Attribute Subset Selection using Distributed Genetic Algorithm

By

Shahzad Bashir

Fatima Shahid Hashmi

Nageen Asghar

Documentation Submitted to the Department of CS&IT.

University of Sargodha – Pakistan

In partial fulfilment of the requirement for the degree of

BS (CS)

Supervisor

Submission Date: 06-01-2020

Sadaf Aslam

University of Sargodha

ABSTRACT

In this era of big data with facilities for advanced real-time data acquisition, the solutions to large-scale optimization problems are strongly desired. An **optimization problem** is the problem of finding the best solution from all feasible solutions. **Evolutionary computation**, the term now used to describe the field of investigation and the focus of research that concerns evolutionary algorithms, offers practical advantages to the researcher facing difficult optimization problems. **Genetic algorithms** are a subset of evolutionary computing that borrow techniques from classic evolution theories to help find solutions to problems within a large search space. **GA** are efficient optimization algorithms that have been successfully applied to solve a multitude of complex problems. **It** is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution.

In this project, we will compare our Distributed Genetic Algorithm with its Serial version on Benchmark functions like Ackley, Square Sum and Shubert.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Need for Attribute Subset Selection:

The data set may have many attributes. But some of those attributes can be irrelevant or redundant. The goal of attribute subset selection is to find a minimum set of attributes, such that dropping of those irrelevant attributes does not much affect the utility of data and the cost of data analysis could be reduced. Mining on a reduced data set also makes the discovered pattern easier to understand.

1.2 Significant Scope:

The scope of attribute subset selection is to find a minimum set of attributes such that dropping of those irrelevant attributes which do not much affect the utility of data and the cost of data analysis could be reduced which make it significant. Genetic algorithms (GAs) are a heuristic search and optimisation technique inspired by natural evolution. They have been successfully applied to a wide range of real-world problems of significant complexity.

1.3 Complexity:

The core complexity in our project is that to extract meaningful compact information from multidimensional data through central pre-processing steps. Due to massive data from heterogenous sources the predictive accuracy of algorithms also may be reduced due to the presence of irrelevant attributes.

1.4 Innovation:

In our project we are going to implement a hierarchical implementation of a Distributed genetic algorithm (Data Pre-processing, GA population evaluation, and GA operations) to find the minimal rough set reduction. The process of evaluating the entire Data set is performed in several iterations. Furthermore, with the help of this model results will be more accurate and faster as compared to any other approach.

1.5 Applicable:

Being population-based characteristics of GA, it would solve evolutionary problem It has a plethora of decision-making applications and intrusion decision in scientific, engineering and business domains. Mining on a reduced data set also makes the discovered pattern easier to understand that's why our project is applicable.

1.6 RESEARCH QUESTIONS:

Our research can be evaluated based on the following questions:

I. Is the DGA time efficient?
II. Is the DGA scalable?
III. Will the DGA in fact compete with the serial version?

Chapter 2

Literature Review

2.1 Distributed implementation using Apache Spark of a genetic algorithm applied to test data generation [1]

Main Contribution:

- Parallel implementation of a genetic algorithm in Apache Spark.
- The approach in this research was rather generic, so it had potential to be adapted to other types of GA.
- A fitness function was proposed based on some "probabilities" that certain branch conditions occur in some order and use them to guide the tests towards areas not yet explored. This was different from existing "goal-oriented" approaches, which attempted to find test data for a given path in the control-flow graph.

Drawbacks:

The loading time of the fitness evaluation process took significant time due to operation system dependencies and initialization times.

Reading and Writing data to disk also took significant time, especially in the case of distributed computing when the same machine had several physical processes that could execute parallel tasks but usually shared the same disk.

2.1 Spark-Based Parallel Genetic Algorithm for Simulating a Solution of Optimal Deployment of an Underwater Sensor Network [2]

Main Contribution:

- The Shubert multi-peak function (SMPF) is used to simulate deployment of underwater sensor network. By calculating the extremums of the Shubert multi-peak function (ESMPF), the simulating optimal deployment sites can be obtained.
- Based on RDD computation model of the Spark framework, a parallel GA for optimizing the deployment of a UWSN (DUWSN) is designed and implemented.
- By the comparison with the GAs based on single-node and Hadoop, it is verified that the proposed GA runs more efficiently while showing a higher accuracy.

Conclusions:

A Spark-based parallel GA is proposed to simulate the optimal deployment of an underwater sensing network. Compared with single-node-based GA and Hadoop-based GA, the Spark-

based parallel GA can significantly shorten the computation time of iterative evolution when dealing with large-scale underwater sensing nodes. Thus, a remarkable improvement in the timeliness of solving the optimal deployment of a UWSN is achieved. Encouragingly, the innate natural randomness and distribution of the Spark-based parallel GA entitles it to avoid local optimization effectively, which further improves the accuracy of the proposed method and finally results in optimal deployment of the UWSN

2.3 Scalable Distributed Genetic Algorithm Using Apache Spark (S-GA) [3]

S-GA has been tested on several numerical benchmark problems for large-scale continuous optimization containing up to 3000 dimensions, 3000 population size, and one billion generations. S-GA presents a variant of island model and minimizes the materialization and shuffles in RDDs for minimal and efficient network communication. At the same time it maintains the population diversity by broadcasting the best solutions across partitions after specified Migration Interval. They have tested and compared S-GA with the canonical Sequential Genetic Algorithm (SeqGA).

Conclusion:

S-GA has been found to be more scalable and it can scale up to large dimensional optimization problems while yielding comparable results.

Chapter 3

Research Methodology

3.1 Implementation:

Scala in context of spark while results will be evaluated on cluster. **Scala** helps to dig deep into the **Spark's** source code that aids developers to easily access and implement new features of **Spark**.

Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way. Scala has been created by Martin Odersky and he released the first version in 2003. Scala smoothly integrates the features of object-oriented and functional languages.

3.2 Scala is Object-Oriented:

Scala is a pure object-oriented language in the sense that every value is an object. Types and behaviour of objects are described by classes and traits. Classes are extended by **subclassing** and a flexible **mixin-based composition** mechanism as a clean replacement for multiple inheritance.

3.3 Scala is Functional:

Scala is also a functional language in the sense that every function is a value and every value is an object so ultimately every function is an object. Scala provides a lightweight syntax for defining **anonymous functions**, it supports **higher-order functions**, it allows functions to be **nested**, and supports **currying**.

3.4 What Does Spark Do?

Spark can handle several petabytes of data at a time, distributed across a cluster of thousands of cooperating physical or virtual servers. It has an extensive set of developer libraries and APIs and supports languages such as Java, Python, R, and Scala; its flexibility makes it well-suited for a range of use cases. Spark is often used with distributed data stores such as MapR XD, Hadoop's HDFS, and Amazon's S3, with popular NoSQL databases such as MapR Database, Apache HBase, Apache Cassandra, and MongoDB, and with distributed messaging stores such as MapR Event Store and Apache Kafka.

Spark supports the following resource/cluster managers:

- **Spark Standalone** – a simple cluster manager included with Spark

- **Apache Mesos** – a general cluster manager that can also run Hadoop applications
- **Apache Hadoop YARN** – the resource manager in Hadoop 2
- **Kubernetes** – an open source system for automating deployment, scaling, and management of containerized applications

3.5 Technical Details:

Being population-based characteristics of GA, it would solve evolutionary problems.

Techniques used in our research are:

- Genetic Algorithm

Algorithm 1. Sequential Genetic Algorithm

1. P <- Generate Initial Population
2. While Stopping Criteria not met do
3. P' <- Select Parents (P)
4. P' <- Crossover (P')
5. P' <- Mutate (P')
6. P' <- Survival- Selection (P U P')
7. P <- P'

Algorithm 2. Pseudo Code for S GA

N: Population Size

P: Population

Pi: Sub-Population at partition i

D: Dimensions

G: Generations

m: Number of Partitions

Mi: Migration Interval / gap

f : Fitness Function

Ms: Migration Size

1: Randomly initialise population of size P

2: Distribute P among m partitions

3: G = 0+ 4: while stopping criteria not met do

5: at each partition i

6: for k: 1 to Mi do

7: Pi" Select Parents (Pi')

8: Pi" Crossover (Pi')

9: Pi" Mutate (Pi')

10 Calculate Fitness (Pi'')

11: Pi' Survival_ Selection (Pi' U Pi")

12: end for

13: BroadcastSolutions

13: End at each partition i

14. Pi' =(Pi - (weak (m* Ms) solutions)) BroadcastSolutions

15: G = G + Mi

16: end while

- Benchmark Functions
- Island Model
- Crossover
- Mutation
- Selection
- Replacement
- Evolutionary Computation
- Objective/Optimization Function
- Map Reduce
- RDD

Experimental Setup:

The experiments are performed on a three node cluster: DELL PowerEdge R815, 2x AMD Opteron 6376 (64 Cores), 256 GB .RAM, 3 TB SATA RAID-5 with spark-2.1.0 and Scala 2.11.8.

Chapter 4

1. References

REFERENCES:

[1] Paduraru, Ciprian & Melemciuc, Marius-Constantin & Stefanescu, Alin. (2017). A distributed implementation using apache spark of a genetic algorithm applied to test data generation. 1857-1863. 10.1145/3067695.3084219.

[2] Liu, P., Ye, S., Wang, C., & Zhu, Z. (2019). Spark-Based Parallel Genetic Algorithm for Simulating a Solution of Optimal Deployment of an Underwater Sensor Network. *Sensors*, *19*(12), 2717.

[3] Maqbool F., Razzaq S., Lehmann J., Jabeen H. (2019) Scalable Distributed Genetic Algorithm Using Apache Spark (S-GA). In: Huang DS., Bevilacqua V., Premaratne P. (eds) Intelligent Computing Theories and Application. ICIC 2019. Lecture Notes in Computer Science, vol 11643. Springer, Cham