Scalability and Fairification of Evolutionary Algorithms

By

Fahad Maqbool

A thesis submitted to the Computer Science & Information Technology.

University of Sargodha - Pakistan

In partial fulfillment of the requirement for the degree of

PhD (Computer Science)

Supervisor

Dr. Muhammad Ilyas
Assistant Professor of Computer Science & Information Technology
University of Sargodha

Submission Date: February 21, 2024

Declaration

I Fahad Maqbool (SCSF17E002) student of the Computer Science & Information Technology, University of Sargodha hereby declare that I have completed the work in due course of degree time frame. I have not used any material without proper referencing and following the acceptable plagiarism limit guidelines. I have followed all the guidelines required to complete the thesis.

Date: February 21, 2024 Signature: .

Fahad Maqbool

SCSF17E002

DEDICATION

I dedicate this thesis to my parents...

ACKNOWLEDGEMENTS

I would like to thanks Allah Almighty who gave me the courage, and serenity to complete this milestone. Moreover I am thankful to my parents whose prayers has always been the source of strength for me. Moreover I am thankful to my supervisor Dr. Muhammad Ilyas for continuous motivation and support throughout my degree time frame. I am specially thankful for the time and efforts by Dr. Hajia Jabeen (Co-Supervisor) for her research collaboration and guideline through out the research work. With out the support of my supervisors it would have been difficult for me to complete this thesis on time. Finally I would like to thanks to my friend and colleague Mr. Saad Razzaq, whose continuous support and encouragement through out the degree time.

Fahad Maqbool

SCSF17E002

THESIS SUBMISSION CERTIFICATE

It is certified that the research thesis entitles "**Scalability and Reproducibility of Evolutionary Algorithms**" that has been submitted by **Fahad Maqbool** is an original research work that is not copied from or submitted to any other university for any degree requirements. The candidate has compiled the thesis according to the University of Sargodha plagiarism guidelines. Also, it is certified that the candidate has made all necessary amendments in thesis as pointed out by supervisor and co supervisor.

_____

Dr. Muhammad Ilyas (Supervisor I)
Assistant Professor
University of Sargodha
Sargodha, Pakistan

_____

Dr. Hajira Jabeen (Supervisor II)
Lead Big Data Team
GESIS-Leibniz Institute for the Social Sciences
Cologne, Germany

_____

Incharge/Chairman
Dept. of Computer Science & IT
University of Sargodha

ABSTRACT

In this era of big data with advanced real-time data acquisition, the solutions to large-scale optimization problems are strongly desired. Evolutionary Algorithms are efficient optimization algorithms that have been successfully applied to solve a multitude of complex problems. Evolutionary algorithms (EA) have been found efficient in solving complex optimization problems. However, the performance of conventional EAs degrades with the increasing number of decision variables due to the lack of scalability. The growing need for large-scale optimization and inherent parallel evolutionary nature of the algorithms calls for exploring them for parallel processing using existing parallel, in-memory, distributed computing frameworks like Apache Spark. Large Scale Global Optimization (LSGO) is interesting for its applications in machine learning, e.g. in deep learning or knowledge graph embeddings. Despite the performance, popularity and simplicity of Evolutionary algorithms, not much attention has been paid towards reproducibility and reusability of Evolutionary algorithms.

In this thesis we have proposed a Blind Naked Mole Rat based Coloring (BNMR-Col) for solving complex optimization problem. BNMR-Col uses both exploitation and exploration to find the best solution in search space. Exploitation uses both local moves and global moves to find a better solution in the surroundings of an existing solution. On the other hand, exploration generates new solution by combining various solutions from the search space. We have also proposed a Scalable Genetic Algorithms using Apache Spark (S-GA). S-GA makes liberal use of rich APIs offered by Spark. We have tested S-GA on several benchmark functions for large-scale continuous optimization containing up to 3000 dimensions, 3000 population size, and one billion generations. We have also proposed a scalable, parallel, distributed and hybrid EA named Distributed Scalable Shade-BatScalable Shade Bat (DistSSB) to solve LSGO problems. DistSSB is inspired by the exploration capability of the SHADE algorithm and exploitation feature of the Bat algorithm (BA). To achieve scalability, DistSSB is implemented using the popular distributed in-memory framework, Apache Spark. DistSSB distributes its population into multiple sub-populations using the island model. Each sub-population is independently evolved using SHADE or BA. DistSSB is scalable for for up to *"one million"* dimensions.

Finally we have presented an Evolutionary algorithm i.e., Genetic Algorithm (GA) as a usecase to FAIR-Algorithms. We have extended Findable, Accessible, Interoperable and Reusable (FAIR) data principles to enable the reproducibility and reusability of algorithms. Additionally, to enable FAIR algorithms, we propose a vocabulary (i.e. evo) using lightweight RDF format, facilitating the reproducibility.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this era of big data with advanced real-time data acquisition, solutions to large-scale optimization problems are strongly desired. Real-life optimization problems are becoming increasingly complex due to an increase in the number of decision variables as a result of large-scale problems. Knowledge Graph Embedding (KGE) is an example of a large-scale problem where large-scale knowledge graphs like DBpedia (2 million entities and 10 million facts), Wikidata (1.5 million entities and 3 million facts), and Google KG (500 million entities and 3.5 billion facts) are transformed to multidimensional metrics Ji et al. (2020).

Evolutionary algorithms are the biologically inspired set of algorithms that uses the natural evolution process to solve complex and multiple optimization problems. Evolutionary algorithms have the ability to traverse the search space both in an exploratory and exploitative manner, enabling them to find solutions from the explored search space, as well as to discover new regions of search space to find better solutions. The growing need for large-scale optimization and the inherent parallel evolutionary nature of the Evolutionary algorithms calls for exploring them for parallel processing using existing parallel, in-memory, distributed computing frameworks like Apache Spark.

With the advancement of distributed and parallel cluster computing frameworks like Apache Hadoop, Apache Spark, and Apache Flink there is a growing need to leverage these frameworks for large-scale optimization problems. Apache Hadoop map-reduce framework is useful for batch processing of large-scale data, while Apache Spark is faster than Hadoop due to its in-memory computing power and capability to work with iterative algorithms and interactive data processing. Apache Flink is useful for real-time and event-driven large-scale applications (Ferrucci et al., 2018, Zaharia et al., 2016a). In this thesis, we are using Apache SPARK with evolutionary algorithms for solving Large Scale Global Optimization (LSGO) problems due to its fast in-memory data processing power and capability to work with algorithms that are iterative in nature.

A lot of scientific and research data is produced by scientific, educational, and research institutes annually. In the recent past FAIR guidelines have been proposed by Wilkinson et al. Wilkinson et al. (2016). They have highlighted the importance of making scientific data findable, accessible, interoperable, and reusable. FAIR guidelines are highly appreciated and followed by researchers and scientific communities in making scientific data FAIR. This will help them in improving the reusability and reproducibility of data. After the successful adoption of FAIR data principles, FAIR research software guidelines Gruenpeter et al. (2020), FAIR ontologies Guizzardi (2020), and FAIR SmartAPI Vita et al. (2018) have been proposed. In this thesis, we have extended FAIR data principles for algorithms and proposed $FAIR-Algorithms$.

In this thesis, we have also proposed BNMR-Col to solve a complex optimization problem (i.e. graph coloring). Moreover, we have proposed Scalable Genetic Algorithms (S-GA) and Distributed Scalable Shade-Bat (DistSSB) using Apache Spark for solving LSGO problems. We have tested them on 15 (fifteen) benchmark functions. To improve the reproducibility and reusability of Evolutionary algorithms, we have proposed $FAIR-Algorithms$ and presented $FAIR-GA$ as a use case of $FAIR-Algorithms$.

1.1 Motivation

Real-life optimization problems are becoming increasingly complex due to an increase in the number of decision variables as a result of big data and large-scale data availability. Evolutionary algorithms have been found efficient in solving complex optimization problems due to the inherently parallel nature of evolutionary algorithms. However, these conventional techniques are not scalable to large problem sizes and their performance degrades with the increasing number of decision variables. Moreover, with the advancement of distributed and parallel cluster computing frameworks(i.e. Apache Hadoop, Apache Spark, Apache Flint etc.) in the last few years, researchers have shifted their focus toward the development of scalable techniques. This motivates us to develop scalable evolutionary algorithms leveraging the existing parallel, in-memory, and distributed computing frameworks like Apache Spark.

Another motivation behind this work is that a considerable portion of scientific data and research manuscripts remain unnoticed every year due to partial findability, accessibility, reusability, and interoperability by humans or machines. Only one-fifth

of the published manuscripts also publish experimental data on some data repositories. In current research practices, most of the data used in research articles is not findable. Hence, it cannot easily be reused by the research community. Similarly, the act of sharing research software (i.e. code and hyper-parameter settings) is not a common practice due to little to no attribution mechanisms for the research software developers. In most cases, research software's details are briefly shared in research manuscripts and hence are far from being findable and reproducible. Same naming conventions of different digital artifacts, the different naming conventions of the same digital artifact, ignored practices of publishing dataset and software code with research articles, and sharing code in repositories without rich metadata adds a challenge to code findability and hence compromises the algorithm reusability and reproducibility. Metadata is an important factor to improve reproducibility and hence reusability. The lack of algorithms-specific metadata limits the researchers to reproduce the results. In recent years, efforts have been made in research, academia, development, and industry to make scientific data FAIR (Findable, Accessible, Interoperable, Reusable) for both humans and machines. This inspires us to not only develop algorithm-specific metadata but also extend FAIR data principles to conform with algorithms.

1.2 Problem Statement & Challenges

In this era of big data with advanced real-time data acquisition, solutions to LSGO problems are strongly desired. LSGO is interesting for its applications in machine learning, like knowledge graph embeddings and computational creativity. Evolutionary algorithms have been found efficient in solving complex optimization problems like Graph Coloring Problems (GCP). However, there are still many evolutionary algorithm techniques whose potential has still been under-explored for various optimization problems. Moreover, the performance of conventional evolutionary algorithms degrades with the increasing number of decision variables due to the lack of scalability.

The growing need for large-scale optimization and the inherent parallel evolutionary nature of the algorithms call for exploring them for parallel processing using existing parallel, in-memory, distributed computing frameworks like Apache Spark. Evolutionary algorithms are popular meta-heuristics that use stochastic operators to find

optimal solutions and have proved their effectiveness in solving many complex optimization problems (such as classification, optimization, and scheduling). However, despite their performance, popularity, and simplicity, not much attention has been paid to the reproducibility and reusability of evolutionary algorithms. Scalability, reusability, and reproducibility are the major challenges that still need to be addressed by the evolutionary algorithm research community.

1.3 Research Questions

This thesis covers the following research questions.

> RQ1. How complex optimization problems can be solved using evolutionary algorithms?

We have used Blink Naked Mole Rat (BNMR) algorithm to find the optimal chromatic number for GCP. BNMR uses both exploration and exploitation to find optimal/near-optimal solutions. Exploitation uses both local moves and global moves to find a better solution in the surroundings of an existing solution. whereas exploration generates new solutions by combining various solutions from the search space.

> RQ2. How we can exploit Apache Spark frame work to improve the scalability of evolutionary algorithms?

Apache SPARK is useful for iterative and interactive data processing. In Spark Core API, high-level abstraction is provided using RDD to distribute data across the cluster for further processing. Due to this parallel processing in-memory distributed computing power of Apache SPARK, EA are a suitable choice for solving LSGO problems. This is due to the reason that EA are inherently parallel in nature. We have proposed Scalable Genetic Algorithms (S-GA) and Distributed Scalable Shade-Bat (DistSSB) using Apache Spark.

> RQ3. Real-world problems (e.g., Knowledge Graphs, Deep Learning, NLP) are increasing in size. Can we optimize conventional EA techniques for very large dimensions?

We have proposed two scalable EA techniques (i.e. S-GA and DistSSB) in this thesis. S-GA has been successfully tested for 3000 dimensions, while DistSSB scales to one million dimensions.

RQ4. Can we reproduce the results of EA to improve the reusability?

Metadata is an important factor to improve reproducibility and hence reusability. Lack of techniques specific metadata limits the researchers to reproduce the results. In this thesis we have propose a specialized metadata for Genetic Algorithms(GA). Also, we have proposed a specialized vocabulary (i.e., evo) for GA. The evo vocabulary covers detailed parameters related to GA that helps in making GA FAIR.

RQ5. Can we make algorithms FAIR?

We have extended FAIR data principles and applied them to make FAIR algorithms. Moreover, we have applied the extended FAIR principles to GA as a usecase.

1.4 Thesis Overview

1.4.1 Contributions

This thesis contributes in the areas of using evolutionary algorithms on real life problem, Large Scale Global Optimization, and FAIR algorithms.

1. We have proposed Blind Naked Mole Rat based Coloring (BNMR-Col) for graphs. BNMR-Col uses both exploitation and exploration to find the best solution in search space.
2. We have presented an algorithm for Scalable Genetic Algorithms using Apache Spark (S-GA). S-GA makes liberal use of rich APIs offered by Apache Spark. We have tested S-GA on several benchmark functions for large-scale continuous optimization containing up to 3000 dimensions, 3000 population size, and one billion generations. S-GA minimizes the materialization and shuffles on RDD in order to reduce network communication overhead. However, it maintains population diversity by broadcasting the best solutions across partitions after a specified migration interval.

3. We have proposed a scalable, parallel, distributed, and hybrid evolutionary algorithm named Distributed Scalable Shade-Bat (DistSSB) to solve LSGO problems. DistSSB is inspired by the exploration capability of the SHADE algorithm and the exploitation feature of the Bat algorithm (BA). To achieve scalability, DistSSB is implemented using the popular distributed in-memory computing framework, Apache Spark. DistSSB distributes its population into multiple sub-populations using the island model. Each sub-population is independently evolved using SHADE or BAT. After the migration interval, the best solutions are broadcasted employing the mesh topology.
4. We have extended Findable, Accessible, Interoperable and Reusable (FAIR) data principles to enable the reproducibility and reusability of algorithms. We have chosen GA as a use case to demonstrate the applicability of the proposed principles. Also, we have presented an overview of methodological developments and variants of GA that makes it challenging to reproduce or even find the right source. Additionally, to enable FAIR algorithms, we propose a vocabulary (i.e. evo) using a lightweight RDF format, facilitating reproducibility.

1.4.2 List of Publications

- ***Fahad Maqbool***, Muhammad Fahad, Muhammad Ilyas, and Hajira Jabeen. "Graph Coloring using Evolutionary Algorithm" published in Expert Systems. Wiley, 2023. https://doi.org/10.1111/exsy.13262
- ***Fahad Maqbool***, Saad Razzaq, Jens Lehmann, and Hajira Jabeen. "Scalable distributed genetic algorithm using apache spark (s-ga)." In International conference on intelligent computing, pp. 424-435. Springer, Cham, 2019. https://doi.org/10.1007/978-3-030-26763-6_41
- ***Fahad Maqbool***, Saad Razzaq, Asif Yar, and Hajira Jabeen. "Large Scale Distributed Optimization using Apache Spark: Distributed Scalable Shade-Bat (DistSSB)." In 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 2559-2566. IEEE, 2021. https://doi.org/10.1109/CEC45853.2021.9504853
- ***Fahad Maqbool***, Saad Razzaq, and Hajira Jabeen. "FAIRy Tale of Genetic Algorithm" to be submitted. *Preprint:* https://arxiv.org/abs/2305.00238

1.5 Thesis Structure

This thesis is comprised of eight chapters. Chapter 1 briefly explains problem & its significance, problem statement, challenges, research questions, thesis structure, and list of publications. In Chapter 2, we have briefly explained the preliminary concepts related to our thesis. Chapter 3 discusses related work and identifies the potential research gap. In Chapter 4, we have explained EA-based Graph Coloring. Here we have presented a BNMR-Col approach to solve GCP. In Chapter 5, we have presented Scalable Genetic Algorithm (S-GA) that uses Apache Spark to deal high dimensional problems. S-GA has been successfully tested up to 3000 dimensions. In Chapter 6, we have proposed a hybrid scalable Evolutionary Algorithm (i.e. Distributed Scalable Shade-BAT (DistSSB)) that uses Apache Spark framework, exploration capability of BAT algorithm, and exploitation power of SHADE algorithm to solve Large Scale Global Optimization (LSGO) problems. DistSSB experiments reveal its scalability up to one million dimensions. In Chapter 7, we have extended the FAIR data principles for making algorithms FAIR. Moreover, we have presented a use case of FAIR algorithms (i.e.FAIR-GA). Finally, in Chapter 8 we present a summary of the thesis. Also, we float interesting future directions in this chapter.

Main contribution of this chapter is as follows.

Chapter 2

Preliminaries

Evolutionary algorithms have been found efficient in solving complex optimization problems due to the inherently parallel nature of these algorithms. In this chapter, we have explained the evolutionary algorithms that are used in this thesis to solve complex optimization and LSGO problems. The brief detail of these algorithms and related concepts is explained in the following subsections.

2.1 Sequential Genetic Algorithm (SeqGA)

SeqGA, also known as Canonical GA is a stochastic search method that is used to find the optimal solution for a given optimization problem using the Darwinian principle of evolution "Survival of the Fittest". It creates a single pool of possible solutions population (panmixia) and applies stochastic operators (i.e. Selection, Crossover, Mutation, and Survival Selection) to create a new evolved solution. This process of new population evolution continues until the population has converged to an optimal solution, or desired time/effort has elapsed. For large-scale, or complex problems, SeqGA may require more computational effort like more memory and long execution time. Algorithm 1. explains the working of SeqGA. Gao and Neumann (2014), Keco and Subasi (2012), Lim et al. (2007), Liu and Wang (2015), Trivedi et al. (2015), Whitley et al. (1999)

Algorithm 1 Sequential Genetic Algorithm

$P \leftarrow GenerateInitialPopulation$

while Stopping Criteria not met **do**
  $P' \leftarrow SelectParents(P)$
  $P' \leftarrow Crossover(P')$
  $P' \leftarrow Mutate(P')$
  $P' \leftarrow Survival - Selection(PUP')$
end while

Select Parents specifies the individual selection mechanism for reproduction or recombination. *Crossover* helps to explore the search space by generating new solutions after recombination, while *mutation* exploits the solutions for improvement by random perturbation of the selected solution. The *Survival Selection* scheme decides the number of individuals to be selected from parents and offsprings for the next generation.

2.2 BAT Algorithm

Yang and Xin-She proposed a novel meta-heuristic based Bat algorithm (BA) in 2010 Yang (2010). BA uses the echolocation behavior of microbats to find the optimal solution. Following control parameters are used in BA.

- N: Population size
- P: Actual population comprising N bats.
- x_i: Position of Bat_i
- v_i : Velocity of Bat_i
- f_r^{max} & f_r^{min} : Upper and lower frequency bounds
- ϵ: Bandwidth for the calculation of a reasonable gradient.
- r: Pulse rate that determines the exploitation rate.
- it : Current iteration
- A_i : Loudness rate of Bat_i
- $\widehat{A}$: Mean loudness of all bats
- r_i^0 : Initial pulse rate of Bat_i
- α & γ : The two constant parameters of range [0,1] used to update A and r

The velocity v_i (Eq. 2.2) of Bat_i is influenced by a random frequency f_r (Eq. 2.1). New bat position x_i can be calculated using Eq. 2.3.

$$f_r = f_r^{min} + (f_r^{max} - f_r^{min}) * rand[0,1] \tag{2.1}$$

$$v_i = v_i + (x_i - x_{best}) * f_r \tag{2.2}$$

$$x_i = x_i + v_i \tag{2.3}$$

Local search is part of the exploitation process. The new bat position is calculated using a random walk around the x^{best} position using Eq. 2.4.

$$x_i^{new} = x^{best} + \epsilon \widehat{A} \tag{2.4}$$

x_i is replaced by x_i^{new} if fitness of x_i^{new} is better than the fitness of x_i and loudness is less than $U[0,1]$. x^{best} is replaced by x^{new} if fitness of x^{new} is better than x^{best}. x_i, r_i, and A_i are updated using Eq. 2.5 and Eq. 2.6 respectively. algorithm 2 presents the pseudo-code of the bat algorithm.

$$r_i = r_i^0 * (1 - e^{(-\gamma * it)}) \tag{2.5}$$

$$A_i = \alpha A_i \tag{2.6}$$

Algorithm 2 Bat Algorithm

```
Initialize the population P
while stopping criteria do
    for x_i: 1 to N do
        Calculate f_r using Eq. 2.1
        Calculate v_i using Eq. 2.2
        Find x_i using Eq. 2.3
        if rand[0,1] ≥ r_i then
            x_i^new ← Perform local search using Eq. 2.4
        end if
        if rand[0,1] ≤ A_i & f(x_i^new) ≤ f(x_i) then
            x_i = x_i^new
            Update r_i using Eq. 2.5
            Update A_i using Eq. 2.6
        end if
    end for
end while
```

2.3 Differential Evolution

DE is a population-based EA, designed for solving continuous numerical optimization problems Storn and Price (1997), Wang et al. (2013). The three control parameters of DE are population size (N), crossover rate (C_R), and scaling factor (F). Mutation,

crossover, and selection are the three commonly used DE operators. A new candidate (donor) solution (v_d) is generated by applying the mutation operator. In the crossover phase, the donor solution interacts with the currently selected candidate solution x_i to generate a trail vector u_t. In the selection phase fitness of u_t is compared with x_i. u_t replaces x_i, if $f(u_t)$ is better than $f(x_i)$. $DE/rand/1/bin$ is the classical mutation strategy of DE. Here rand is base vector, 1 specifies the number of difference vector(s), and bin is the recombination type. In DE two recombination types (binomial (bin) and exponential (exp) are commonly used.

2.4 Success-History Based Parameter Adaptation for Differential Evolution (SHADE)

Among many variants of DE, SHADE has gained much popularity as it computes the control parameters (crossover CR and scaling factor F) dynamically Tanabe and Fukunaga (2013a), based on the history. It maintains a history data structure H that contains N entries for crossover (M_{CR}) and scaling factor (M_F). Initially, all values of H are set to a constant value, i.e., 0.5. A random index r_i is selected from $[1 : H]$ and is used to compute values of both control parameters F_i and CR_i in each iteration. F_i is calculated at line 8 in algorithm 3 using Eq. 2.7.

$$F_i = rand_c(M_{F,r_i}, 0.1) \tag{2.7}$$

Algorithm 3 SHADE

1: Initialize Population (P), Memory (A') and History (H)
2: **while** Termination Criteria **do**
3: **for** $x_i : 1$ *to* N **do**
4: Calculate parameters F_i and CR_i and by selecting index r_i randomly from $[1 : H]$ using Eq. 2.7 and Eq. 2.8
5: Generate trail vector u_t
6: $x_{r1} \leftarrow rand(P)$
7: $x_{r2} \leftarrow rand(A')$
8: $x^{best} \leftarrow [P, N * p]$
9: $v_i^d = x_i + F_i * (x^{best} - x_i) + F_i * (x_{r1} - x_{r2})$ using Eq. 2.7
10: Generate trail vector $u_i^t = \begin{cases} v_i^d & \text{if } rand[0,1] < CR_i \\ x_i & \text{otherwise} \end{cases}$
11: **if** $f(u_i^t) \leq f(x_i)$ **then**
12: $x_i = u_i^t$
13: Update ((M_{CR}),(M_F)) based on F_i and CR_i
14: **end if**
15: **end for**
16: **end while**

$$CR_i = rand_n(M_{CR,r_i}, 0.1) \tag{2.8}$$

CR_i in algorithm 3 at line 4 is calculated using Eq. 2.8. F_i is selected randomly using Cauchy distributions, and CR_i is selected using uniform (normal) distribution. The upper limit for selection of F_i and CR_i is 0.1.

Instead of selecting the fittest individual as the best, SHADE considers $N * p$ fittest solutions and selects a random solution from them as the best solution x^{best} used at line 8 in algorithm 3. The value of p is calculated using Eq. 2.9.

$$p = rand[p_{min}, 0.2] \tag{2.9}$$

p is random number between p_{min} and 0.2. Value of p_{min} is set to 2/N, which guarantees that at least two fittest solutions are selected. 0.2 is the highest value for p as suggested in Zhang et al. (2020). SHADE maintains an external archive called memory (A') that is used for the mutation process. A candidate vector x_i that is worse than the trial vector u_t is placed in A'. Pseudo code of SHADE is given in algorithm 3.

2.5 Apache Spark and Resilient Distributed Dataset (RDD)-Distributed Computing Framework

The Apache Spark is an open-source, in-memory unified cluster computing framework and an analytic engine that has gained popularity in recent years due to its fast, in-memory data processing capability Zaharia et al. (2016a). Spark is useful for iterative and interactive data processing. In Spark core API, high-level abstraction is provided using RDD to distribute data across the cluster for further processing. The driver node distributes data among available worker nodes and is responsible for the task scheduling and monitoring of worker nodes. Each worker node receives one or more partitions of the RDD. RDD has two types of operations. (i) Transformations (ii) Actions. Transformations perform a task on an existing RDD to create a new RDD. They are in fact logical execution plans that don't materialize the RDD. Actions materialise the execution by creating a new RDD. The actions result in network communication and may become a performance bottleneck if not used carefully.

RDD is an immutable data structure, once created it can't be modified. The only way to update RDD data is to create a new RDD. The creation is always done at the master node and then it is distributed among the worker nodes in the form of logical units called Partitions (Pti). Each RDD is divided into logical partitions and each worker node can contain one or more partitions. Reducing the number of actions on RDD reduces the communication overhead as each action results in data communication between master and worker nodes.

2.6 Island Model

Island model is a population distribution paradigm Corcoran and Wainwright (1994) that divides the initial population into multiple sub-populations (islands). Each island executes an algorithm independently. Depending on the Migration Rate (M_r), islands share solutions with other islands after specified Migration Interval (M_i). Migrated solution(s) replace the worst solutions on each island. The island model enhances algorithms' global search ability by exploring search space in multiple trajectories and improving the execution (convergence) time.

2.7 Evolution Architectures

2.7.1 Cooperative Coevolution (CC)

The LSGO approaches have been categorized Sabar et al. (2019) as Cooperative Co-evolution (CC) and Non-Cooperative Co-evolution (NCC). In CC, a high dimensional problem is divided into low dimensional sub-problems, making them capable of being solved using Evolutionary algorithms. Sub-problems maintain separate sub-populations to ensure diversity Gong et al. (2015). Each sub-problem is solved separately and help in finding the solution of the high dimensional problem. At the end solutions are combined from sub-problems to have a final solution. Figure 2.1 show that CC divides the problem into multiple sub-problem each of which solved independently to find the sub solution.

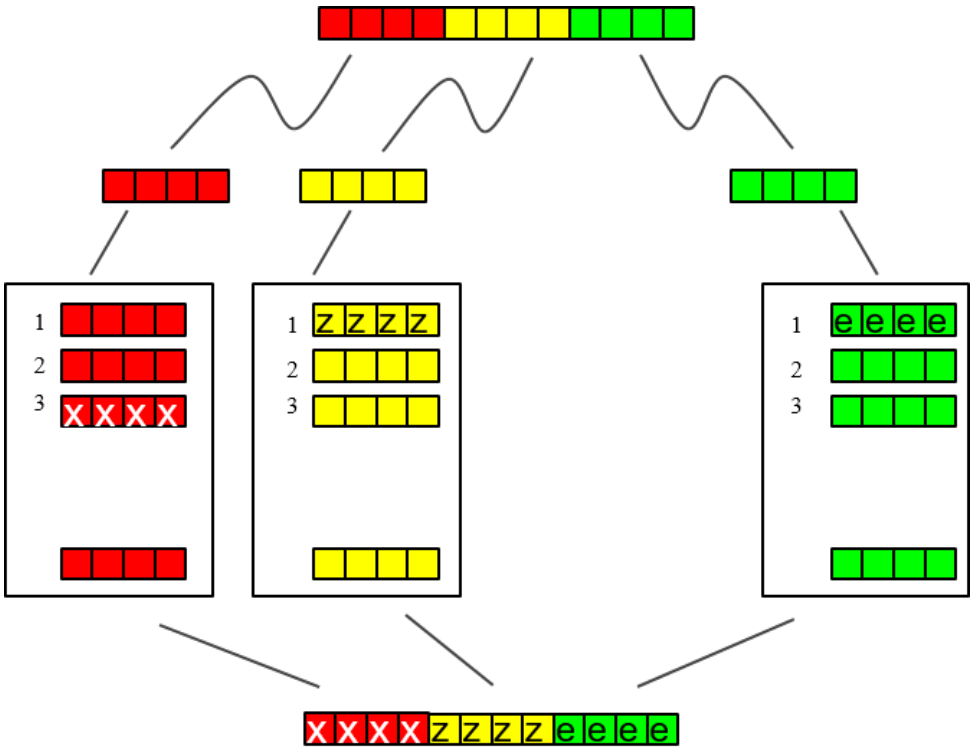


FIGURE 2.1: Cooperative coevolution

2.7.2 Non Cooperative Coevolution (NCC)

Non Cooperative Coevolution divides a population into multiple groups. Each group is working to find the best solutions based on the fitness value and best solutions are transferred to next generation. The problem does not require any grouping strategy

as it divides the population rather than decomposing the problem in multiple sub-problems.

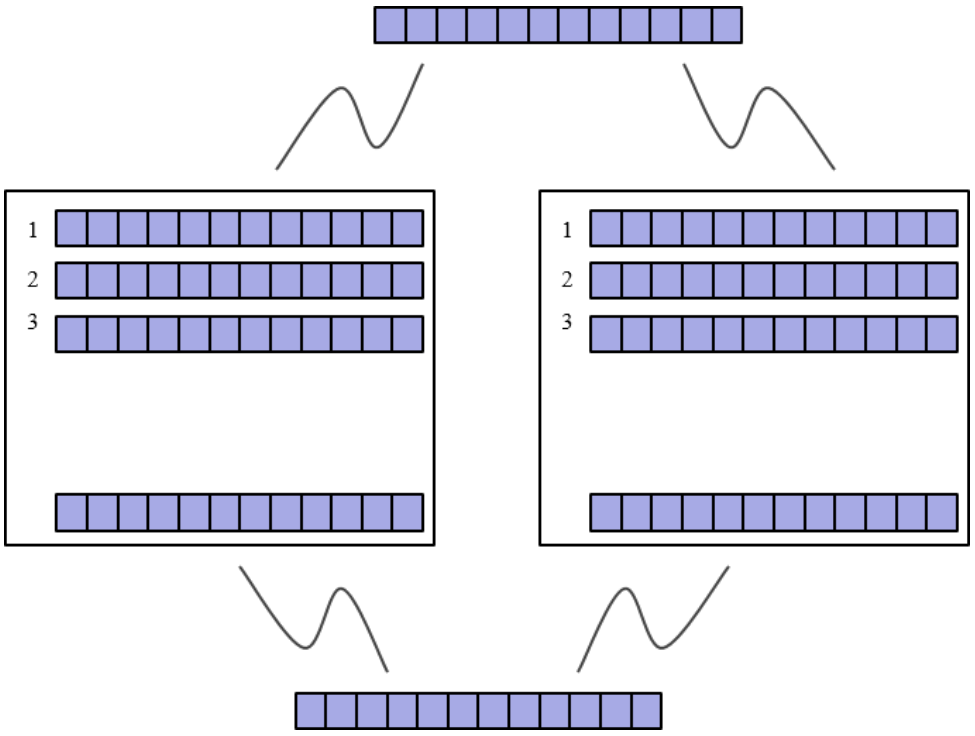


FIGURE 2.2: Non cooperative coevolution

NCC is a suitable choice for many overlapping and complex optimization problems. Figure 2.2 shows that NCC divides the initial population into mulitpe groups each of which is working on finding the best solutions.

Chapter 3

Related Work

This Chapter highlights the importance of evolutionary algorithms, and there real life applications in various fields. Also with the increase in number of dimensions and large scale data problems, importance of exploring parallel and distributed evolutionary approaches is twofold. We have also discussed parallel and distributed evolutionary algorithms that are used with parallel and distributed frameworks to solve such a large scale global optimization problem. Finally to improve the Findability, accessibility, interoperability and reusability (FAIR) of evolutionary algorithms, we have extended the FAIR data principles and map them on making algorithms FAIR. We have also presented a usecase of making GA-FAIR.

3.1 Applications of Evolutionary Algorithms

Swarm Intelligence (SI) (Fister Jr et al., 2013) and evolutionary algorithms (Maier et al., 2019, Yu and Gen, 2010) are the famous bio-inspired algorithms that have gained popularity in the last few years and have been successfully applied to many real world problems (including but not limited to link prediction (Chen and Chen, 2014), clustering (Razzaq et al., 2016), and model tuning & hyper-parameter optimization (Jabeen and Baig, 2010)). SI technique comprises a population of candidate solutions and have shown competitive results for GCP. Gravitational Swarm Intelligence (GSI) was applied to solve GCP (Rebollo-Ruiz and Graña, 2014) and was found that the chromatic number found by GSI was either equivalent or better than Degree of Saturation (Brélaz, 1979), Tabu Search (Li et al., 2003), Simulated Annealing (Johnson et al., 1991), and Ant Colony Optimization (ACO) (Ge et al., 2010). A famous SI algorithm discrete cuckoo optimization was also applied to GCP that achieved a success rate but was unable to produce optimal results for all benchmark graph instances (Mahmoudi and Lotfi, 2015). In another effort a proper coloring of a graph was generated using the collective experience of an artificial ant colony ANTCOL (Costa and Hertz, 1997). Each ant in the colony iteratively builds a coloring with the probability of selecting a

vertex in each step based on the accumulated coloring experience of the ant colony. Empirical results revealed that the number of colors used by ANTCOL is closer to the probabilistically estimated chromatic number for a given graph (Johri and Matula, 1982).

A hybrid algorithm comprising discrete version of PSO and local search was also proposed to solve GCP (Qin et al., 2011). The hybrid technique used a distance measure defined over a discrete solution space to evaluate the difference between two particle positions (i.e. coloring). Once the position of each particle is updated, the Tabucol algorithm (Galinier and Hao, 1999) is used to reduce the number of conflicts. However, this hybrid technique worked on limited instances of discrete solution space. In another approach, ABC algorithm used an order based technique for solving GCP. Food sources were represented as a set of n-component vectors where each component corresponds to a node in the graph. The algorithm decodes each food source as a potential solution to GCP by ordering each component in ascending order and then coloring the graph using the first-fit strategy. The ABC algorithm exhibits a slightly better performance than Largest Degree Ordering (LDO) and Saturation Degree Ordering (SDO). LDO gives preference to the nodes that have the highest number of edges incident to them while SDO prioritizes the nodes based on the number of colors in the neighborhood of a node. However, the authors selected a few graphs instances that were of limited edge density (Tomar et al., 2013). Karim Baiche et. al. enhanced the dragonfly algorithm to ensure the diversity of solutions that potentially avoids local optima (Baiche et al., 2019). The approach was tested on a limited number of DIMACS graph instances.

A Cat Swarm Optimization (CSO) based technique chromaticats was proposed that comprises two phases (i.e. seeking and tracing mode). The seeking mode is designed to identify proper coloring and further optimizing the solution by minimizing the number of used colors. The tracing mode explores new possible colorings by using random mutation. Chromaticats converged in fewer iterations when compared to Cultural Algorithm (CA) and Evolutionary Programming (EP). However, it takes more computational time to complete each iteration (Bacarisas and Yusiong, 2011). Kui Chen and Hitoshi Kanoh used adaptive artificial bee colony algorithm to find better graph coloring compared to different variants of artificial bee colony algorithms (Chen and Kanoh, 2019). Authors tested the effectiveness of the proposed approach on 30 graph instances.

Although these already attempted SI algorithms effectively find good solutions for limited graph instances (e.g. Hybrid algorithm (Qin et al., 2011) performs for limited graph instances of discrete solution space, ABC algorithm (Tomar et al., 2013) performs better better for graph instances with limited edge density, and enhanced binary dragonfly algorithm (Baiche et al., 2019) was also tested for a limited number of DIMACS graph instances) but they suffer with slow convergence rate. Moreover few SI algorithms like chromaticats (Bacarisas and Yusiong, 2011) has high computational cost per iteration.

Taherdangkoo, M. and Taherdangkoo, R. proposed a modified Blind Naked Mole Rat algorithm (i.e. M-BNMR) to deal Loney's Solenoid Optimization (LSO) problem and incorporated the idea of searching areas which have not been examined due to the stochastic nature of the search process. The algorithm exhibits a better convergence rate and effectively avoids the local optima (Taherdangkoo and Taherdangkoo, 2014). M-BNMR revealed better results compared to other SI algorithms (including PSO, ABC, Gaussian ABC, and stem cells algorithm). Faster convergence rate and better computational time of BNMR and M-BNMR inspired us to explore the potential of BNMR algorithm for graph coloring problem. In this paper we have proposed BNMR-Col to solve graph coloring problem that have competitive results for various graph instances.

3.2 Evolutionary Algorithm for solving Large Global Optimization Problems

Evolutionary algorithms have been used for optimization from last few decades and several parallel and distributed models have been proposed. With the advancements in distributed and scalable tools to handle big data, like Apache Hadoop and Apache Spark, researchers proposed implementations using these cloud computing frameworks. Out of evolutionary algorithms Genetic Algorithms (GA) were widely implemented in solving parallel and distributed problems due to its intrinsic nature. Generally, there are three main models to parallelize Genetic Algorithm i.e. global single-population master-slave (global model), single-population fine-grained (grid model), multiple-population coarse-grained (island model) (Luque and Alba, 2011). The **Global Model** works like SeqGA with one population. The master is responsible for handling the population by applying GA operators while slave manages the fitness evaluation of individuals. In **Grid Model**, GA operators are applied within each sub-population

and each individual is assigned to only one sub-population. This helps in improving the diversity. However, this model suffers from the problem of getting stuck in a local optima, and it has high communication overhead due to frequent communication between the sub-populations. **Island Model** (Whitley et al., 1999) uses a large population divided among different sub-populations, called islands, by preserving genetic diversity among them. GA operates on these islands independently with the ability to exchange/migrate some of the individuals. This helps in increasing the diversity of chromosome and avoid getting stuck in a local optima. Mostly, PGA divides a population into multiple sub-populations. Each population independently searches for an optimal solution using stochastic search operators like crossover and mutation. The **SeqGA** uses single large population pool and apply stochastic operators on them. More detail about SeqGA is given in section 3.2. Whitley et. al expected that Island model would outperform SeqGA, because of the diversity of chromosomes and migration of individuals among several islands. However, results revealed that Island model may perform better only if migration among sub-populations is handled carefully. A comparison of Hadoop Map Reduce based implementations of the three main PGA models, global single-population master-slave GAs (global model), single-population fine-grained (grid model), multiple-population coarse-grained (island model) is discussed by F. Ferrucci et.al (Ferrucci et al., 2018). They observed that overhead of Hadoop distributed file system(HDFS) make Global and Grid models less efficient as compared to Island model for parallelizing GA for HDFS access, communication and execution time. Island model performs less HDFS operations, resulting in optimized resource utilization and efficient execution time. However, they reported experimental results of Global, Grid, and Island models on population size of 200 only, with a run of maximum 300 generations on small problems with limited number of dimensions (only up to 18). Their results concluded that distributed frameworks provide efficient support for data distribution, parallel processing, and memory management but they incur significant overhead of communication delays.

Verma et. al used Hadoop's Map Reduce framework to make GA scalable (Verma, 2010). Their experiments were performed on OneMax problem and they addressed scalability and convergence as decreased time per iteration. This was achieved by increasing the number of resources while keeping the problem size fixed. D. Kečo and A. Subasi (Keco and Subasi, 2012) presented PGA using Hadoop map-reduce. Their focus was to improve final solution quality and cloud resource utilization. They

obtained improved performance in term of convergence but there were no improvements in the solution quality. Edgar et.al (Osuna et al., 2017) proposed a diversity based parent selection mechanism for speeding up the multi-objective optimization using Evolutionary Algorithm. The diversity-based parent selection mechanism helps to find the Pareto front faster than the classical approaches. Osuna et al. focused on individuals having high diversity located in poor explored areas of the search space. Gao et al. (Gao and Neumann, 2014) contributed to maximizing the diversity of population in GA, by favouring the solutions whose fitness value is better than a given threshold. They worked on OneMax and Leading One's (Witt, 2006) problems. The results revealed that algorithm efficiently maximizes the diversity of a population. They have presented a theoretical framework and haven't addressed the contribution of diversity in large-scale optimization problems. Both, GA and PGAs are widely used in several applications. Amaro B. et.al (Amaro Junior et al., 2017) applied to parallel biased random-key GA with multiple populations on irregular strip packing problem. In this problem, items of variable length and fixed width need to be placed in a container. For an efficient layout scheme, they have used the collision-free region as a partition method along with a meta-heuristic and a placement algorithm. F. Gronwald et.al (Gronwald et al., 2017) determined the location and amount of pollutant sources in the air by using Backward Parallel Genetic Algorithm (BPGA). A concentration profile was compiled by considering the readings from different points in an area. BPGA utilizes multiple guesses in a generation, and the best one is determined by a fitness function. This best guess is used in the reproduction of the next generation.

3.3 Evolutionary algorithms using Parallel and Distributed Frameworks

A parallel approach to the evolutionary algorithm should make optimal resource utilization and reduce communication overhead to gain an ideal speedup. Several other evolutionary algorithms have been also applied and performed in solving LSGO problems. In this regard, Spark based SparkPSODE (Fan and Lee, 2019) combined the dominating features of both Particle Swarm Optimization (PSO) and Differential Evolution (DE) where individual's position and velocity was updated using PSO while mutation, crossover, and selections operators were adopted from DE. Authors did not address the complex functions with rotations or overlapping factors. Increasing

partitions in Spark results in more computational time. This could be improved by utilizing proper computational power of Spark framework. PGA using Apache Spark framework (Qi et al., 2016) was proposed for the pairwise test suite generation. Parallel operations were used for fitness evaluation and genetic operations. They did not address the large-scale data problems and only focused on test suite size generation. Results were compared with SeqGA on synthetic and real-world datasets. Previously proposed parallel implementations of GA majorly differ in the structure of their population. This structure affects the execution time. The topology of PGA determines the sub-populations model and sharing of their solutions, i.e.(sending and receiving solutions from each other). Following these models in GA, when executed using distributed frameworks like Apache Spark, suffers from network overhead. This network overhead is not affected by the topology being used. Topology defines the structural basis for exchanging solutions between partitions. As there is no uni-cast mechanism available in Apache Spark, therefore, solutions are broadcasted to all the partitions independent of the topology. Although based on the topology, each partition will select the subset of broadcasted solutions for further processing. Although parallelization is intrinsic in the nature of GAs, it comes at the cost of significant network latency overhead for data transfer as sharing is required across the partitions or sub-populations. This overhead disturbs the ideal speed-up that we may achieve by using parallel/distributed techniques. Fahad et al., (Maqbool et al., 2019a) proposed a scalable GA using Apache Spark named (S-GA) for high dimensional problems. They have used the Island model to split the population into sub-populations. Results have shown that an increase in the number of islands increased the number of migrants causing higher network traffic, and increased execution time of S-GA. However, this resulted in faster convergence in terms of iterations. They have used five simple functions for evaluations and did not cover the complex optimization problems. AlJame et al., (AlJame et al., 2020) proposed an Apache Spark implementation of Whale Optimization algorithm (Spark-WOA). Spark-WOA materialized RDD containing evolved solutions after each iteration. Spark-WOA broadcasted the best solution after each iteration to all the partitions. Frequent materialization of RDD is a bottleneck that resulted in network communication and overall increased execution time. Authors have evaluated their technique on four simple benchmark functions for upto seventy (70) iterations only, whereas the number of decision variables was missing. Spark-based DE with grouping topology (SgtDE) was proposed by He et al., (He et al., 2020) to solve LSGO. The

initial population was divided into fifteen sub-populations and three equal groups. Ring topology was used for migration among the same group's islands, while mesh topology was used for communication between the groups. SgtDE used a population size of 300. Although Apache Spark is a cluster computing framework, SgtDE was tested on a single machine only.

SHADE (Tanabe and Fukunaga, 2013b), a variant of DE, maintained history through the memory of Crossover Rate (M_{CR}) and scaling Factor (M_F). Additionally, it preserved a collection of Crossover Rate (CR) and scaling Factor (F) values that have performed well in the past. These values helped producing new (CR, F) pairs by sampling the domain of the parameters adjacent to one of these preserved pairs. SHADE showed a limited evaluation on 100 population size, 30 decision variables, and $3.0 * 10^5$ fitness function evaluations. Also, the execution time for convergence has not been discussed. Oscar and Carlos proposed a Global and Local search using SHADE (GL-SHADE) (Pacheco-Del-Moral and Coello, 2020) that divided the population into two parts. SHADE algorithm was executed on one part for exploration, while eSHADEls was executed on other part for exploitation. After certain iterations, both sub-populations shared their best individuals. The GL-SHADE was evaluated on the CEC-2013 benchmark function suite, and it achieved better results. GL-SHADE was tested with 100 population size and the execution time was not reported.

SHADE-ILS (Molina et al., 2018) combined the exploration power of SHADE with the exploitation power of local search strategies for optimization. SHADE-ILS adaptively adjusted its parameters, and the mutation operator selected the best variable among p best solutions. With a population size of 100, SHADE-ILS obtained good results for CEC-2013 benchmarks functions and was selected as the winner of CEC-2018. An Efficient Recursive Differential Grouping for Large-Scale Continuous Problems (ERDG), a CC-based approach, was proposed by Yang et al. (Yang et al., 2020). Their focus was to reduce the computation cost by examining historical information used for interrelationship examination and reducing the function evaluation. They have used CEC 2013 benchmark functions for evaluation. SHADE-ILS performed better than ERDG on half of the functions. Cooperative Co-evolution with Recursive Differential Grouping (CC-RDG3) (Sun et al., 2019) has comparable results to SHADE-ILS. CC-RDG3 is a CC-based technique where the focus is on division of dimensions.

In summary, most of the examples discussed above do not target the scalability and handle problems with large dimensions, even though some of them used big data

frameworks. For the comparison, we have selected SHADE-ILS, a NCC algorithm, as it is the winner of CEC-2018 (Molina et al., 2018) and GL-SHADE (Pacheco-Del-Moral and Coello, 2020) as it has obtained better results for some of the functions compared to SHADE-ILS using CEC-2013 benchmark function suite (Li et al., 2013). Although CC-RDG3 is the winner of CEC-2019, we have not selected it for comparison as it is a CC based technique that divides the dimensions and decomposes the overlapping problems, whereas in NCC based technique the whole population is decomposed into multiple groups and hence these methods are not directly comparable.

3.4 Challenges and Opportunities of FAIR Algorithms

Researchers usually follow different steps in their research process ranging from problem analysis, literature review, data collection, research software development/usage, experiments, and result analysis. By following FAIR guidelines, researchers can easily reuse published data, research software, and results. It also helps to increase the focus on extending the existing work and achieving their research goals at rapid pace. Recently, efforts have been made by research community to support the FAIR research culture. Recent development in academia (Wilkinson et al., 2016), Life Science (Vogt et al., 2019), FAIR ontologies (Guizzardi, 2020), SmartAPI (Vita et al., 2018), Immune Epitope Database (Spoor et al., 2019) and Health Care (Sinaci et al., 2020), have been made to make data, webAPI, ontologies and biological databases FAIR.

Academia and publishing industry must play an active and vibrant role in making such efforts to publish data, research software, and research manuscripts according to FAIR principles. In this regard few journals (JORS, IPOL, JOSS, eLife, and science direct) have already started to review the research software during the peer review process and publish it along with the article (Gruenpeter et al., 2020). Also Association for Computing Machinery (ACM), has started to review research software, datasets, experiments, and other related files, along with research manuscripts (Krishnamurthi and Vitek, 2015). ACM has introduced the policy badges to review the research articles for results replication i.e. (same results of an article) or results reproduction (results generated independently) mechanism. ACM also encouraged reproducibility, reusability, and replicability in which the experimental setup of software can be used or extended by the same or a different team (Boisvert, 2016). Moreover few recommendations for

executing FAIR practices including (training, education, fundraising, incentives, rewards, recognition, development, and monitoring of policies) were suggested by Hong et.al. (Hong et al., 2020). Also FAIR has far reaching benefits for different domains. In agriculture, Basharat et.al (Ali and Dahlhaus, 2022) discussed role of FAIR data in agriculture industry. They have applied FAIR guidelines to ensure data findabilty and reusabilty of agriculture data, used for decision making and agriculture performance. For making a molecular plant data FAIR a check list is compiled by Reiser et.al (Reiser et al., 2018), it includes placing data at a stable repository, using unique identifiers for genes and its products, by using standard file formats, reproducible computational technique by mentioning (software versions, raw data files, citing data source, parameter settings). Different industries have started their projects of making data FAIR (van Vlijmen et al., 2020). A project on making life science data FAIR FAIRplus [1] is in progress.

FAIR guidelines are independent of tools, technologies, and implementation platforms (Wilkinson et al., 2016). There are few common and some exclusive details for FAIR data and FAIR research software's suggested by Lamprecht et al. (Lamprecht et al., 2020). They have adopted some of the existing FAIR principles where they fit in for research software's and modify/extend the remaining one. The list of recommendations for FAIR research software based on existing FAIR guidelines for data is proposed by Hasselbring et al. (Hasselbring et al., 2020). Software development community also suggested that FAIR research software principles should be separately defined (Gruenpeter et al., 2020). There are different challenges (i.e. software documentation, accessibility, licensing issues, software dependencies, environment, quality control, and software sustainability) in making research software findable and reusable (Org et al., 2020). All these efforts are made in recent years for making scientific data, software, and related objects FAIR. Different digital artifact repositories (i.e. Github[2], GitLab[3], Zenodo[4],SourceForge[5],and Bitbucket[6]) are used to store and publish the data and software. The findability of data and research software is enforced by the relevant conference, workshop, or journal at the time of publishing of manuscript. Similarly,

[1] https://fairplus-project.eu/
[2] https://guides.github.com/features/pages/,Accessed Oct 8,2021
[3] https://gitlab.com/gitlab-org/gitlab, Accessed Oct 8,2021
[4] https://zenodo.org/,Accessed Oct 8,2021
[5] https://sourceforge.net/,Accessed Oct 8,2021
[6] https://bitbucket.org/,Accessed Oct 8,2021

the accessibility of data and research software has its own challenges. Data and software usage license and copyrights details should be clearly stated and permission of access should be granted to the research community where admissible. Reproducability of research software is also a challenge and this is due to the lack of availability of software code, its Persistent IDs, and reproducing the complete software environment as highlighted by Alliez et al. (Alliez et al., 2019). To improve the reusability by the research community for the photovoltaic time series data, a set of recommendation's were suggested by Arafath et.al. (Nihar et al., 2021). It includes clearly defined dataset, accessibility and availability of metadata in human and machine readable format i.e JSON-LD.

Another challenge related to research software is not a well-defined attribution mechanism for the developers by the research community. This results in less focus on quality research software but on research manuscripts (Lamprecht et al., 2020). Current citation mechanisms, impact factor policies, and promotion/hiring in universities are research publication centric. A preliminary work on software citation principles was highlighted by Smith et al. (Smith et al., 2016). They have identified, importance, credit, attribution, unique identification, persistence, accessibility and specificity as the major software citation principals. Format of citing software, metadata of software for citation, criteria for peer review of software, and acceptance of software as digital product were the few challenges related to software citations that were highlighted by Niemeyer et al. (Niemeyer et al., 2016). The research community, journals, conferences, workshops, and research and project funding agencies have to initiate such reforms that help in making research data, software, and related research objects FAIR. All the relevant stake holders have to develop/encourage practices, like citation incentives and scholarly attribution for research software developers / data analysts / researchers for following FAIR principles.

To the best of our knowledge we are unable to find any application of FAIR guidelines to algorithms, so in this article we have extended FAIR data guidelines to develop FAIR-Algorithms. Moreover to justify the applicability of FAIR-Algorithms we have presented a usecase (i.e FAIR-GA) and validated our FAIR- Algorithms proposed guidelines.

Chapter 4

Graph Coloring using Blind Naked Mole Rat Algorithm

Evolutionary algorithms have gained popularity in the last few years due to their capability to solve complex optimization problems efficiently and effectively. Researchers have proposed numerous applications of these algorithms in resource optimization (Deng et al., 2022), economic dispatch (Qu et al., 2018), large scale global optimization (Maqbool et al., 2021), parameter estimation (Gao et al., 2021), and engineering problems (Slowik and Kwasnicka, 2020). These population-based algorithms are inspired by the mechanism of biological evolution and maintain a set of candidate solutions. Crossover, mutation, and reproduction are central to many of these algorithms. Genetic Algorithm (GA) is a widely used population-based evolutionary algorithm that also has its applications in graph problems. The problems include graph partitioning (Bui and Moon, 1996, Mishra et al., 2014, Talbi and Bessiere, 1991), steiner graph problem (Esbensen, 1995, Rojas and Meza, 2015), and Graph Coloring Problem (GCP) (Zhang et al., 2014).

Let $G = (V, E)$ be a simple graph, where V and E represent the set of vertices and the set of edges respectively. GCP is an assignment of minimum colors (labels) to the vertices of a graph such that the adjacent vertices do not have the same color. Solving GCP with the minimum possible colors makes it a hard problem. GCP has been mapped to numerous application areas including resource scheduling (De, 2022), Parking Algorithm (Agizza et al., 2022), wireless network channel allocation problem (Marappan, 2021), video synopsis problem (He et al., 2017), and multiple stacks traveling salesman person problem (Demange et al., 2015). A specialized case of GCP is k-GCP. Let k be the number of colors assigned to vertices of a graph then the conflict optimization version of k-GCP minimizes the number of conflicts in the graph. The optimal solution of k-GCP is a coloring configuration with zero conflicts and is termed as proper k-coloring.

Blind Naked Mole-Rats (BNMR) algorithm (Mohammad and Mohammad, 2012) is a meta-heuristic that is inspired by the social behavior and population rich colony culture of blind naked mole rats. BNMR uses both exploration and exploitation to search the optimal solution in the search space. Exploration helps to explore the far away areas in search space while exploitation helps to converge to a solution by exploring the neighborhood of a fittest solution. Attenuation coefficient controls the rate of exploitation and decreases with number of iterations. It has been observed empirically that BNMR prevents from trapping in local optima and outperforms several competing algorithms (such as particle swarm optimization, artifical bee colony , genetic algorithm, stem cells algorithm, and simulated annealing) when applied to numerical function optimization, data clustering (Taherdangkoo et al., 2013), traveling salesman person problem (Yildirim et al., 2016), and electromagnetic designing (Taherdangkoo, 2021).

In this chapter, we have addressed the following research question.

> RQ1. How complex optimization problems can be solved using evolutionary algorithms?

Our contributions in this chapter are as follows.

- We have modified BNMR algorithm and proposed BNMR-Col to solve k-GCP.
- We have tested BNMR-Col on DIMACS graphs instances (Johnson and Trick, 1996, Zhang et al., 2014).
- We have tested BNMR-Col with ANTCOL (Ferrández et al., 2011) and Chromaticats (Bacarisas and Yusiong, 2011).

This chapter is based on the following publication.

- **Fahad Maqbool**, Muhammad Fahad, Muhammad Ilyas, and Hajira Jabeen. "Graph Coloring using Evolutionary Computation: A case study of Blind Naked Mole Rat Algorithm" in Expert Systems, pp. 1-17. Wiley, 2023.

This chapter is organized as follows. Section 4.1 presents the BNMR algorithm and its major components. In section 4.2, we have explained BNMR-Col algorithm. Section 4.3 presents the results of BNMR-Col, ANTCOL, Chromaticats on popular graph instances. Finally, we have summarized this chapter and its possible extensions in section 4.4.

4.1 Blind Naked Mole Rat (BNMR)

BNMR is a meta-heuristic based on social behavior of mole-rats. They searched the food and protects the colony against the attacks. BNMR worked at random on the complete problem space. Number of mole rats(i.e. population) are twice the number of food sources. Whereas each food source represents a response for problem space i.e. target to be found by mole-rats. BNMR has focused to overcome the issues faced by the conventional optimization algorithms like low convergence rate and getting stuck in local optima. Algorithm 4 presents the optimization process of BNMR algorithm Qin et al. (2011).

Algorithm 4 Blind Naked Mole Rat (BNMR) Algorithm

Require:
Define Objective function $f(X_i)$
Define constants $\alpha,\ \beta,\ \zeta$, and maximum number of iterations T
Initialize population $X = \{X_i\}$, removal rate b_i where $i = 1 \cdots N$ and $X_i = [x_1^i \cdots x_d^i]$
▷ d represents the dimensions of the search space
Ensure: X_{best} ▷ Optimal or Near Optimal Solution
Sort X_i such that $f(X_i) \geq f(X_{i+1})$
$t \leftarrow 1$ ▷ Reset iteration counter
while $t \leq T$ **do** ▷ While stopping criteria not met i.e. maximum iterations
  $X_{new} \leftarrow [x_j^{new}] \leftarrow x_j^{Min} + \beta(x_j^{Max} - x_j^{Min})$
  if $rand \leq A_i$ **then** ▷ Exploitation phase
    Select a solution X_{best} among the best solutions using the probability $p_i = \frac{f(X_i)=FS_i*R_i}{\sum_{k=1}^{N} f(X_k)}$
    Find X_{local} around X_{best}
  end if
  Generate a new solution X_{rand} using random search ▷ Exploration phase
  if $rand \leq B_i \quad \& \quad f(X_i) < f(X_{opt})$ **then**
    Accept and save the new solutions in the memory of mole-rat
  end if
end while

Let M_i be a mole rat with a food source X_i and removal rate b_i, where $i = 1 \cdots m$. M_i performs both exploration and exploitation in each iteration of BNMR. In exploitation M_i select another mole-rat M_j with food source X_j (based on relative fitness probability). M_i generates a neighboring food source X_j' for M_j. In case of exploration, BNMR considers M_i a potential invader. Therefore, it generates a new random food source X_i'. If $f(X_i')$ is better than $f(X_i)$ then M_i is mark as invader and replaces its food source X_i with X_i'.

TABLE 4.1: Average number of conflicts using 10 runs of Random-k and k-ColorWalk for Leighton graphs with known chromatic number (△)

Instance	△	Number of conflicts	
		Random-k	k-ColorWalk
le450_5a	5	1144	815
le450_5b	5	1127	838
le450_5c	5	1988	1619
le450_5d	5	1927	1583
le450_15a	15	548	156
le450_15b	15	543	158
le450_15c	15	1124	698
le450_15d	15	1106	685
le450_25a	25	334	31
le450_25b	25	328	27
le450_25c	25	703	231
le450_25d	25	693	236

4.1.1 Population (*P*)

Population $P = M_1 \cdots M_N$ consists of N mole-rats with no division of labor mole-rats as suggested by M. Taherdangkoo et al. (Taherdangkoo et al., 2013). Each mole-rat $M_j = X_j, T_j, s_j, b_j$ consist of an assigned food source X_j, a restricted list of vertices T_j, a stagnation counter s_j, and removal rate b_j. Each food source X_j represents a potential solution to the problem from the solution space S. Each food source X_j is initialized by a randomized algorithm, k-ColorWalk as described in Algorithm 5. The algorithm uses the structure of the graph to initially build k color classes without any conflict. After that remaining uncolored vertices, if any, are assigned random color c_r such that $c_r \in \{c_1 \cdots c_k\}$

Another approach Random-k is used to create the initial solutions by assigning assignment colors to the vertices. Random-k results in higher number of conflicts as compared to k-ColorWalk. Table 1 displays the average number of conflicts using 10 runs of Random-k and k-ColorWalk for Leighton graphs. Generating k-colorings for the given graph using k-ColorWalk helps to keep large-sized color classes and generates the coloring with fewer conflicts compared to Random-k. This strategy helps to improve the convergence speed. However, number of conflicts in initial coloring does not effect the final solution (Lü and Hao, 2010).

4.1.2 Attenuation Coefficient

Attenuation coefficient A is a variable that controls the probability of a mole-rat to select one of the best solutions and improve it. Attenuation rate starts from a higher

Algorithm 5 k-ColorWalk for Population Initialization

Require:

$Graph\ G \leftarrow (V, E)$ ▷ Graph with vertex set V and edge set E
$Initialize\ k$ ▷ Maximum number of colors
$c_i\ where\ i\ \leftarrow 0 \cdots k$ ▷ Number of colors currently assigned
$C_j \leftarrow 0\ where\ j = \{1 \cdots |V|\}$ ▷ Color of vertex u
$U \leftarrow V$ ▷ Set of vertices to visit
$S \leftarrow \phi$ ▷ Set of visited vertices
$T \leftarrow \phi$ ▷ Set of neighbor vertices that still needs to be explore
$N(u) \leftarrow \{v\}\ such\ that\ \forall v\ there\ exists\ (u, v)$ ▷ Neighborhood of a vertex u

Ensure:

while $i \leq k$ **do**
  $i \leftarrow i + 1$
  while $U \neq \phi$ **do**
    $Randomly\ select\ u\ \in\ U$
    if $c_i \notin C_{N(u)}$ **then**
      $C_u = c_i$
    end if
    $T \leftarrow N(u)$
    $S \leftarrow S \cup u$
    $U \leftarrow U - u$
    while $T \neq \phi$ **do**
      $Randomly\ select\ t\ \in T$
      if $c_i \notin C_{N(t)}$ **then**
        $C_t = c_i$
      end if
      $S \leftarrow S \cup t$
      $T \leftarrow T \cup N(t)$
      $T \leftarrow T - S$
    end while
  end while
end while

initial value and gradually settles down to a lower probability value. It is defined as $A = \alpha * e^{\frac{-\alpha i}{z}}$, where α is a user-specified value that controls the rate of decrement, i is the current iteration and z is half of the maximum iteration value set by the user.

4.1.3 Conflict Resolution

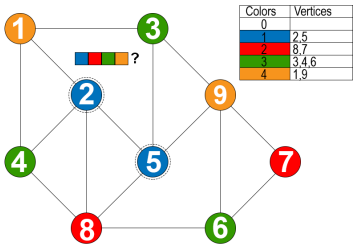
Conflict resolution in GCP checks the adjacent vertices having the same color. Various conflict resolution strategies have been proposed. We have used SDO (Brélaz, 1979) for conflict resolution due to it's simplicity and effectiveness. In SDO, saturation degree of a vertex is defined as the number of it's adjacent differently colored vertices. Let C be the set of all conflicting vertices in the graph then set of available conflicting vertices ($C_A = C - l$) is obtained by removing all tabu vertices(l) from C. Tabu vertices

are the vertices that have been visited recently and belongs to set C. To generate a neighboring solution, a vertex $v_i \in C_A$ is randomly selected. Using probability β, v_i is assigned either lowest color ($c = 1$) or a random color cr such that $2 \leq cr \leq k$. Then we uncolor the vertices adjacent to v_i. After this, we iteratively select a vertex $vi \in C_A$ with largest saturation degree and assign it either a lowest legal color or a random color in the range $[2 \cdots k]$. After coloring all the vertices vi, these are added in the restricted list. The length of restricted list l is adjusted dynamically using $l = \sum_{c=1}^{k} h(v_i^c) + r$. Here r is a random number in the range [1-10] and $h(v_i^c)$ is the number of conflicts in the food source X_i given by equation 4.2.

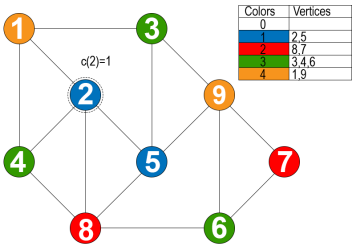
Figure 4.1 shows the step by step working of One k-SDO move using a k-coloring graph with two conflicting vertices v_2 and v_5 as shown in figure 4.1(a). Figure 4.1(b) shows the first step of One k-SDO move which is to select a conflicting vertex and assign it the lowest color. In the next step we uncolor all the vertices in the neighborhood of selected conflicting vertex as shown in figure 4.1(c). In the next step (figure 4.1(d)), we iteratively color all the vertices in the neighborhood of the selected conflicting vertex v2 using Saturation Degree Ordering (SDO) with k as a fixed bound not to exceed while assigning a color value. Then we select uncolored vertex with maximum saturation degree (i.e. v_5) and assign it lowest legal color (c_1) as shown in figure 4.1(e). In a similar fashion all the remaining uncolored vertices are assigned colors using saturation degree as the order of priority (figure 4.1(f) - 4.1(i)). If no legal color is available for assignment to a given vertex, any random color is assigned within 1 to k value. In the final step we assign lowest legal color to all uncolored vertices as shown in figure 4.1(j).

4.1.4 Exploitation

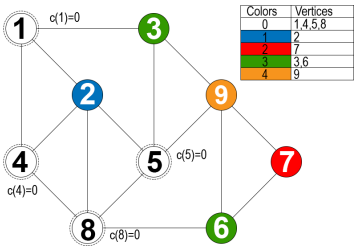
Exploitation is the process of improving assigned food source towards a better solution by a given mole-rat. It is further divided into two phases, (i.e. Self-Exploitation (SE) and Probabilistic-Exploitation (PE)). In SE, a mole-rat M_i applies the local search procedure on its assigned food source X_i to generate a neighboring solution. While in PE, a mole-rat M_i generates neighboring solutions for other mole-rats M_j using tabu search (Hertz and de Werra, 1987). Both of these phases are briefly discussed in the following sub-sections.



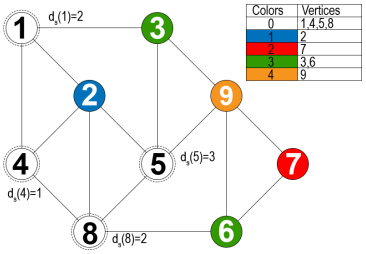
(a) k-coloring with two conflicting vertices



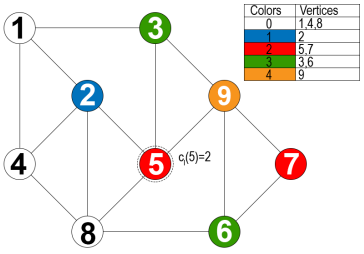
(b) Step I of "One k-SDO" move



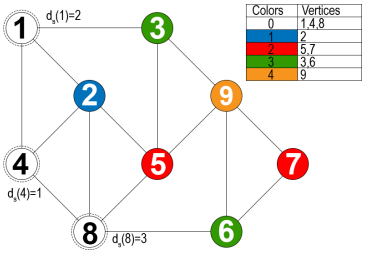
(c) Step II of "One k-SDO" move



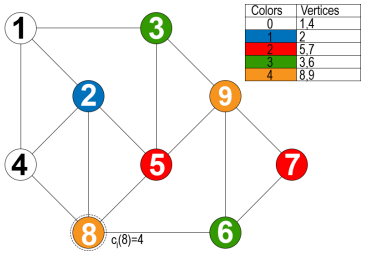
(d) Step III of "One k-SDO" move



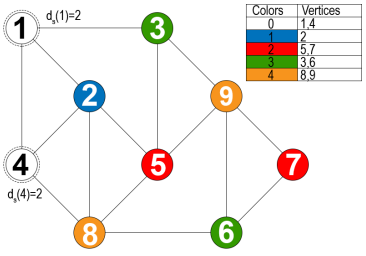
(e) Step IV of "One k-SDO" move



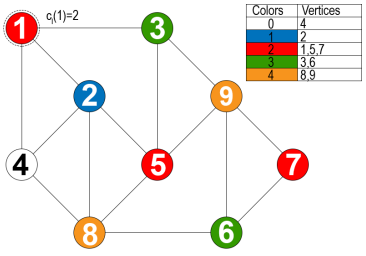
(f) Step V of "One k-SDO" move



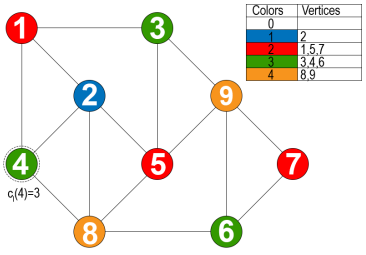
(g) Step VI of "One k-SDO" move



(h) Step VII of "One k-SDO" move



(i) Step VIII of "One k-SDO" move



(j) Step IX of "One k-SDO" move

FIGURE 4.1: Conflict resolution steps through One k-SDO move

4.1.4.1 Self-Exploitation

In self-exploitation phase, a mole-rat M_j generates a neighborhood of its assigned food source X_j using SDO technique. A neighboring solution of X_j is generated by selecting a conflicting vertex $v_i \in C$ having maximum conflict degree $d_c(v_i)$. Conflict degree of a vertex v_i is the number of vertices adjacent to v_i, which have the same color label as of v_i. After generating Y Neighboring Solutions $\{X_{j^1}, X_{j^2} \cdots X_{j^Y}\}$ for X_j, a neighboring

solution $X_{j^{best}}$ with the least number of conflicts (i.e. $\sum_{c=1}^{k} h(v^c_{j^{best}}) > \sum_{c=1}^{k} g(v^c_{jy})$ where $best \neq n$ and $y = 1 \cdots Y$) compared to X_j is selected to replace X_j.

Figure 4.5 shows the working of self-exploitation using an example. Let X_j be the initial coloring of the mole-rat M_j as shown in figure 4.2. A set of conflicting vertices $C = 1, 2, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 18, 19$ represents all the distinct vertices that have conflicting colors in adjacent vertices. A set of restricted list T consist of all those vertices that are forbidden to be selected and is empty at the start. Set of available conflicting vertices C_A is obtained by removing all vertices in T from C (i.e. $C_A = C - T$). A neighbor move is generated by selecting a vertex from C_A and performing one k-SDO move to that vertex. Let us consider that the first neighbor move is generated by selecting the vertex with the largest conflict degree. All the other moves are generated by selecting a vertex randomly from CA. Two moves N1 using vertex v11 and N2 using vertex v4 have been generated in Figure 4.2. Here v11 is the vertex with largest conflict degree and v4 is a randomly selected vertex. N1 reduces the conflicts to 7 and N2 results in 8 conflicts. Subsequently, the best move among the neighbor moves is selected. It is clear that N1 has better fitness than N2. Also, the fitness of N1 is better than initial coloring (i.e. 13 conflicts) of mole-rat M_j. Thus, current coloring X_j of mole-rat is replaced with N1. Conflicting vertex v11 selected to generate N1 is considered as tabu and added in T. This completes one single Local Move step for a mole-rat in an iteration. In the second iteration, the same procedure is repeated and two neighbor moves are generated using modified coloring of mole-rat. Figure 4.3 shows the modified coloring X_j and two new neighbor moves N1 and N2. Note that v11 is not available as a conflicting vertex as it is in T. New neighbor moves N1 and N2 are generated using v6 and v13. Both N1 and N2 produce coloring with the same number of conflicts and both are better than current coloring of the mole-rat. Therefore, any one of them can be selected. Let N2 be the coloring which updates the mole-rat. Figure 4.4 shows the next iteration of local move that results in elimination of all the conflicts.

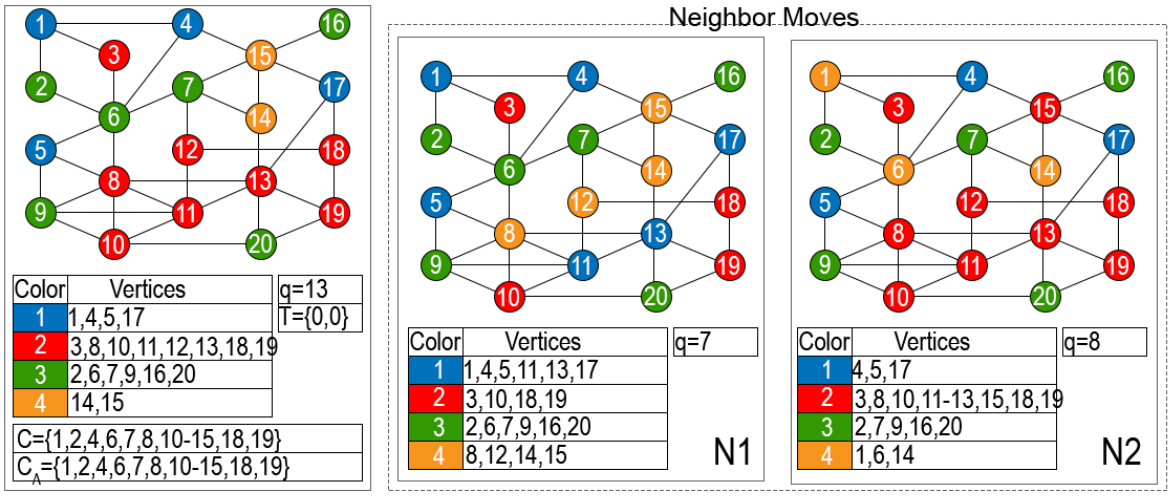


FIGURE 4.2: Step I - Neighborhood move

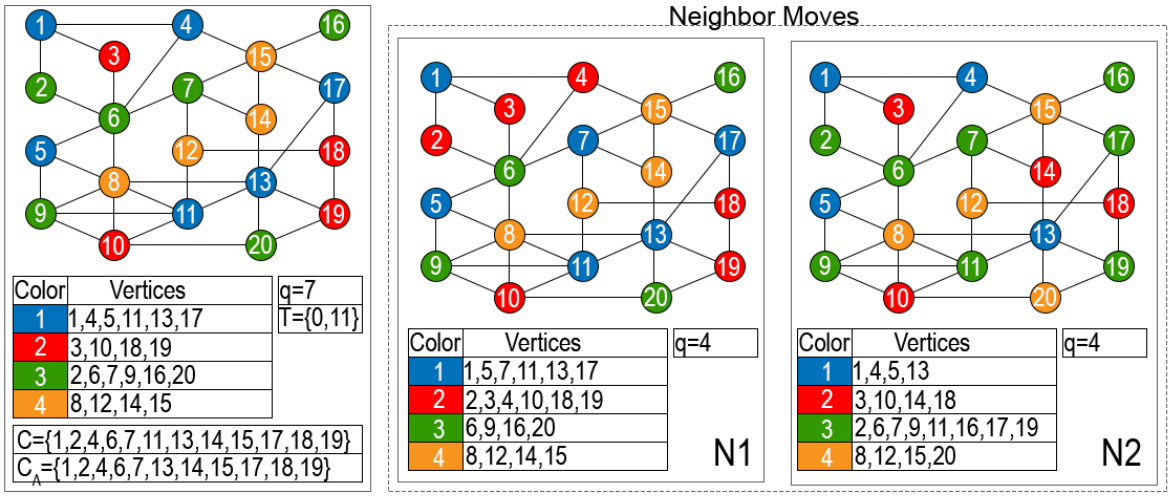


FIGURE 4.3: Step II - Neighborhood move

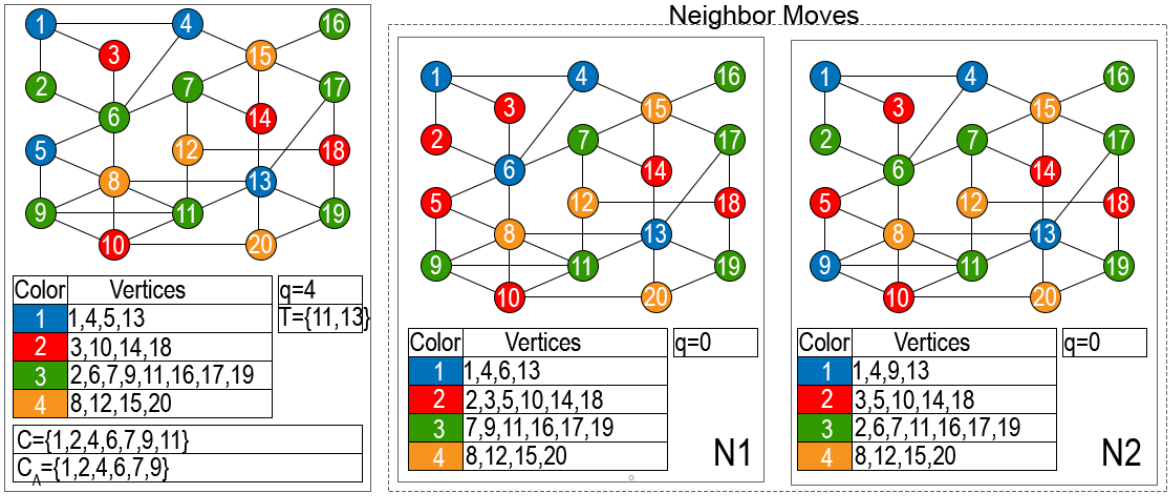


FIGURE 4.4: Step III - Neighborhood move

FIGURE 4.5: Self-exploitation of a given food source

4.1.4.2 Probabilistic-Exploitation

In this phase, a mole-rat M_j selects another target mole-rat M_t from the population. Selection is based on probability which is highest for the fittest mole-rat. The probability ρ for each mole-rat in the population is defined as $\rho((M_t)) = \frac{f(M_t)}{\sum_{u=1}^{N} f(M_u)}$.
After selection of M_t, tabu search (Hertz and de Werra, 1987) is applied to food source X_t for a limited number of iterations. Combining self-exploitation with tabu search helps M_t to avoid getting stuck in local optima. Attenuation coefficient A controls the probability of a mole-rat to apply tabu search on another mole-rat's food source.

4.1.5 Exploration

Crossover operator is employed to build a new food source X_{new} utilizing the knowledge of best mole-rats in the colony. We have used Adaptive Multi-Parent Crossover (AMPaX) operator (Chowdhury et al., 2013) with a modified class evaluation function. AMPaX considers only the cardinality of the color class. However, we extend AMPaX to count the no of conflicts also in order to determine the quality of the color class. If a mole-rat M_i has below average fitness in the colony and is unable to improve it's fitness for λ number of iterations, then the food source Xi is replaced with $Xnew$. In order to create X_{new}, we select m mole-rats as parents (i.e. $M_{selected-1}, M_{selected-2} \cdots M_{selected-m}$), such that $m \geq 2$, and the selected mole-rats are the fittest mole-rats of the population. We evaluate each color class c in X_j (where $j = 1 \cdots m$) using equation 4.1.

$$g(V_j^c) = |V_j^c| \times h(V_j^c) \tag{4.1}$$

Where $|V_j^c|$ is the number of vertices in X_j that belongs to color class c, and $h(V_j^c)$ is a function that evaluates the fitness of color class V_j^c based on $\complement_j^c$ number of conflicting vertices in V_j^c as elaborated in equation 4.2 and equation 4.3.

$$h(V_j^c) = \begin{cases} 1 & if\ \complement_j^c = 0 \\ 0.667 & if \complement_j^c = 1 \\ \frac{1}{\complement_j^c} & otherwise \end{cases} \tag{4.2}$$

$$\complement_j^c = \sum_{i=1}^{E} \left| \forall (v_a, v_b) \in E \mid v_a, v_b \in V_j^c \right| \tag{4.3}$$

After evaluating the color classes for all the selected parent mole-rats, a mole-rat $M_a \in M_{selected-1}, M_{selected-2} \cdots M_{selected-m}$ is selected that has the best color class $c*$ i.e. $h(V_a^{c*}) \geq h(V_j^c)$, where $j \neq a$ and $j = 1 \cdots m$. All the vertices V_a^{c*} of the best-selected color class are then transferred to X_new as its first color class. A restricted list with size $m/2$ is maintained to keep diversity in selection. Mole-rat M_a is added in the restricted list and vertices in V_a^c* are then removed from all the selected parent mole-rats. In next step, another parent mole-rat $M_b \in M_{selected-1}, M_{selected-2} \cdots M_{selected-m}$ with best color class V_b^{c**} among all the parent mole-rats is selected. Note that M_b should not be the part of restricted list. All the vertices in V_b^{c**} are then transferred to X_{new} as its second color class. In similar manner, k number of color classes are transferred to X_{new},. Any uncolored vertices in X_{new} are then assigned random color c_r such that $1 \leq c_r \leq k$. The new food source X_{new} finally replaces X_i.

We have explained the concept of exploration using an example as shown in figure 4.10. we have selected 4 mole-rats $M_1, M_2, M_3, and M_4$ to build a new graph coloring X_{new}. Three color classes has been transferred from selected parent mole-rats to X_{new} as $k = 3$. In the first step, the best color class is determined among all the mole-rats. Consider mole-rat M_1 and its color classes C_1, C_2 and C_3. Quality of a color class is determined using equation 4.1. There are three vertices in C_1 and no conflict in the class, that results in the quality of color class $g(C_1) = 3$. Similarly, in C_2, there are two vertices and no conflict. Quality of color class C_2 is given as $h(C_2) = 2$. Lastly, quality of color class C_3 is equal to $h(C_3) = 3.335$ as there are five vertices and one conflict between v_3 and v_6. Therefore, the best color class of M_1 is C_3. In the similar fashion, we determine the best color classes of all mole-rats. The best color classes of M_2 is C_1 with $h(C_1) = 4$, M_3 is C_1 with $h(C_1) = 3$, and M_4 is C_2 with $g(C_2) = 2.668$. The best color class among mole-rats $M_1 \cdots M_4$ is C_1 of M_2. This color class is then transferred to X_{new} as its first color class. M_2 is considered as tabu for the next two iterations. All the vertices of C_1 (v_2, v_3, v_9, v_{10}) are uncolored for mole-rats $M_1 \cdots M_4$. This completes first step of exploration, with a single class transferred to X_{new}. Figure 4.7 shows the first color class of X_{new} after transferring C_1 of M_2.

In following steps C_3 of M_1 and C_2 of M_4 is transferred to X_{new} as shown in figure 4.8 and figure 4.9 respectively. After transferring all three classes to X_{new}, one vertex v_4 still remains uncolored. This vertex is assigned a random color within bounds of $1 \cdots k$. Figure 4.11 shows the final coloring solution.

4.1.6 Invader Removal

Colony defense mechanism (Taherdangkoo et al., 2013) considers low-cost data points as invaders. They are replaced by randomly selected data points from search space, thus causing mutation. In BNMR-Col vertices with the highest conflict degree are considered as invaders. In mutation we select a conflicting vertex v_i (with color c_s) that has maximum conflict degree. We replace color of v_i with a random color c_r such that $1 \leq c_r \leq k$ and $c_r \neq c_s$.

4.1.7 Removal Rate

Removal rate b_i of a mole-rat M_i defines the number of times mutation is performed. Initially, the removal rate is determined as in equation 4.4.

$$b_j = 1 + (r \times |V|) \tag{4.4}$$

$$b_j = 1 + (\zeta \times b_j) \tag{4.5}$$

Where r is a random number in the range $0.1 - 0.25$ and $|V|$ is the number of vertices in the graph. In each iteration, the removal rate is updated as mentioned in equation 4.5. ζ specifies the rate of decrement or increment in the removal rate and can vary in the range of $0 - 1$.

4.1.8 Fitness Function

The fitness of each mole-rat $f(M_i)$ is determined on the basis of the number of conflicts in its assigned food source X_i. We use equation 4.6 to find the fitness of each mole-rat.

$$f(M_i) = \sum_{c=1}^{k} h(V_i^c) \tag{4.6}$$

4.2 BNMR-Col

Natural inspiration for the BNMR algorithm comes from the social behavior of blind naked mole-rats in a colony. Two key aspects of mole-rat colony behavior depicted in the BNMR meta-heuristic are digging of the tunnels and colony defense. Mole-rats dig tunnels in search of large tubers to feed on. This behavior is modeled as a stochastic

search for food sources. The colony defense mechanism is formulated as a mutation process to enforce exploration in a search space.

Each mole-rat in a colony independently searches for a configuration (k-coloring) with fewer conflicts. The queen mole-rat acts as a space to store the best coloring as well as the fitness values of all the mole-rats as shown in figure 4.12. This enables queen mole-rat to facilitate other mole-rats to coordinate with each other. BNMR-Col consists of four phases: (i) Population Initialization, (ii) Exploitation of Food Sources, (iii) Mutation, and (iv) Crossover. Detailed working of BNMR-Col has been explained in algorithm 6.

Algorithm 6 Blind Naked Mole Rat Coloring (BNMR-Col) Algorithm

Require:
$G \leftarrow (V, E)$ ▷ V is the set of vertices, E is the set of edges
Initialize k, α, β, ζ ▷ Number of colors, decrement factor for A, mutation probability, mole-rat removal rate's decrement factor
Initialize $X_i = \{C_1^i, C_1^i \cdots C_k^i\}$ ▷ i^{th} Solution with vertices using k colors
Define M_i with food source X_i ▷ Each mole rat initailed with a k coloring solution
Initialize b_i and s_i ▷ Initialize removal rate and stagnation counter of M_i
Define $t = 0$, T, A ▷ Define current iteration, maximum iteration, and attenuation variable
Ensure: $X_{best} = \{C_1^{best}, C_2^{best} \cdots C_k^{best}\}$ ▷ Optimal or Near Optimal Solution

while $t \leq T$ **do** ▷ While stopping criteria not met i.e. maximum iterations
  for M_i **do** ▷ where $i = 1 \cdots n$
    if $rand \leq A_i$ **then** ▷ Exploitation phase
      M_i performs probabilistic exploitation on M_j
    end if
    M_i performs self exploitation on X_i
    if $rand < \beta$ and $f(M_i) < f(M_{best})$ **then**
      Assign random colors to b_i vertices
    end if
    $b_i \leftarrow 1 + (\zeta + b_i)$
    if $s_i > s_l$ **then**
      if $f(M_i) < \frac{\sum_{j=1}^{n} f(M_j)}{n}$ **then**
        Apply invader removal mechanism to M_i
        $s_j \leftarrow 0$
      end if
    end if
    Determine and update best mole rat M_{best}
    $A = \alpha \; exp^{\frac{-\alpha t}{0.5T}}$
    $t \leftarrow t + 1$
  end for
end while

4.3 RESULTS AND DISCUSSION

We have implemented BNMR-Col in C#. Moreover, we have also implemented BNMR-Col, ANTCOL and Chromaticats for comparison. We have conducted experiments on a machine having 3.4 GHz Intel Core i3 4130 processor and 16 GB RAM, using 64-bit Windows 10 Operating System. We have selected graph instances for experiments from a more recent collection of DIMACS instances by "COLOR02/03/04:

TABLE 4.2: Characteristics of graph instances from different graph families.

Mycielski Graphs (MYC)	These triangle free graphs are based on Mycielski transformation.
Register Graphs (REG)	Graph files are based on problem of register allocation for variables in real codes.
Book Graphs (SGB)	Each vertex in a graph represents a fictional character and is connected by an edge with another character.
Game Graph (SGB)	Each vertex represents a college football team and two teams are connected by an edge if they play a match.
Queen Graphs (SGB)	Each vertex represents a square on chessboard and has an edge with all other vertices in same row, column, or diagonal.
Miles Graphs (SGB)	Each vertex represents a USA city. Two vertices are connected by an edge if they are close enough, based on road mileage.
Insertion Graphs (CAR)	These graphs are generalization of Mycielski graphs with inserted vertices to increase graph size but not density.
Random Graphs (DSJ)	These are randomly generated graphs by David Johnson.
Multi Coloring Graphs	These are quasi-random graphs intended for multi coloring problem and used for graph coloring by ignoring node weights.

Graph Coloring and its Generations" symposium Bensouyad et al. (2015). Graph instances are grouped together into various families based on their sources. A brief description of graph families are discussed in table 4.2. Table 4.3 shows the characteristics of various graph instances from different graph families. Where $|V|$ denotes the number of vertices in graph, $|E|$ denotes the number of edges in graph, $\triangle$ gives the degree of graph, and lowest k is the known minimum value of colors for a graph with which it can be properly colored.

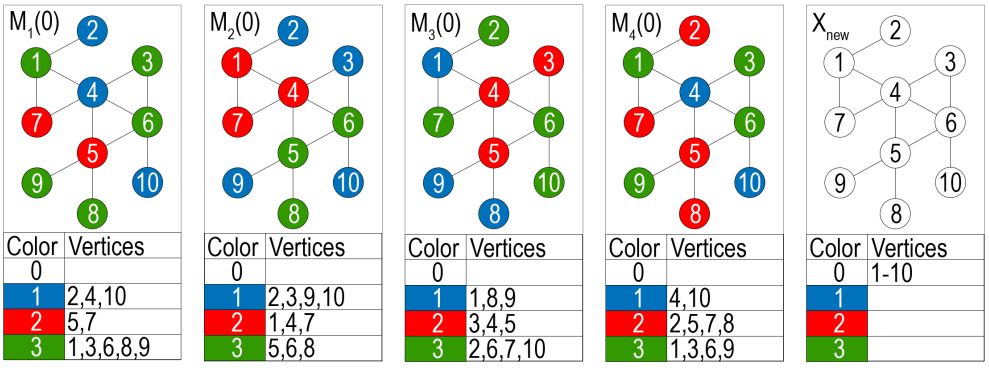


FIGURE 4.6: Selected mole-rats for exploration phase

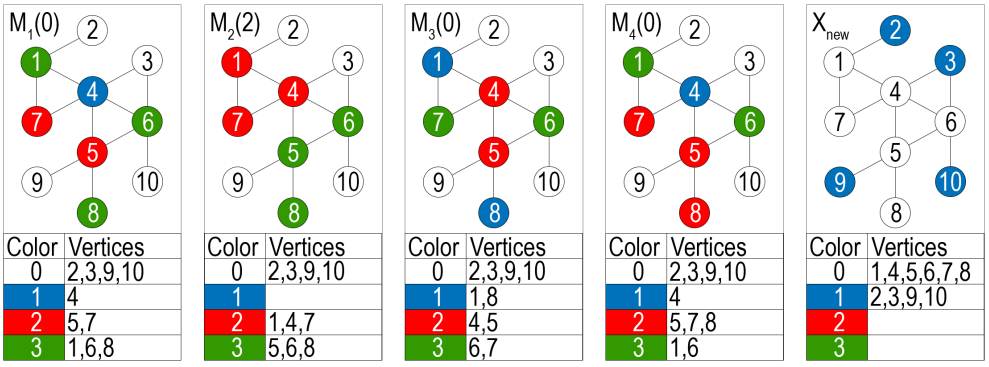


FIGURE 4.7: First step of exploration

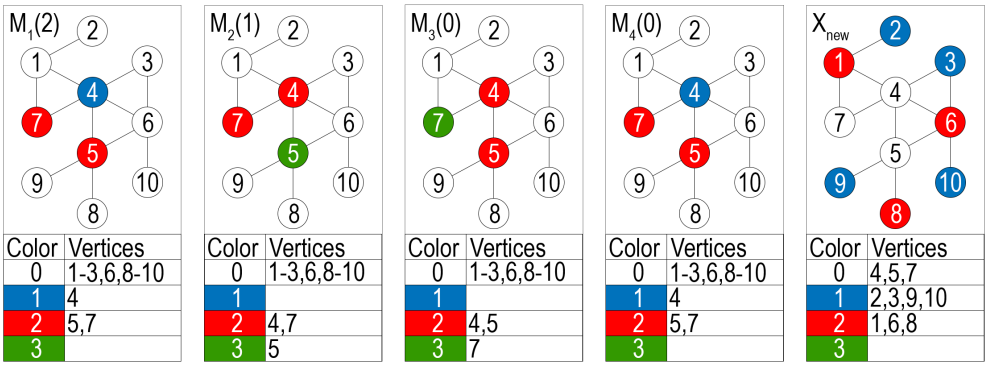


FIGURE 4.8: Second step of exploration

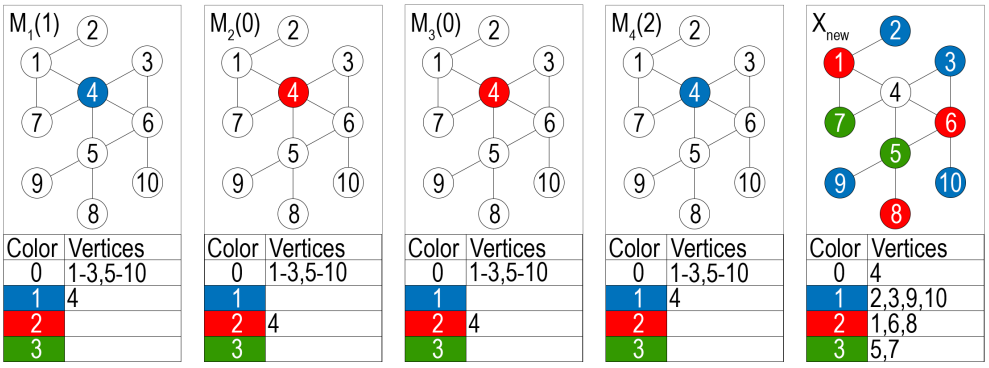
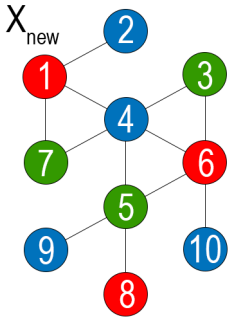


FIGURE 4.9: Third step of exploration

FIGURE 4.10: Exploration Phase comprising four parent mole rats to generate new food source.



Color	Vertices
0	
1	2,4,9,10
2	1,6,8
3	3,5,7

FIGURE 4.11: Solution obtained from exploration phase

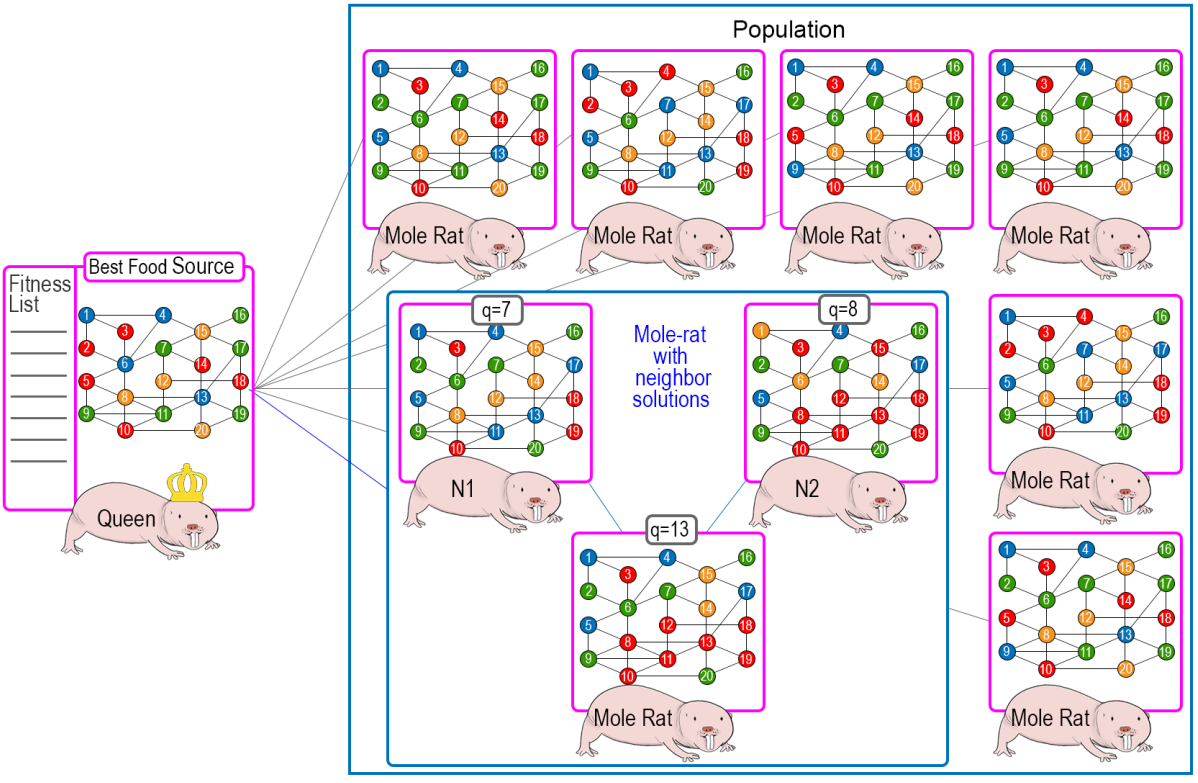


FIGURE 4.12: Overview of BNMR colony for k-GCP

TABLE 4.3: Characteristics of various graph instances from MYC, REG, CAR and DSJ families along with uncategorized instances. $|V|$ denotes the number of vertices in the graph, |E| denotes the number of edges in the graph, $\triangle$ gives the degree of the graph, and lowest k is the known minimum value of colors for a graph with which it can be properly colored.

Sr.	instance	family	|V|	|E|	$\triangle$	lowest k	Sr.	instance	family	|V|	|E|	$\triangle$	lowest k
1	myciel3	MYC	11	20	5	4	21	1-FullIns_5	CAR	282	3247	95	5
2	myciel4	MYC	23	71	11	5	22	2-FullIns_3	CAR	52	201	15	4
3	myciel5	MYC	47	236	23	6	23	2-FullIns_4	CAR	212	1621	55	5
4	myciel6	MYC	95	755	47	7	24	3-FullIns_3	CAR	80	346	19	5
5	myciel7	MYC	191	2360	95	8	25	4-FullIns_3	CAR	114	541	23	6
6	mulsol.i.1	REG	197	3925	121	49	26	5-FullIns_3	CAR	154	792	27	7
7	mulsol.i.2	REG	188	3885	156	31	27	R100_1g	-	100	509	20	5
8	mulsol.i.3	REG	184	3916	157	31	28	R100_5g	-	100	2456	61	15
9	zeroin.i.1	REG	211	4100	111	49	29	R100_9g	-	100	4438	45	36
10	zeroin.i.2	REG	211	3541	140	30	30	R75_1g	-	70	251	12	4
11	zeroin.i.3	REG	206	3540	140	30	31	R75_5g	-	75	1407	48	14
12	1-Insertions_4	CAR	67	232	22	5	32	R75_9g	-	75	2513	71	33
13	1-Insertions_5	CAR	202	1227	67	6	33	R50_1g	-	50	108	8	3
14	2-Insertions_3	CAR	37	72	9	4	34	R50_5g	-	50	612	36	10
15	2-Insertions_4	CAR	149	541	37	5	35	R50_9g	-	50	1092	47	21
16	3-Insertions_3	CAR	56	110	11	4	36	r125.1	-	125	209	8	5
17	3-Insertions_4	CAR	281	1046	56	5	37	r125.5	-	125	3838	99	36
18	4-Insertions_3	CAR	79	156	13	3	38	DSJC125.1	DSJ	125	736	23	5
19	1-FullIns_3	CAR	30	100	11	3	39	DSJC125.5	DSJ	125	3891	75	12
20	1-FullIns_4	CAR	93	593	32	4	40	DSJC125.9	DSJ	125	6961	120	30

TABLE 4.4: Characteristics of selected graph instances from Donald Knuth's Stanford Graph Base

games120	SGB	120	638	26	9
queen5_5	SGB	25	320	32	5
queen6_6	SGB	36	580	38	7
queen7_7	SGB	49	952	48	7
queen8_8	SGB	64	1456	54	9
queen8_12	SGB	96	2736	64	12
queen9_9	SGB	81	2112	64	10
queen10_10	SGB	100	2940	70	11
queen11_11	SGB	121	3960	80	11
queen12_12	SBG	144	5192	86	13
miles250	SGB	128	774	32	8
miles500	SGB	128	2340	76	20
miles750	SGB	128	4226	128	31
miles1000	SGB	128	6432	172	42
miles1500	SGB	128	10396	212	73
anna	SGB	138	493	142	11
huck	SGB	74	301	106	11
jean	SGB	80	254	72	10
david	SGB	87	406	164	11

Graph instances from Donald Knuth's Stanford Graph Base (SGB) are further divided into book, queen, miles graphs and a game graph. Table 4.4. shows number of vertices, number of edges, degree of graph and best known k for selected SGB graphs. Two basic strategies are employed in BNMR-Col i.e. fixing k when chromatic number is known, and sequentially decreasing k until algorithm termination criteria is met where the upper bound on k is provided by LDO. Rational behind using these graph instances for result comparison was the popularity and common use of these graph instances by state of the art graph coloring techniques. Table 4.5 depicts the parameters for ANTCOL, BNMR-Col and Chromaticats. Parameters for ANTCOL and Chromaticats are used as suggested in their respective implementations.

TABLE 4.5: Parameters for AntCol, BNMR-Col and Chromaticats.

BNMR-Col Parameters	Chromaticats Parameters	ANTCOL Parameters
max_iteration = 100	Number of Cats = 30	nants = 100
N = 10	Number of Cycles = 100	$ncycles_{max}$ = 100
‘α = 0.5	MR=2%	α=2
β = 0.1	CDC = 80	β=4
ζ=0.95	SMP=20	ρ=0.5
ρ = 40%		
sl = 20		
move_count = 50		
depth_of_search= 20		

ANTCOL, Chromaticats and BNMR-Col are executed five times on 18 graph instances for maximum 100 iterations. Success ratio is defined as the number of times algorithm

is able to reach best known number of colors denoted by k. Results have been shown in Table 4.6.

Table 4.7 show that none of the algorithm was able to find k for FullIns and DSJC125.1 graph instances. MSPGCA and W-GA were unable to find optimal coloring for queen6_6 and queen8_8 instances. W-GA cannot find optimal coloring for queen7_7. MSPGCA reported coloring with 14 colors for queen8_12 instance, for which best known coloring is 12. BNMR-Col is able to match k for all of these instances. For queen10_10 instance MA-GCP and MSPGCA reported coloring of 13 and 14 respectively. BNMR-Col was unable to match best known coloring (i.e. 11) for this instance. However, BNMR-Col reported coloring of 12, which is better than MA-GCP and MSPGCA. These results show that BNMR-Col is able to match or produce better colorings than MSPGCA, MA-GCP and W-GA.

TABLE 4.6: Computational results of ANTCOL, Chromaticats, and BNMR-Col.

instance	k	BNMR-Col		ANTCOL		Chromaticats	
		lowest k	Success Ratio	lowest k	Success Ratio	lowest k	Success Ratio
myciel5	6	6	5/5	6	5/5	6	5/5
myciel6	7	7	5/5	7	5/5	7	5/5
queen5_5	5	5	5/5	5	5/5	5	5/5
queen6_6	7	7	5/5	7	5/5	7	3/5
queen7_7	7	7	3/5	7	1/5	7	1/5
miles250	8	8	5/5	8	5/5	8	2/5
miles500	20	20	5/5	20	5/5	20	2/5
jean	10	10	5/5	10	5/5	10	5/5
huck	11	11	5/5	11	5/5	11	5/5
1-FullIns_3	3	4	0/5	4	0/5	4	0/5
2-FullIns_3	4	5	0/5	5	0/5	5	0/5
R50_1g	3	3	5/5	3	5/5	3	1/5
R50_5g	10	10	5/5	10	5/5	10	5/5
R50_9g	21	21	5/5	21	5/5	21	2/5
R75_1g	4	4	5/5	4	5/5	4	2/5
R75_5g	13	13	5/5	13	5/5	15	0/5
R75_9g	33	33	5/5	33	5/5	34	0/5
mulsol.i.1	49	49	5/5	49	5/5	49	3/5

4.4 Summary

We have proposed a new algorithm BNMR-Col, an implementation of Blind Naked Mole-Rats metaheuristics for GCP. BNMR-Col is tested on a wide range of DIMACS graph instances. We compared the results with other state ot the art swarm intelligence and evolutionary algorithms. BNMR-Col is competitive and in 83% of the cases it approaches the best known value of k.

TABLE 4.7: Results of BNMR-Col, MA-GCP, MSPGCA and W-GA.

instance	k	MA-GCP[20]	MSPGCA[18]	W-GA[19]	BNMR-Col
1-FullIns_5	5	-	6	-	6
2-FullIns_4	5	-	6	-	6
3-FullIns_4	6	-	7	-	7
4-FullIns_3	6	-	7	-	7
5-FullIns_3	7	-	8	-	8
1-Insertions_5	6	-	6	-	6
2-Insertions_4	5	-	5	-	5
3-Insertions_4	5	-	5	-	5
4-Insertions_4	5	-	5	-	5
myciel3	4	4	-	4	4
myciel4	5	5	-	5	5
myciel5	6	6	6	6	6
myciel6	7	7	7	-	7
myciel7	8	8	8	-	8
games120	9	9	9	9	9
huck	11	11	11	11	11
jean	10	10	10	10	10
david	11	11	11	11	11
queen5_5	5	5	5	5	5
queen6_6	7	7	8	8	7
queen7_7	7	7	7	8	7
queen8_8	9	-	11	10	9
queen8_12	12	-	14	-	12
queen9_9	10	-	10	-	10
queen10_10	11	13	14	-	12
miles250	8	8	-	8	8
miles500	20	20	-	-	20
miles750	31	31	31	-	31
miles1000	42	42	42	42	42
miles1500	73	73	73	-	73
anna	11	11	11	11	11
DSJC125.1	5	-	6	-	6
mulsol.i.1	49	49	-	-	49
zeroin.i.1	49	49	-	-	49
fpsol2.i.1	65	-	-	65	65

Chapter 5

Scalable Genetic Algorithm

Owing to inherently decentralized nature of Genetic Algorithms (GA), a multitude of variants of Parallel GA (PGA) have been introduced in the literature (Knysh and Kureichik, 2010, Luque and Alba, 2011). However, the application of GAs has remain limited to moderately sized optimization problems and the research focused mostly on speeding up the performance of otherwise time-consuming and inherently complex applications e.g. assignment and scheduling (Lim et al., 2007, Liu and Wang, 2015, Trivedi et al., 2015), or prediction (Ferrucci et al., 2018) tasks. To deal with large-scale optimization problems, multi-core systems and standalone clusters architectures have been proposed (Zheng et al., 2011). Zheng et. al have used distributed storage file system and distributed processing frameworks like Apache Hadoop to achieve scalability in PGA (Chávez et al., 2016, Di Geronimo et al., 2012, Salza et al., 2016, Zheng et al., 2011). Hadoop Map Reduce (Hashem et al., 2016) is a reliable, scalable and fault tolerant framework for large scale computing. Hadoop requires writing data to HDFS after each iteration to achieve its fault tolerance. In case of CPU bound iterative processing, e.g. in case of Genetic Algorithms, this I/O overhead is undesirable and substantially dominates the processing time. PGA has been explored for numerous interesting applications like software fault prediction (Ferrucci et al., 2018), test suite generation (Qi et al., 2016), sensor placement (Hu et al., 2017) assignment and scheduling (Lim et al., 2007, Liu and Wang, 2015, Trivedi et al., 2015), dynamic optimization (Lissovoi and Witt, 2017), adapting offspring population size and number of islands (Lässig and Sudholt, 2011). Mostly, PGAs have been implemented using distributed frameworks and the effectiveness is evaluated in terms of execution time, computation effort, solution quality in comparison with the Sequential Genetic Algorithm (SeqGA). However, the above mentioned techniques have been tested on simpler problems which can be solved using limited population size and less number of generations, overlooking the scalability that can be achieved by these frameworks to solve large-scale optimization problems. Apache Spark (Gu and Li, 2013) is an open source distributed cluster

computing framework that has gained popularity in the recent years. It performs better than Hadoop, for large-scale distributed operations due to its faster in-memory operations. It performs well for iterative operations besides its high memory requirement (Gu and Li, 2013). Contrary to Hadoop, Spark keeps data in memory and uses lineage graphs to achieve resilience and fault tolerance. This makes computing faster and eliminates the I/O overhead incurred in case of map-reduce. Spark provides APIs for generic processing in addition to specialised libraries for SQL like operations, stream processing using concepts of mini-batches (Zaharia et al., 2016b), iterative machine learning algorithms, and a Graph processing library (Wani and Jabin, 2018). Spark's efficient data processing has proven to be 100 times faster for in-memory operations and 10 times faster for disk operations when compared to Hadoop Map Reduce (Shoro and Soomro, 2015). Researchers have made significant efforts to explore the intrinsic parallel nature of genetic algorithms using island model (Whitley et al., 1999), and several PGA models (Keco and Subasi, 2012). A lot of efforts have been made by implementing and testing these models on Hadoop framework (Ferrucci et al., 2018, Keco and Subasi, 2012, Shoro and Soomro, 2015, Verma, 2010) by using map reduce strategy to improve scalability. In this paper, we propose a scalable GA (S-GA) for large-scale optimization problems using Apache Spark. We have designed S-GA to have a high frequency of in-memory operations compared to disc operations. The proposed algorithm is compared for scalability with the SeqGA and found to be efficient. The details can be found in experiment section. S-GA aims to reduce the communication between master and worker nodes of Apache Spark by optimal resource utilization for large scale optimization tasks. This is contrary to the traditional island model Whitley et al. (1999) of PGA, where the communication among different population islands is directly proportional to the population and solution size. In S-GA, the communication is independent of the population size and is limited by the *migration rate* and *migration interval*. Hence, reducing a significant amount of data transfer between parallel computations making it scalable and applicable to large scale problems. We have compared S-GA with SeqGA for continuous numeric optimization problems. The experiments have been performed on five different large scale optimization problems.

In this chapter, we have addressed the following research question.

RQ2. How we can exploit Apache Spark framework to improve the scalability of evolutionary algorithms?

Our contributions in this chapter are as follows.

- We have presented an algorithm for Scalable Genetic Algorithms using Apache Spark (S-GA)
- We have tested S-GA on several benchmark functions for large-scale continuous optimization
- S-GA is tested on 3000 dimensions, 3000 population size and upto one billion generations.

This chapter is based on the following publication.

- **Fahad Maqbool**, Saad Razzaq, Jens Lehmann, and Hajira Jabeen. "Scalable distributed genetic algorithm using apache spark (s-ga)." in International conference on intelligent computing, pp. 424-435. Springer, Cham, 2019.

The remaining chapter is structured as follows: In section 5.1, S-GA: Scalable Distributed Genetic Algorithm is explained. In section 5.2 Experimental setup, evaluation metrics and results are elaborated. We have summarized this chapter in section 5.3.

5.1 S-GA: Scalable Distributed Genetic Algorithm using Apache Spark

S-GA creates an initial random population of solutions and distributes them on different partitions as an RDD. Each partition performs GA operations and fitness evaluations independent of other partitions. For experiments, we have used roulette wheel, uniform, interchange, and weak parent as selection criteria, crossover technique, mutation mechanism for generating new solutions, and survival selection for deciding offsprings selection for next generation. Based on the selection scheme, individuals from each partition are selected for crossover and mutation. The crossover and mutation rate decides number of individuals to be selected for the particular operator. In S-GA each partition replaces its weakest solution by the fittest solution broadcasted by

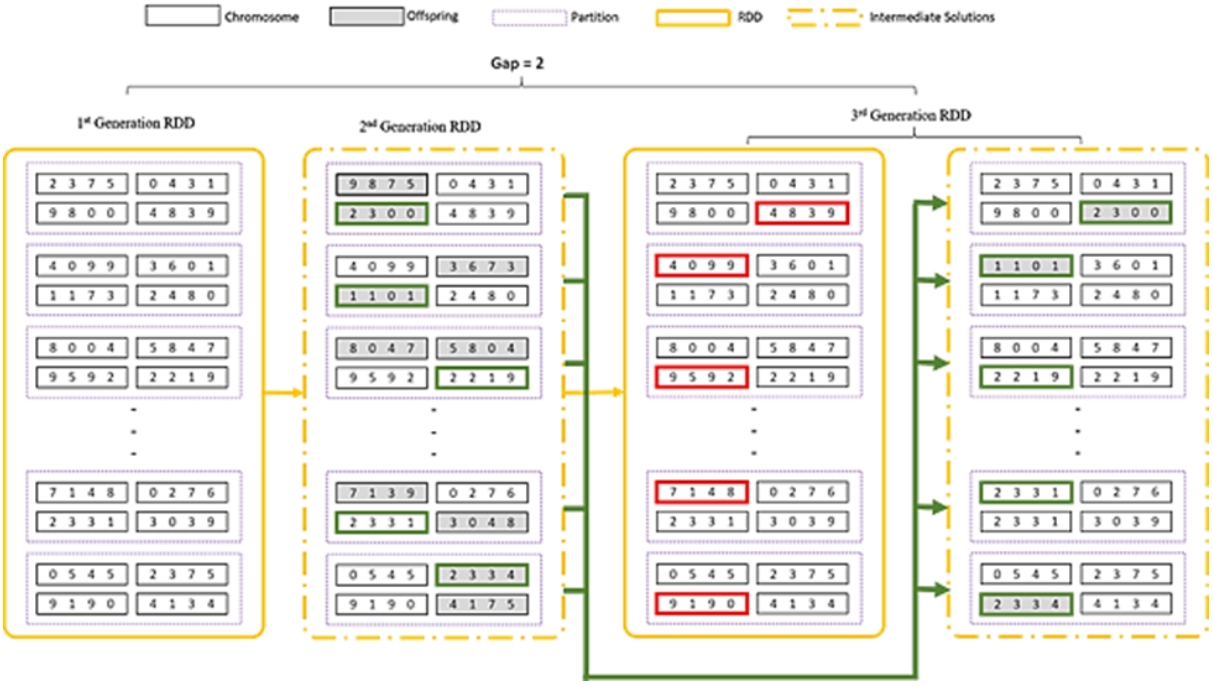


FIGURE 5.1: Evolution process of S-GA

other partitions. **Migration Size** (*Ms*) specifies the number of solutions to be broadcasted to other partitions during each migration step. S-GA significantly reduces the communication overhead by minimizing actions on RDD. Broadcasted solutions are evolved after each migration interval. These evolved solutions are further combined with RDD solutions to produce further evolving solutions for next broadcast.

The pseudo code of S-GA is elaborated in algorithm. 2. Population is initialized at line (1) using random solutions and then distributed among m partitions at line (2). At line (3) a generation counter has been reset. Weak solutions are removed at each partition considering *Ms* and the number of partitions *m* at line(6). Also best broadcasted solutions of all partitions are concatenated with remaining solutions (solutions after removing weak individuals) at each partition. Solutions are evolved using stochastic operators at line (7-12). At line 13, SGA broadcasts evolved best solutions(s) to other partitions. Finally the above steps are iterated until the stopping criteria is met.

Migration Interval/Gap *(Mi)* defines the number of generations after which S-GA broadcasts fittest individual(s) of each partition to other partitions. Afterwards, S-GA replaces its weakest solutions at each partition by broadcasted best solutions. S-GA evolves the individual(s)/solution(s) at each partition for *Mi* number of generations. S-GA avoids materialising RDD to evolved solutions after each migration interval. Afterwards, S-GA migrates best solution of each partition to all the partitions. Figure

Algorithm 7 Pseudo code of S-GA

Require:
N: Population Size
P: Population
Pi: Sub-Population at Partition i
D: Dimensions
G: Generations
Mi: Migration Interval / Gap
f: Fitness Function
Ms: Migration Size

1: $P \leftarrow Randomly\ initialise\ population$
2: Distribute *P* among m partitions
3: $G = 0$
4: **while** stopping criteria not met **do**
5: at each partition *i*
6: $Pi' \leftarrow PickPi - (Ms \times m)bestsolutionsfromPi \cup \ (Ms \times m)migratedsolutions$
7: **for** $k \leftarrow 1toMi$ **do**
8: Pi'' ← Select Parents (Pi')
9: Pi'' ← Crossover (Pi'')
10: Pi'' ← Mutate (Pi'')
11: Pi' ← Survival_Selection (Pi'U Pi'')
12: **end for**
13: Migrate *Ms* individuals from each partition *i*
14: End At each partition *i*
15: *G = G + Mi*
16: **end while**

5.1 explains the idea with an example. Let's assume value of *Mi = 2* and fitness function as sphere (i.e. $f(x_i) = \sum_{i=1}^{n} x_i^2$). Initial RDD is created using a population of random solutions. These initial solutions are then evolved using crossover and mutation operators. After every 2nd generation, best solutions from each partition are migrated to other partitions. As the solution migrates, each partition at the start of very next generation picks all migrated solutions and replace them with weakest solutions at each partition. As RDD with replaced solutions is never materialised through out the execution of S-GA, so this replacement is temporary until the next broadcast.

5.2 Experiments

5.2.1 Experimental Setup

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. The configuration parameters of S-GA and GA are detailed in in Table 1.

5.2.2 Evaluation Metrics

5.2.2.1 Speed Up

It is the ratio of sequential execution time to the parallel execution time. It reflects how much the parallel algorithm is faster than the sequential algorithm. Table 5.1 reflects speed up for all the cases where SeqGA and S-GA converge to VTR (Value To Reach). VTR defines the threshold for convergence. We have used $\frac{1}{Numberof Dimensions}$ as VTR in experimentation. Speed up values in Table 5.1 shows that in few cases speedup has increased. With careful considerations of partitions, migration interval (Mi), and migration size (m); A significant speedup can be achieved for S-GA. However these factors can be exploited more to see their effects on speedup and improved execution time.

5.2.2.2 Execution Time

The execution time of SeqGA and S-GA was measured using system clock time. This time was recorded for a maximum of 1 billion generations. Table 2. Shows average execution time over 5 runs for each configuration of S-GA. We can observe that execution time reduces significantly when we increase Mi from 50000 to 100000, however the fitness error also decreases significantly. This difference in time reduces with an increase in the number of partitions. Migration overhead defines the total number of migrated individual(s) by all partitions after Mi. Increase in m and Mi results in increased network overhead (m*Mi) and hence execution time. But on the other hand this also helps S-GA to converge in a less number of generations. Table 5.2 lists the execution time of Sphere, Ackley, Griewank, Rastrigin, Zakharov, and Sum of Different Power functions for optimization upto 3000 Dimensions (D). For simplicity, the population size (N) has been assumed to be equivalent to Number of Dimensions. G represents the number of generations that has been consumed using specified configurations. VTR as mentioned earlier is reciprocal to D. Hence VTR would be lesser for 3000 dimensions compared to 2000 and 1000 dimensions. Bold values in Table 5.2 represents the fitness error that has decreased beyond specified threshold i.e. VTR.

TABLE 5.1: Experimental Results of S-GA and SeqGA. Function: Griewank, P=D, Crossover scheme: Uniform, Mutation: Interchange, Replacement Scheme: Weak parent, Selection Scheme: Roulette Wheel, Crossover Probability: 0.5, Mutation Probability: 0.05

D	SeqGA			S-GA						
	G	Time	Error	Mi	m	Ms	G	Time	Error	SpeedUp
1000	748	247	2.45e-4	25000	18	1	106850000	712	8.25e-4	-
						2	39050000	352	9.28e-4	-
						3	27275000	327	1.4e-4	-
					24	1	46050000	356	6.68e-4	-
						2	22675000	263	8.08e-4	-
						3	19925000	311	9.55e-4	-
					30	1	37500000	328	2.87e-4	-
						2	15750000	228	2.33e-4	1.08
						3	13150000	269	7.01e-4	-
				50000	18	1	194500000	650	2.79e-4	-
						2	86500000	393	7.99e-4	
						3	54850000	317	4.15e-4	-
					24	1	92800000	565	9.37e-5	-
						2	44250000	262	2.52e-4	
						3	38750000	311	3.69e-4	-
					30	1	81850000	344	3.42e-4	-
						2	33550000	244	2.01e-5	1.01
						3	30350000	309	3.24e-4	-
2000	989	1064	2.03e-4	25000	18	1	242650000	3479	1.94e-4	-
						2	95400000	1845	4.17e-4	-
						3	62225000	1472	1.67e-4	-
					24	1	127075000	2112	2.65e-4	-
						2	61875000	1466	3.22e-4	-
						3	36650000	1061	3.51e-4	1.002
					30	1	95950000	1714	1.88e-4	-
						2	41250000	1042	2.64e-4	1.02
						3	26275000	970	1.65e-4	1.1
				50000	18	1	448200000	3309	1.5e-4	-
						2	179300000	1713	2.37e-4	-
						3	133200000	1541	4.15e-5	-
					24	1	246500000	2162	3.62e-4	-
						2	120250000	1429	2.47e-4	-
						3	74300000	1052	1.37e-4	1.01
					30	1	185250000	1837	2.91e-4	-
						2	78900000	1076	3.2e-4	-
						3	54900000	989	2.56e-4	1.07

TABLE 5.2: Experimental Results of S-GA and SeqGA. Crossover scheme: Uniform, Mutation: Interchange, Replacement Scheme: Weak parent, Selection Scheme: Roulette Wheel, Crossover Probability: 0.5, Mutation Probability: 0.05, P=D, m: 24, Ms: 2

f	Mi	D = 1000 VTR = 0.001			D = 2000 VTR = 5.0 E-4			D = 3000 VTR = 3.33 E-4		
		G	Time	Error	G	Time	Error	G	Time	Error
Sphere	50000	1e9	556	8.28	1e9	11531	0.004	1e9	18904	0.017
	100000	1e9	282	265.05	1e9	5932	0.003	1e9	12227	5.967
Ackley	50000	1e9	5616	0.009	1e9	11799	0.095	1e9	17609	1.27
	100000	1e9	2613	0.02	1e9	5819	0.015	1e9	9456	2.07
Griewank	50000	4.4e7	262	**2.52e-4**	1.2e8	1429	**2.47e-4**	2.1e9	3917	**1.25e-4**
	100000	9.6e7	277	**6.23e-4**	2.4e6	1417	**1.41e-4**	4.1e9	3732	**2.47e-4**
Rastrigin	50000	1e9	5339	0.024	1e9	11513	1.443	1e9	18594	0.067
	100000	1e9	2623	2.081	1e9	5809	0.907	1e9	9447	52.45
Zakharov	50000	1e9	5575	17035.15	1e9	11779	33111.93	1e9	16048	10249.73
	100000	1e9	2896	16803.21	1e9	5783	33205.55	1e9	9036	50674.89
Sum of Diff Powers	50000	200000	6	**4.59e-4**	250000	10	**3.16e-4**	700000	19	**1.6e-4**
	100000	400000	6	**2.82e-4**	400000	8	**4.5e-4**	600000	11	**1.42e-4**

It can be seen from Table 5.2 that for higher values of Mi (i.e. 100000), each function consumes less time in most of the cases. Broadcasts are also important as they help each sub-population Pj to increase it's diversity and helps each Pj to get out of it's local optima. Increased Mi values reduces frequent broadcasts and hence the network overhead. In case of increased Mi, the higher number iterations may not improve the optima significantly, due to reduced diversity in the particular sub population. Table 5.2 reveals the discussed fact as Error is less for Mi=50000 compared to Mi=100000 in most of the cases.

5.3 Summary

In this chapter, we have proposed initial results of Scalable Distributed GA (S-GA) using Apache Spark for large-scale optimization problems. The results have been compared with SeqGA. We have tested S-GA for Sphere, Ackley, Griewank, Rastrigin, Zakharov, and Sum of Different Powers functions that are typical benchmarks for continuous optimization problems. We have used Population size of up to 3000, Dimensions of up to 3000, Partition Size up to 30, migration size up to 03, and migration interval to 100000. For few cases S-GA has outperformed SeqGA for higher population, partitions, migration size, and migration interval in term of execution time.

Chapter 6

Distributed Scalable Shade-Bat Algorithm

Real-life optimization problems are becoming increasingly complex due to an increase in the number of decision variables as a result of digitalization. The traditional EA, i.e., Differential Evolution (DE) (Storn and Price, 1997), Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), and Genetic algorithm (GA) (Holland, 1975) show promising results for relatively low dimensional problems. For high dimensions, EA suffers from performance bottleneck (slow convergence speed and convergence to local optima), termed as the curse of dimensionality (Chen et al., 2015). Therefore it is important to develop distributed and scalable techniques for Large Scale Global Optimization (LSGO). LSGO problems have arisen in many fields, including but not limited to engineering (Molina et al., 2018), machine learning, resource scheduling, and vehicle routing in large scale traffic network (Mahdavi et al., 2017), business intelligence, & data mining (Mohamed, 2017). Knowledge Graph Embedding (KGE) modelling is a large scale optimization problem where large scale knowledge graphs like DBpedia (2 million entities and 10 million facts), Wikidata (1.5 million entities and 3 million facts), and Google KG (500 million entities and 3.5 billion facts) are transformed to multidimensional matrices. KGE is used to find missing, or new facts in the knowledge graphs (Ji et al., 2020). Deep transfer learning is another application area to optimize Neural Network weights and then to use these pre-trained networks for new problems. Keras's [1] VGG16, VGG19, NASNetLarge, and EfficientNetB7 are few large scale networks that contain 13.8 million, 14.3 million, 8.8 million, and 6.6 million decision variables respectively.

A problem is categorized as an LSGO problem (Li et al., 2013), if the number of decision variables increases beyond 1000. The LSGO problems can range from fully separable problems to non-separable problems, where separability describes the extent to which a problem can be divided into sub-problems, such that a fitness function can evaluate them independently. A problem is fully separable if all its variables are independent of each other (Blanchard et al., 2019).

[1] https://keras.io/api/applications/

The established LSGO approaches have been categorized (Sabar et al., 2019) as Co-operative Co-evolution (CC) and Non-Cooperative Co-evolution (NCC). In CC, a high dimensional problem is divided into low dimensional sub-problems, making them capable of being solved using conventional EA. Sub-problems maintain separate sub-populations to ensure diversity (Gong et al., 2015). Although CC has been used in different meta-heuristics (e.g. Ant colony optimization (ACO) (Doerner et al., 2001), GA (Chang, 2010), DE (Antonio and Coello, 2016) and PSO (Tan et al., 2007)) but the performance of CC changes abruptly in case of complex and overlapping sub-problems. Additionally, CC is sensitive to the grouping strategy. In NCC, a problem in never decomposed and hence doesn't require a grouping strategy. The problem is solved without requiring detailed information about the decomposition. This makes NCC a suitable choice for many overlapping and complex optimization problems. NCC techniques include SHADE with Iterative Local Search (SHADE-ILS) (Molina et al., 2018), adaptive population differential evolution with dual control strategy (Zhang and Sanderson, 2009), enhanced adaptive differential evolution algorithm (Mohamed, 2017), and SHADE-SPA memetic framework (Hadi et al., 2019).

In this article, we have proposed a NCC technique, Distributed, Scalable Shade BAT (DistSSB). DistSSB is a parallel and scalable approach. DistSSB combines Success-History based parameter Adaptation for Differential Evolution (SHADE) (Tanabe and Fukunaga, 2013a) and Bat algorithm (BA) (Yang, 2010) to achieve adept exploration and exploitation, respectively. Using the Island model (Al-Betar and Awadallah, 2018), we have divided the population into multiple sub-populations where each sub-population is evolved in parallel. SHADE is used for exploration in a few partitions, whereas BAT is used for exploitation in the remaining partitions. The performance of DistSSB is tested on the CEC-2013 benchmark function suite (Li et al., 2013), and the results have been found better from SHADE-ILS and GL-SHADE for most functions with improved execution time for 1000 and 2000 decision variables. To verify the scalability, we have compared DistSSB, SHADE-ILS, and GL-SHADE on one million dimensions for $f12$ and found that SHADE-ILS, and GL-SHADE are unable to scale up to this problem size while DistSSB shows convincing results. To the best of our knowledge, DistSSB is the first, open source algorithm that addresses the problem of scalability.

This chapter covers the following research question.

RQ3. Real-world problems (e.g., Knowledge Graphs, Deep Learning, NLP) are increasing in size. Can we optimize conventional EA techniques for very large dimensions?

Our contributions in this chapter are as follows.

- We have proposed a scalable, parallel, distributed and hybrid EA named Distributed Scalable Shade Bat (DistSSB) to solve LSGO problems.
- DistSSB distributes its population into multiple sub-populations using the island model. Each sub-population is independently evolved using SHADE or BAT
- DistSSB is implemented using the popular distributed in-memory framework, Apache Spark.
- DistSSB is scalabale upto one million dimensions.

This chapter is based on the following publication.

- **Fahad Maqbool**, Saad Razzaq, Asif Yar, and Hajira Jabeen. "Large Scale Distributed Optimization using Apache Spark: Distributed Scalable Shade-Bat (DistSSB)." in 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 2559-2566. IEEE, 2021.

The rest of this chapter is organized as follows. In section 6.1, Distributed Scalable Shade-Bat is explained. Experimental setup, Optimization, Execution time, Scalability, effect of islands and migration interval, is explained in section 6.2. We have summarized this chapter in section 6.3.

6.1 Distributed Scalable Shade-Bat (DistSSB)

In this section, we detail the proposed Scalable Shade Bat (DistSSB) algorithm. DistSSB uses island model by dividing the population among islands. This helps in achieving speedup and avoiding stagnation through solution sharing among islands. The SHADE algorithm has a strong exploration capability and helps finding new solutions in unexplored search space, whereas, BA algorithms offers an intensive exploitation

strategy. Therefore DistSSB combines SHADE (for exploration) and BA (for exploitation) for value-added optimization. Inspired from center-based sampling strategy (Rahnamayan and Wang, 2009), we select the m fittest solutions and calculate their centroid as the best solution, as given in Eq. 6.1 for islands utilizing BA. It has been statistically proven that there is a higher chance of finding an unknown optimal solution when initial solutions are closer to the center of the search space and this chance increases with increasing number of dimensions (Rahnamayan. and Wang, 2009).

$$x^{best} = \frac{x_1^{best} + x_2^{best} + ... + x_m^{best}}{m} \tag{6.1}$$

Additionally, we have proposed a local search operator as explained in algorithm 2 for BA that helps in fast convergence by maintaining diversity as given in Eq. 6.2. Eq. 2.4 is replaced by Eq. 6.2 in algorithm 2 to avoid premature convergence.

$$x_{new} = x^{best} + \epsilon A(x^{best,n} - x_{k,n}) \tag{6.2}$$

Where k is a random number between $[0, N-1]$ and n is number between $[0, d-1]$ such that $k \neq n$ where d represents dimensions. The main steps of the DistSSB are given in algorithm 8. DistSSB creates RDD from the population of random solutions.

Algorithm 8 Distributed Scalable SHADE Bat

```
Initialize Population (P),
      Migration Rate (M_r), Migration Interval (M_i)
population = sc.parallelize (P)
while stoppingCriteria do
   bestSolutions = population.mapPartitionsWithIndex { ( index, iterator ) {
      popData = loadData()
      popData = popData.eliminateWeakSolutions()
      popData = broadcastedSol.union(popData)
   for ‘ do  t : 1 to M_i
      if index % 2 == 0  then
         SHADE(popData)
         best = selectRandom(popData.take(M_r))
      end if
      if index % 2 == 1  then
         BA(popData)
         best = findCentroid(popData.take(M_r))
      end if
   end for
      save.popData.iterator
   best.iterator
   } } . collect()
   broadcastedSol = sc.broadcast(bestSolutions)
end while
```

The code within mapPartitionsWithIndex (line 5-19) is evolved in a parallel fashion on multiple partitions. It is an internal feature of Spark that returns a new RDD by applying a function to each partition in distributed manner. Within the parallel execution, half of the sub-populations (partitions) are evolved using BA (line 14-15) while others are evolved using SHADE (line 10-11). After each migration interval, the best solution(s) from each partition (line 19) are collected and placed in bestSolutions (line 4). bestSolutions are then broadcasted to other partitions (line 21). Each partition at the start of the next migration interval picks all the broadcasted solutions and uses them to replace its weakest solutions (line 6-7). This creates a mesh topology between the partitions. The evolved RDD (popData) with replaced and evolved solutions is not materialized throughout the DistSSB execution. This RDD remains in-memory until the next broadcast. In-memory execution reduces the network communication between worker and master nodes and decreases the overall execution time. The evolved population at each partition is saved temporarily on the local (partition) file system after every migration interval (line 18). The saved popData and bestSolutions are retrieved again at the start of the next migration interval (line 5). The algorithm terminates when stopping criteria is met.

6.2 Experimental Setup

To test and evaluate the performance of the DistSSB, we have used the Large Scale Global Optimization benchmark functions suite and the experimental conditions used in CEC-2013 (Li et al., 2013). CEC benchmark contains 15 functions with 1000 decision variables except for $F13$ and $F14$ where decision variables are 905 due to overlapping factors. These benchmark functions are divided into fully separable, partially separable, overlapping, and non-separable functions. We briefly discuss these functions below, and additional details can be found in (Li et al., 2013, Zhang et al., 2020).

1. Fully separable functions: $(f1 - f3)$
2. Partially separable functions
   (a) with a separable subcomponent: $(f4 - f7)$
   (b) with no separable subcomponent: $(f8 - f11)$
3. Overlapping Functions: $(f12 - f14)$

4. Non-separable Functions: ($f15$)

We have tested all CEC-2013 benchmark functions upto 2,000 dimensions (2000D) while tested a few functions for $5.0*10^4$ dimensions. To verify the scalability of DistSSB we have evaluated $f12$ for 10^6 dimensions. Table 6.1 shows the parameter settings of DistSSB. Maximum number of function evaluations are $3.0*10^6$ and VTR is 0.0. All the

TABLE 6.1: Parameter settings for DistSSB

Parameter	**Value**	**Parameter**	**Value**
Population size (NP)	10^3	ϵ	0.003
Number of Islands (I_n)	10	r_i^0	0.1
Migration Interval (M_i)	$2 * 10^4$	α , γ	0.95
Migration Rate (M_r)	1	CR, F	0.95, 0.55

results are reported as average of 25 executions against each configuration. In Table 6.1 parameters for BA (ϵ, r_i^0, α and γ) are initialized as in (Al-Betar and Awadallah, 2018)

TABLE 6.2: DistSSB, GL-SHADE, and SHADE-ILS fitness comparison

f	Details	1000D			2000D		
		DistSSB	GL-SHADE	SHADE-ILS	DistSSB	GL-SHADE	SHADE-ILS
$f1$	Best	1.83E-05	3.07E-10	0.00E+00	4.48E-08	1.17E+00	4.76E-27
	Median	2.19E-02	5.38E-06	0.00E+00	4.06E-03	1.92E+00	6.66E-25
	Worst	4.79E-01	8.71E-05	8.81E-03	1.20E-01	4.68E+00	2.73E-23
	Mean	1.52E-01	1.32E-05	1.48E-03	3.00E-02	2.32E+00	5.85E-24
	StDev	2.08E-01	2.65E-05	2.94E-03	5.15E-02	1.43E+00	1.20E-23
	P-Value		2.12E-04$^-$	1.50E-03$^-$		**1.57E-04$^+$**	1.57E-04$^-$
$f2$	Best	4.69E+02	0.00E+00	9.08E+02	7.26E+02	1.49E-01	9.18E+02
	Median	5.07E+02	1.05E+00	1.09E+03	1.63E+03	2.83E-01	4.92E+03
	Worst	6.03E+02	4.97E+00	1.30E+03	8.53E+03	9.24E-01	6.77E+05
	Mean	5.18E+02	1.89E+00	1.11E+03	3.32E+03	4.13E-01	1.41E+05
	StDev	4.29E+01	2.15E+00	1.89E+02	3.34E+03	3.03E-01	3.00E+05
	P-Value		1.57E-04$^-$	**1.570E-04$^+$**		1.57E-04$^-$	**2.33E-02$^+$**
$f3$	Best	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.02E+01	2.03E+01
	Median	2.00E+01	2.01E+01	2.01E+01	2.00E+01	2.05E+01	2.07E+01
	Worst	2.02E+01	2.02E+01	2.02E+01	2.00E+01	2.05E+01	2.07E+01
	Mean	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.04E+01	2.06E+01
	StDev	7.04E-02	8.38E-02	6.07E-02	0.00E+00	1.64E-01	2.23E-01
	P-Value		**2.46E-04$^+$**	**4.59E-03$^+$**		**1.57E-04$^+$**	**3.41E-03$^+$**
$f4$	Best	2.54E+08	1.72E+08	8.90E+09	2.58E+09	1.50E+10	3.97E+10
	Median	4.09E+08	7.30E+08	1.73E+10	3.84E+09	2.42E+10	9.07E+10
	Worst	5.42E+08	1.06E+09	2.16E+10	6.26E+09	3.78E+11	1.68E+11

DistSSB, GL-SHADE, and SHADE-ILS fitness comparison (Continued).							
	Mean	3.99E+08	6.84E+08	1.62E+10	4.17E+09	9.36E+10	9.57E+10
	StDev	1.05E+08	2.74E+08	5.40E+09	1.50E+09	1.59E+11	4.60E+10
	P-Value		**1.56E-02**$^{+}$	**1.570E-04**$^{+}$		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$
	Best	1.58E+06	1.04E+06	1.72E+06	2.49E+06	1.42E+06	1.71E+06
	Median	1.80E+06	1.17E+06	1.99E+06	5.23E+06	3.07E+06	4.18E+06
$f5$	Worst	2.86E+06	1.48E+06	2.23E+06	2.73E+07	1.04E+07	6.20E+06
	Mean	2.02E+06	1.18E+06	2.02E+06	9.19E+06	4.19E+06	4.22E+06
	StDev	3.73E+05	1.14E+05	2.25E+05	1.03E+07	3.57E+06	1.68E+06
	P-Value		1.57E-04^{-}	4.96E-01$^{=}$		1.51E-01^{-}	2.84E-02^{-}
	Best	1.00E+06	1.04E+06	1.02E+06	1.02E+06	1.01E+06	1.02E+06
	Median	1.01E+06	1.05E+06	1.04E+06	1.04E+06	3.58E+06	3.45E+06
$f6$	Worst	1.02E+06	1.06E+06	1.04E+06	1.07E+06	7.24E+06	6.54E+06
	Mean	1.01E+06	1.05E+06	1.04E+06	1.04E+06	3.48E+06	3.35E+06
	StDev	7.89E+03	7.38E+03	8.58E+03	1.552E+04	2.45E+06	2.02E+06
	P-Value		**1.57E-04**$^{+}$	**5.07E-04**$^{+}$		**3.61E-03**$^{+}$	**1.40E-02**$^{+}$
	Best	1.43E+04	4.90E+05	9.27E+06	2.40E+07	2.29E+08	8.24E+07
	Median	3.87E+04	5.49E+05	9.79E+06	3.77E+07	4.54E+08	2.92E+08
$f7$	Worst	6.78E+04	6.25E+05	2.49E+07	6.27E+07	7.06E+08	5.42E+08
	Mean	4.16E+04	5.47E+05	1.17E+07	4.07E+07	4.83E+08	3.08E+08
	StDev	1.95E+04	4.20E+04	4.78E+06	1.33E+07	1.94E+08	1.73E+08
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$
	Best	2.84E+10	6.11E+12	4.04E+13	1.42E+13	5.27E+13	1.83E+14
	Median	5.16E+10	7.10E+12	3.39E+14	2.57E+13	1.93E+15	2.35E+15
$f8$	Worst	6.26E+10	7.40E+12	3.28E+14	3.85E+13	8.92E+15	6.71E+15
	Mean	5.26E+10	7.40E+12	3.28E+14	2.57E+13	1.93E+15	2.35E+15
	StDev	2.65E+10	9.65E+11	1.83E+14	8.65E+12	3.23E+15	2.42E+15
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$
	Best	2.25E+08	2.65E+08	1.83E+08	5.71E+08	1.02E+09	4.65E+08
	Median	2.55E+08	3.55E+08	1.94E+08	6.77E+08	1.58E+09	6.79E+08
$f9$	Worst	2.70E+08	5.25E+08	2.20E+08	8.01E+08	4.26E+09	9.01E+08
	Mean	2.54E+08	3.65E+08	1.98E+08	9.13E+07	9.31E+07	9.27E+07
	StDev	1.44E+07	9.68E+07	1.55E+07	8.51E+07	1.23E+09	1.49E+08
	P-Value		**1.50E-03**$^{+}$	1.57E-04^{-}		**1.57E-04**$^{+}$	**6.70E-04**$^{+}$
	Best	9.05E+07	9.18E+07	9.15E+07	9.02E+07	8.60E+07	8.81E+07
	Median	9.06E+07	9.29E+07	9.24E+07	9.15E+07	9.29E+07	9.16E+07
$f10$	Worst	9.19E+07	9.08E+07	9.30E+07	9.20E+07	9.83E+07	9.98E+07
	Mean	9.07E+07	9.28E+07	9.24E+07	9.13E+07	9.31E+07	9.27E+07
	StDev	2.35E+05	5.68E+05	5.71E+05	6.33E+05	4.29E+06	4.01E+06
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**5.21E-01**$^{+}$	**7.91E-01**$^{+}$
	Best	5.71E+06	9.16E+11	2.17E+08	9.49E+07	2.37E+12	1.08E+10
	Median	9.17E+06	9.32E+11	3.03E+08	4.58E+08	4.78E+12	3.04E+10
$f11$	Worst	1.70E+07	9.44E+11	4.13E+08	6.84E+08	7.86E+12	6.84E+10
	Mean	9.99E+06	9.31E+11	2.92E+08	4.07E+08	4.95E+12	3.40E+10

DistSSB, GL-SHADE, and SHADE-ILS fitness comparison (Continued).							
	StDev	3.95E+06	1.15E+10	5.29E+07	2.00E+08	1.79E+12	1.80E+10
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$
	Best	1.81E+03	2.90E+01	2.18E+02	3.10E+03	9.27E+01	1.01E+03
	Median	2.06E+03	1.20E+02	5.95E+02	4.63E+03	3.53E+03	1.20E+03
$f12$	Worst	2.46E+03	2.52E+02	6.86E+02	5.82E+03	9.79E+03	4.65E+03
	Mean	2.09E+03	1.25E+02	4.96E+02	4.48E+03	4.31E+03	1.97E+03
	StDev	2.05E+02	7.78E+01	1.67E+02	9.98E+02	3.32E+03	1.35E+03
	P-Value		1.57E-04^{-}	1.57E-04^{-}		4.06E-01$^{=}$	1.94E-03^{-}
	Best	4.05E+06	4.13E+08	4.47E+08	1.91E+09	1.63E+10	1.64E+09
	Median	6.61E+06	4.44E+08	5.20E+09	3.19E+09	3.18E+10	8.45E+09
$f13$	Worst	8.98E+06	4.69E+08	5.50E+09	4.70E+09	5.38E+10	9.98E+09
	Mean	6.63E+06	4.42E+08	3.82E+09	3.22E+09	3.29E+10	7.37E+09
	StDev	1.4E+06	1.62E+07	2.12E+09	8.27E+08	1.32E+10	2.95E+09
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**5.06E-03**$^{+}$	**3.64E-01**$^{+}$
	Best	6.63E+06	5.58E+07	1.13E+10	2.58E+07	2.52E+10	2.76E+10
	Median	7.58E+06	8.78E+07	1.68E+10	3.08E+08	4.35E+10	4.97E+10
$f14$	Worst	8.56E+06	9.03E+07	2.35E+10	4.26E+08	7.29E+10	8.31E+10
	Mean	7.60E+06	7.94E+07	1.66E+10	2.27E+08	4.57E+10	5.34E+10
	StDev	6.96E+05	1.45E+07	5.24E+09	1.76E+08	1.74E+10	2.03E+10
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$
	Best	2.89E+06	1.01E+07	7.28E+06	1.53E+07	1.93E+08	8.54E+06
	Median	3.09E+06	1.37E+07	8.85E+06	3.29E+07	3.19E+08	7.02E+07
$f15$	Worst	3.38E+06	1.59E+07	1.67E+07	7.17E+07	5.38E+08	9.81E+07
	Mean	3.09E+06	1.35E+07	1.04E+07	3.81E+07	3.58E+08	6.15E+07
	StDev	1.46E+05	1.99E+06	3.40E+06	2.05E+07	1.32E+08	3.14E+07
	P-Value		**1.57E-04**$^{+}$	**1.57E-04**$^{+}$		**1.57E-04**$^{+}$	**3.26E-01**$^{+}$
+ / = / -			11 / 0 / 4	11 / 1 / 3		12 / 1 / 2	12 / 0 / 3

while two control parameters (CR, F) of SHADE are set as suggested in (Molina et al., 2018). All the DistSSB 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. GL-SHADE and SHADE-ILS experiments have been performed on Google Cloud (12GB RAM) that is GPU enabled.

The source code[1] of DistSSB using Scala and Apache Spark framework is available on github.

[1] https://github.com/HajiraJabeen/SparkOP

6.2.1 Optimization

The optimization results of DistSSB are compared with GL-SHADE and SHADE-ILS. For 1000D, DistSSB has obtained better convergence as compared to SHADE-ILS for eleven functions and performed equivalent on one function while SHADE-ILS performed better for $f1, f9, \& \ f12$. When compared to GL-SHADE, DistSSB performed better for eleven functions, while GL-SHADE performed better on $f1, f2, f5, \& \ f12$.

For 2000D, DistSSB's performance improved and it converged better than SHADE-ILS on twelve functions, while SHADE-ILS is better for $f1, f5, \& \ f12$. In comparison to GL-SHADE, DistSSB performed better on twelve functions, equal on one function while GL-SHADE performed better for $f2, \& \ f5$. Table 6.2 shows a comparison of DistSSB, GL-SHADE and SHADE-ILS on 1000D and 2000D with a maximum of $3 * 10^6$ function evaluations.

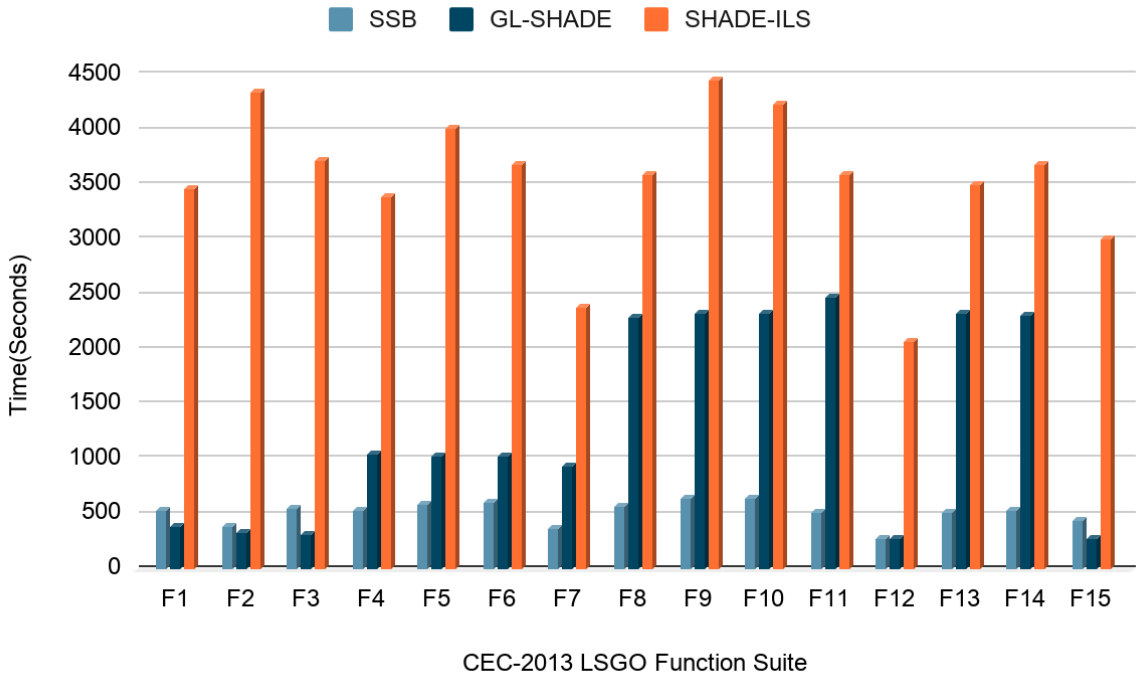


FIGURE 6.1: Execution time of DistSSB, GL-SHADE, and SHADE-ILS on 1000D

This can be observed that DistSSB has performed better for majority of functions on high dimensions. We have used Wilcoxon rank sum test to analyze the accuracy of results using significance level of 0.05. The last row of Table 6.2 indicates the number of functions for which DistSSB is better $+$, worse $-$, or equal $=$ to the compared

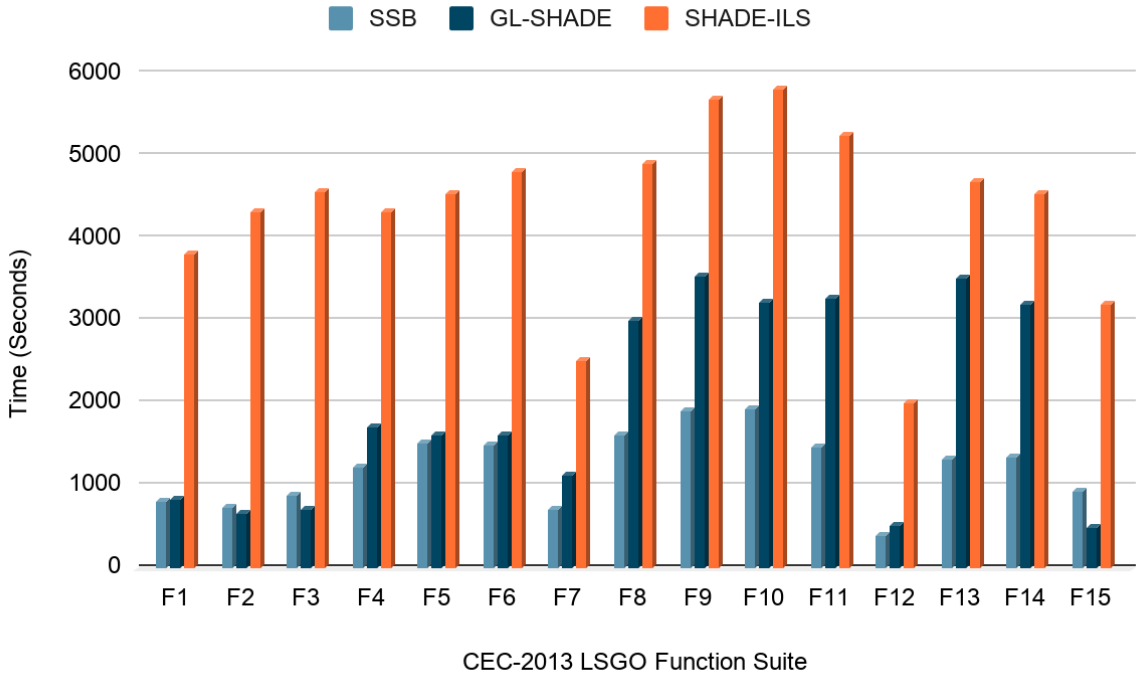


FIGURE 6.2: Execution time of DistSSB, GL-SHADE, and SHADE-ILS on 2000D

algorithms. Table 6.3, shows the average ranking of algorithms based on friedman statistical ranking test. DistSSB has the smallest mean rank that reflects DistSSB is better than GL-SHADE and SHADE-ILS.

6.2.2 Execution Time of DistSSB

The execution time of SHADE-ILS, GL-SHADE, and DistSSB is measured using system clock time. Time comparison is shown in Figure. 6.1 and Figure. 6.2 for 1000D, and 2000D respectively. DistSSB outperformed SHADE-ILS on all functions for both dimensions. While comparing with GL-SHADE for 1000D, DistSSB performed better on ten functions, equal for one function while GL-SHADE performed better on $f1$, $f2$, $f3$ & $f15$, while equal for $f12$. In case of 2000D DistSSB performed better than GL-SHADE on twelve functions while GL-SHADE performed better for $f2$, $f3$, & $f15$. Here it is pertinent to mention that by increasing dimensions from 1000 to 2000, DistSSB improves in execution time for $f1$, & $f12$ over GL-SHADE.

6.2.3 Scalability of DistSSB

We have tested the scalability of DistSSB on higher dimensions. On 50,000D results show that DistSSB converged faster for $f2$, $f3$, & $f15$ functions, whereas the execution time of GL-SHADE was better for these functions on lower dimensions. Figure. 6.3 shows the execution time of SHADE-ILS, GL-SHADE and DistSSB on $f2$, $f3, \& f15$ for 50000D. To check the scalabilty of DistSSB experiments are performed on one million dimensions for overlapping function $f12$, which is rather simple and does not involve a rotation matrix (to reduce the overall complexity). We also tested the scalability over one million dimensions. The results in Table 6.4 show that Shade-ILS and GL-SHADE resulted in memory-outage, while DistSSB and proves to be scalable and suitable for high dimensional problems.

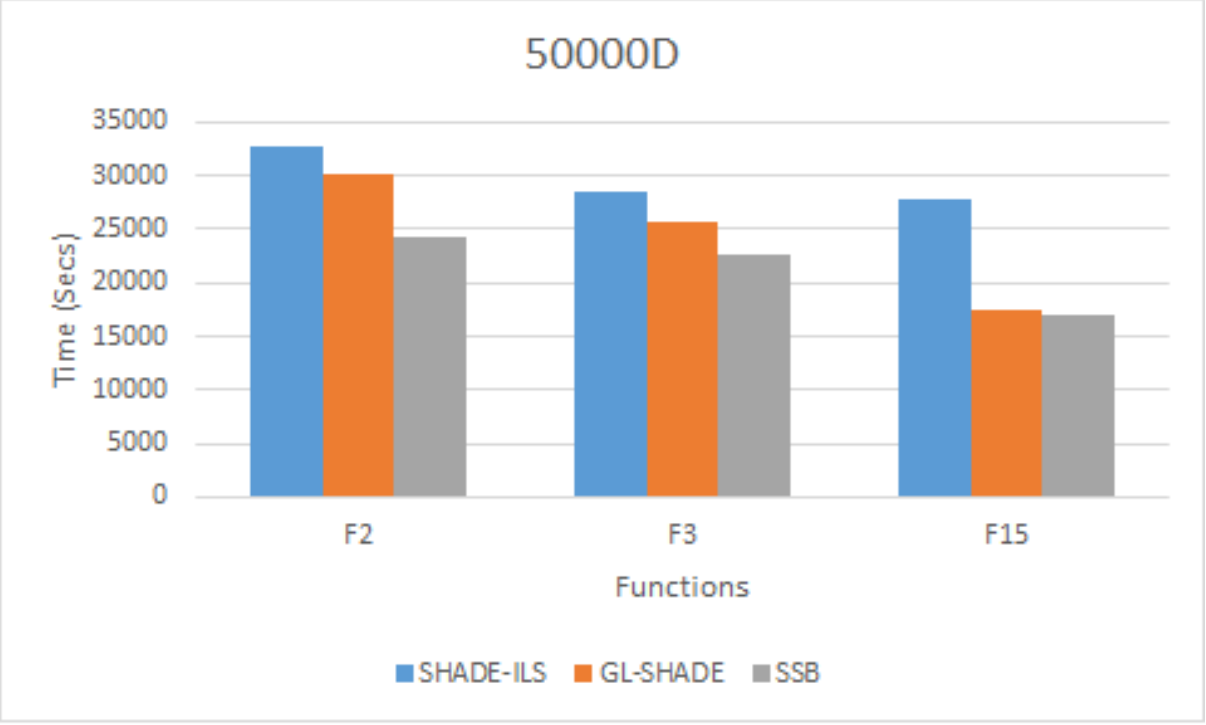


FIGURE 6.3: Time comparison of $f2$, $f3$, & $f15$ at 50000D

6.2.4 Effect of Islands

We have tested the behaviour of DistSSB using 10, 15, and 20 islands for $f3$, against 50000D. The results are shown in Figure. 6.4. Increasing the number of islands improves data parallelism across partitions but this decreases diversity at each partition and decreases the chances of finding optima during a migration interval. This is the

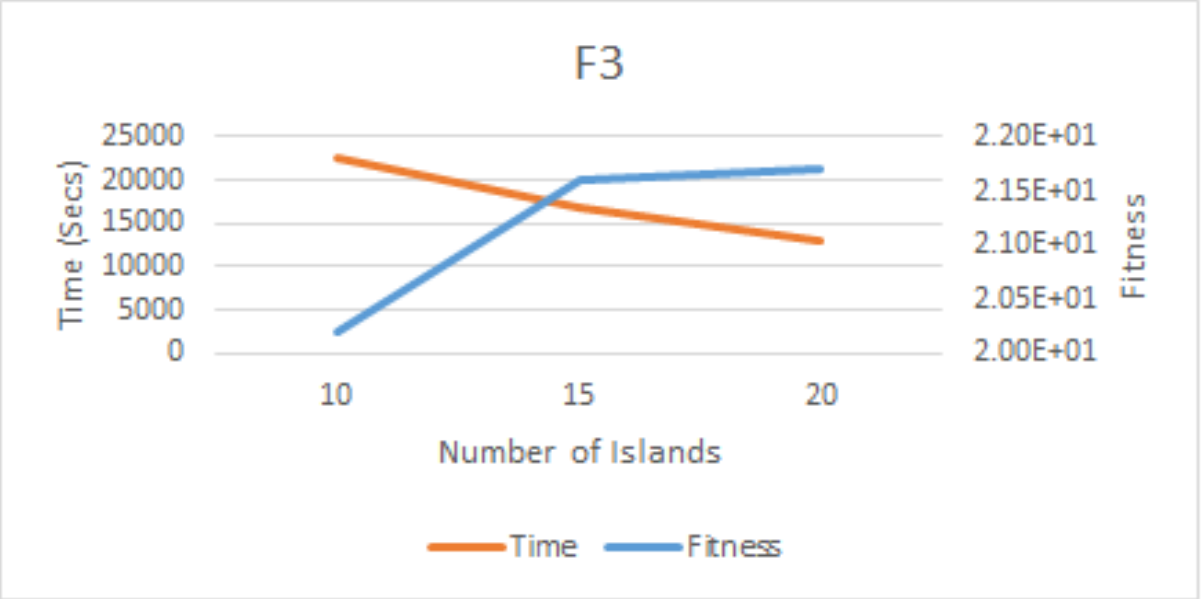


FIGURE 6.4: Behaviour of DistSSB, on 10,15 and 20 islands on $f3$ against 50000D

reason that $f3$ has got better optimization value on 10 islands. On the other hand, execution time decreases with the increase in the number of islands due to the increased parallelism. Execution time of DistSSB for $f1, f2, f3, f12$, and $f15$ against 50000D and varying islands also support the argument as shown in Figure. 6.5. The number of islands may be carefully decided keeping in view the trade off between execution time and optimization value.

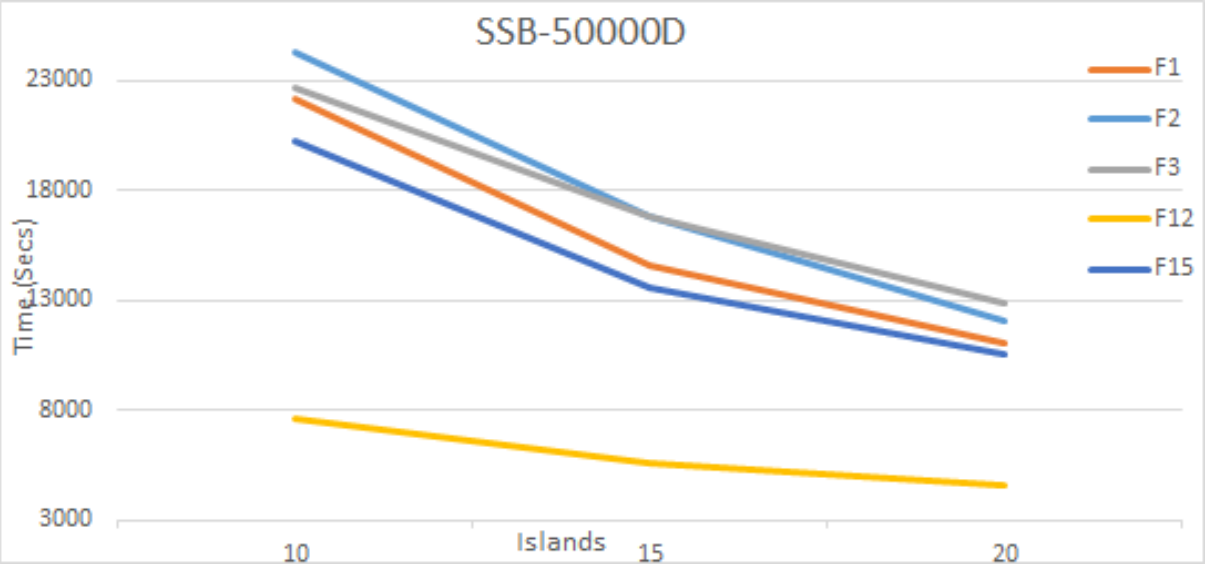


FIGURE 6.5: Execution time of DistSSB for 50000D

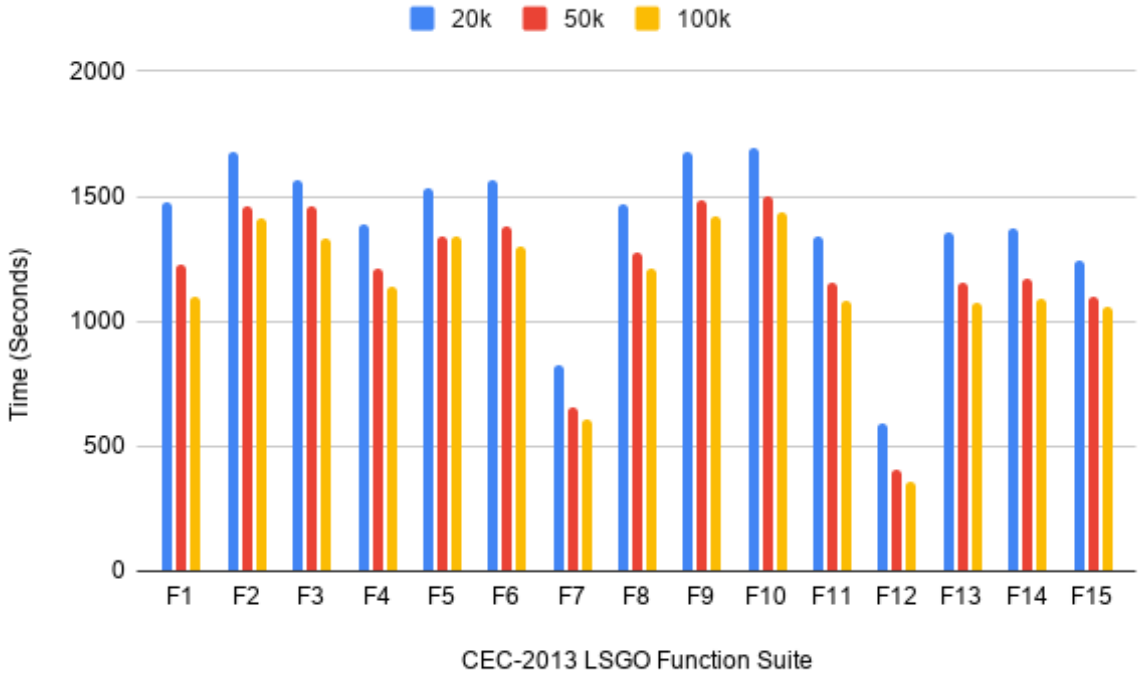


FIGURE 6.6: Execution time of DistSSB for 20k, 50k and 100k migration interval on 1000D

TABLE 6.3: Algorithms ranking using Friedman statistical ranking

Dimensions	DistSSB	GL-SHADE	SHADE-ILS
1000	1.50	1.93	2.56
2000	1.46	2.26	2.26

6.2.5 Effect of Migration Interval

Migration interval of 20K, 50K, and 100K has been used to study the impact on optimization value and execution time. Table 6.5 shows that decreasing migration interval results in improved optimization value in most of the cases. Once solutions at a partition are stuck in local optima, then more iterations will be wasted in case of higher migration intervals, hence effecting the overall convergence towards global optima. On the other hand higher migration interval improves the execution time as shown in Figure. 6.6 and Figure. 6.7. This is due to the reason that higher migration interval results in reduced network overhead for constant island size.

TABLE 6.4: Fitness Value of $f12$ on 10^6 dimensions

DistSSB	GL-SHADE	SHADE-ILS
2.73E+01	Out of memory exception	Unable to allocate memory

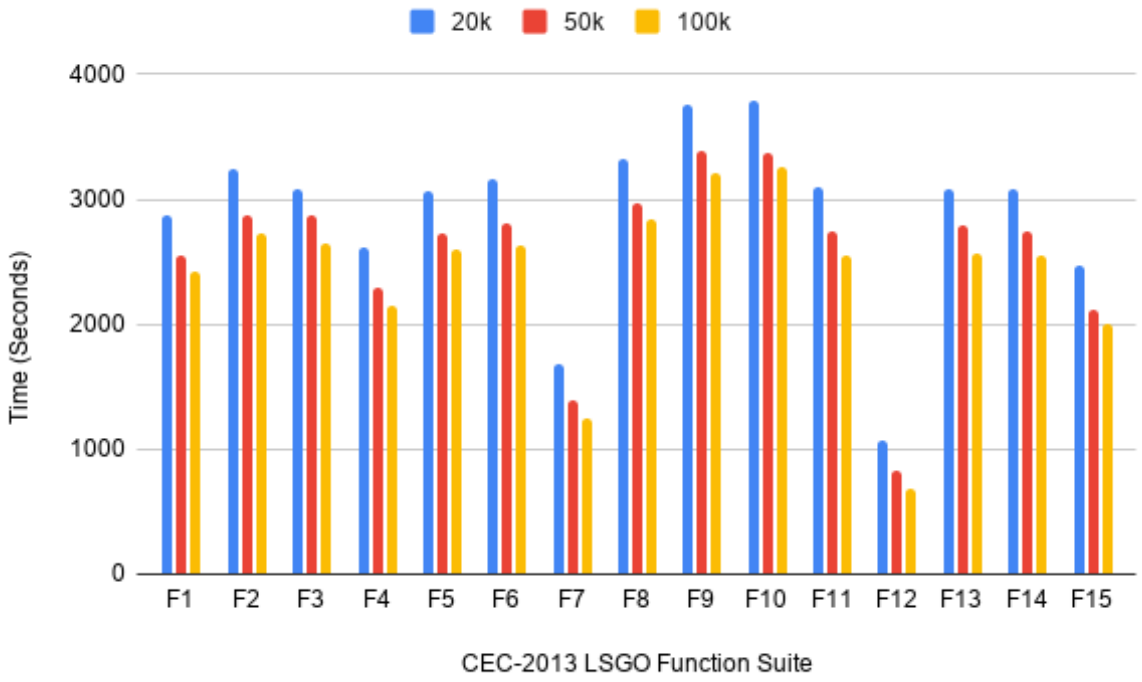


FIGURE 6.7: Execution time of DistSSB for 20, 50 & 100k migration interval on 2000D

TABLE 6.5: Average fitness comparison over 25 runs using 10^7 iterations, 1 migration rate, and 10 islands

f	1000D			2000D		
	M_i: 20k	M_i: 50k	M_i: 100k	M_i: 20k	M_i: 50k	M_i: 100k
$f1$	4.13E-18	1.21E-20	**1.58E-25**	1.03E+00	1.34E-08	**6.77E-18**
$f2$	1.25E+03	6.74E+02	**3.67E+02**	1.15E+04	1.96E+03	**1.40E+03**
$f3$	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01
$f4$	1.18E+08	2.19E+08	**7.68E+07**	**1.83E+09**	3.18E+09	3.67E+09
$f5$	3.66E+06	**2.95E+06**	3.39E+06	**8.24E+06**	7.76E+06	9.15E+06
$f6$	**9.98E+05**	9.99E+05	1.03E+06	1.04E+06	1.05E+06	1.05E+06
$f7$	**4.16E-05**	1.76E-03	1.01E-01	**1.70E+05**	5.52E+05	3.48E+06
$f8$	**1.47E+09**	3.76E+09	2.63E+09	**1.26E+12**	2.12E+12	1.41E+12
$f9$	2.64E+08	2.70E+08	**2.43E+08**	**6.07E+08**	7.07E+08	6.63E+08
$f10$	**9.05E+07**	9.08E+07	9.26E+07	9.18E+07	**9.05E+07**	9.06E+07
$f11$	9.25E+05	**1.70E+06**	2.77E+06	**1.57E+08**	2.11E+08	4.10E+08
$f12$	2.16E+03	2.15E+03	**2.05E+03**	7.94E+04	3.45E+05	**7.27E+04**
$f13$	**3.40E+05**	6.66E+05	1.79E+06	**1.35E+08**	1.41E+08	1.06E+09
$f14$	**7.06E+06**	8.29E+06	8.37E+06	**1.64E+08**	1.74E+08	2.99E+08
$f15$	**4.90E+05**	1.04E+06	1.40E+06	**6.83E+06**	1.42E+07	1.91E+07

6.3 Summary

Various evolutionary techniques have been proposed to solve LSGO problems. However, the evaluations of these methods have remained limited. While the problem sizes are increasing with time, scalable and distributed techniques are required to solve ever-growing LSGO problems. In this paper, we have proposed DistSSB, a scalable and distributed technique to solve LSGO problems. DistSSB uses Apache Spark for scalability and offers efficient exploration and exploitation using SHADE and BAT algorithms. We have compared the performance of DistSSB for varying number of dimensions over functions of different complexity with two state-of-the-art algorithms GL-SHADE and SHADE-ILS. DistSSB outperformed both algorithms in scalability while delivering comparable results. We have tested DistSSB for up to one million dimensions, while the other two algorithms failed to scale. It should be noted that the execution time of DistSSB remains reasonable for a million dimensions.

Chapter 7

FAIR-GA: FAIRy Tale of Genetic Algorithm

Traditional optimization techniques (random search, univariate method, stochastic gradient decent, quasi-Newton) are good at solving simple optimization problems (Chambers, 2019, Mitchell, 1998). However, they suffer from the performance bottleneck with an increase in the problem complexity. Also, these techniques require a well-defined deterministic path at the start. On the other hand, stochastic optimization techniques like GA perform well with non-smooth and ill-conditioned objective functions. It is capable to find good solutions while avoiding local optima. GA is based on the idea of "Survival of the fittest". Given a population, it has three main operators i.e. Selection, Mutation, and Crossover. Selection chooses potentially promising solutions to proceed to the next generation, while crossover combines the traits of parent chromosomes to create the offsprings. Mutation changes a certain value of a gene within a chromosome, and this helps in avoiding local optima. GA has a wide application range including scheduling, planning, assignment, and prediction in various industry and business problems (Liu and Wang, 2015, Trivedi et al., 2015).

Currently, we are in the middle of the golden jubilee and diamond jubilee of Genetic Algorithms but far away from the standardization of GA. Even after 50+ years, we are unable to decide and agree on the name that corresponds to a particular set of hyperparameters. Genetic Algorithm was the term coined by John Holland in 1965 (Chambers, 2019, Mitchell, 1998). Since then, Genetic Algorithms (Chambers, 2019), Simple Genetic Algorithms (Mitchell, 1998), Canonical Genetic Algorithms (Chambers, 2019), and Sequential Genetic Algorithms (Mitchell, 1998) are the several names given to GA. One might be confused if these are different GA variants, but these all are the same and refer to John Holland's GA. Similar is the case with GA software (source code of a GA research publication). One may find different code repositories of GA[1]

[1] https://github.com/bz51/GeneticAlgorithm

[2], Simple GA Simple Genetic Algorithm[3] [4] [5] [6], Canonical Genetic Algorithm[7] [8] [9] [10] [11] [12] , and Sequential Genetic Algorithm[13] on GitHub[14].These various implementations of the same approach with different naming conventions may decrease the findability, accessibility and reusability of GA. Also, one may find various implementations of the GA[15] [16] [17] [18] [19] with different hyperparameters but with the same naming conventions. This make it difficult in understandability and reusability.

A motivation behind this work is that a considerable portion of scientific data and research manuscripts remain unnoticed every year due to partial findability, accessibility, reusability, and interoperability by humans or machines (Fantin, 2020, Guizzardi, 2020, Hasnain and Rebholz-Schuhmann, 2018, Stall et al., 2019, Vogt, 2019, Vogt et al., 2019, Wilkinson et al., 2016, 2018). Only one-fifth of the published manuscripts also publish experimental data on some data repositories (Stall et al., 2019). In current research practices, most of the data used in research articles is not findable. Hence, it cannot easily be reused by the research community. Similarly, the act of sharing research software (i.e. code and hyper-parameter settings) is not a common practice due to little to no attribution mechanisms for the research software developers (Hasselbring et al., 2020). In most cases, research software's details are very briefly shared in research manuscripts and hence are far from being findable and reproducible. Same naming conventions of different digital artifacts, different naming convention of same digital artifact, ignored practices of publishing dataset and software code with research articles, sharing code in repositories without rich metadata adds a challenge to code findability and hence compromises the algorithm reusability and reproducibility. In recent years, efforts have been made in research, academia,

[2] https://github.com/ezstoltz/genetic-algorithm
[3] https://github.com/tmsquill/simple-ga
[4] https://github.com/yetanotherchris/SimpleGeneticAlgorithm
[5] https://github.com/afiskon/simple-genetic-algorithm
[6] https://github.com/ajlopez/SimpleGA
[7] https://github.com/GMTurbo/canonical-ga
[8] https://github.com/nanoff/Canonical-Genetic-Algorithm
[9] https://github.com/sanamadanii/Canonical-Genetic-Algorithm
[10] https://github.com/sajjadaemmi/Canonical-Genetic-Algorithm
[11] https://github.com/yareddada/Canonical-Genetic-Algorithm
[12] https://github.com/UristMcMiner/canonical_genetic_algorithm
[13] https://github.com/regicsf2010/SequentialGA
[14] https://guides.github.com/features/pages/
[15] https://github.com/ezstoltz/genetic-algorithm
[16] https://github.com/strawberry-magic-pocket/Genetic-Algorithm
[17] https://github.com/memento/GeneticAlgorithm
[18] https://github.com/ShiSanChuan/GeneticAlgorithm
[19] https://github.com/lagodiuk/genetic-algorithm

development, and industry to make scientific data FAIR (Findable, Accessible, Interoperable, Reusable) for both humans and machines (Stall et al., 2019, Wilkinson et al., 2016). Table 7.1 briefly covers FAIR data principles proposed by Wilkinson et al. (Wilkinson et al., 2016) in 2016 for making data FAIR. These principles focus on machine action-ability with minimum to nil human intervention.

TABLE 7.1: FAIR principles for Data

FAIR	Id	Description
F	1	metadata are assigned a globally unique and persistent identifier.
	2	data are described with rich metadata.
	3	metadata clearly and explicitly include the identifier of the data it describes.
	4	metadata are registered or indexed in a searchable resource.
A	1	metadata are retrievable by their identifier using a standardized Communications protocol.
	1.1	the protocol is open, free, and universally implementable.
	1.2	the protocol allows for an authentication and authorization procedure, where necessary.
	2	metadata are accessible, even when the data are no longer available.
I	1	metadata use a formal, accessible, shared, and broadly applicable language to facilitate machine readability and data exchange.
	2	metadata use vocabularies that follow FAIR principles.
	3	metadata include qualified references to other (meta)data.
R	1	metadata are richly described with a plurality of accurate and relevant attributes.
	1.1	metadata are released with a clear, and accessible data usage license.
	1.2	metadata are associated with detailed provenance.
	1.3	metadata meet domain-relevant community standards.

FAIR principles revolve around three main components (i.e Digital Artifact, Metadata about the digital artifact and Infrastructure). The FAIR guidelines emphasize automated discovery (Findability) of the digital artifact (mainly data). Once discovered one should have a clear idea of how these artifacts can be accessed including authentication and authorization. Metadata should be well defined to assist reusability. Data is more productive if its accessibility, interoperability, and reusability details are clearly documented in its metadata. This chapter addresses the following research questions.

RQ4. Can we reproduce the results of EA to improve the reusability?

RQ5. Can we make algorithms FAIR?

The contributions of this chapter are as follows:

1. We have extended FAIR principles beyond data, so that these could be applied to methods, algorithms and software artifacts.
2. We have presented GA as a usecase to demonstrate the applicability of proposed FAIR principles for algorithms.

3. We have proposed specialized metadata for GA to ensure FAIR practice using light weight RDF format.
4. We demonstrate the application of proposed principles for a Python based GA code [20] and published its associated metadata [21] through zenodo.

This chapter is based on the following article.

- **Fahad Maqbool**, Saad Razzaq, Hajira Jabeen. "FAIRy Tale of Genetic Algorithms." submitted in Applied Soft Computing, Elsevier, 2023.

The rest of the chapter is arranged as following. In Section 7.1, Genetic Algorithm is explained and need of making it FAIR. Section 7.2, briefed about FAIR principles. Section 7.3, covers the FAIR common and exclusive principles for algorithms and GA, while section 7.4, presents the metadata of GA. Usecase of mapping of FAIR Algorithms principles on GA is highlighted in Section 7.5. In section 7.6, we have summarized this chapter.

7.1 Genetic Algorithm (GA)

GA is an evolutionary algorithm that has gained much importance in the last few decades due to its simplicity and effectiveness for complex optimization problems. GA is a directed randomization technique based on Charles Darwin's theory of "Natural's Selection" Chambers (2019), Mitchell (1998). Randomization helps GA to avoid local optima while the directed approach helps to converge to an optimal solution. GA uses stochastic operators (i.e. crossover and mutation) that helps to explore the search space and exploit the solutions respectively. GA starts by initializing a population of candidate solutions. Each candidate solution represents a string of feature/decision variables. The population is evolved by applying GA operators on the candidate solutions. The fitness of the candidate solution is evaluated using a fitness function that is mainly problem dependent. The termination criteria is based on the maximum number of generations, the maximum amount of time, or the specified convergence criteria. Different variants of GA (i.e Sequential GA, Parallel GA and Distributed GA) are briefly explained in Table 7.2. GA has different population initialization methods (i.e Random, Feasible Individuals, and Random and Greedy) as explained in Table

[20] https://doi.org/10.5281/zenodo.7096663
[21] https://doi.org/10.5281/zenodo.7095155

7.3. Moreover GA has also different population structures and how individual solutions communicate with each other as shown in Table 7.4.

Table 7.2: Different variants of GA

S.NO	GA Variant	GA Variant Detail	Reference
1	Sequential GA	It starts with a single population of solutions and evolves it over time by applying GA operators. The process continues until the desired convergence, or required generations are reached.	Chambers (2019), Mitchell (1998)
2	Parallel GA	The initial population is divided into subpopulations. Multiple GA operations are performed in parallel like fitness evaluation, selection, crossover, and mutation.	Cantú-Paz et al. (1998), Ferrucci et al. (2018), Lim et al. (2007), Liu and Wang (2015), Qi et al. (2016), Trivedi et al. (2015)
3	Distributed GA	In this variant, dimensions of individuals or population are distributed. For dimension distribution multi-agent and coevolution methods are used. In population distribution Island, Hierarchical, Master slave, Cellular, and Pool model are used.	Akopov and Hevencev (2013), Alba et al. (2005), Artyushenko (2009), Cao et al. (2017), Dubreuil et al. (2006), Eklund (2004), Folino et al. (2008), Maqbool et al. (2019b), Roy et al. (2009), Sefrioui and Périaux (2000)

While working with GA, researchers must carefully select and specify essential parameters like population initialization, population structure, encoding scheme, selection criteria, crossover technique, crossover rate, mutation rate, mutation technique, and replacement criteria as shown in detail in Figure 7.1. The suggested details helps the researchers to express GA metadata more appropriately and use the existing GA techniques to reproduce the results effectively.

Table 7.3: Population initialization methods in GA

S.NO	Population Initialization	Initialization details	Reference
1	Random	The initial population is randomly selected without any heuristic and constraint.	Li (2009), Salhi and Petch (2007), Skok et al. (2000)
2	Feasible Individuals	The initial population contains selected/possible individuals as an initial population set.	Alvarenga et al. (2007), Ghoseiri and Ghannadpour (2010)
3	Random and Greedy	The initial population is based on a random and greedy mixed approach.	Ombuki et al. (2006), Santos et al. (2006)

We have also performed a limited survey to support our claim that most of the GA algorithms are not FAIR. We only selected top 50 articles against the keyword search[22] "Genetic Algorithm" from Google Scholar, published in year 2021. From 50 articles, only 03 articles Hooft et al. (2021), Oliveira et al. (2021), Weißbrich et al. (2021) mention the source code [23], [24], [25] of their proposed technique. Moreover most of the GA-based code repositories available on Github [1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14] do not provide the required hyper-parameter settings, configuration parameters, and metadata details. We reiterate here that the purpose of this survey is neither to perform a exhaustive overview of Genetic Algorithm, nor to provide a thorough understanding of the algorithm. Rather it is aimed to merely highlight the main challenges in findability and reproducibility of GA research software by selecting a sample. Another challenge to the findability is multiple naming conventions of GA and its variants available in literature Hooft et al. (2021), Oliveira et al. (2021). Researchers may intermix multiple conventions leading to poor findability.

7.2 FAIR

The journey of making data FAIR (Findable, Accessible, Interoperable, Reusable) data started from the guidelines initially proposed by Wilkinson et al. Wilkinson et al. (2016) enlisted in Table 7.1. Research communities were suggested to follow the guidelines and agree upon common data and metadata storage framework. FAIR not only helps

[22] https://scholar.google.com/scholar?as_ylo=2021&q=Genetic+Algorithm&hl=en&as_sdt=0,5 Accessed: 30-08-2022, 12:00
[23] https://github.com/Ensing-Laboratory/FABULOUS
[24] https://github.com/tubs-eis/VANAGA
[25] http://mauricio.resende.info/src/coEvolBrkgaAPI

Table 7.4: Population structures in different variants of GA

S.NO	Population Structure	Description	Reference
1	Conventional GA	A combined population pool where each individual can interact with any other individual.	Lim (2014)
2	Island Model	The initial population is divided into multiple subpopulations/islands. On each island, GA operators work independently.	Artyushenko (2009)
3	Cellular Model	Each individual can interact within a defined small neighborhood, and GA operations are applied to them.	Alba et al. (2005)
4	Terrain-Based	Parameters are available across the population. At each generation, an individual can interact with the best individual in a close neighborhood.	Krink et al. (1999)
5	Spatially-Dispersed	Once the first individual/parent is selected, the second individual is chosen based on its spatial coordinates visibility from the first individual/parent.	Dick (2003)
6	Multilevel Cooperative	The population is divided into multiple groups. Offsprings in sub populations evolve and are updated by replacing weak individuals.	Reza et al. (2010)

FIGURE 7.1: Detailed metadata parameters to improve the reusability and reproducibility of GA.

the researchers to get the maximum potential (by attracting new research partnerships and increasing citations/visibility) from the data set. It also helps in to improve the reusability and reproducibility of the data by building novel resources and tools

by taking maximum benefits from the existing datasets. FAIRification of data is based on four key components i.e Findable, Accessible, Interoperable, and Reusable.

Findability suggests that the necessary practices that should be carried out to make your data and metadata easily findable by both man and machines. In order to ensure data availability, it should be placed in such a way that every element of data and metadata is accessible using a unique and persistent URI. This will help to avoid the ambiguity about the elements. Search engines are the key source of information nowadays. In order to make data/metadata findable by the search engines, it should not only be placed on index-able sources so that search engine bots may read and index them in their SERP (Search Engine Result Pages) but also metadata should be rich enough so that it may contain the necessary information/keywords based on which you want to be found in search engines. Including data identifiers in metadata file will help increase clarification about the data for which metadata is being defined.

Accessibility doesn't state that data should be freely accessible to everyone but rather there should be a clear machine readable guidelines available that states who can access the data. Data accessibility using standard communications protocols (e.g. HTTP, FTP, SMTP) not only ensures data reusability but also help to increase this if the protocols are available free of cost. Data storage carries a cost and may become unavailable overtime. Metadata should even remain available in case of broken links to dataset so that if someone interested in the dataset then he/she may track the publisher or author of the dataset using the metadata.

To ensure the interoperability, a formal, shared, and broadly applicable, metadata format is required, that follows standard vocabularies and qualified references to other metadata. Interoperability helps machine to understand exchanged data format from other machines with the help of standardized ontology and vocabularies. In reusability, metadata is enriched with detailed attributes about the data along with a clear data usage license. Complete details and conditions under which data was generated through experiments/sensors/machine/protocol and citation details is mentioned in metadata. In this article we have adapted and extended FAIR data principles for algorithms(FAIR-Algorithms). Later we have presented a usecase by applying FAIR-Algorithms using GA.

7.3 FAIR-Algorithms: FAIR Principles for Algorithms

An algorithm is a set of instructions to complete a specific task. It is usually designed to solve a specialized problem / sub-problem and consists of inputs, tasks, outputs, and parameters settings. Inputs are the data and parameters, while the task is the main description of the work that uses inputs to generate outputs(reports, computational outcome, models).

Sharing the data, research software, algorithm and related metadata is required in improving, reusing, or reproducing algorithms with a purpose to improve efficiency, efficacy, or resource utilization. Hence a recent trend of developing FAIR principles for software may be of interest to those researchers who view software as a black box or an atomic entity. This motivated us to work on FAIR guidelines for algorithms.

The idea behind FAIR-Algorithms is that if a research is FAIR then not only others should be able to reproduce data and software but also should be able to reproduce, reuse, extend, or build on top of the algorithm.

We have enlisted FAIR principles for algorithms in Table 7.5 and mentioned the action (i.e. adapted, and extended) against each principle. Adapted is used where FAIR data principle is followed for algorithm and extended is used when existing FAIR principles is modified to cover the algorithm and its related details. FAIR-Algorithms guidelines are presented in Table 7.5.

7.3.1 Findability

F1:- Algorithm and its metadata is assigned a globally unique and persistent identifier.

Existing digital artifact repositories (i.e. Github[26], GitLab[27], Zenodo[28], SourceForge[29], Bitbucket[30]) do not assign a unique identifier to the algorithm, rather to a software repository that may contain one or more algorithms. Therefore it is recommended that there should be a unique metadata file related to each algorithm. Also a unique identifier should be assigned to each metadata file and algorithm. An algorithm has a

[26] https://github.com/
[27] https://gitlab.com/
[28] https://zenodo.org/
[29] https://sourceforge.net/
[30] https://bitbucket.org/

TABLE 7.5: FAIR principles for Algorithms

FAIR	ID	FAIR for Algorithms	Action
F	1	Algorithm and its metadata is assigned a globally unique and persistent identifier.	extended
	2	Algorithms are described with rich metadata.	adapted
	3	Algorithm metadata clearly and explicitly include the identifier of the algorithm it describes.	extended
	4	Algorithm metadata are indexed on a searchable repository.	extended
A	1	Algorithm metadata are retrievable by their identifier using a standardized communications protocol.	adapted
	1.1	The protocol is open, free, and universally implementable.	adapted
	1.2	The protocol allows free and easy access to algorithms' metadata and details.	extended
	2	Algorithm-metadata remains available, even when the algorithms are modified.	extended
I	1	Algorithm metadata uses a formal, accessible, shared, and broadly applicable language for knowledge representation.	adapted
	2	Metadata uses vocabulary that follows FAIR principles.	adapted
	3	Algorithm metadata include qualified references to other metadata.	adapted
R	1	Algorithm metadata is richly described with a plurality of accurate and relevant attributes.	adapted
	1.1	Usually, algorithm metadata is freely accessible. However, if required, algorithm metadata is released with a clear and accessible usage license.	extended
	1.2	Algorithm metadata includes detailed provenance. It includes its basics, execution and performance attributes	extended
	1.3	Algorithm metadata meet domain-relevant community standards.	adapted

unique identifier, If its sub algorithms have no unique ID (UID), and the owner wants to assign UID, he can opt to do so. On the other hand, if an algorithms is used in another algorithms it UID will be used, and the new/parent algorithms will be assigned a new UID. We do not imagine allocation of UIDs retrospectively. Moreover different implementation of same algorithms have different UIds and in case of updates in an implementation its versioning control should also be maintained.

F2:- Algorithms are described with rich metadata

The metadata for an algorithm includes its input, tasks/steps, output, implementation details, parameter settings, execution environment, and execution duration. We have extended the MEX vocabulary Esteves et al. (2015) to define the metadata for algorithms as given in Table 7.6. Algorithm's metadata file also includes the UID of the algorithm in it.

F3:- Algorithm metadata clearly and explicitly include the identifier of the algorithm it describes.

Currently there is no practice of publishing algorithm's metadata. However we have recommended that algorithm's metadata should clearly and explicitly point to the algorithm that is being described, these identifiers include the identifier, author, usage

information, citation, and other related properties as suggested in metadata for algorithms in Table 7.6.

F4:- Algorithm metadata are indexed on a searchable repository
Publishing algorithm's metadata is not in practice and hence not indexed. Therefore, we have suggested that algorithm's metadata should be placed on digital artifact repositories (i.e Zenodo, Github, GitLab, and BitBucket) that quickly index the published resources and make them searchable.

7.3.2 Accessability

A1:- Algorithm metadata are retrievable by their identifier using a standardized Communications protocol.
Existing artifact repositories (i.e Zenodo, Github, GitLab, and BitBucket) are accessible using standard communication protocols like http/https. So, metadata placed on these repositories is also accessible. Hence, we recommend to use these for algorithms.

A1.1:- The protocol is open, free, and universally implementable.
Generally, the https protocols are open, free, and universally used. The algorithms that are published on above mentioned digital artifact repositories are using these free access protocols. In case of private publishing of Algorithms or its metadata, the metadata must be made accessible through universally acceptable protocols.

A1.2:- The protocol allows free and easy access to algorithms' metadata and details.

Mostly algorithms don't have authentication and authorization issues. In case of privacy related issues in publishing a particular algorithm, the metadata should still remain accessible, even after following an authentication and authorization procedure. However, the authentication and authorization procedure may be adapted where necessary.

A2:- Algorithm-metadata remains available, even when the algorithms are modified.

New variants of algorithms are proposed over time and older variants may reduce their public visibility. Therefore, metadata for the earlier versions of an algorithm should remain accessible and available even after its new version or extension has become more prevalent.

7.3.3 Interoperability

I1:- Algorithm metadata uses a formal, accessible, shared, and broadly applicable language for knowledge representation

Current artifacts repositories support XML[31], JSON[32], JSON-LD[33], and Rest APIs[34] as broadly applicable languages. Therefore we recommend to use above mentioned broadly applicable formats for algorithm's metadata.

I2:- Metadata uses vocabularies that follow FAIR principles.

Fairification of vocabularies to define algorithm's metadata have been under explored. In this regards, we have suggested to use and extend (where necessary) MEX Vocabulary (comprising $mexcore$[35], $mexalgo$[36], $mexperf$[37]) Esteves et al. (2015) for algorithm metadata as it maximally satisfies FAIR principles. $mexcore : Context$, $mexalgo : AlgorithmClass$, $mexcore : model$, $mexalgo : AlgorithmParameter$, and $mexalgo : Implementation$ are few of the main classes of the vocabulary.

I3:- Algorithm metadata include qualified references to other metadata.

Currently there is no practice of publishing algorithms metadata. Once it is started, focus on referencing other related metadata would also be in practice. It will help in increasing algorithm reusability.

7.3.4 Reusability

R1:- Algorithm metadata is richly described with a plurality of accurate and relevant attributes.

[31] https://www.w3.org/XML/
[32] https://www.json.org/
[33] https://json-ld.org/
[34] https://restfulapi.net/
[35] https://github.com/mexplatform/mex-vocabulary/blob/master/vocabulary/mexcore.ttl
[36] https://raw.githubusercontent.com/mexplatform/mex-vocabulary/master/vocabulary/mexalgo.ttl
[37] https://github.com/mexplatform/mex-vocabulary/blob/master/vocabulary/mexperf.ttl

Rich metadata is helpful in understanding an algorithm. We have combined metadata specifications in detail in Table 7.6. It includes basic attributes from schema[38]. Algorithm, parameters, learning methods, tool, class from $mexalgo$. Performance measure and user defined measures from $mexperf$. All these details $mexcore$, $mexalgo$ and $mexperf$ are taken from mex vocabulary Esteves et al. (2015). Algorithm metadata is richly described by following these detailed set of attributes.

R1.1:- Usually, algorithm metadata is freely accessible. However, if required, algorithm metadata is released with a clear and accessible usage license.

Copyright protects the creative work or expression of ideas but not the ideas themselves. An algorithm is an abstract idea and not subject to licensing Cormen et al. (2009). However, the code of the algorithm should have licensing information. An algorithm code is part of the research software, so the algorithm's usage license is as per the discretion of the research software team.

R1.2:- Algorithm metadata includes detailed provenance. It includes its basics, execution and performance attributes

Currently metadata specification for algorithm is not in practice. We have specified metadata specifications in detail in Table 7.6. These metadata specifications helps in making algorithm FAIR.

R1.3:- Algorithm metadata meet domain-relevant community standards.

To the best of our knowledge, there doesn't exist any algorithm relevant community standards. Therefore it is suggested to follow the guidelines from algorithm related vocabularies and ontologies.

7.4 Metadata for Genetic Algorithm

Rich metadata of a digital artifact specified using a well known data format plays an important role in machine readability. Also it plays a vital role in reusability and reproducibility using well defined hyper-parameter values. Algorithm metadata based on mex-vocabulary covers all general-purpose attributes(like tools, dataset, feature etc) that are common among algorithms. Defining GA specific attributes (like population size, fitness function, crossover rate) is not workable using mex-vocabulary and demands an extension of attributes in mex-vocabulary. In this section we have

[38] https://schema.org/

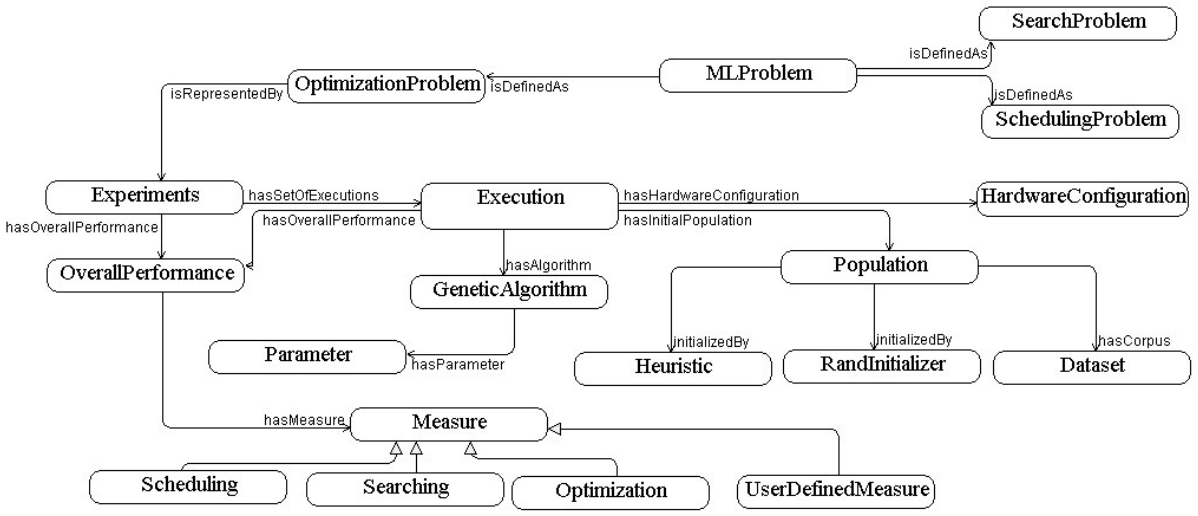


FIGURE 7.2: GA iteration cycle starting from the problem specification till the performance measures.

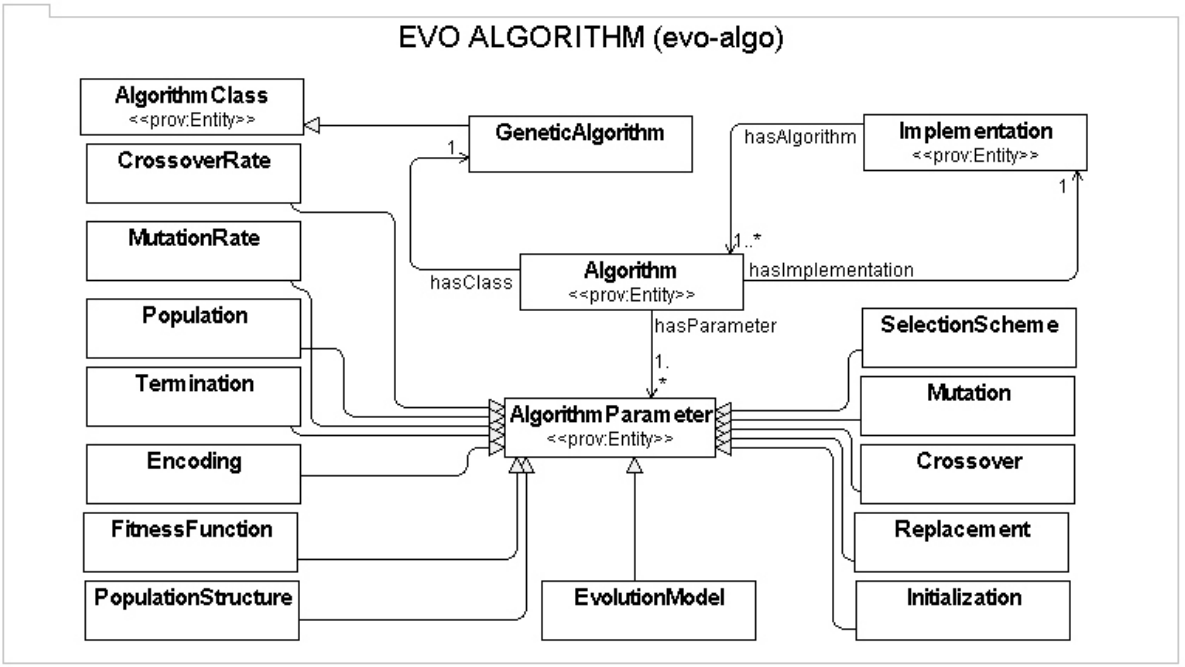


FIGURE 7.3: Algorithm related parameter specification for GA

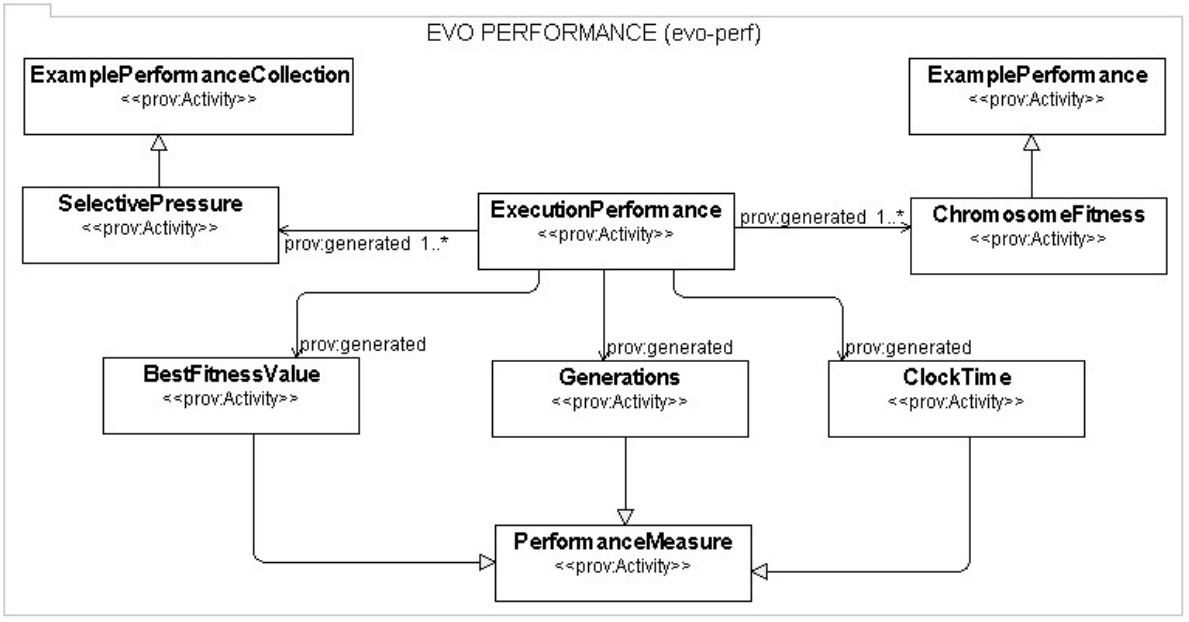


FIGURE 7.4: Performance related parameter specification for GA

have suggested specific metadata [21] for GA as listed in table 7.7. Listing 7.1 shows a metadata of a Python based GA [39]. We have represented this metadata using minimal classes and properties from prov-o[40], mex-core[41], mex-algo[42], and mex-perf[43]. Moreover we also suggest to improve upon MEX vocabulary and add few more terms to metadata related to GA as listed below. A hierarchical representation of these parameters has been shown in Figure 7.1.

1. $evo : SelectionScheme$
2. $evo : Encoding$
3. $evo : FitnessFunction$
4. $evo : Error/LossFunction$
5. $evo : PopulationSize$
6. $evo : Crossover$
7. $evo : CrossoverPoint$
8. $evo : CrossoverProbability$
9. $evo : Mutation$
10. $evo : MutationProbability$
11. $evo : Replacement$

[39] https://colab.research.google.com/drive/1t_kUu6l3a4F1sP6oK92CHZwqANsTMijP?usp=sharing
[40] https://www.w3.org/TR/prov-o/
[41] http://mex.aksw.org/mex-core#
[42] http://mex.aksw.org/mex-algo#
[43] http://mex.aksw.org/mex-perf#

12. $evo : Elitism$
13. $evo : Termination$
14. $evo : Generations$
15. $evo : Time$
16. $evo : Fitness$

Figure 7.1 shows the classical execution iteration of ML problem using GA. The iteration includes the problem (i.e. scheduling, searching, or optimization), GA, population (initialized using dataset, random initializer, or heuristic), input parameters, and performance measures as shown in Figure 7.3 and 7.4 respectively.

TABLE 7.6: Metadata specification for Algorithm.

Property	Description	Type
schema:identifier	DOI of algorithm	schema:URL
schema:url	URL of software repository	schema:URL
schema:name	Name of the algorithm	schema:Text
schema:description	Algorithm description that discusses Algorithm purpose and application areas.	schema:Text
schema:author	Person / organization that has created the code and holds its intellectual copyrights.	schema:Organization schema:Person
schema:usageInfo	Limitation of the algorithm.	schema:Text
schema:keywords	Keywords and tags that describe the key terms to define a software.	schema:Text
schema:citation	Source code attribution (i.e. link to the article where the particular algorithm has been discussed).	schema:Text
schema:license	It helps to protect the intellectual property by defining the guidelines for use and distribution of software.	schema:URL
mexcore:ApplicationContext	Basic information about algorithm that may provide a high-level overview (including goals, aims, objectives, and scope) of the software	mexcore:ApplicationContext { :trustyURI rdfs:Literal mexcore:trustyURIHash }
mexcore:Context	The problem for which algorithm has been employed e.g data clustering, neural network optimization, and protein folding.	mexcore:Context { prov:wasAttributedTo mexcore:ApplicationContext }
mexcore:Experiment	The class represents some basic information about the experiment.	mexcore:Experiment { mexcore: attributeSelectionDescription xsd:string :dataNormalizedDescription xsd:string :noiseRemovedDescription xsd:string :outliersRemovedDescription xsd:string }
mexcore:Execution	A single run of an algorithm-based program. Each run is based on specific parameter specification and hardware configurations.	mexcore:Execution { mexcore:endsAtPosition xsd:string :targetClass xsd:string prov:wasInformedBy mexcore:ExperimentConfiguration }

Metadata specification for Algorithm (Continued).		
mexcore:ExperimentConfiguration	It represents execution detail (on different algorithm configuration and hardware environments) of an experiment.	mexcore: ExperimentConfiguration { prov:wasStartedBy mexcore:Experiment }
mexcore:HardwareConfiguration	Detail about hardware configuration	mexcore: HardwareConfiguration { mexcore:cpu xsd:string mexcore:cpuCache xsd:String mexcore:hdType xsd:string mexcore:memory xsd:string :videoGraphs xsd:string }
mexcore:DataSet	Initial population/ dataset for algorithm experiments	owl:Class
mexcore:Example	An individual solution or a chromosome	mexcore:Example { mexcore:datasetColumn rdfs:Literal mexcore:datasetRow rdfs:Literal }
mexcore:ExampleCollection	ExampleCollection is a collection of chromosomes and represents a population at a particular generation.	mexcore:ExampleCollection { mexcore:startsAt rdfs:Literal mexcore:endsAt rdfs:Literal mexcore:hasPhase mexcore:Phase }
mexalgo:LearningMethod	This defines the learning approach of the algorithm i.e. evolution in case of genetic algorithm.	mexalgo:LearningMethod { mexalgo:isLearningMethodOf mexalgo:Algorithm }
mexalgo:LearningProblem	GA is a metaheuristic based algorithm	mexalgo:LearningProblem { :isLearningProblemOf :Algorithm }
mexalgo:AlgorithmClass	The algorithm class (e.g.:GeneticAlgorithm)	mexalgo:AlgorithmClass { :isAlgorithmClassOf :Algorithm }
mexalgo:AlgorithmParameter	The representation of GA parameter with its associated values (e.g. encoding, population, crossover scheme)	mexalgo:AlgorithmParameter { }
mexalgo:Tool	It describes the libraries for GA (e.g.: PyGAD, GAlib, GeneAI).	mexalgo:Implementation { }
mexperf:PerformanceMeasure	It describes the evaluation measure to check the performance of GA (e.g.: Fitness function).	mexperf:PerformanceMeasure { }
mexperf:UserDefinedMeasure	This property is used to mention domain relevant metrics.	mexperf:UserDefinedMeasure { mexperf:formula xsd:string }

TABLE 7.7: Proposed *evo* vocabulary for GA.

Property	Description	Type
evo:Initialization	Their are different initilization methods (i.e random, heuristic and dataset).	schema: Text
evo:Encoding	GA works on the encoding of the solutions rather than solutions. These may include binary encoding, value encoding, and permutation encoding.	schema: Text
evo:Bound	It represents the upper and lower bound values for each dimension in the state space.	schema: Text
evo:PopulationSize	Population size is generally dependent on the problem domain and may be decided by hit and trial method. It has a significant role to speed up the convergence.	schema: Number
evo:FitnessMeasure	It is used to represent the final fitness value.	schema: Number
evo:TimeMeasure	This represents the clock time used by the GA	schema: Duration
evo:GenerationMeasure	It represents the total generation consumed during the execution.	schema: Number
evo:Evolution	It represents the learning method used by the GA	schema: Text
evo:Crossover	Parent chromosomes recombine to create new offsprings. Crossover property is used to specify the crossover operator (i.e. single point crossover, multi-point crossover, uniform crossover, or three parent crossover).	schema: Text
evo:CrossoverRate	It is used to to decide the number of parents involved in the crossover process	schema: Text
evo:Mutation	Mutation operator helps to avoid getting stuck in local optima by maintaining the population diversity. Popular mutation includes bit flip, inversion, scramble, and random resetting.	schema: Text
evo:MutationRate	It is the frequency measure by which value of randomly selected genes would be modified	schema: Text
evo:Selection	Selection scheme specifies the criteria through which parent chromosome will be selected from the current generation to produce offsprings using crossover and mutation. This may include rank selection, roulette wheel selection, and tournament selection.	schema: Text
evo:PopulationUpdate	It can be steady state (i.e. off springs would be added to the population as they are created) or generational (i.e. off springs would be added to the population after the generation)	schema: Text
evo:Replacement	Replacement is the scheme/strategy (weak parent, both parent, random parent) through which parent chromosomes will be replaced by newly created offsprings.	schema: Text
evo:Termination	Termination specifies the criteria (i.e. generations, time, fitness, convergence) that stops the execution of genetic algorithm.	schema: Text
evo:MaxGenerations	It specify the maximum number of iterations after which genetic algorithm will be terminated	schema: Number

Proposed *evo* vocabulary for GA (Continued).		
evo:FitnessFunc	It is used to mention the fitness function name (e.g. Sphere, Ackley or Griewank). Fitness function takes a solution as input and evaluates how close a given solution is to the optimal solution.	schema: Text
evo:FitnessFuncDef	This class would be used to mention the fitness formula or fitness function definition.	schema: Text
evo:Time	Maximum amount of clock time after which we terminate the execution of genetic algorithm.	schema: Duration

7.5 $FAIR-GA$: FAIR Genetic Algorithm (A usecase of FAIR-Algorithms)

Genetic algorithm is a popular optimization algorithm that is very effective in solving complex optimization problems. It initially starts with a population of encoded solutions, evolve these solutions using stochastic reproduction operators (i.e. crossover & mutation), evaluate solutions using fitness function, eliminate less fit solutions, and proceeds with fittest solutions to next generations until termination criteria is met. Although GA and its variants have proved their significance in many optimization problems but lack of focus on reproducibility limits reusability of these techniques. In this section we have presented FAIR-GA (i.e. a use case of FAIR-Algorithms). This will not only help in supporting FAIR research culture but also helps researchers to increase their citation index and reusability of their proposed GA variants. Also, we have suggested guidelines that should be performed while mapping FAIR-Algorithms on GA (i.e. $FAIR-GA$). These steps relate to the application of $FAIR-Algorithms$ on GA with some customization in F2, I2, R1, and R1.3 as discussed below.

F2:- GA are described with rich metadata

Taking care of FAIR standards, we suggest that GA metadata should contain detailed information about the input, output, algorithm name, algorithm task, parameters, parameters settings, citation detail, hardware details, and software dependencies. Algorithms metadata as listed in Table 7.6 are not inlined with the vocabulary requirements related to GA. GA has more specialized attributes (like population initialization, population size, solution encoding, generations, and termination criteria). Therefore

proposed *evo* vocabulary illustrated in Table 7.7 has to be collectively used with metadata specifications for algorithm described in Table 7.6 to improve the reusability and reproducibility of GA.

I2:- GA software use vocabularies that follow FAIR principles

For Fair-GA, we have proposed *evo* vocabulary to support GA. The *evo* vocabulary comprises of twenty properties as illustrated in table 7.7 along with the description and type of each property.

R1:- Metadata are richly described with a plurality of accurate and relevant attributes

GA metadata must include accurate and relevant attributes. We have proposed extended metadata for GA as shown in Figure 7.1 and listed in *evo* vocabulary given in table 7.7. This is a set of relevant and related attributes and suggested to be the part of GA techniques.

R1.3:- Metadata meet domain-relevant community standards

Our proposed *evo* vocabulary is developed by looking into different state of the art GA variants. Detailed metadata has been suggested by taking care of all GA related hyperparameters configurations. However, we are unable to find any domain-relevant community standard for GA.

In Listing 7.1 we have applied, proposed *evo* vocabulary along with other suggested FAIR-Algorithms guidelines for representing the experiments of GA one max search optimization problem.

```
{
  {
  "@context": {
  "prov": "http://www.w3.org/ns/prov#",
  "mexperf": "http://mex.aksw.org/mex-perf#",
  "mexcore": "http://mex.aksw.org/mex-core#",
  "mexperf": "http://mex.aksw.org/mex-algo",
    "evo": "http://mex.aksw.org/evo"
  },
  "@id": "mexperf:ExecutionPerformance",
  "prov:generated": [
    {
    "@id": "evo:FitnessMeasure",
    "evo:hasFitness": "-20"
    },
```

```
    {
      "@id": "evo:TimeMeasure",
      "evo:elapsedTime": "3",
      "evo:timeUnit": "nsec"
    },
    {
      "@id": "evo:GenerationMeasure",
      "evo:generationCount": "8"
    }
                    ],


  "prov:wasInformedBy": {
    "@id": "mexcore:Execution",
    "prov:wasInformedBy": {
      "@id": "mexcore:ExperimentConfiguration",
      "prov:used": {
          "@id": "mexcore:HardwareConfiguration",
          "mexcore:hardDisk": "108GB",
          "mexcore:memory": "36GB"
        },
      "prov:wasStartedBy": {
      "@id": "mexcore:Experiment",
      "prov:wasAttributedTo": {
      "@id": "mexcore:ApplicationContext"
      }
      }
    },
    "prov:used": {
    "@id": "mexalgo:Algorithm",
      "schema:identifier": "https://doi.org/10.5281/zenodo.7096663",
      "schema:name": "Simple Genetic Algorithm",
      "schema:description": "A python implementation of a simple genetic algorithm to
  optimize the numerical functions.",
      "schema:author": [{
        "@id": "schema:Person",
        "name": "Saad Razzaq",
        "email": "saad.razzaq@uos.edu.pk"
      },
      {
        "@id": "schema:Person",
        "name": "Fahad Maqbool",
        "email": "fahad.maqbool@uos.edu.pk"
      },
      {
        "@id": "schema:Person",
        "name": "Hajira Jabeen",
        "email": "hajira.jabeen@gesis.org"
      }],
```

```
    "schema:keywords": "Evolutionary Optimization; Mutation; Crossover",
    "schema:license": "https://www.gnu.org/licenses/gpl-3.0-standalone.html",

    "mexalgo:hasClass": {
      "@id": "mexalgo:GeneticAlgorithms"
    },
    "mexalgo:hasLearningProblem": {
      "@id": "mexalgo:MetaHeuristic"
    },
    "mexalgo:hasLearningMethod": {
      "@id": "evo:Evolution"
    },
    "mexalgo:hasTool": {
      "@id": "mexalgo:Python"
    },
    "mexalgo:hasHyperParameter": [
      {
        "@id": "evo:Initialization",
        "prov:value": "Random"
      },
      {
        "@id": "evo:Bound",
        "prov:value": ["-5","+5"]
      },
      {
        "@id": "evo:Encoding",
        "prov:value": "Bit-String"
      },
      {
        "@id": "evo:PopulationSize",
        "prov:value": "100"
      },
      {
        "@id": "evo:Dimensions",
        "prov:value": "20"
      },
      {
        "@id": "evo:Crossover",
        "prov:value": "One-point Crossover"
      },
      {
        "@id": "evo:CrossoverRate",
        "prov:value": "0.9"
      },
      {
        "@id": "evo:Mutation",
        "prov:value": "Bit Flip"
      },
      {
```

```
113           "@id": "evo:MutationRate",
114           "prov:value": "1.0 / (float(n_bits) * len(bounds))"
115         },
116         {
117           "@id": "evo:Selection",
118           "prov:value": "Tournament"
119         },
120         {
121           "@id": "evo:PopultionUpdate",
122           "prov:value": "Generational"
123         },
124         {
125           "@id": "evo:Replacement",
126           "prov:value": "BothParent"
127         },
128         {
129           "@id": "evo:Termination",
130           "prov:value": "Generations"
131         },
132         {
133           "@id": "evo:MaxGenerations",
134           "prov:value": "100"
135         },
136         {
137           "@id": "evo:FitnessFunc",
138           "prov:value": "One-Max"
139         },
140         {
141           "@id": "evo:FitnessFuncDef",
142           "prov:value": "def onemax(x): return -sum(x)"
143         }
144       ]
145     }
146   }
147 }
148 }
```

Listing 7.1: RDF representation of GA experiments for solving one max search optimization problem using MEX vocabulary and *evo* vocabulary

7.6 Summary

We have extended FAIR principles beyond data, so that these could be applied to methods, algorithms and software artifacts. Our focus in this article is to ensure the reproducibility and reusability of algorithms. We have presented $FAIR - GA$ as

a usecase to demonstrate the applicability of the proposed principles for algorithms. Additionally, to ensure $FAIR-Algorithms$ we propose a metadata schema using light weight RDF format, facilitating the reproducibility. Finally, we have demonstrated the application of proposed $FAIR-GA$ using a python based GA code for solving one max search optimization problem. Moreover, we have also proposed a specialized vocabulary (i.e evo) for GA.

Chapter 8

Conclusion & Future Directions

In this thesis we have used different Evolutionary Algorithms for solving complex optimization problem like GCP and LSGO problems. We have proposed Blind Naked Mole Rat based Coloring (BNMR-Col) for graphs. BNMR-Col uses both exploitation and exploration to find the best solution in search space. Exploitation uses both local moves and global moves to find a better solution in the surroundings of an existing solution. On the other hand, exploration generates new solution by combining various solutions from the search space. BNMR-Col shows better convergence rate and approaches the lowest color value in 83% of the cases when tested on DIMACS graph instances.

We have proposed Scalable Distributed GA (S-GA) using Apache Spark for LSGO problems. The results have been compared with SeqGA. We have tested S-GA on continuous optimization problems functions for population size of up to 3000, Dimensions of up to 3000, Partition Size up to 30, migration size up to 03, and migration interval to 100000. For few cases S-GA has outperformed SeqGA for higher population, partitions, migration size, and migration interval in term of execution time. We have also proposed another evolutionary algorithm (DistSSB, a scalable and distributed technique) for solving LSGO problems as problem size is continuously increasing and we need more scalable and distributed algorithms. DistSSB uses Apache Spark for scalability and offers efficient exploration and exploitation using SHADE and BAT algorithms. We have compared the performance of DistSSB for varying number of dimensions over functions of different complexity with two state-of-the-art algorithms GL-SHADE and SHADE-ILS. DistSSB outperformed both algorithms in scalability while delivering comparable results. We have tested DistSSB for up to one million dimensions, while the other two algorithms failed to scale. The execution time of DistSSB also remains reasonable for a million dimensions.

Moreover in this thesis we have extended FAIR principles beyond data, so that these could be applied to methods, algorithms and software artifacts. Our focus in this article is to ensure the reproducibility and reusability of algorithms. We have presented $FAIR-GA$ as a usecase to demonstrate the applicability of the proposed principles

for algorithms. Additionally, to ensure $FAIR - Algorithms$ we propose a metadata schema using light weight RDF format, facilitating the reproducibility. Finally, we have demonstrated the application of proposed $FAIR - GA$ using a python based GA code for solving one max search optimization problem. Moreover, we have also proposed a specialized vocabulary (i.e evo) for GA.

Further research can be conducted to investigate the various aspects that this study was unable to address. More extensive computational experimentation involving large graph instances and using more efficient graph library implementation is always desirable. Research can be conducted by using different and more complex fitness functions. To improve diversification in population, a distance criterion can be used to distinguish between similar individuals of BNMR-Col. Reduction of neighborhood size with the reduction of conflicts is a major issue for local move which can be explored further. Another area of possible future research is to investigate cyclic behaviors and develop methods to avoid cycling in the search space. SI algorithms are intrinsically parallel in nature and recent development of cluster computing frameworks like Apache Spark also opens new horizons for the development of scalable BNMR-Col algorithm.

In future, we plan to extend S-GA and evaluate different migration and distribution strategies for larger scale and more complex optimization problems. We will also investigate local search methodologies to augment the performance of DistSSB and search for the optimal island size in comparison to the problem size. In future we will propose more specialized for different machine learning algorithms for exhibiting them FAIR.

Bibliography

M. Agizza, W. Balzano, and S. Stranieri. An improved ant colony optimization based parking algorithm with graph coloring. In *International Conference on Advanced Information Networking and Applications*, pages 82–94. Springer, 2022.

A. S. Akopov and M. A. Hevencev. A multi-agent genetic algorithm for multi-objective optimization. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1391–1395. IEEE, 2013.

M. A. Al-Betar and M. A. Awadallah. Island bat algorithm for optimization. *Expert Systems with Applications*, 107:126–145, 2018.

E. Alba, H. Alfonso, and B. Dorronsoro. Advanced models of cellular genetic algorithms evaluated on sat. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1123–1130, 2005.

B. Ali and P. Dahlhaus. The role of fair data towards sustainable agricultural performance: A systematic literature review. *Agriculture*, 12(2):309, 2022.

M. AlJame, I. Ahmad, and M. Alfailakawi. Apache spark implementation of whale optimization algorithm. *Cluster Computing*, 23(3):2021–2034, 2020.

P. Alliez, R. Di Cosmo, B. Guedj, A. Girault, M.-S. Hacid, A. Legrand, and N. Rougier. Attributing and referencing (research) software: Best practices and outlook from inria. *Computing in Science & Engineering*, 22(1):39–52, 2019.

G. B. Alvarenga, G. R. Mateus, and G. De Tomi. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research*, 34(6):1561–1584, 2007.

B. Amaro Junior, P. R. Pinheiro, and P. V. Coelho. A parallel biased random-key genetic algorithm with multiple populations applied to irregular strip packing problems. *Mathematical problems in engineering*, 2017, 2017.

L. M. Antonio and C. A. C. Coello. Indicator-based cooperative coevolution for multi-objective optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 991–998. IEEE, 2016.

B. Artyushenko. Analysis of global exploration of island model genetic algorithm. In *2009 10th International Conference-The Experience of Designing and Application of CAD Systems in Microelectronics*, pages 280–281. IEEE, 2009.

N. D. Bacarisas and J. P. T. Yusiong. The effects of varying the fitness function on the efficiency of the cat swarm optimization algorithm in solving the graph coloring problem. *Annals. Computer Science Series*, 9(2), 2011.

K. Baiche, Y. Meraihi, M. D. Hina, A. Ramdane-Cherif, and M. Mahseur. Solving graph coloring problem using an enhanced binary dragonfly algorithm. *International Journal of Swarm Intelligence Research (IJSIR)*, 10(3):23–45, 2019.

M. Bensouyad, N. Guidoum, and D.-E. Saïdouni. A new and fast evolutionary algorithm for strict strong graph coloring problem. *Procedia Computer Science*, 73: 138–145, 2015.

J. Blanchard, C. Beauthier, and T. Carletti. A surrogate-assisted cooperative coevolutionary algorithm using recursive differential grouping as decomposition strategy. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 689–696. IEEE, 2019.

R. F. Boisvert. Incentivizing reproducibility. *Communications of the ACM*, 59(10):5–5, 2016.

D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on computers*, 45(7):841–855, 1996.

E. Cantú-Paz et al. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

B. Cao, J. Zhao, Z. Lv, and X. Liu. A distributed parallel cooperative coevolutionary multiobjective evolutionary algorithm for large-scale optimization. *IEEE Transactions on Industrial Informatics*, 13(4):2030–2038, 2017.

L. D. Chambers. *Practical Handbook of Genetic Algorithms: Complex Coding Systems, Volume III*, volume 3. CRC press, 2019.

Y.-H. Chang. Adopting co-evolution and constraint-satisfaction concept on genetic algorithms to solve supply chain network design problems. *Expert Systems with Applications*, 37(10):6919–6930, 2010.

F. Chávez, F. Fernández, C. Benavides, D. Lanza, J. Villegas, L. Trujillo, G. Olague, and G. Román. Ecj+ hadoop: an easy way to deploy massive runs of evolutionary algorithms. In *European Conference on the Applications of Evolutionary Computation*, pages 91–106. Springer, 2016.

B. Chen and L. Chen. A link prediction algorithm based on ant colony optimization. *Applied Intelligence*, 41(3):694–708, 2014.

K. Chen and H. Kanoh. Solving the graph coloring problem using adaptive artificial bee colony. *Transactions of the Society for Evolutionary Computation*, 9(3):103–114, 2019.

S. Chen, J. Montgomery, and A. Bolufé-Röhler. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence*, 42(3):514–526, 2015.

H. A. R. Chowdhury, T. Farhat, and M. H. Khan. Memetic algorithm to solve graph coloring problem. *International Journal of Computer Theory and Engineering*, 5(6): 890, 2013.

A. L. Corcoran and R. L. Wainwright. A parallel island model genetic algorithm for the multiprocessor scheduling problem. In *Proceedings of the 1994 ACM symposium on applied computing*, pages 483–487, 1994.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

D. Costa and A. Hertz. Ants can colour graphs. *Journal of the operational research society*, 48(3):295–305, 1997.

S. De. An efficient technique of resource scheduling in cloud using graph coloring algorithm. *Global Transitions Proceedings*, 3(1):169–176, 2022.

M. Demange, T. Ekim, B. Ries, and C. Tanasescu. On some applications of the selective graph coloring problem. *European Journal of Operational Research*, 240(2):307–314, 2015.

W. Deng, H. Ni, Y. Liu, H. Chen, and H. Zhao. An adaptive differential evolution algorithm based on belief space and generalized opposition-based learning for resource allocation. *Applied Soft Computing*, 127:109419, 2022.

L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro. A parallel genetic algorithm based on hadoop mapreduce for the automatic generation of junit test suites. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 785–793. IEEE, 2012.

G. Dick. The spatially-dispersed genetic algorithm. In *Genetic and Evolutionary Computation Conference*, pages 1572–1573. Springer, 2003.

K. Doerner, R. F. Hartl, and M. Reimann. Cooperative ant colonies for optimizing resource allocation in transportation. In *Workshops on Applications of Evolutionary Computation*, pages 70–79. Springer, 2001.

M. Dubreuil, C. Gagné, and M. Parizeau. Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(1):229–235, 2006.

S. E. Eklund. A massively parallel architecture for distributed genetic algorithms. *Parallel Computing*, 30(5-6):647–676, 2004.

H. Esbensen. Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm. *Networks*, 26(4):173–185, 1995.

D. Esteves, D. Moussallem, C. B. Neto, T. Soru, R. Usbeck, M. Ackermann, and J. Lehmann. Mex vocabulary: a lightweight interchange format for machine learning experiments. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 169–176, 2015.

D. Fan and J. Lee. A hybrid mechanism of particle swarm optimization and differential evolution algorithms based on spark. *KSII Transactions on Internet and Information Systems (TIIS)*, 13(12):5972–5989, 2019.

J. Fantin. *A Distributed Fair Random Forest*. PhD thesis, University of Wyoming, 2020.

J. M. Ferrández, V. Lorente, J. Cuadra, J. R. Álvarez-Sánchez, E. Fernández, et al. A biological neuro processor for robotic guidance using a center of area method. *NeuroComputing*, 74(8):1229–1236, 2011.

F. Ferrucci, P. Salza, and F. Sarro. Using hadoop mapreduce for parallel genetic algorithms: A comparison of the global, grid and island models. *Evolutionary computation*, 26(4):535–567, 2018.

I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister. A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*, 2013.

G. Folino, C. Pizzuti, and G. Spezzano. Training distributed gp ensemble with a selective algorithm based on clustering and pruning for pattern classification. *IEEE Transactions on Evolutionary Computation*, 12(4):458–468, 2008.

P. Galinier and J.-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

S. Gao, K. Wang, S. Tao, T. Jin, H. Dai, and J. Cheng. A state-of-the-art differential evolution algorithm for parameter estimation of solar photovoltaic models. *Energy Conversion and Management*, 230:113784, 2021.

W. Gao and F. Neumann. Runtime analysis for maximizing population diversity in single-objective optimization. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 777–784, 2014.

F. Ge, Z. Wei, Y. Tian, and Z. Huang. Chaotic ant swarm for graph coloring. In *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 512–516. IEEE, 2010.

K. Ghoseiri and S. F. Ghannadpour. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing*, 10(4):1096–1107, 2010.

Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300, 2015.

F. Gronwald, S. Chang, and A. Jin. Determining a source in air dispersion with a parallel genetic algorithm. *International Journal of Emerging Technology and Advanced Engineering*, 7(8):174–185, 2017.

M. Gruenpeter, R. Di Cosmo, H. Koers, P. Herterich, R. Hooft, J. Parland-von Essen, J. Tana, T. Aalto, and S. Jones. M2. 15 assessment report on 'fairness of software'. *Zenodo*, 2020.

L. Gu and H. Li. Memory or time: Performance evaluation for iterative operation on hadoop and spark. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 721–727. IEEE, 2013.

G. Guizzardi. Ontology, ontologies and the "i" of fair. *Data Intelligence*, 2(1-2):181–191, 2020.

A. A. Hadi, A. W. Mohamed, and K. M. Jambi. Lshade-spa memetic framework for solving large-scale optimization problems. *Complex & Intelligent Systems*, 5(1):25–40, 2019.

I. A. T. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan. Mapreduce: Review and open challenges. *Scientometrics*, 109(1):389–422, 2016.

A. Hasnain and D. Rebholz-Schuhmann. Assessing fair data principles against the 5-star open data principles. In *European Semantic Web Conference*, pages 469–477. Springer, 2018.

W. Hasselbring, L. Carr, S. Hettrick, H. Packer, and T. Tiropanis. From fair research data toward fair and open research software. *it-Information Technology*, 62(1):39–47, 2020.

Y. He, C. Gao, N. Sang, Z. Qu, and J. Han. Graph coloring based surveillance video synopsis. *Neurocomputing*, 225:64–79, 2017.

Z. He, H. Peng, J. Chen, C. Deng, and Z. Wu. A spark-based differential evolution with grouping topology model for large-scale global optimization. *Cluster Computing*, pages 1–21, 2020.

A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

J. Holland. Adaptation in natural and artificial systems: an introductory analysis with application to biology. *Control and artificial intelligence*, 1975.

N. C. Hong, S. Cozzino, F. Genova, M. Hoffmann-Sommer, R. Hooft, L. Lembinen, J. Martilla, M. Teperek, and A. Holl. Six recommendations for implementation of fair practice. Technical report, European Commission, 2020.

F. Hooft, A. Perez de Alba Ortiz, and B. Ensing. Discovering collective variables of molecular transitions via genetic algorithms and neural networks. *Journal of chemical theory and computation*, 17(4):2294–2306, 2021.

C. Hu, G. Ren, C. Liu, M. Li, and W. Jie. A spark-based genetic algorithm for sensor placement in large scale drinking water distribution systems. *Cluster Computing*, 20(2):1089–1099, 2017.

H. Jabeen and A. R. Baig. Particle swarm optimization based tuning of genetic programming evolved classifier expressions. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 385–397. Springer, 2010.

S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *arXiv preprint arXiv:2002.00388*, 2020.

D. S. Johnson and M. A. Trick. Introduction to the second dimacs challenge: cliques, coloring, and satisfiability. *Cliques, coloring, and satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:1–7, 1996.

D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406, 1991.

A. Johri and D. W. Matula. Probabilistic bounds and heuristic algorithms for coloring large random graphs. Master's thesis, Southern Methodist University, Dallas, Texas, USA, 1982.

D. Keco and A. Subasi. Parallelization of genetic algorithms using hadoop map/reduce. *SouthEast Europe Journal of Soft Computing*, 1(2), 2012.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

D. Knysh and V. M. Kureichik. Parallel genetic algorithms: a survey and problem state of the art. *Journal of Computer and Systems Sciences International*, 49(4):579–589, 2010.

T. Krink, B. H. Mayoh, and Z. Michalewicz. A patchwork model for evolutionary algorithms with structured and variable size populations. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1321–1328, 1999.

S. Krishnamurthi and J. Vitek. The real software crisis: Repeatability as a core value. *Communications of the ACM*, 58(3):34–36, 2015.

A.-L. Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico, V. Dominguez Del Angel, S. Van De Sandt, J. Ison, P. A. Martinez, et al. Towards fair principles for research software. *Data Science*, 3(1):37–59, 2020.

J. Lässig and D. Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th workshop proceedings on Foundations of genetic algorithms*, pages 181–192, 2011.

C. Li, H. Xu, X. Liao, and J. Yu. Tabu search for cnn template learning. *Neurocomputing*, 51:475–479, 2003.

G. Li. Research on open vehicle routing problem with time windows based on improved genetic algorithm. In *2009 International conference on computational intelligence and software engineering*, pages 1–5. IEEE, 2009.

X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, and H. China. Benchmark functions for the cec 2013 special session and competition on large-scale global optimization. *gene*, 7(33):8, 2013.

D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23 (4):658–670, 2007.

T. Y. Lim. Structured population genetic algorithms: a literature survey. *Artificial Intelligence Review*, 41(3):385–399, 2014.

A. Lissovoi and C. Witt. A runtime analysis of parallel evolutionary algorithms in dynamic optimization. *Algorithmica*, 78(2):641–659, 2017.

Y. Y. Liu and S. Wang. A scalable parallel genetic algorithm for the generalized assignment problem. *Parallel computing*, 46:98–119, 2015.

Z. Lü and J.-K. Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

G. Luque and E. Alba. *Parallel genetic algorithms: Theory and real world applications*, volume 367. Springer, 2011.

S. Mahdavi, S. Rahnamayan, and M. E. Shiri. Multilevel framework for large-scale global optimization. *Soft Computing*, 21(14):4111–4140, 2017.

S. Mahmoudi and S. Lotfi. Modified cuckoo optimization algorithm (mcoa) to solve graph coloring problem. *Applied soft computing*, 33:48–64, 2015.

H. R. Maier, S. Razavi, Z. Kapelan, L. S. Matott, J. Kasprzyk, and B. A. Tolson. Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. *Environmental modelling & software*, 114:195–213, 2019.

F. Maqbool, S. Razzaq, J. Lehmann, and H. Jabeen. Scalable distributed genetic algorithm using apache spark (s-ga). In *International Conference on Intelligent Computing*, pages 424–435. Springer, 2019a.

F. Maqbool, S. Razzaq, J. Lehmann, and H. Jabeen. Scalable distributed genetic algorithm using apache spark (s-ga). In *International conference on intelligent computing*, pages 424–435. Springer, 2019b.

F. Maqbool, S. Razzaq, A. Yar, and H. Jabeen. Large scale distributed optimization using apache spark: Distributed scalable shade-bat (distssb). In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2559–2566. IEEE, 2021.

R. Marappan. A new multi-objective optimization in solving graph coloring and wireless networks channels allocation problems. *Int. J. Advanced Networking and Applications*, 13(02):4891–4895, 2021.

A. Mishra, D. Vakharia, A. J. Hati, and K. S. Raju. Hardware software partitioning of task graph using genetic algorithm. In *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, pages 1–5. IEEE, 2014.

M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

A. W. Mohamed. Solving large-scale global optimization problems using enhanced adaptive differential evolution algorithm. *Complex & Intelligent Systems*, 3(4):205–231, 2017.

T. M. H. S. Mohammad and H. B. Mohammad. A novel meta-heuristic algorithm for numerical function optimization: Blind, naked mole-rats (bnmr) algorithm. *Scientific Research and Essays*, 7(41):3566–3583, 2012.

D. Molina, A. LaTorre, and F. Herrera. Shade with iterative local search for large-scale global optimization. In *2018 IEEE congress on evolutionary computation (CEC)*, pages 1–8. IEEE, 2018.

K. E. Niemeyer, A. M. Smith, and D. S. Katz. The challenge and promise of software citation for credit, identification, discovery, and reuse. *Journal of Data and Information Quality (JDIQ)*, 7(4):1–5, 2016.

A. Nihar, A. J. Curran, A. M. Karimi, J. L. Braid, L. S. Bruckman, M. Koyutürk, Y. Wu, and R. H. French. Toward findable, accessible, interoperable and reusable (fair) photovoltaic system time series data. In *2021 IEEE 48th Photovoltaic Specialists Conference (PVSC)*, pages 1701–1706. IEEE, 2021.

B. B. Oliveira, M. A. Carravilla, J. F. Oliveira, and M. G. Resende. A c++ application programming interface for co-evolutionary biased random-key genetic algorithms for solution and scenario generation. *Optimization Methods and Software*, pages 1–22, 2021.

B. Ombuki, B. J. Ross, and F. Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.

L. A. Org, R. H. DTL, S. Kuijpers, and J. Parland-von Essen. D2. 4 2nd report on fair requirements for persistence and interoperability. Technical report, FAIRsFAIR project, 2020.

E. C. Osuna, W. Gao, F. Neumann, and D. Sudholt. Speeding up evolutionary multi-objective optimisation through diversity-based parent selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 553–560, 2017.

O. Pacheco-Del-Moral and C. A. C. Coello. A shade-based algorithm for large scale global optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 650–663. Springer, 2020.

R.-Z. Qi, Z.-J. Wang, and S.-Y. Li. A parallel genetic algorithm based on spark for pairwise test suite generation. *Journal of Computer Science and Technology*, 31(2): 417–427, 2016.

J. Qin, Y. Yin, and X. Ban. Hybrid discrete particle swarm algorithm for graph coloring problem. *J. Comput.*, 6(6):1175–1182, 2011.

B.-Y. Qu, Y. Zhu, Y. Jiao, M. Wu, P. N. Suganthan, and J. J. Liang. A survey on multi-objective evolutionary algorithms for the solution of the environmental/economic dispatch problems. *Swarm and Evolutionary Computation*, 38:1–11, 2018.

S. Rahnamayan and G. G. Wang. Center-based sampling for population-based algorithms. In *2009 IEEE Congress on Evolutionary Computation*, pages 933–938. IEEE, 2009.

S. Rahnamayan. and G. G. Wang. Toward effective initialization for large-scale search spaces. *Trans Syst*, 8(3):355–367, 2009.

S. Razzaq, F. Maqbool, and A. Hussain. Modified cat swarm optimization for clustering. In *International conference on brain inspired cognitive systems*, pages 161–170. Springer, 2016.

I. Rebollo-Ruiz and M. Graña. An empirical evaluation of gravitational swarm intelligence for graph coloring algorithm. *Neurocomputing*, 132:79–84, 2014.

L. Reiser, L. Harper, M. Freeling, B. Han, and S. Luan. Fair: a call to make published data more findable, accessible, interoperable, and reusable. *Molecular plant*, 11(9): 1105–1108, 2018.

A. Reza, Z. Vahid, and Z. Koorush. Mlga: a multilevel cooperative genetic algorithm. bio-inspired computing: theories and applications (bic-ta). In *IEEE fifth international conference*, pages 271–277, 2010.

F. Rojas and F. Meza. A parallel distributed genetic algorithm for the prize collecting steiner tree problem. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 643–646. IEEE, 2015.

G. Roy, H. Lee, J. L. Welch, Y. Zhao, V. Pandey, and D. Thurston. A distributed pool architecture for genetic algorithms. In *2009 IEEE Congress on Evolutionary Computation*, pages 1177–1184. IEEE, 2009.

N. R. Sabar, A. Turky, and A. Song. Adaptive multi-optimiser cooperative co-evolution for large-scale optimisation. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 705–712. IEEE, 2019.

S. Salhi and R. Petch. A ga based heuristic for the vehicle routing problem with multiple trips. *Journal of Mathematical Modelling and Algorithms*, 6(4):591–613, 2007.

P. Salza, F. Ferrucci, and F. Sarro. Develop, deploy and execute parallel genetic algorithms in the cloud. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 121–122, 2016.

H. G. Santos, L. S. Ochi, E. H. Marinho, and L. M. d. A. Drummond. Combining an evolutionary algorithm with data mining to solve a single-vehicle routing problem. *Neurocomputing*, 70(1-3):70–77, 2006.

M. Sefrioui and J. Périaux. A hierarchical genetic algorithm using multiple models for optimization. In *International Conference on Parallel Problem Solving From Nature*, pages 879–888. Springer, 2000.

A. G. Shoro and T. R. Soomro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.

A. A. Sinaci, F. J. Núñez-Benjumea, M. Gencturk, M.-L. Jauer, T. Deserno, C. Chronaki, G. Cangioli, C. Cavero-Barca, J. M. Rodríguez-Pérez, M. M. Pérez-Pérez, et al. From raw data to fair data: the fairification workflow for health research. *Methods of information in medicine*, 59(S 01):e21–e32, 2020.

M. Skok, D. Skrlec, and S. Krajcar. The genetic algorithm method for multiple depot capacitated vehicle routing problem solving. In *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516)*, volume 2, pages 520–526. IEEE, 2000.

A. Slowik and H. Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32(16):12363–12379, 2020.

A. M. Smith, D. S. Katz, and K. E. Niemeyer. Software citation principles. *PeerJ Computer Science*, 2:e86, 2016.

S. Spoor, C.-H. Cheng, L.-A. Sanderson, B. Condon, A. Almsaeed, M. Chen, A. Bretaudeau, H. Rasche, S. Jung, D. Main, et al. Tripal v3: an ontology-based toolkit for construction of fair biological community databases. *Database*, 2019, 2019.

S. Stall, L. Yarmey, J. Cutcher-Gershenfeld, B. Hanson, K. Lehnert, B. Nosek, M. Parsons, E. Robinson, and L. Wyborn. Make scientific data fair, 2019.

R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

Y. Sun, X. Li, A. Ernst, and M. N. Omidvar. Decomposition for large-scale optimization problems with overlapping components. In *2019 IEEE congress on evolutionary computation (CEC)*, pages 326–333. IEEE, 2019.

M. Taherdangkoo. Modified blind naked mole-rat algorithm applied to electromagnetic design problems. *Archives of Electrical Engineering*, 70(2), 2021.

M. Taherdangkoo and R. Taherdangkoo. Modified bnmr algorithm applied to loney's solenoid benchmark problem. *International Journal of Applied Electromagnetics and Mechanics*, 46(3):683–692, 2014.

M. Taherdangkoo, M. H. Shirzadi, M. Yazdi, and M. H. Bagheri. A robust clustering method based on blind, naked mole-rats (bnmr) algorithm. *Swarm and Evolutionary Computation*, 10:1–11, 2013.

E.-G. Talbi and P. Bessiere. A parallel genetic algorithm for the graph partitioning problem. In *Proceedings of the 5th International Conference on Supercomputing*, pages 312–320. ACM, 1991.

C. Tan, C. K. Goh, K. C. Tan, and A. Tay. A cooperative coevolutionary algorithm for multiobjective particle swarm optimization. In *2007 IEEE Congress on Evolutionary Computation*, pages 3180–3186. IEEE, 2007.

R. Tanabe and A. Fukunaga. Evaluating the performance of shade on cec 2013 benchmark problems. In *2013 IEEE Congress on evolutionary computation*, pages 1952–1959. IEEE, 2013a.

R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE congress on evolutionary computation*, pages 71–78. IEEE, 2013b.

R. S. Tomar, S. Singh, S. Verma, and G. S. Tomar. A novel abc optimization algorithm for graph coloring problem. In *2013 5th International Conference and Computational Intelligence and Communication Networks*, pages 257–261. IEEE, 2013.

A. Trivedi, D. Srinivasan, S. Biswas, and T. Reindl. Hybridizing genetic algorithm with differential evolution for solving the unit commitment scheduling problem. *Swarm and Evolutionary Computation*, 23:50–64, 2015.

H. van Vlijmen, A. Mons, A. Waalkens, W. Franke, A. Baak, G. Ruiter, C. Kirkpatrick, L. O. B. da Silva Santos, B. Meerman, R. Jellema, et al. The need of industry to go fair. *Data Intelligence*, 2(1-2):276–284, 2020.

A. Verma. Scaling simple, compact and extended compact genetic algorithms using mapreduce. Master's thesis, University of Illinois at Urbana-Champaign, 2010.

R. Vita, J. A. Overton, C. J. Mungall, A. Sette, and B. Peters. Fair principles and the iedb: short-term improvements and a long-term vision of obo-foundry mediated machine-actionable interoperability. *Database*, 2018, 2018.

L. Vogt. Anatomy knowledge graphs: Toward fair morphological data. *Biodiversity Information Science and Standards*, 3:e37203, 2019.

L. Vogt, S. Auer, T. Bartolomaeus, P. L. Buttigieg, P. Grobe, P. Michalik, M. Stocker, and R. Usbeck. Fair. red: Semantic knowledge graph infrastructure for the life sciences. *Biodiversity Information Science and Standards*, 2019.

H. Wang, S. Rahnamayan, and Z. Wu. Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems. *Journal of Parallel and Distributed Computing*, 73(1):62–73, 2013.

M. A. Wani and S. Jabin. Big data: issues, challenges, and techniques in business intelligence. In *Big data analytics*, pages 613–628. Springer, 2018.

M. Weißbrich, J. A. Moreno-Medina, and G. Payá-Vayá. Using genetic algorithms to optimize the instruction-set encoding on processor cores. In *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–6. IEEE, 2021.

D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology*, 7(1):33–47, 1999.

M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

M. D. Wilkinson, S.-A. Sansone, E. Schultes, P. Doorn, L. O. Bonino da Silva Santos, and M. Dumontier. A design framework and exemplar metrics for fairness. *Scientific data*, 5(1):1–4, 2018.

C. Witt. Runtime analysis of the (μ+ 1) ea on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

M. Yang, A. Zhou, C. Li, and X. Yao. An efficient recursive differential grouping for large-scale continuous problems. *IEEE Transactions on Evolutionary Computation*, 2020.

X.-S. Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

T. Yildirim, C. B. Kalayci, and Ö. Mutlu. A novel metaheuristic for traveling salesman problem: blind mole-rat algorithm. *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 22(1):64–70, 2016.

X. Yu and M. Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016a.

M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016b.

J. Zhang and A. C. Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

K. Zhang, M. Qiu, L. Li, and X. Liu. Accelerating genetic algorithm for solving graph coloring problem based on cuda architecture. In *Bio-Inspired Computing-Theories and Applications*, pages 578–584. Springer, 2014.

X. Zhang, Z.-H. Zhan, and J. Zhang. Adaptive population differential evolution with dual control strategy for large-scale global optimization problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE, 2020.

L. Zheng, Y. Lu, M. Ding, Y. Shen, M. Guoz, and S. Guo. Architecture-based performance evaluation of genetic algorithms on multi/many-core systems. In *2011 14th IEEE International Conference on Computational Science and Engineering*, pages 321–334. IEEE, 2011.