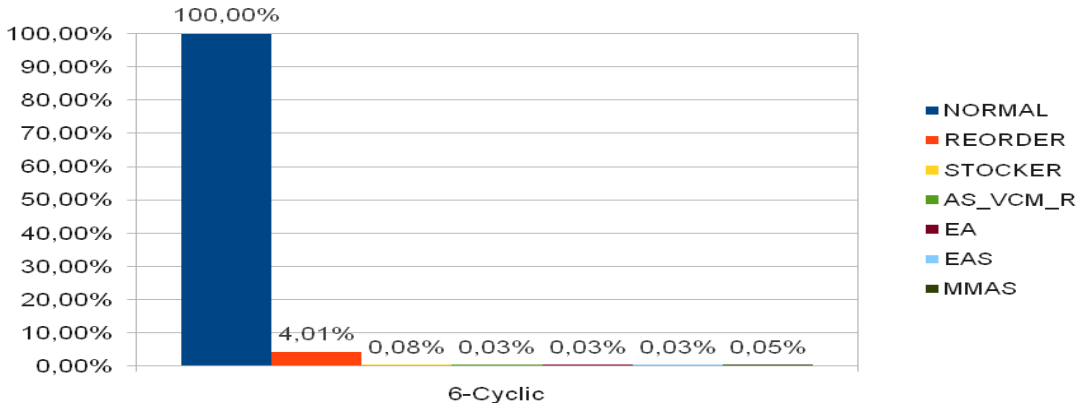

Figure 8.11 Performances for Q2 (cyclic query with 6 triple patterns)

The experiment results show that proposed solutions reduced the query execution time distinctively for the queries with 6 triple patterns except the last one (Q3). Q3 has nearly optimal triple pattern order in the default. So trying to optimize triple pattern order of Q3 causes adding optimization time to the execution time, which is simply the lowest execution time (Figure 8.12).

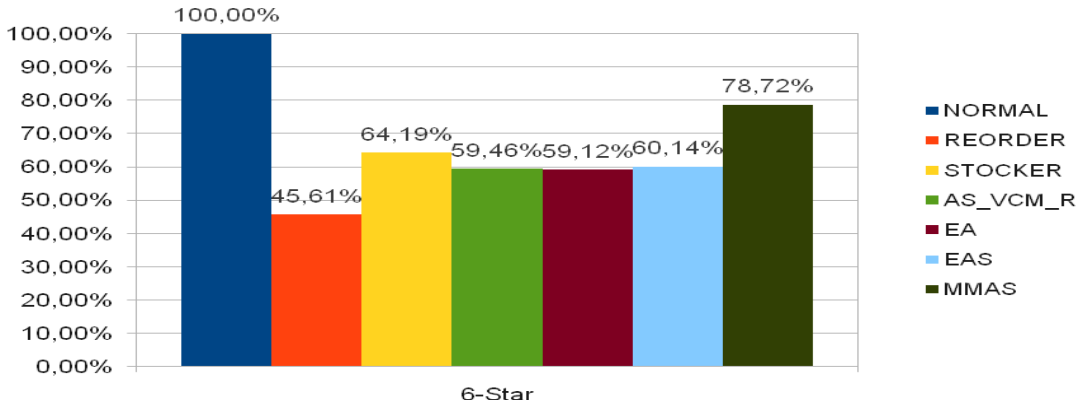

Figure 8.12 Performances for Q2 (star query with 6 triple patterns)

Table 8.11 LUBM queries for 8 triple patterns

	NE	RE	STO	AS	EA	EAS	MMAS
Q1	21857	21856	**5015**	9559	9409	9424	12102
Q2	37690	37712	**337**	1745	1628	1654	1694
Q3	674	**186**	256	346	345	345	394

Even if they can't provide the minimum execution time in comparison to STO, the ACO algorithms reduced the execution time more than by half for Q1 (chain query with 8 triple patterns). Figure 8.13 depicts the execution time percents for different algorithms.

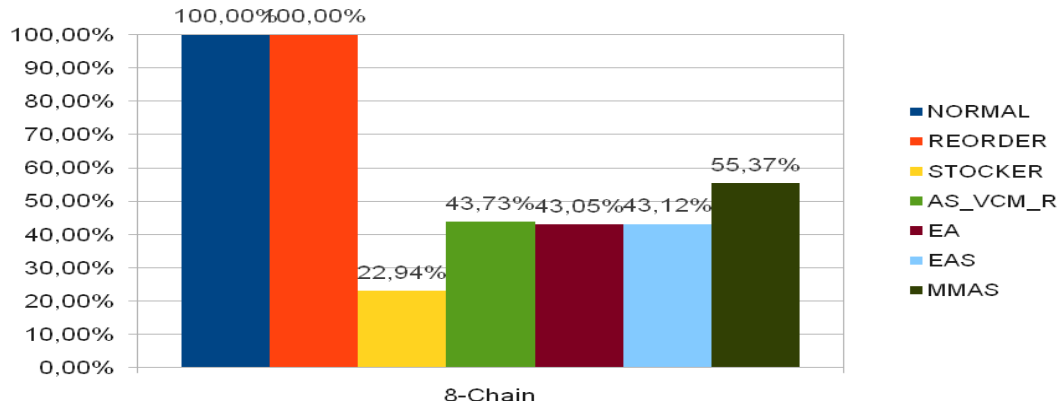

Figure 8.13 Performances for Q1 (chain query with 8 triple patterns)

For Q2 (cyclic query with 8 triple patterns), AS provides the best execution time and all ACO algorithms find nearly optimal orders (Figure 8.14).

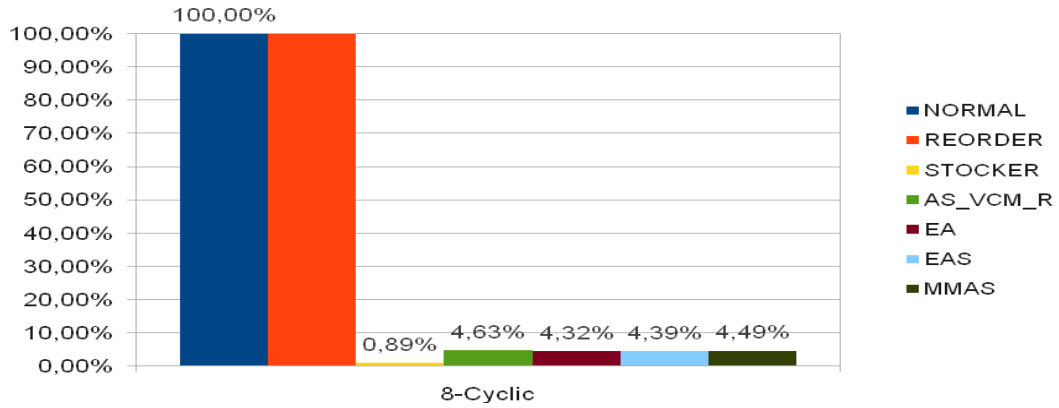

Figure 8.14 Performances for Q2 (cyclic query with 8 triple patterns)

Even if performance of RE is the best for Q3 (Figure 8.15), all ACO algorithms provide nearly optimal orders. Beside ACO algorithms, the algorithm STO which is proposed by Stocker and others (2008) also provides nearly optimal orders for each query with 8 triple patterns.

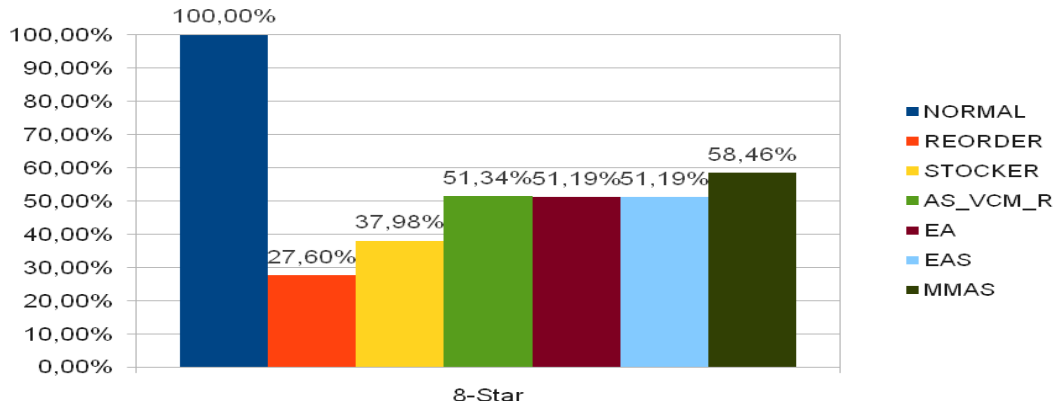

Figure 8.15 Performances for Q3 (star query with 8 triple patterns)

Table 8.12 LUBM qeries for 10 triple patterns

	NE	RE	STO	AS	EA	EAS	MMAS
Q1	14430	14319	392	340	255	**251**	332
Q2	86785	44324	**4603**	6845	6689	6817	6923
Q3	338	**148**	224	293	290	293	344

As presented in Table 8.12, for Q1 (chain query with 10 triple patterns), EAS is the fastest algorihm as often has been so far (Figure 8.16) and other algorithms provide nearly optimal execution plans.

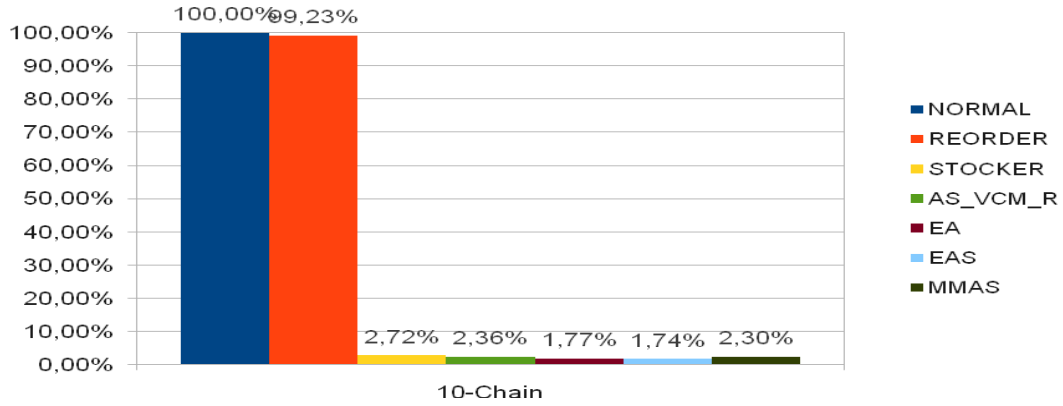

Figure 8.16 Performances for Q1 (chain query with 10 triple patterns)

Although for Q2 (cyclic query with 10 triple patterns) STO is the fastest algorithm; the ACO algorithms provide significant decrease in execution time (Figure 8.17). For example execution time for EA is 7.71% of NE.

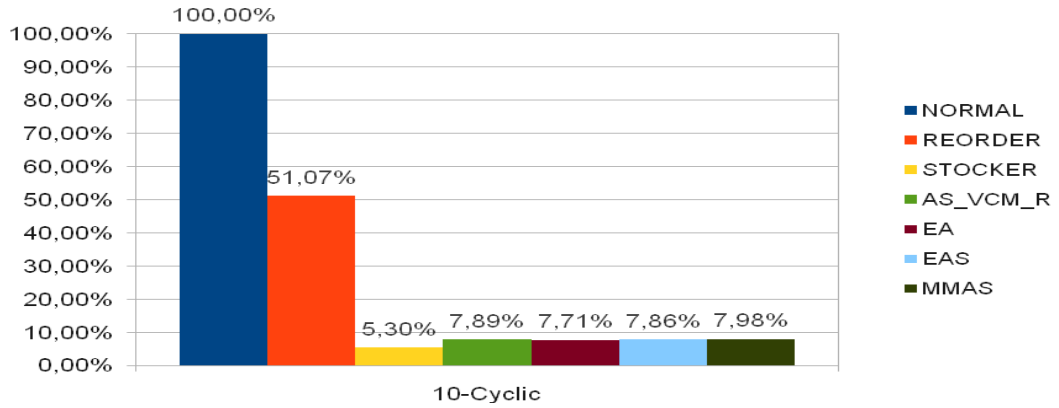

Figure 8.17 Performances for Q2 (cyclic query with 10 triple patterns)

Normal execution of Q3 (star query with 10 triple patterns) only takes 338 ms and the fastest algorithm is RE for Q3, which executes the query in 148 ms. Still, ACO algorithms provide better execution plans than NE (Figure 8.18).

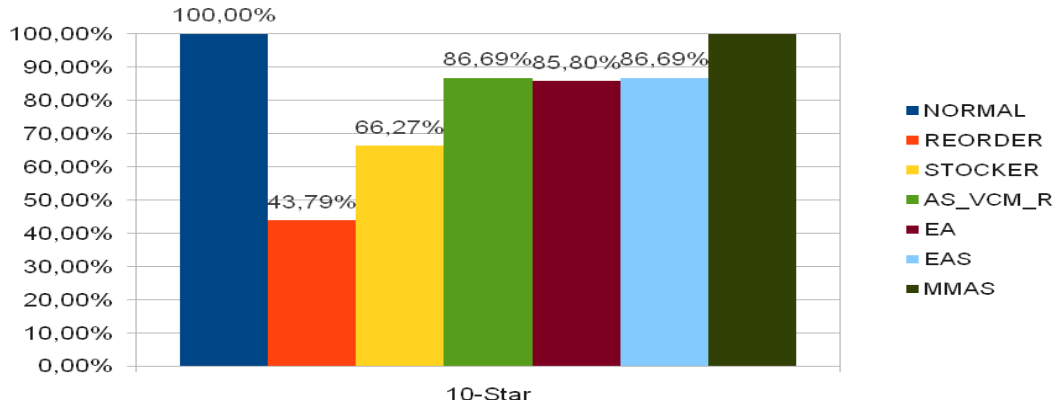

Figure 8.18 Performances for Q3 (cyclic query with 10 triple patterns)

Table 8.13 LUBM queries for 12 triple patterns

	NE	RE	STO	AS	EA	EAS	MMAS
Q1	32864	26745	**306**	2349	2344	2334	2456
Q2	750542	750449	**463**	585	586	592	665
Q3	43821	43870	**321**	426	447	444	447

For 12 triple pattern queries STO is the fastest algorithm. For Q1 even if ACO algorithms can't find the best execution orders, they provide significant improvement for order of triple patterns (Figure 8.19).

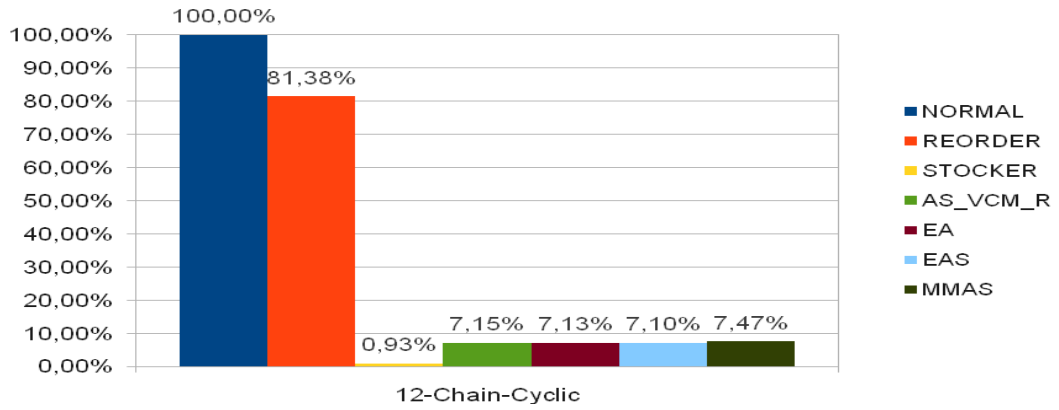

Figure 8.19 Performances for Q1 (chain-cyclic query with 12 triple patterns)

For Q2 (Figure 8.20) and for Q3 (Figure 8.21) ACO algorithms are very successful again. For example execution time of EAS algorithm is 0.08% of NE execution time for Q2 and 1.01% of NE execution time for Q3.

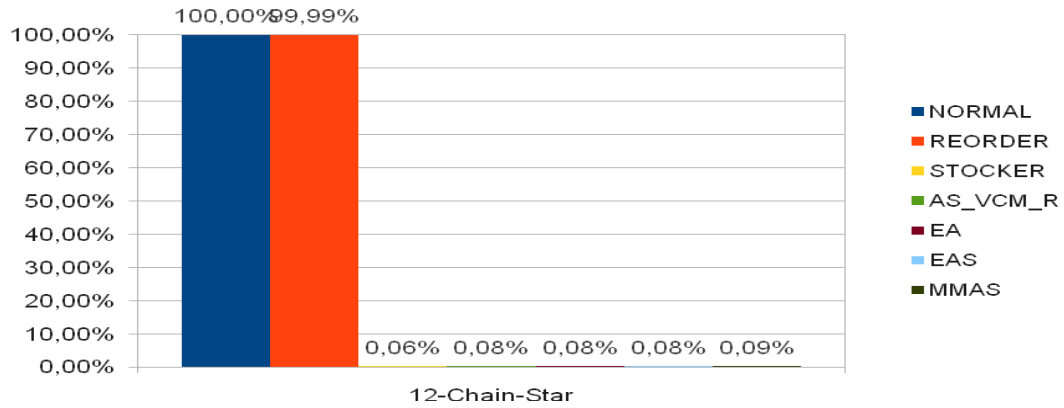

Figure 8.20 Performances for Q2 (chain-star query with 12 triple patterns)

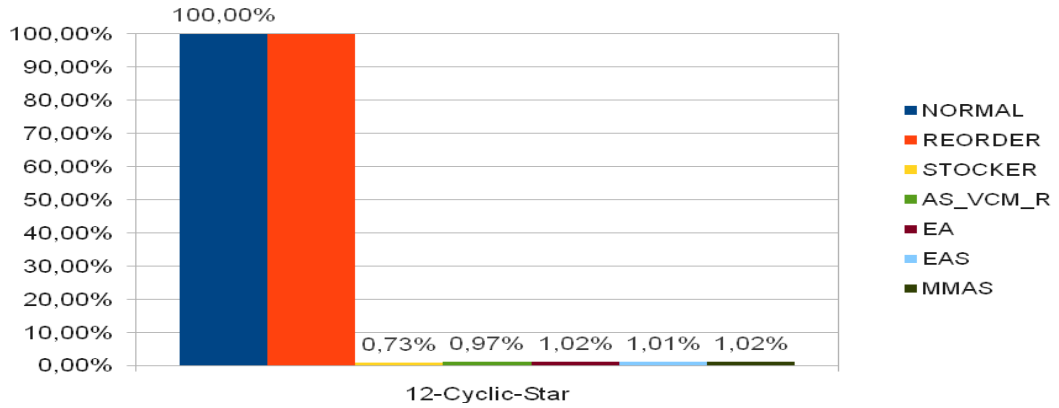

Figure 8.21 Performances for Q3 (cyclic-star query with 12 triple patterns)

Table 8.14 LUBM test queries

	NE	RE	STO	AS	EA	EAS	MMAS
Q2	308747	12386	260	**82**	84	84	135
Q8	1161	1143	404	289	**288**	291	346
Q9	43636	43649	314	233	233	**232**	302

LUBM test queries (Lehigh University, 2012b) which have more than 5 triple patterns are also examined. For LUBM test query 2, AS algorithm provides the best query execution order and other ACO algorithms provide close results to AS (Figure 8.22). STO also provides nearly optimal execution order.

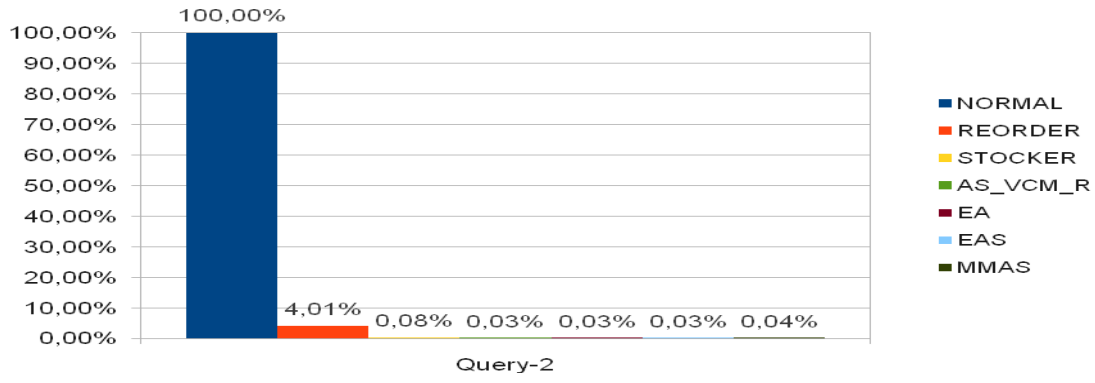

Figure 8.22 Performances for Query-2 LUBM test query

For query-8 and query-9 LUBM test queries, ACO algorithms provide fastest execution orders.

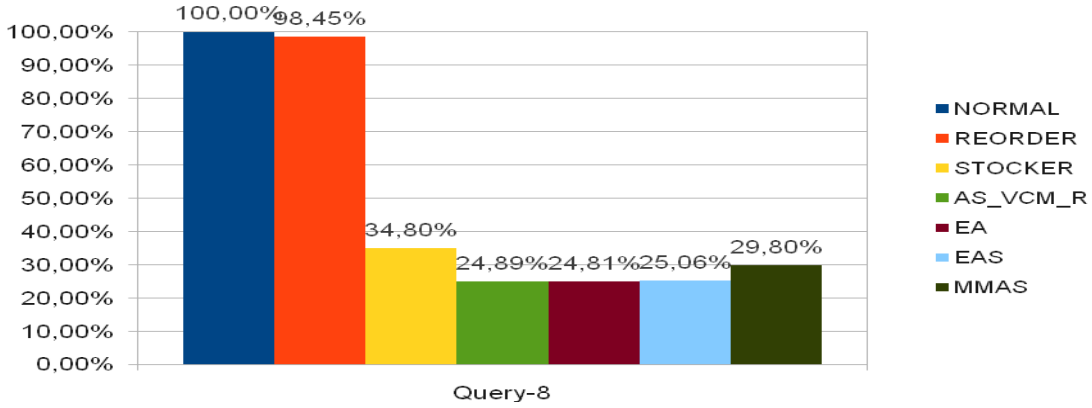

Figure 8.23 Performances for Query-8 LUBM test query

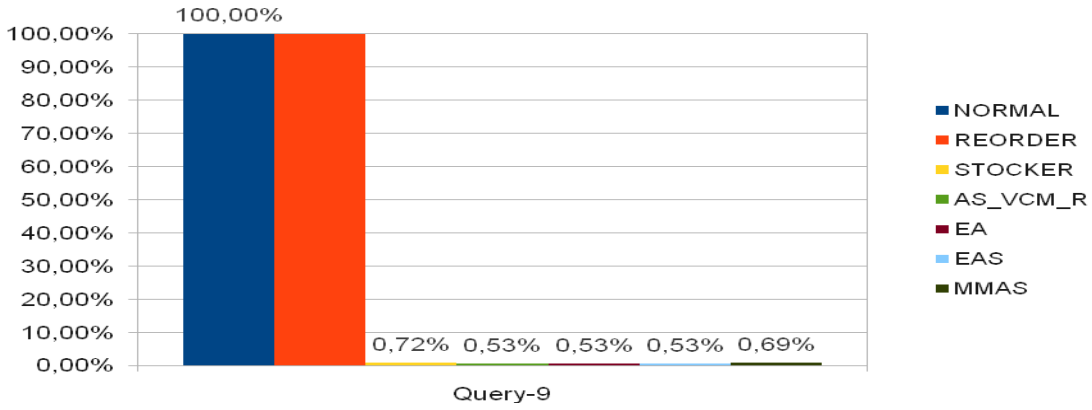

Figure 8.24 Performances for Query-9 LUBM test query

Naturally, the proposed approach can't improve the optimal query triple patterns order. Apart from that experiment results show that proposed approach often produces the best result. Eventhough it can't produce the best result, it can achieve remarkable improvement in comparison to Normal Execution.

CHAPTER NINE
CONCLUSION & FUTURE WORK

9.1 Conclusion

In this study a SPARQL query optimization approach with Ant Colony Optimization is presented. Several experiments have been performed and presented to assess the success of the proposed approach. Contributions of this study can be summarized as follows:

- optimizing SPARQL triple patterns order using ant colony optimization for better execution
- real time optimization without requiring any prior knowledge

Experiments have been performed in CIA Factbook for chain, chain-star queries with 4, 6, 8, 10, 12, 14 triple pattern queries and in LUBM ontologies for chain, cyclic, star, chain-star, chain-cyclic, cyclic-star queries with 4, 6, 8, 10, 12 triple pattern queries. LUBM experiments are also performed for LUBM test queries (2, 8, and 9). Proposed approach is compared with Stocker and others' (2008) approach. Experiment results show that proposed approach is performing well for the majority of the queries. The queries that it does not improve considerably are which already has the optimal order.

Experiments which have been performed on two ontologies show that applicability of proposed approach is satisfactory and this applicability can be improved by finding suitable parameters that are obtained from preliminary parameter analysis.

In this study an opportunity for comparing the ACO algorithms for this problem has been showed up. MMAS, the algorithm which is accepted as a better algorithm for other problems, is failed to satisfy the expectations. It did not provide the best results; on the other hand EAS provided the best results mainly. Two factors are thought to be the

reason to this result. The first one is additional computational load of the MMAS and the second one is the problem specific inapplicability of MMAS.

In conclusion, this proposed novel approach does not require any prior knowledge and reduces execution time considerable for majority of the queries as shown in the experiments.

9.2 Future Work

Although experiments show the success of the proposed method, still there is a room for improving heuristics - selectivity estimation and cost calculation - to optimize all the queries without requiring any prior graph information.

Secondly, different benchmark queries and different ontologies experiments using this method must be conducted. These experiments will help to show the scalability and usability of this method more effectively.

Finally integration of local search techniques to ACO techniques and developing better optimization techniques are planned for the same problem to evaluate if any of these improve current results.

REFERENCES

Antoniou, G. & van Harmelen, F. (2003). *A Semantic Web Primer.* London, England: The MIT Press.

Apache Jena (2012a). Apache Jena. Retrieved June 09, 2012 from http://apache.jena.org.

Apache Jena (2012b). Jena architecture overview. Retrieved June 09, 2012 from http://jena.apache.org/about_jena/architecture.html.

Apache Jena (2012c). ARQ – A Sparql processor for Jena. Retrieved June 09, 2012 from http://jena.apache.org/documentation/query.

Apache Jena (2012d). ARQ- Extending query execution. Retrieved June 09, 2012 from http://jena.apache.org/documentation/query/arq-query-eval.html

Astrahan, M. M., Blasgen, Ht. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P. & others (1976). System R: Relational Approach to Database Management. *ACM Transactions on Database Systems,* (1), 97-137.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284* (5), 34-43.

Bernstein, A., Kiefer, C. & Stocker, M. (2007). OptARQ: Sparql optimization approach based on triple pattern selectivity estimation. *Technical Report,* Department of Informatics, University of Zurich.

Bonabeau, E. & Theraulaz, G. (2008). Swarm Smarts. *Scientific American, 18*, 40-47.

Breitmann K. K., Casanova M. A. & Truszkowski, W., (2007). *Semantic Web: Concepts, Technologies and Applications*. Heidelberg, Germany: Springer-Verlag.

Caliusco, M. L. & Stegmayer, G. (2010). Semantic web technologies and artificial neural networks for intelligent web knowledge source discovery. *Emergent Web Intelligence: Advanced Semantic Technologies, Advanced Information and Knowledge Processing,* 17-36.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. & Wilkinson, K. (2004). Jena: implementing the semantic web recommendations. *In Proceedings of the 13th International World Wide Web Conference on alternate track papers & posters, 04*, 74-83.

Chaudhuri, S. (1998). An overview of query optimization in relational systems. *In Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 34-43.

Che, D., Aberer, K. & Özsu, M. T. (2006). Query optimization in xml structured-document databases. *The VLDB Journal, 15* (3), 263-289.

Central Intelligence Agency (2012). The world Factbook. Retrieved June 09, 2012 from https://www.cia.gov/library/publications/the-world-factbook/index.html.

Davies, J., Studer, R. & Warren, P. (2006). *Semantic Web Technologies : Trends and Research in Ontology-Based Systems*. Chichester, UK: Wiley.

Dollin, C. (2006, March 8). Re: Speed Question. Message posted to http://tech.groups.yahoo.com/group/jena-dev/message/21438.

Dorigo, M., Maniezzo, V. & Colorni, A. (1991). Positive feedback as a search strategy. *Technical Report*. Politecnico di Milano.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms. *PhD Thesis*. Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEE Transactions on Systems, Man, and Cybernetics,* Part B *26* (1), 29-41.

Dorigo, M., Birattari, M. & Stützle T. (2006). Ant colony optimization – artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine,* (1), 28-39.

Dökeroğlu, T. & Coşar, A. (2011). Dynamic Programming with ant colony optimization metaheuristic for optimization of distributed database queries. *In proceedings of 26th International Symposium on Computer and Information Sciences*, 107-113.

Dréo, J. (2006). Shortest path find by an ant colony. Retrieved June, 18, 2012 from http://en.wikipedia.org/wiki/File:Aco_branches.svg.

Haase, P., Broekstra, J., Eberhart, A. & Volz, R. (2004). Acomparison of rdf query languages. *In Proceedings of the Third International Semantic Web Conference,* (3298), 502-517.

Hartig, O. & Heese, R. (2007). The sparql query graph model for query optimization. *In Proceedings of the 4th European Conference on the Semantic Web: Research and Applications*, 564-578.

Ioannidis, Y. E. (1996). Query Optimization. *Computing Surveys, 28* (1), 121-123.

Kader, R. A. & van Keulen, M. (2007). Overview of query optimization in xml database systems. *Technical Report.* University of Twente, Enschede, Netherlands.

Koide, S. & Kawamura, M. (2004). SWCLOS: A Semantic Web Processor on Common Lisp Object System. *In Proceedings of the Third International Semantic Web Conference.*

Lehigh University (2012a). SWAT Projects – the Lehigh University Benchmark (LUBM). Retrieved June 09, 2012 from http://swat.cse.lehigh.edu/projects/lubm/.

Lehigh University (2012b). LUBM Test queries. Retrieved June 09, 2012 from http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt.

Li, N., Liu, Y., Dong, Y. & Gu J. (2008) Application of ant colony optimization algorithm to multi-join query optimization. *In Proceedings of the 3rd International Symposium on Advances in Computation and Intelligence*, 189-197.

Maduko, A., Anyanwu, K., Sheth, A. & Schliekelman, P. (2007). Estimating cardinality of rdf graph patterns. *In proceedings of 16th international conference on World Wide Web, WWW*, 1233-1234.

Maniezzo V., Fabio D. & Gambardella, L. M. (2004). Ant Colony Optimization. In G. C. Onwubolu & B. V. Babu (Ed.). *New Optimization Techniques in Engineering*, 101-117.

Mitchell, G., Zdonik, S. B. & Dayal, U. (1991). Object-oriented query optimization: What's the problem?. *Technical Report*. Brown University, Providence, RI, USA.

Neumann, T. & Weikum, G. (2008). Rdf-3x: A risc style engine for rdf. *In proceedings of VLDB Endowment, 1* (1), 647-659.

Neumann, T. & Weikum, G. (2009). Scalable Join Processing on Very Large RDF Graphs, *In Proceedings of SIGMOD'09 International Conference on Management of Data,* 627-640.

Özsu, M. T. & Blakeley J. A. (1995). Query processing in object oriented database systems. *In Modern Database Systems*, 146-174.

Piatetsky-Shapiro, G. & Connell, C. (1984). Accurate estimation of the number of tuples satisfying a condition. *In proceedings of the ACM SIGMOD International Conference on Management of Data,* 256-276.

Pirahesh, H., Hellerstein, J. M. & Hasan, W. (1992). Extensible/rule based query rewrite optimization in Starburst. *In Proceedings of SIGMOD'92 International Conference on Management of Data,* 39-48.

Ruckhaus, E., Ruiz, E. & Vidal, M. (2008). Query evaluation and optimization in the semantic web. *Theory and Practice of Logic Programming, 8* (3), 393-409.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. & Price, T. G. (1979). Access path selection in a relational database management system. *In Proceedings of the SIGMOD'79 International Conference on Management of Data*, 23-34.

Shironoshita, E. P., Ryan, M. T. & Kabuka, M. R. (2007). Cardinality estimation for the optimization of queries on ontologies. *SIGMOD Record, 36* (2), 13-18.

Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C. & Reynolds, D. (2008). Sparql basic graph pattern optimization using selectivity estimation. *In Proceedings of the 17th International Conference on World Wide Web*, 595-604.

Stuckenschmidt, H., Vdovjak, R., Broekstra, J. & Houben, G. (2005). Towards distributed processing of rdf path queries. *International Journal of Web Engineering and Technology, 2* (2/3), 207-230.

Stützle, T. & Hoos, H. H. (1997). The MAX-MIN Ant System and local search for the traveling salesman problem. *In Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'97),* 309-314.

Stützle, T. & Hoos, H. H. (2000) MAX-MIN Ant System. *Future Generation Computer Systems, 16* (8), 889-914.

Web World Wide Consortium. (2012). Sparql 1.1 query language. Retrieved June 09, 2012, from http://www.w3.org/TR/sparql11-query/.

Wolf, P. (2006, March 8). Speed Question [Msg 1]. Message posted to http://tech.groups.yahoo.com/group/jena-dev/message/21436.

APPENDICES

A. List of Abbreviations

B. List of SPARQL Queries

a. CIA Factbook Queries

b. LUBM Queries

A. LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
API	Application Programming Interface
AS	Ant System
AS-VC	Ant System – Variable Counting
AS-VCM	Ant System – Modified Variable Counting
BGP	Basic Graph Pattern
CIA	Central Intelligence Agency
CPU	Central Processing Unit
DBMS	Database Management System
DOB	Deductive Ontology Base
DP-ACO	Dynamic Programming – Ant Colony Optimization
EAS	Elitist Ant System
GNU	GNU's Not Unix
GSH	Graph Statistics Handler
HP	Hewlett Packard
IRI	Internationalized Resource Identifier
KKE	Karınca Kolonisi Eniyilemesi
LUBM	Lehigh University Benchmark
MMAS	MAX-MIN Ant System
NE	Normal Execution
OpenJDK	Open Java Development Kit
OWL	Web Ontology Language
RCQ-GA	RDF Chain Query - Genetic Algorithm
RDF	Resource Description Framework
RE	Reordered Execution
RISC	Reduced Instruction Set Computer
SOP	Sequential Ordering Problem
SPARQL	SPARQL Protocol and RDF Query Language

SQGM	SPARQL Query Graph Model
SQL	Structured Query Language
STO-VC	Stocker – Variable Counting
TP	Triple Pattern
TSP	Travelling Salesman Problem
VC	Variable Counting
VC-M	Variable Counting – Modified
WWW	World Wide Web
XML	Extensible Markup Language
2PO	Two Phase Optimization

B. LIST OF SPARQL QUERIES

a. CIA Factbook Queries

4 Chain Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?cName ?ethGroup
WHERE {
?countries rdf:type c:Country.
?ethGroup rdf:type o:EthnicGroupPercent.
?countries c:name ?cName.
?countries o:ethnicGroup ?ethGroup.
}
```

4 Chain Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?iName ?industry ?countries
WHERE {
?countries rdf:type c:Country.
?industry rdf:type o:Industry.
?industry o:name ?iName.
?countries o:industry ?industry.
}
```

4 Chain Query 3

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?partner ?agrProduct
WHERE {
?impPartner o:country ?partner.
?partner o:agricultureProduct ?agrProduct.
c:CA o:importPartner ?impPartner.
?agrProduct o:name ?aName.
}
```

4 Chain Query 4

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?partner ?agrProduct
WHERE {
?impPartner o:country ?partner.
?partner o:agricultureProduct ?agrProduct.
?cou rdf:type c:Country.
?cou o:importPartner ?impPartner.
}
```

6 Chain Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?neighbour
WHERE {
?expPartner o:country ?partner.
?partner o:border ?border.
?cou o:exportPartner ?expPartner.
?border o:country ?neighbour.
?cou rdf:type c:Country.
?cou c:name "TURKEY".
}
```

6 Star Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cName ?eIssue ?nHazard ?climate ?dProduct
WHERE {
?cou o:environmentalIssue ?eIssue.
?cou o:naturalHazard ?nHazard.
?cou o:climate ?climate.
?cou rdf:type c:Country.
?cou c:name ?cName.
?cou o:grossDomesticProductAgriculture ?dProduct.
}
```

6 Star Query 3

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cName ?eIssue ?nHazard ?climate ?dProduct
WHERE {
?cou o:environmentalIssue ?eIssue.
?cou o:naturalHazard ?nHazard.
?cou o:climate ?climate.
?cou rdf:type c:Country.
?cou c:name ?cName.
?cou o:grossDomesticProductAgriculture ?dProduct.
}
```

6 Star Query 4

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cName ?tSea ?terrain ?lArea ?eIssue ?nResources
WHERE {
?cou o:territorialSea ?tSea.
?cou o:terrain ?terrain.
?cou o:landArea ?lArea.
?cou o:environmentalIssue ?eIssue.
?cou o:naturalResources ?nResources.
?cou rdf:type c:Country.
?cou c:name ?cName. }
```

6 Chain – Star Query 5

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?nHaz ?eIssue ?aProduct ?div ?d ?label
WHERE {
c:PO o:naturalHazard ?nHaz.
c:PO o:environmentalIssue ?eIssue.
c:PO o:agricultureProduct ?aProduct.
c:PO o:administrativeDivision ?div.
?div rdf:type ?d.
?d o:label ?label.
}
```

8 Chain – Star Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?cou ?rel ?language ?lMale ?lFemale ?mFemale ?mTotal
WHERE {
?cou rdf:type c:Country.
?religion o:religion ?rel.
?cou o:literacyMale ?lMale.
?cou o:literacyFemale ?lFemale.
?cou o:language ?language.
?cou o:religion ?religion.
?cou o:medianAgeFemale ?mFemale.
?cou o:medianAgeTotal ?mTotal. }
```

8 Chain – Star Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?iPartner ?label ?imp ?exp
WHERE {
?iPartner o:participatesIn ?intOrg.
?iPartner o:imports ?imp.
?iPartner o:exports ?exp.
c:VE o:importPartner ?impPartner.
?impPartner o:country ?iPartner.
?iPartner o:administrativeDivision ?div.
?div rdf:type ?d.
?d o:label ?label.
}
```

8 Chain – Star Query 3

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?bord ?imp ?exp ?ind
WHERE {
?bord o:importsCommodity ?impCommodity.
?impCommodity o:commodity ?iCommodity.
?bord o:industry ?ind.
?bord o:imports ?imp.
?bord o:exports ?exp.
?bord o:budgetExpenditures ?budget.
```

```
c:AE o:border ?border.
?border o:country ?bord.
}
```

8 Chain – Star Query 4

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cou ?rel
WHERE {
?relper rdf:type o:ReligionPercent.
?cou o:religion ?relper.
?relper o:religion ?rel.
?cou o:netMigrationRate ?nRate.
?ethGroup o:ethnicGroup ?eGroup.
?eGroup o:name ?eName.
?cou o:populationGrowthRate ?pRate.
?cou  o:ethnicGroup ?ethGroup.
}
```

10 Chain – Star Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cou ?iOrg ?rel ?pRate ?uRate ?eName ?pop
WHERE {
?cou o:religion ?relper.
?relper o:religion ?rel.
```

```
?cou o:netMigrationRate ?nRate.
?cou o:populationGrowthRate ?pRate.
?cou o:participatesIn ?iOrg.
?cou o:unemploymentRate ?uRate.
?cou o:population ?pop.
?cou o:location ?location.
?cou o:economyOverview ?eView.
?cou o:legalSystem ?lSystem.
}
```

10 Chain – Star Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?country ?cName ?eDebt ?iRate ?eView ?eRec ?laborForce
WHERE {
?country o:participatesIn ?intOrg.
?otherCountries o:laborForce ?laborForce.
?otherCountries o:participatesIn ?intOrg.
?otherCountries o:inflationRate ?iRate.
?otherCountries o:externalDebt ?eDebt.
?otherCountries c:name ?cName.
?intOrg rdf:type o:InternationalOrganization.
?intOrg o:name "UNESCO".
?otherCountries o:economyOverview ?eView.
?otherCountries o:economicAidRecipient ?eRec. }
```

10 Chain – Star Query 3

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cName ?ind ?iPartner ?iCommodity ?imp ?bRev ?bExp
WHERE {
?cou c:name ?cName.
?cou o:industry ?ind.
?cou o:importPartner ?impPartner.
?impPartner o:country ?iPartner.
?iPartner o:importsCommodity ?impC.
?impC o:commodity ?iCommodity.
?cou o:imports ?imp.
?cou o:budgetRevenues ?bRev.
?cou o:budgetExpenditures ?bExp.
}
```

10 Chain – Star Query 4

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?neighbour ?eCons ?oCons ?partner
WHERE {
?expPartner o:country ?partner.
?partner o:border ?border.
?cou o:exportPartner ?expPartner.
?border o:country ?neighbour.
?cou rdf:type c:Country.
```

```
?cou c:name "JAPAN".
?cou o:exclusiveEconomicZone ?eZone.
?cou o:oilConsumption ?oCons.
?cou o:electricityConsumption ?eCons.
?cou o:industry ?ind.
}
```

12 Chain Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?countries ?neighbour ?name ?lang
WHERE {
?cBorder o:country ?cBord.
?countries o:border ?cBorder.
?countries c:code "SP".
?countries o:exportPartner ?ePartner.
?ePartner o:country ?ePart.
?ePart o:border ?border.
?border o:country ?neighbour.
?neighbour o:importPartner ?impPartner.
?impPartner o:country ?iPartner.
?iPartner c:name ?name.
?language o:language ?lang.
?cBord o:language ?language.
}
```

12 Chain – Star Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?country ?bord ?bo ?band ?cons ?ind ?suf ?male ?land
WHERE {
?country c:code "GM".
?border o:country ?bord.
?bord o:border ?bor.
?bor o:country ?bo.
?bo o:radioStations ?radio.
?radio o:band ?band.
?country o:constitution ?cons.
?country o:independence ?ind.
?country o:suffrage ?suf.
?country o:medianAgeMale ?male.
?country o:arableLand ?land.
?country o:border ?border.
}
```

14 Chain Query 1

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?country ?ePart ?eBord ?currency ?cCode ?oView
WHERE {
?eBorder o:country ?eBord.
?eBord o:importPartner ?impPartner.
```

```
?impPartner o:country ?iPartner.
?iPartner o:border ?iBorder.
?iBorder o:country ?iBord.
?iBord o:importsCommodity ?commodity.
?commodity o:commodity ?com.
?com o:name ?name.
?country c:code "DA".
?country o:border ?border.
?border o:country ?neighbour.
?neighbour o:exportPartner ?ePartner.
?ePartner o:country ?ePart.
?ePart o:border ?eBorder.
}
```

14 Chain – Star Query 2

```
PREFIX c: <http://www.daml.org/2001/09/countries/fips#>
PREFIX o: <http://www.daml.org/2003/09/factbook/factbook-ont#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?country ?ePart ?eBord ?currency ?cCode ?oView
WHERE {
?ePart o:border ?eBorder.
?eBorder o:country ?eBord.
?eBord o:currency ?currency.
?eBord o:currencyCode ?cCode.
?eBord o:economyOverview ?oView.
?eBord o:budgetRevenues ?bRev.
?eBord o:permanentCrops ?perCrop.
?eBord o:imports ?import.
?eBord o:laborForce ?lForce.
```

```
?country rdf:type c:Country.
?country o:border ?border.
?border o:country ?neighbour.
?neighbour o:exportPartner ?ePartner.
?ePartner o:country ?ePart.
}
```

b. LUBM Queries

6 – Chain Query 1

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?student ?advisor ?course ?tName
WHERE {
?advisor ub:teacherOf ?course.
?tAsst ub:teachingAssistantOf ?course.
?tAsst ub:name ?tName.
?std rdfs:subClassOf ub:Student.
?student rdf:type ?std.
?student ub:advisor ?advisor.
}
```

6 – Cyclic Query 2

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?X ?Y ?Z
WHERE {
?X rdf:type ub:GraduateStudent.
?Y rdf:type ub:University.
?Z rdf:type ub:Department.
?X ub:memberOf ?Z.
?Z ub:subOrganizationOf ?Y.
?X ub:undergraduateDegreeFrom ?Y.
}
```

6 – Star Query 3

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?Y ?course ?eMail ?name ?phone
WHERE {
?Y ub:name  ?name.
?Y ub:emailAddress ?eMail.
?Y ub:telephone ?phone.
?Y ub:teacherOf ?course.
?Y rdf:type ub:AssistantProfessor.
?Y ub:worksFor <http://www.Department0.University0.edu>.
}
```

8 – Chain Query 1

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?professor ?course ?student ?faculty
WHERE {
?faculty ub:worksFor ?dept.
?faculty rdf:type ?f.
?f rdfs:subClassOf ub:Faculty.
?prof rdfs:subClassOf ub:Professor.
?professor rdf:type ?prof.
?professor ub:teacherOf ?course.
?student ub:takesCourse ?course.
```

```
?student ub:memberOf ?dept.
}
```

8 – Cyclic Query 2

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?professor ?course ?gradStudent ?dept
WHERE {
?professor ub:teacherOf ?course.
?gradStudent rdf:type ub:GraduateStudent.
?gradStudent ub:takesCourse ?course.
?gradStudent ub:advisor ?professor.
?professor ub:worksFor ?dept.
?gradStudent ub:memberOf ?dept.
?dept ub:subOrganizationOf <http://www.University0.edu>.
?course ub:name ?name.
}
```

8 – Star Query 3

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?Y ?course ?name ?degree ?tel ?unv
WHERE {
?Y ub:name ?name.
?Y ub:emailAddress ?eMail.
?Y ub:takesCourse ?course.
```

```
?Y ub:telephone ?tel.
?Y ub:memberOf ?unv.
?Y ub:undergraduateDegreeFrom ?degree.
?Y ub:advisor ?advisor.
?Y rdf:type ub:ResearchAssistant.
}
```

10 – Chain Query 1

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?teacher ?dept ?course ?adv ?uName
WHERE {
?faculty rdfs:subClassOf ub:Faculty.
?fac rdfs:subClassOf ?faculty.
?teacher rdf:type ?fac.
?teacher ub:worksFor ?dept.
?student ub:memberOf ?dept.
?student ub:takesCourse ?course.
?tAsst ub:teachingAssistantOf ?course.
?tAsst ub:advisor ?adv.
?adv ub:doctoralDegreeFrom ?unv.
?unv ub:name ?uName.
}
```

10 – Cyclic Query 2

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?sName ?dept ?eMail ?aName
WHERE {
?adv ub:name ?aName.
?adv ub:worksFor ?dept.
?dept ub:subOrganizationOf <http://www.University0.edu>.
?dept rdf:type ub:Department.
?A rdfs:subClassOf  ub:Student.
?student rdf:type ?A.
?student ub:name ?sName.
?student ub:memberOf ?dept.
?student ub:emailAddress ?eMail.
?student ub:advisor ?adv.
}
```

10 – Star Query 3

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?Y ?course ?name ?degree ?tel  ?unv
WHERE {
?Y ub:name ?name.
?Y ub:emailAddress ?eMail.
?Y ub:telephone ?tel.
?Y ub:teacherOf ?course.
?Y ub:undergraduateDegreeFrom ?uDegree.
?Y ub:mastersDegreeFrom ?mDegree.
?Y ub:doctoralDegreeFrom ?dDegree.
?Y ub:worksFor ?unv.
```

```
?Y ub:researchInterest ?interest.
?Y rdf:type ub:FullProfessor.
}
```

12 – Chain – Cyclic Query 1

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?professor ?prof ?rInt  ?student  ?empType ?type
WHERE {
?student ub:takesCourse ?course.
?student ub:advisor ?professor.
?student ub:memberOf ?unv.
?employee ub:worksFor ?unv.
?employee rdf:type ?empType.
?empType rdfs:subClassOf  ?type.
?type rdfs:label ?label.
?professor rdf:type ub:AssociateProfessor.
?professor ub:researchInterest ?rInt.
?prof ub:researchInterest ?rInt.
?prof rdf:type ub:AssistantProfessor.
?prof ub:teacherOf ?course.
}
```

12 – Chain – Star Query 2

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?name ?unv0 ?unv1 ?unv2 ?int ?publication ?course2  ?course
WHERE {
?publication ub:publicationAuthor ?author.
?author rdf:type ub:AssociateProfessor.
?author ub:name ?name.
?author ub:undergraduateDegreeFrom ?unv0.
?author ub:mastersDegreeFrom ?unv1.
?author ub:doctoralDegreeFrom ?unv2.
?author ub:researchInterest ?int.
?professor ub:teacherOf ?course.
?student ub:takesCourse ?course.
?student ub:teachingAssistantOf ?course2.
?professor2 ub:teacherOf ?course2.
?publication ub:publicationAuthor ?professor2.
}
```

12 – Cyclic – Star Query 3

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?student ?teacher  ?course ?sName ?dept ?eMail
WHERE {
?A rdfs:subClassOf  ub:Student.
?student rdf:type ?A.
?C rdfs:subClassOf ub:Faculty.
?D rdfs:subClassOf ?C.
?teacher rdf:type ?D.
?student ub:advisor ?teacher.
?course rdf:type ub:Course.
```

```
?student ub:takesCourse ?course.
?teacher ub:teacherOf ?course.
?student ub:name ?sName.
?student ub:memberOf ?dept.
?student ub:emailAddress ?eMail.
}
```

LUBM – Test Query – 2

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?X ?Y ?Z
WHERE {
?X rdf:type ub:GraduateStudent.
?Y rdf:type ub:University.
?Z rdf:type ub:Department.
?X ub:memberOf ?Z.
?Z ub:subOrganizationOf ?Y.
?X ub:undergraduateDegreeFrom ?Y.
}
```

LUBM – Test Query – 8

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?student ?dept ?eMail
WHERE {
?A rdfs:subClassOf  ub:Student.
?student rdf:type ?A.
```

```
?dept rdf:type ub:Department.
?student ub:memberOf ?dept.
?dept ub:subOrganizationOf <http://www.University0.edu>.
?student ub:emailAddress ?eMail.
}
```

LUBM – Test Query -9

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?student ?teacher  ?course
WHERE {
?A rdfs:subClassOf  ub:Student.
?student rdf:type ?A.
?C rdfs:subClassOf ub:Faculty.
?D rdfs:subClassOf ?C.
?teacher rdf:type ?D.
?student ub:advisor ?teacher.
?course rdf:type ub:Course.
?student ub:takesCourse ?course.
?teacher ub:teacherOf ?course.
}
```