Applications of Fireworks-based Evolutionary Algorithms for Computationally Challenging Network Problems

by

Hafiz Munsub Ali

MS (CS), PAF-Karachi Institute of Economics and Technology, Karachi, Pakistan, 2007

BS (CS), The Islamia University of Bahawalpur, Pakistan, 2003

Thesis Submitted in Partial Fulfillment of

the Requirement for the Degree of

Doctor of Philosophy

in the

School of Engineering Science

Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Summer 2019

Approval

Name:	**Hafiz Munsub Ali**
Degree:	**Doctor of Philosophy**
Title:	***Applications of Fireworks-based Evolutionary Algorithms for Computationally Challenging Network Problems***
Examining Committee:	**Chair:** Glenn H. Chapman Professor

Daniel C. Lee
Senior Supervisor
Professor

Jiangchuan Liu
Co-Supervisor
Professor
School of Computing Science

Qianping Gu
Supervisor
Professor
School of Computing Science

Jie Liang
Supervisor
Professor

Rodney G. Vaughan
Internal Examiner
Professor

Kui Wu
External Examiner
Professor
Department of Computer Science
University of Victoria

Date Defended/Approved: June 4, 2019

Abstract

This thesis covers two types of contributions: formulation of network optimization problems and algorithms to solve these optimization problems. We propose resource assignment problem in Internet of Things network (IoTN) with three nodes: IoT, core cluster node (CCN) and base station (BS). The assignment of resources, such as CPU and memory, from IoTs to CCNs, and CCNs to BSs is a challenging task. The objective of the problem is to minimize the weighted sum of computational power at CCNs and transmission power between IoTs-CCNs and CCNs-BSs radio connections. We also propose a broadband wireless network (BWN) wherein the planning of BSs, relay stations (RSs), and their connections to subscribers minimizes the overall (i.e., weighted sum of the hardware and operational) cost of the network and reformulate a virtual machine (VM) placement to minimize power consumption in a datacenter. The (re)formulated problems are integer programming problem and finding optimal solutions for these problems by using exhaustive search is not practical due to demand of high computing resources. The practical approach is to minimize power in IoT network and VM placement, and plan broadband wireless network using population-based heuristic algorithms. We propose swarm intelligence-based algorithms, that is, two versions of the discrete fireworks algorithm (DFWA) and its variants. The performance of these new algorithms is compared against the low-complexity Biogeography-based Optimization (LC-BBO) algorithm, the Discrete Artificial Bee Colony (DABC) algorithm, and the Genetic Algorithm (GA). Our simulation results and statistical test demonstrate that the proposed algorithm can comparatively find good-quality solutions with moderate computing resources.

Keywords: Computationally challenging network problems; integer programing problems; discrete fireworks algorithm; population-based heuristic algorithms; exhaustive search

To my beloved parents mother: Zuhra, and father: Abdul Haq for their unconditional support, affection, encouragement and guidance.

Acknowledgements

The culmination of this thesis has been the product of a long and exciting journey, which dates as far back as Fall 2009, when I first commenced my studies in Engineering. Since then, I have been consistently inspired and enlightened by the professors I have had throughout my studies.

Firstly, I would like to thank my thesis Senior Supervisor, Professor Daniel Lee for the endless support, guidance and understanding that he provided to me throughout the course of the work for this thesis. Especially, Dr. Daniel Lee's critical feedback and expert analysis were both in my development, and his insight proved invaluable to me.

I would also like to thank Co-Supervisor Professor Jiangchuan Liu for his encouragement and valuable comments on my research. He was both perceptive and intuitive and his guidance has meant a lot to me.

Prof. Qianping Gu and Prof. Jie Liang were also helpful to me in my studies, and I would also like to thank them for their support.

Of course, I am grateful for the support and help of faculty, staff and graduate students of the mobile communication lab. I give my special thanks to Dr. Muhammad Naeem, Saeed Ashrafinia, Jaspreet Oberoi, Saad Mahboob, Dr. Ali Zarei, Dr. Ying Chen, Faryad Ali Rana, Dr. Waleed Ejaz, Abhijit Bhattacharya, Rajveer Brar, Dr. Muhammed Omar, Dr. Ismail M. Khater, and others, who provided me with great help and support during the last few years.

My deepest appreciations go out to my brothers (Ihtesham and Anser), Sisters (Noreen, Shaheen, Sadia, and Nadia), wife Fouzia, and blessed sons (Ghazanfer, Uzzam, Qasim, and Hashim) to whom I owe so much. I would like to thank my parents for the sacrifices they have made, and for the inspiration and support they have provided throughout my life.

Finally, I would also like to thank God for his unwavering devotion and guidance to me throughout the course of this journey.

Table of Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

ABC	Artificial Bee Colony
AI	Artificial Intelligence
BBO	Biogeography-based Optimization
BS	Base Station
CCN	Core cluster node
DFWA-with-Dy-3-LS	Discrete FWA with ensembles of dynamic 3 local search methods
DFWA-with-Dy-2-LS	Discrete FWA with ensembles of dynamic 2 local search methods
DFWA-with-FR-3-LS	Discrete FWA with ensembles of fixed-rate 3 local search methods
DFWA-*Insert*	DFWA with Insert local search method
DFWA-*Swap*	DFWA with Swap local search method
DFWA-*Interchange*	DFWA with Interchange local search method
DFWA	Discrete Fireworks Algorithm
DABC	Discrete Artificial Bee Colony
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EFWA	Enhanced Fireworks Algorithm
FFD	First fit decreasing
FWA	Fireworks Algorithm
GA	Genetic Algorithm
HSI	Habitat Suitability Index
IDFWA	Problem specific information-based discrete fireworks algorithm
IoT	Internet of things
IQEA	Immune Quantum Inspired Evolutionary Algorithm
LC-BBO	Low-complexity Biogeography-based Optimization
limitTrial	Number of times that the nectar of a food source position (a candidate solution) is evaluated.
LTE	Long-Term Evolution
LS	Local Search

MMR	Multi-hop Relay
PM	Physical Machine
PMP	Point to multi-point
p-value	Probability Value
QEA	Quantum Inspired Evolutionary Algorithm
RS	Relay Station
RPM	Maximal Network Capacity
SS	Subscriber Station
SI	Swarm Intelligence
SIV	Suitability Index Variables
Std.	Standard Deviation
TP	Test Point
VM	Virtual Machine
WiMAX	World Inter-operability for Microwave Access

List of Symbols

N	Population/Fireworks
$\mathcal{Z}$	Set of fireworks for mutation
s_i	Number of sparks for the i^{th} firework
A_i	Amplitude for the i^{th} firework
â	A parameter for explosion amplitude
M_e	A parameter for number of explosion sparks
X	A vector that represents the candidate solution
s^{min}	Parameter to set the minimum number of sparks
s^{max}	Parameter to set the maximum number of sparks
A_i^{min}	Represents nonlinearly decreasing amplitude
A^U	Upper limit of the explosion amplitude
A^L	Lower limit of the explosion amplitude
t^{max}	Maximum number of function evaluations (a stopping criterion for an algorithm)
E	Emigration rate
I	Immigration rate
λ	Immigration probability
μ	Emigration probability
M_r	Mutation probability
ϕ	Total computational power at CCN
Φ	Transmission power between IoTs-CCNs and CCNs-BSs
F_i	Fitness value of the i^{th} candidate solution
Z	Number of virtual machines
M	Number of physical machines
v_i	i^{th}virtual machine
p_j	j^{th}physical machine
u_j	The percentage of CPU utilization of j^{th}physical machine
e_j	Power consumption of j^{th}physical machine
e_{max}^j	Maximum power consumption of j^{th}physical machine
e_{idle}^j	Power consumption of j^{th}physical machine in idle status

v_{cpu}^{i}	CPU demand of i^{th}virtual machine
v_{mem}^{i}	Memory demand of i^{th}virtual machine
v_{net}^{i}	Network bandwidth of demand of i^{th}virtual machine
p_{cpu}^{j}	CPU capacity of j^{th}physical machine
p_{mem}^{j}	Memory capacity of j^{th}physical machine
p_{net}^{j}	Network bandwidth capacity of j^{th}physical machine
x_{ij}	the binary value representing whether VM v_i is assigned to PM p_j
θ	User-defined probability for Hybrid IDFWA/LC-BBO algorithm
α_1	User-defined fractional parameter for IDFWA
T	Selected components of vector *X*
Δ	Fractional parameter to determine specified PMs/CCNs in *X*
α	Cut-off point for the p-value
c_b^B	A base station's cost
c_r^R	A relay station's cost
$l_{b,r}^{BR}$	Path-loss associated with BS and RS
$l_{b,t}^{BT}$	Path-loss associated with BS and TP
$l_{r,t}^{RT}$	Path-loss associated with RS and TP
$m_{b,r}^{BR}$	Upper bounds (e.g., channel capacity) on the possible information flow rate associated with BS and RS
$m_{b,t}^{BT}$	Upper bounds (e.g., channel capacity) on the possible information flow rate associated with BS and TP
$m_{r,t}^{RT}$	Upper bounds (e.g., channel capacity) on the possible information flow rate associated with RS and TP
u_t^T	Traffic demand of a TP
C_1	Maximum capacity (in bits per second) for a BS
C_2	Maximum capacity (in bits per second) for a RS
W_1	Weight for the first term of the objective function
W_2	Weight for the 2nd term of the objective function
y_b^B	Binary decision variables that determined whether a BS *b* is deployed
y_r^R	Binary decision variables that determined whether a RS *r* is deployed
$x_{b,r}^{BR}$	Binary decision variables that denote whether a connection is established between BS and RS

$x_{b,t}^{BT}$	Binary decision variables that denote whether a connection is established between BS and TP
$x_{r,t}^{RT}$	Binary decision variables that denote whether a connection is established between RS and TP
$f_{b,t}^{BT}( )$	Flow function for BS and TP
$f_{r,t}^{RT}( )$	Flow function for RS and TP
$f_{b,r}^{BR}( )$	Flow function for BS and RS
$\mathscr{b}_b^B$	Binary variable represents whether a BS is in *X*
S_r^R	Binary variable represents whether a RS is in *X*
Π	Set of all feasible solutions in neighborhood search
π	A solution in neighborhood search ($\pi \in \Pi$)
$\mathcal{N}(\pi)$	An associated set of *neighbors*, $\mathcal{N}(\pi) \subset \Pi$
$\wp$	A set of integers representing the LS operators
ϖ	Select a local search operator
ϱ	Integer vector representing LS methods in an ensemble
∂	Fixed rate probability to a LS method in an ensemble
$\mathcal{H}$	set of IoT nodes (IoTs).
$\mathcal{M}$	set of core cluster nodes (CCNs).
$\mathcal{G}$	set of base stations (BSs).
$\mathcal{S}_i$	denotes an IoT, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
c_j	denotes a CCN, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.
$\mathscr{b}_k$	denotes a BS, where $k = 1,2, \dots, \lvert\mathcal{G}\rvert$.
$\mathbb{u}_j$	represents the percentage of CPU utilization of a CCN c_j.
e_j	power consumption of a CCN c_j.
e_{max}^j	maximum power consumption of a CCN c_j, when $\mathbb{u}_j = 100\%$.
e_{idle}^j	power consumption of a CCN c_j in idle mode.
$\mathcal{S}_{cpu}^i$	CPU demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
$\mathcal{S}_{mem}^i$	memory (RAM) demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
$\mathcal{S}_{data}^i$	data transmission demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
c_{cpu}^j	CPU capacity of a CCN c_j, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.
c_{mem}^j	memory (RAM) capacity of a CCN c_j, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.

x_{ij}	binary value representing whether an IoT, $\mathcal{S}_i$, is assigned to a CCN c_j.
c_{data}^{j}	data transmission demand of a CCN $c_j = \frac{\sum_{i \in \mathcal{H}} \mathcal{S}_{data}^{i} \times x_{ij}}{2}$.
y_{jk}	binary value representing, whether a CCN c_j is assigned to a BS $\mathcal{B}_k$.

Chapter 1. Introduction

Optimization problems are common in many disciplines and various domains such as science, engineering, information technology, finance, and the arts. In general, an optimization problem is a problem of finding the best solution from all possible solutions. In many cases, the space of possible solutions is typically too large to search using brute force or exhaustively. The optimization problems with such a large search space are often considered computationally challenging as their solution demands high computing resources. Optimization problems are modeled with variety of optimization objectives and some common optimization objectives include minimizing cost, maximizing profit, minimizing error, optimizing design, etc. An optimization problem can be formulated with one objective or combination of objectives also known as multi-objective optimization. Some of the challenging network optimization problems include broadband wireless network planning, virtual machine (VM) placement, resources assignment in Internet of things network (IoTN), sensor networks, and mobile ad hoc networks.

Blessings of Internet technology is making our lives better than ever before in many ways. Undoubtedly, our society becomes “network society” and world becomes “global village” due to recent innovations in internet (and mobile) technologies. Internet services are delivered through essential infrastructure such as datacenters, base stations, and relay stations. Verity of optimization issues involve in the Internet (and mobile) technologies and infrastructure. Power consumption in the Internet (and mobile) infrastructure is one of the widely studied optimization issue. Recent goal of this study is to minimize the adverse impact of power consumption on the planet by designing power-efficient algorithms [1]–[6]. In addition, smart ways to mitigate the operational and maintenance cost of this crucial infrastructure is also one of the popular research areas.

1.1. Problems considered in the thesis

The advances in communication technologies provide new ways to communicate and are considered as an opportunity to reduce society's overall environmental impacts. Facts of this environmental impact can also be viewed by a comparison of system-wide environmental impact of communication technologies such as wireless technologies versus traditional applications. Here, a comparison of carbon dioxide (CO_2) emission for two applications is: (1) reading news content on a mobile device versus reading news on a paper and (2) wireless teleconferencing versus business travel. Wireless technologies in both applications create lower environmental impacts such as reading a newspaper on a mobile device results in the release of 32−140 times less CO_2, and teleconferencing results in 1−3 orders of magnitude lower CO_2 [1].

Effectiveness of wireless technologies encourages new trends in mobile computing and open the doors for further innovative applications. The paradigm shifts to mobile computing and its impact on global energy consumption compelled the research community to see how the world consumes energy. Some recent work found that computing devices such as datacenters, desktops, and mobile devices (laptops and mobile phones), accounted for about 3−7 percent of the global electricity usage. The share of mobile devices was about 10−20 percent, and this share is expected to grow as markets are flooded with popular application enabled smart phones. Datacenters and mobile infrastructures like base stations (BSs) or relay stations (RSs) have been considered as main power consumers within the computing sector. Therefore, recently significant focus of research is diverted to provide energy-efficient and sustainable solutions to datacenters, and mobile infrastructure. The energy-efficient and environmentally sustainable solutions include better hardware, economical algorithms/protocols and innovative applications [2]–[4]. Here, we noted two recent examples one each for economic algorithm and innovative applications. On one hand, total power consumption is optimized for cellular systems by jointly considering base station (BS) deployment and power allocation with quality of experience (QoE) guarantees [3]. On the other hand, Self-Powered IoT-Enabled Water Monitoring System is proposed to reduce the wastage of water [4].

According to ACM Ubiquity, 2015 [5], information and communication technologies (ICT) consume 4.7 percent of the worldwide electricity consumption. Electricity represents 15 percent of worldwide energy production but contributes to 37 percent of CO_2 emissions. Scientists predict that mobile communication systems will increase CO_2 equivalent emissions by a factor of three by the year 2020 [6].

The above reported facts and recent trends are the motivating factors to investigate the following problems in this thesis:

- Virtual machine placement.
- Emerging IoT applications.
- Broadband wireless network planning.

1.1.1. Virtual machine placement

Internet services have grown considerably in the last two decades due to inventions and improvements in broadband wired/wireless network technologies and mobile devices. Providers of popular Internet services, such as data storage (e.g., Dropbox), video streaming (e.g., YouTube), and cloud computing (e.g., Amazon) maintain and operate large datacenters, consuming a significant amount of energy [6], [7]. These services help humanity but hurt the planet by emitting excessive carbon in the form of carbon dioxide (CO_2). Excessive carbon emission and its global impact have motivated the research community to minimize energy consumption in datacenters by developing efficient hardware and resource allocation algorithms [7], [8].

The main factors contributing to power consumption in a datacenter are power dissipation in physical servers, cooling systems to normalize the temperature, and inefficient procedures of resource allocation [7]. Datacenters use devoted servers to provide different types of services to consumers, which causes underutilization of servers and increases the overall power cost. Such costs can be mitigated by power efficient hardware resources, which requires computationally economic algorithms.

The underutilization of hardware in datacenters triggers the concept of virtualization technology. In virtualization technology, a physical machine (PM) is virtualized to multiple virtual machines (VMs) having different capacities (e.g., storage, memory, computation), which may be running different operating systems. In modern virtualized datacenters, a single PM can fulfill multiple and variable user requests [7]. Virtualization technology not only improves hardware utilization in datacenters, it saves power by allowing unutilized PMs to be turned off.

The objective of VM placement (discussed in chapter 3) is to minimize the cost of power by using efficient resource utilization in datacenters. VMs assignments to PMs in a datacenter is a computationally challenging optimization problem and exact algorithm to solve this problem in reasonable computing resources is not known to the author [8].

1.1.2. Optimizing power in emerging IoT applications

When extended to machine to machine (M2M) communication, the traditional sensing paradigm of wireless sensor networks (WSNs) can connect billions of things across the globe to the Internet; this type of WSN is known as the Internet of Things (IoT). In a smart system, IoT nodes collaborate to connect physical objects together using diverse technologies such as real-time analytics, machine learning, commodity sensors, and embedded systems [9]. The large number of physically connected devices in the IoT creates mammoth amounts of data—a.k.a. Big Data—which required smart computation, data storage, and management [10]. Like traditional WSNs, IoT networks face challenges in computation, battery, data storage, bandwidth, latency, and reliability [11]. Big Data can be handled using centralized data centers by moving computing, control, and data storage into clouds. However, the scattered nature, latency sensitivity, and lack of reliability challenge the ability of cloud computing to meet the requirements of IoT networks.

New challenges trigger new concepts, and one such concept is the fog computing. Fog computing provides a bridge between IoT nodes and classic cloud computing. The idea behind fog computing is to bring the cloud closer to IoT nodes to mitigate the latency and unreliability of data transfer. Fog computing services include local data processing and

storage at IoT nodes, which improves efficiency and performance and reduces the amount of data transferred to the cloud for processing, analysis, and storage. Instead of sending data into the cloud, data collected by IoT nodes are sent to network edge devices for processing and temporary storage, reducing cloud network traffic and latency. The integration of fog computing and the IoT is called “fog as a service” (FaaS) [12], wherein an array of fog nodes is established across the geographic footprint of the IoT network. Each fog node hosts local computation, networking, and storage capabilities. FaaS enables customers to receive services from many different business models. The fog cloud is basically a server or a set of servers with large computational power and storing capabilities that receives, processes, and analyzes data collected from IoT nodes. IoT gateways act as cluster heads (CHs) that connect to each other and to IoT nodes [13].

We investigate an IoT network in which the core cluster nodes (CCNs) are capable of real-time communication and are called cluster heads (CHs). Clustering involves grouping of IoT nodes into clusters and each cluster has a CHs. A CH collects data from respective cluster’s IoT nodes and forward the aggregated data to base station. For real-time communication, IoT nodes need real-time feedback from CHs and CHs need reasonable computing resources to deliver real-time responses. We propose a cluster-assisted IoT network with a battery powered core cluster node (CCN) that contains computing resources such as a CPU and memory. More specifically, a CCN with computing capabilities is assisting the proposed IoT network as a CH to provide real time feedback. The objective of the resource assignment problem in IoT network is to minimize the weighted sum of transmission power between IoTs-CCNs and CCNs-BSs, and computational power at CCNs (discussed in chapter 4). The exact algorithm for the IoTs-CCNs and CCNs-BSs resource assignment using moderate computing resources is not known to the author.

1.1.3. Single-hop broadband wireless network planning

Planning a new broadband wireless network or extending an existing network are multifaceted tasks. Extensive knowledge of the wireless technology and geography of the service area may facilitate the planning of broadband wireless network. We formulate a

wireless broadband network planning problem with the objective of minimizing its overall cost. We define the cost of the network as the weighted sum of the operational cost and the infrastructure (i.e., base station, relay station, installation, etc.) cost.

WiMAX (World Interoperability for Microwave Access) is a telecommunication technology based on the IEEE 802.16 standard [14]. Many extended versions of this standard have been launched in the market since the publication of the IEEE 802.16 standard in 2001 [14], [15]. Mainly, two infrastructure variations of the IEEE 802.16 standard exist in the market: point to multipoint (PMP), and relay station (RS) based modes of operation. In the PMP mode of operation, a communication link exists between subscribers' stations (SSs) and the base station (BS). However, a communication link in an RS based mode of operation would be between SSs and the BS or between SSs and the RS. IEEE 802.16*j* is a promising RS based solution for the replacement of conventional PMP technology, with features providing capacity and coverage enhancements in a broadband wireless network [16], [17].

Broadband wireless network planning with the single-hop (described in chapter 5), is topologically equivalent to the IEEE 802.16*j* standard that operates in a transparent relay mode. A relay communication allows only one relay between SS and BS and its main goal is to enhance the capacity of the network in densely populated urban centers. In the proposed broadband wireless network planning, two types of links can be established: a SS can communicate from an SS-RS-BS or from an SS-BS.

1.2. Tools studied to solve the problems in the thesis

Recent developments in optimization techniques facilitate the solution of the computationally challenging problems, many of which are characterized by high dimensionality and have a combinatorial nature. Also, finding optimal solutions for most of these problems requires exhaustive search and extensive computing resources. A more practical approach is to find high-quality approximate solutions for computationally challenging problems using reasonable computing resources.

The research community has developed methods to design approximate algorithms for the solution of computationally challenging problems [18]. One such methodology is nature-inspired population-based search technique and is becoming popular from past three decades [19]. These techniques include evolutionary and swarm intelligence-based algorithms [19]. Evolutionary algorithms (EAs) are designed by imitating natural phenomena such as Genetic Evolution, Memetic Evolution, Neuro Evolution, Evolution of Immune Systems, etc. Similarly, swarm intelligence-based algorithms are designed by imitating Ant Colonies, the Foraging of Honey Bees, the Biogeography of Species, Artificial Fish School, Fireworks [19]–[22], etc. These nature-inspired techniques are widely used to solve computationally challenging optimization problems. Inspired from the recent development in swarm intelligence-based techniques, we propose modifications/enhancements in the fireworks algorithm (FWA), an enhanced FWA (EFWA). The FWA was first presented in 2010 and was extended to the enhanced fireworks algorithm (EFWA) in 2013 [23]. We compared the performance of our modified FWA algorithms against the following recently presented EAs:

- Biogeography based optimization (BBO) – presented by Dr. Tan in 2008 [20].
- Artificial bee colony (ABC) – presented by Dr. Karaboğa in 2005 [24].

Classic genetic algorithm (GA) – has existed for decades and has many variations available in the literature.

1.3. Summary of contributions

The contribution in this thesis is to study a methodology for designing networks and develop algorithms with the aim of optimizing power consumption in datacenters, emerging Internet of Things (IoT) applications and plan broadband wireless network.

In chapter 3, we propose discrete FWA (DFWA) and its variants that can operate in integer space and reformulate a binary space VM placement problem [8] as a nonbinary space VM placement problem to reduce the constraint checks. The fireworks algorithm (FWA) and enhanced FWA (EFWA) are originally presented for the optimization problems in continuous domain [22], [23], [25]. We modify the EFWA operators using *'round'* and

'ceil' functions to convert continuous domain to integer domain for the discrete FWA (DFWA). Like any other evolutionary algorithm (EA), DFWA is model-free and do not need any problem specific information or domain-knowledge during their operations [5]. However, incorporating problem specific information in EAs can improve their overall efficiency. We introduce a new problem specific information-based DFWA (IDFWA) that utilizes domain-knowledge of the virtual machine placement. In [22], [25], FWA is hybridized with various EAs for continuous space benchmark optimization problems. In contrast, we propose a hybrid of the IDFWA and low-complexity biogeography-based optimization (LC-BBO) for the VM placement [26]. During the implementation, a candidate solution either generated randomly or evolved by any of these algorithms, may violate one or more constraints of the optimization problem, and therefore become infeasible. We propose a repair algorithm to check feasibility and repair the infeasible candidate solutions. The part of work in this chapter were published in *IEEE-SSCI 2014* and *IEEE-SPECTS 2016* [27], [28].

Summary of contributions in chapter 3 is as follows:

- Reformulate the VM placement as an integer space optimization problem to reduce the constraint checks.
- Propose following new algorithms to solve the VM placement:
  - Discrete fireworks algorithm (DFWA),
  - Problem specific information-based DFWA (IDFWA),
  - Hybrid of the IDFWA and the low-complexity biogeography-based optimization (LC-BBO) algorithm (Hybrid IDFWA/LC-BBO).
- Repair algorithm to repair the infeasible solutions during the implementation of the experimented algorithms.

In chapter 4, we propose an Internet of things (IoT) network model for delay sensitive applications. The IoT network contains three types of nodes: IoT, core cluster node (CCN) and base station (BS). A CCN is a battery powered computing capable node for the real-time feedback to IoT nodes. Optimizing power by assigning efficient resources in the IoT network is a challenging task. The objective of the problem formulation is to minimize the weighted sum of the data transmission power between IoTs to CCNs and between CCNs to BSs, and computational power at CCNs. First, we formulate a binary

space IoTs assignment problem and then reformulate it as an integer space IoTs assignment to reduce the constraint checks. To solve the resources assignment optimization problem, we use same algorithms that are also used in chapter 3 such as discrete fireworks algorithm (DFWA), problem specific information-based DFWA (IDFWA), and hybrid of the IDFWA and low-complexity biogeography-based optimization (LC-BBO). A candidate solution either generated randomly or evolved by any of these algorithms, may violate one or more constraints of the optimization problem, and therefore become infeasible during the implementation of the algorithms. We propose a repair algorithm to check feasibility and repair the infeasible candidate solutions.

To the best of our knowledge, limited work is available in the existing literature that considers the objective of simultaneously minimizing the transmission and computational power in an IoT network. The part of work in this chapter was published in the *IET Network January 2019* [29].

Summary of contributions in chapter 4 is as follows:

- Formulate an IoT-CCN and CCN-BS assignment as a binary space optimization problem.
- Reformulate an IoT-CCN and CCN-BS assignment as an integer space optimization problem to reduce the constraint checks.
- To solve the resources assignment in IoT network, we use three Firework-based evolutionary algorithms (same as in chapter 3) as follows:
  - Discrete Fireworks Algorithm (DFWA).
  - Problem specific information based DFWA (IDFWA).
  - Hybrid of the IDFWA/low-complexity BBO (LC-BBO) algorithm.
- Repair algorithm to repair the infeasible solutions during the implementation of the experimented algorithms.

In chapter 5, we propose a broadband wireless network (BWN) with a single-hop between a subscriber (SS) and a BS. The network model consists of three nodes: a base station (BS), a relay station (RS), and SS. A SS can communicate with a BS directly or via an RS. A BWN can be designed from scratch or can be extended from an existing network. We propose a simultaneous BS and RS single-hop BWN from scratch. This network model

adopts path-loss as a criterion of variation for data rates between a wireless link of two communicating nodes. The objective of BWN planning is to minimize the weighted sum of infrastructure (base stations and relay stations) and the operating cost (path-loss) of the BWN. We use DFWA with '*insert,*' '*interchange,*' and '*swap*' local search (LS) methods for the BWN planning integer domain optimization problem [22], [30]. These LS methods are ranked based on their individual performance in the DFWA while planning the BWN. Then, this predetermined ranking information is used to build an ensemble of LS methods for the DFWA. The newly proposed algorithm is called DFWAs with fixed-rate (FR) ensemble of local search methods (DFWA-with-FR-3-LS). To avoid predetermined ranking information while manually selecting a LS method for the DFWA, we propose an algorithm that is DFWA with dynamic ensemble of LS methods (DFWA-with-Dy-3-LS). In [16], [31], BWN planning problems were formulated to determine BS and RS locations that will enhance network capacity at minimal cost. Two-stage network deployment algorithms are presented to solve these network planning problems. In contrast, our proposed BWN planning is simultaneously deploying BSs and RSs by minimizing path-loss as a criterion of variation for data rates among wireless links of communicating nodes. Perturbation in a candidate solution randomly or through the evolution of the algorithm, may violate one or more constraints of the optimization problem, and therefore a candidate solution may become infeasible during the implementation of these algorithms. We propose a repair algorithm to check feasibility and repair the infeasible candidate solutions. The part of work in this chapter were published in *IEEE-CEC 2013* and *IEEE-VTC-Fall 2013* [32], [33].

Summary of contributions in chapter 5 is as follows:

- Formulate the BWN planning as a binary space optimization problem.
- Reformulate the BWN planning as an integer space optimization problem to reduce the constraint checks.
- We propose following new algorithms to solve the BWN planning as follows:
  - DFWAs with fixed-rate (FR) ensemble of three local search methods (DFWA-with-FR-3-LS).
  - DFWA with dynamic ensemble of three LS methods (DFWA-with-Dy-3-LS).
  - DFWA with dynamic ensemble of two LS methods (DFWA-with-Dy-2-LS).

- Repair algorithm to repair the infeasible solutions during the implementation of the experimented algorithms.

Chapter 6 provides a summary of the thesis and suggests future work that could advance the field of experimental procedures to solve computationally challenging optimization problems.

We conduct hundred (100) independent trials (or experiments) for each of the three problems considered in this thesis and compared the results to test experimented algorithms against metrics such as average cost, average Matlab CPU time, and standard deviation. Note that the average cost value of any two algorithms show the quantitative difference between algorithms but do not depict the quality or the level of reliability of the results. Therefore, the difference in average cost of two (i.e., algorithms) groups of data may not represent the true performance of the algorithms and can be misleading due to random fluctuations. We use T-tests [21] to evaluate the degree of reliability in the performance of the evolutionary algorithms (EAs).

1.3.1. Publications from this research

The following papers have been published from this work:

Problem Type	Problem Description	References
Single objective, and constraint problem	Virtual machine placement problem	Chapter 3: A biogeography-based optimization algorithm for energy efficient virtual machine placement [27].
Single objective, and constraint problem	Virtual machine placement problem	Chapter 3: Optimizing the energy efficient VM placement by IDFWA and hybrid IDFWA/BBO algorithms [28].
Single objective and constraint problem	Optimizing power in emerging IoT applications	Chapter 4: Optimizing power using Fireworks-based evolutionary algorithms for emerging IoT applications [29].
Single objective and constraint problem	BS and RS wireless network planning problem	Chapter 5: Broadband network planning problem [32], [33] using evolutionary algorithms.
Single objective and constraint problem	MAX-SAT problem using EAs	This work is in [34].
Single objective and constraint problem	Solving the MAX-SAT problem	This work is in [35]

Multiobjective, and constraint problem, which is converted to a single objective problem using a weighted sum method	Wireless mesh network planning problem	This work is in [36].
Single objective and constraint problem (co-author)	Sensor selection problem using quantum inspired EA	This work is in [37].

Chapter 2. Review of evolutionary algorithms

In this thesis, integer space optimization problems are considered, and these problems are combinatorial in nature. Based on type and size of the problems, various techniques are used to solve integer space optimization problems in existing work. In addition to heuristic and populations-based heuristic algorithms, some other algorithms to solve these problems include standard branch and bound [14], clustering [99], simplex algorithms [108], [17], etc. Note that CPLEX software package is presented by the IBM and this software package implements optimizers based on the simplex algorithms [124]. Since, optimization problems considered in this thesis are computationally challenging and exact algorithm to solve these problems in reasonable computing resources is not known to the author. Therefore, a practical approach is adopted to solve the proposed computationally challenging problems in moderate computing resources by using approximate algorithms such as evolutionary algorithms. Main focus of this thesis is on evolutionary algorithms, in particular, fireworks algorithms.

2.1. Evolutionary computation

Modern genetics is based on Darwinian evolutionary theory, which explains the evolution of earthly species. This principle is extended to design evolutionary algorithms (EAs) such as the genetic algorithm (GA). The natural evolution of species is a process of learning about and adapting to the environment and thus optimizing [20], [21]. The success of the GA inspired the use of other naturally evolving phenomena such as ant colonies, honey bee foraging, fish schools, bird flocks, and particle swarms to design EAs [24], [38]–[40]. We modified a relatively new fireworks algorithm (FWA) to apply it to computationally challenging optimization problems. These modifications included changes in FWA operators, combining multiple local search methods of the FWA, and hybridization of FWA operators with other EA operators. In this chapter, we review the evolutionary algorithms considered in this thesis.

2.1.1. Entities and operations of evolutionary algorithms

A number of procedures and operators must be specified to define an EA [41]. Regardless of origin, most EAs contain a flowchart similar to the chart shown in Figure 2.1. In the following subsection, entities and operations of EAs are described [40], [42].

2.1.1.1. Individuals

A candidate solution for an EA can be represented by considering the problem structure and employing a search algorithm. The efficiency and complexity of a search algorithm largely depends on how suitably the problem has been represented in the search method [41], [43]. A difficult problem must be represented suitably in order to work efficiently with an algorithm.

2.1.1.2. Objective/Fitness function

The quality of a candidate solution is determined using a mathematical function called an objective function. The objective function has an important role in an EA because the evolutionary operators usually make use of the cost or fitness evaluation of candidate solutions. EAs use fitness evaluations of a population to make operational decisions [42].

2.1.1.3. Population

An EA population consists of several individual or candidate solutions. The standard way of generating an initial population is to assign a random value from the allowed domain to each component of each candidate solution. The purpose of random selection is to ensure that the initial population is a uniform representation of the entire search space. If some regions of the search space are not covered by the initial population, these parts may be neglected by the search process [42]. In addition to the initial population, EAs generate a new population at every generation.

2.1.1.4. Parent/mate selection

The parent selection or mate selection mechanism distinguishes among individuals based on their quality [41]. This allows the better individuals to become parents of the next generation. An individual is considered a parent if it has been selected to produce offspring.

In EAs, the parent selection mechanism is typically probabilistic. Thus, high-quality individuals have more chance of becoming parents than low quality individuals. Nevertheless, low-quality individuals are often given a small chance to become parents, otherwise, the population may get stuck in a local optimum because of a too greedy search.

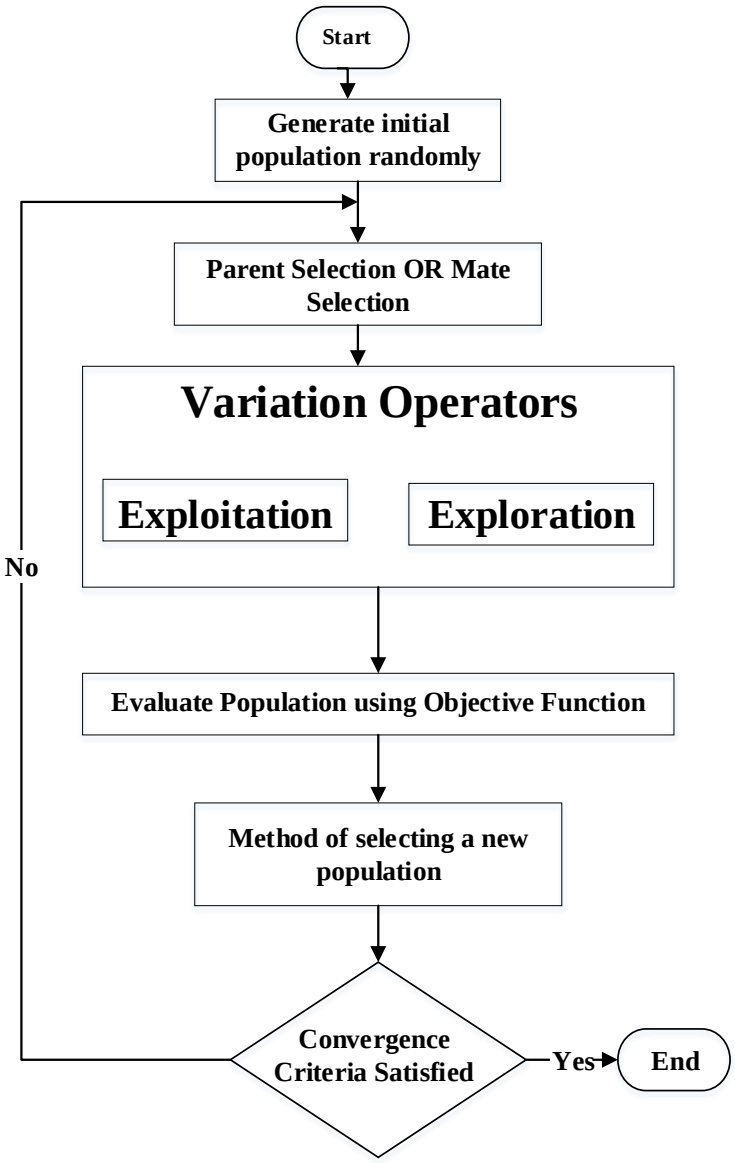


Figure 2.1 EA Flowchart.

Different EAs implement the selection operation differently. In some EAs, selection operators sort the solution population according to fitness and deterministically choose the best solutions for the next algorithm generation. In other EAs, selection operators assign a

probability of selection to each solution according to fitness and generate candidate solutions using a probability distribution [41], [43].

2.1.1.5. Variation operators

EAs generate new candidate solutions by perturbing solutions in the current population. The perturbation process is executed by variation operators. Regardless of name and origin of variation operators, they perform exploitation and exploration in the search space. In EAs, exploitation refers to the use of better solutions (i.e., solutions with a better objective value) for a thorough search in a small region of a big search space, while exploration means to investigate promising regions in the whole search space.

2.1.1.6. Select/generate a new population

The aim of a selection operator is to classify solutions of a population in terms of its objective function values. Then, it selects relatively good solutions from the population and discards the remaining solutions, the rationale being that a solution with better fitness must have a higher probability of selection [41], [43]. The method of selecting/generating a new population is sometimes also called survivor selection, environmental selection, or a population replacement strategy [41]. Like parent selection, new population selection classifies individuals in terms of their objective function values. However, new population selection is used at different stages of the evolutionary cycle. New population selection is used after offspring are generated from the selected parents. The population size after offspring generation may or may not be the same as the population size before offspring generation. If the number of individuals after offspring generation is greater than the population size before offspring generation, a choice is made about which individuals will be allowed to form the next algorithm generation. This decision is often based on objective function values of individuals, and favors individuals having higher fitness values. In contrast to parent selection, which is typically stochastic, the method of selecting a new population is often deterministic.

2.1.1.7. Stopping criteria

Like any other algorithm, an EA must have a stopping criterion. With the use of one stopping criterion at time, an EA can be operated with various stopping criteria. EA operators are iteratively applied until a stopping condition is satisfied. Some common stopping criteria are [40]:

- Maximum generations: the EA iterates for a predefined number of algorithm generations.
- Optimal value: if the optimal value of the objective function is known, the EA search is terminated when it comes to that optimal value.
- Time limit: a user defined maximum running time has elapsed. Other related measures, such as CPU time, the number of generations, or the number of objective function evaluations can be used as well.
- Convergence: the search has converged; convergence is loosely defined as the event when the population becomes stagnant [40].

2.1.2. Intelligence in EAs

Evolutionary algorithms (EAs) are intelligent tools to solve computationally challenging optimization problems. The population is the unit of evolution in an EA [30]. Candidate solutions individually are static objects that do not change or adapt, but individuals in a population can change or adapt. Some typical characteristics of intelligence are adaptation, randomness, communication, feedback, exploration, and exploitation [20]. These characteristics are implemented in an EA for an intelligent algorithm [40].

2.1.2.1. Adaptation

Adaptation to changing environments is considered to be a feature of intelligence. However, adaptation is a necessary but not sufficient condition for intelligence. An EA that can solve a wide class of optimization problems is considered to be more intelligent than

an EA that can solve only a few optimization problems. Adaptability is only one of many criteria for a successful EA.

2.1.2.2. Randomness

We usually think of randomness in negative terms, but it is useful in solving computationally challenging optimization problems. A degree of randomness is a necessary part of an intelligent EA, however, too much randomness is counterproductive [20].

2.1.2.3. Communication

Communication is a feature of intelligence. Communication within a population is a collective behavior of a population-based EA. Intelligence not only involves communication, but it is emergent. That is, intelligence arises from a population of individuals due to collective behavior. A single individual cannot be intelligent. For example, a single ant wanders aimlessly and accomplishes nothing, but a colony of ants can find the shortest path to food, build networks of tunnels, and organize a self-sustaining community. Likewise, a single individual will never accomplish anything if he has no interaction with a community. In candidate solutions to incorporate a communication feature in an EA, the individuals communicate with each other and learn from each other's successes and failures. After a certain time, the population of individuals evolves a good-quality solution to the optimization problem.

2.1.2.4. Feedback

A system is responding, if it senses and reacts to its environment. The new environment cannot be adapted without feedback. Like adaptation and learning, feedback is a fundamental characteristic of EA intelligence and is often recognized in intelligent control theory [20]. Feedback is a necessary, but not sufficient, condition for intelligence. EA designs must incorporate positive and negative feedback.

2.1.2.5. Exploration and Exploitation

Exploration is a search for new ideas or new strategies, and exploitation is the use of existing ideas and strategies that have proven successful in the past. Exploitation is closely related to the feedback strategies discussed above. EA intelligence requires a proper balance of exploration and exploitation. Too much or too little exploration or exploitation is similar to too much randomness, and will probably not lead to good optimization results.

2.2. Swarm intelligence-based evolutionary algorithms

The fireworks algorithm (FWA), the enhanced fireworks algorithm (EFWA), the biogeography-based optimization (BBO) algorithm, and the artificial bee colony (ABC) algorithm are swarm intelligence based evolutionary algorithms in which a population of simple agents behave collectively in a decentralized, self-organized manner [20], [22], [23], [28]. Individual agents in a typical swarm intelligence-based EA can communicate either directly or indirectly with each other by acting on their local environment. An individual agent of a swarm follows very simple rules; however, interactions between such agents can become complicated, causing global behavior that is far beyond the capability of individual agents. This collective behavior of agents in swarm intelligence algorithms inspired researchers to propose a class of evolutionary algorithms that can solve optimization problems. In swarm intelligence based evolutionary algorithms, a swarm is made up of multiple artificial agents. These agents can exchange information in the form of local interactions directly or indirectly (via the environment). In addition to certain stochastic elements, such interaction among agents generates the behavior of adaptive search, and finally leads to global optimization [22].

Motivation: Mainly two factors motivate the author to contribute in the ongoing development of the fireworks algorithm (FWA) to solve the proposed computationally challenging network problems. The enhanced FWA (EFWA), an improved version of the FWA—presented in 2013 [23], was a relatively new development in the area of swarm intelligence and performance of the EFWA was encouraging for the continuous space benchmark problems. This encouraging performance of the EFWA (for continuous

problems) was one of the motivating factors to develop discrete FWA (DFWA) and its variants for the proposed discrete space optimization problems considered in chapter 3 and 4 of the theses. In 2015, two new discrete FWAs were presented by incorporating local search methods for the combinatorial optimization problems [22], [30]. In [22], DFWA was not a better performing algorithm for the traveling salesman problem (TSP) and in [30], performance of the DFWA for a real world combinatorial problem was not compared against state-of-the-art algorithms. Limited work on discrete version of the FWA and inadequate experimental results in existing work [22], [30] was another motivating factor for experimental exploration and development of the various versions of the discrete FWA in the chapter 5. In the following subsections, FWA, EFWA are discussed in detail, and BBO algorithm, ABC algorithm, and Genetic algorithm are discussed briefly.

2.2.1. Fireworks algorithm

The fireworks algorithm (FWA) was first presented in 2010, and it is inspired by the phenomena displayed in real fireworks [22], [23], [28]. In the FWA, a firework or a spark (i.e., candidate solution) can be mathematically represented by a vector of m components. The FWA has four operations: the explosion operator, the mutation operator, the repair mechanism, and the selection operator. In the FWA, the explosion operator is used as an exploitation procedure, and the Gaussian mutation operator is used as an exploration procedure. If a candidate solution is out of the feasible space, the FWA adopts a repair mechanism to allow the candidate solutions to move into the feasible space. The FWA adopts a selection operator to select the population in each algorithm generation [22], because the number of candidate solutions generated in one FWA generation is greater than the population.

2.2.1.1. Explosion operator

To solve an optimization problem, the FWA initially randomly generates a population of N fireworks (i.e., candidate solutions), and each firework is evaluated by using the cost function of the optimization problem. In the population of N fireworks, a firework with a lower cost value is considered a good firework and a firework with a higher

cost value is considered a bad firework. The cost value determines the quality of each firework, which plays an important role in specifying the criteria of the explosion operator. In the FWA, the explosion operator is used to perturb a firework to generate sparks, using offset displacement and two parameters: explosion strength and explosion amplitude [23].

A. Explosion strength

The fireworks algorithm (FWA) determines the number of sparks for each firework in the population of *N* fireworks. The explosion strength refers to the number of sparks generated by a firework explosion. The cost of a firework and user defined parameters are used to determine the number of sparks that are generated by a firework. The authors in [22] designed the FWA in such a way that a firework with a lower cost (good firework) generated more sparks, and a firework with a higher cost (bad firework) generated fewer sparks. The rationale behind generating more sparks around a good firework is to exploit the good firework, therefore a thorough search is conducted to find a better solution around the good firework. However, a bad firework that generates fewer sparks avoids unnecessary computing. Therefore, the sparks generated from the bad fireworks were used to explore the search space and prevent the algorithm from being trapped in a local minimum. The FWA computes the explosion strength s_i for the i^{th} firework as follows:

$$s_i = \text{round}\left(M_e \times \frac{\text{Y}_{max} - f(X^i) + \varepsilon}{\sum_{i=1}^{N}\left(\text{Y}_{max} - f(X^i)\right) + \varepsilon}\right), \text{ where } i = 1,2,\dots,N, \tag{2.1}$$

where s_i is the number of sparks for the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{max} is the maximum cost of the *N* fireworks in the current algorithm generation, $f(X^i)$ represents the cost of the i^{th} firework, M_e is a constant that controls the total number of sparks generated by *N* fireworks, and ε is a small constant used to avoid a division by zero in (2.1).

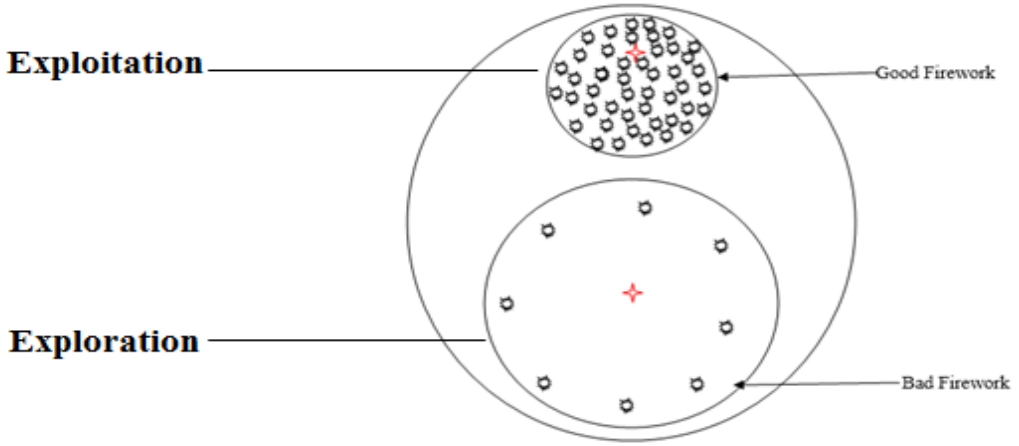


Figure 2.2 Good/Bad fireworks.

Initially, the cost, $f(X^i)$, of the population of N fireworks is computed, where $i = 1,2,\dots,N$. Then, the number of sparks for each firework is computed using equation (2.1). The fireworks with lower cost produce more sparks (good explosion) and the fireworks with higher cost produce fewer sparks (bad explosion) as indicated by equation (2.1) [22], [23]. To avoid the overwhelming effects of outstanding fireworks on good locations, a bound for the number of sparks s_i is applied as follows [22], [23]:

$$s_i = \begin{cases} s^{min} & if\ s_i < s^{min}, \\ s^{max} & esle\ if\ s_i > s^{max}, \\ s_i & else \end{cases} \text{ where } i = 1,2,\dots,N. \qquad (2.2)$$

B. Offset displacement

In the explosion of a firework, an offset displacement is added probabilistically in one or more components of a firework with user determined probability to generate a spark. The offset displacement of a firework is determined randomly within the explosion amplitude of that firework to ensure diversity in the newly generated sparks. The offset displacement ΔX^i for the i^{th} firework is computed once for each generation of the FWA, where $i = 1,2,\dots,N$. In the FWA, the offset displacement for the i^{th} firework, where $i = 1,2,\dots,N$, is computed as follows [23], [25]:

$$\Delta X^i = A_i \times rand(-1,1), \text{ where } i = 1,2,\dots,N, \qquad (2.3)$$

where A_i is the explosion amplitude of the i^{th} firework. An X_q^i is a probabilistically selected component of the i^{th} firework with a user determined probability, and X_q^i is updated using the offset displacement ΔX as follows:

$$\widetilde{X_q^i} = X_q^i + \Delta X, \text{ where } i = 1,2,\dots,N, \tag{2.4}$$

where $\widetilde{X_q^i}$ is the q^{th} component value of a newly generated spark. Pseudo code of the Algorithm 2.1 is run once to generate an explosion spark.

Algorithm 2.1: Generating explosion sparks in the FWA

Inputs:

- X: a vector of *m* component. Note that X is a firework (a candidate solution).

Algorithm parameters:

- *sparkProb:* spark probability [0,1] *// user determined explosion probability*
- *A:* Explosion amplitude *(see 2.2.1.1-C)*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. Calculate the offset displacement: $\Delta X = A \times rand(-1,1)$
2. ***for*** q = 1 to m // *m is number of components in X*
3. ***if*** *rand ()* < *sparkProb*
4. $\widetilde{X_q} = X_q + \Delta X$ // *perturbing the q^{th} component (see 2.2.1.1-B)*
5. ***end if***
6. ***if*** $\widetilde{X_q}$ is out of feasible search space
7. $\widetilde{X_q} = X_q^{min} + |X_q|\%\ (X_q^{max} - X_q^{min})$ // *Repair mechanism (see 2.2.1.3)*
8. ***end if***
9. ***end for***

C. Explosion amplitude

The explosion amplitude quantifies the range of the displacement that is used to perturb one or multiple components of a firework. The explosion of a firework produces sparks by adding displacement probabilistically in one or more components of a firework with a user determined probability. The cost values of fireworks and user defined parameters are used to determine the amplitude of generated sparks. The authors in [22] designed the FWA in such a way that a firework with a lower cost value should generate sparks with smaller amplitude and a firework with a higher cost should generate sparks with larger amplitude. The rationale behind generating sparks with smaller amplitude is to exploit the good firework and conduct a thorough search to find a better solution around

the good firework. The rationale behind generating sparks with larger amplitude from the bad fireworks is to explore the search space and prevent the algorithm from being trapped in a local minimum. The following expression is used to determine the amplitude for each of the *N* fireworks:

$$A_i = \hat{a} \times \frac{f(X^i) - Y_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X^i) - Y_{min}) + \varepsilon}, \text{ where } i = 1,2,\dots,N, \quad (2.5)$$

where A_i is the amplitude associated with the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{min} is the minimum cost among the *N* fireworks in the current algorithm generation, $f(X^i)$ represents the cost of the i^{th} firework, â is a constant used to control the amplitude, and ε is a small constant used to avoid division by zero in (2.5).

2.2.1.2. Gaussian mutation operator

The Gaussian mutation operator is introduced into the FWA to improve the diversity of the population. The number of fireworks (for Gaussian mutation) is a user defined parameter, which can be set to less than or equal to population of the *N* fireworks. In the FWA, $\mathcal{Z}$ denotes the set of fireworks for Gaussian mutation and these fireworks are randomly selected from population of the *N* fireworks where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. Unlike the sparks generated by the explosion operator, each of the fireworks $X^i \in \mathcal{Z}$ generates only one spark using the Gaussian mutation operation. The Gaussian mutation operator is used to perturb one or more components of a firework $X^i \in \mathcal{Z}$ to generate a spark. The component X_q^i of the Gaussian firework $X^i \in \mathcal{Z}$ is probabilistically selected with a user determined probability, and is updated as follows [22], [23]:

$$\widetilde{X_q^i} = X_q^i \times Gaussian(1,1), \quad (2.6)$$

where $\widetilde{X_q^i}$ is a component value of a newly generated spark. The function $Gaussian(1,1)$ denotes the Gaussian random variable with the mean and the standard deviation both set to 1. Pseudo code of the Algorithm 2.2 is run once to generate a Gaussian spark.

2.2.1.3. Repair mechanism

All the fireworks and sparks that are generated by the explosion operation or the Gaussian mutation operation may fall out of the feasible space. Candidate solutions (fireworks and sparks) that fall out of a feasible space are considered infeasible and the infeasible candidate solutions need to be moved back into the feasible space [22]. A repair mechanism is used to deal with infeasible candidate solutions to ensure that all candidate solutions are in the feasible space. Suppose that X_q^i is the component of a candidate solution, which falls in the infeasible space. The FWA uses an operator with a modulo operation (remainder of division), %, to update the component X_q^i as follows [23]:

$$\widetilde{X_q^i} = X_q^{min} + |X_q^i|\% \, (X_q^{max} - X_q^{min}), \tag{2.7}$$

where $\widetilde{X_q^i}$ is a component of a newly generated spark, X_q^{max} and X_q^{min} refer to the lower and upper bounds of the search space in the dimension q, and % denotes that the modulo operation refers to the remainder of division. Quotient q and remainder r of numbers ℓ_1 divided by ℓ_2 satisfy the following: $\ell_1 = q \times \ell_2 + r$ and $|r| < |\ell_2|$. However, the operator in (2.7) is too general and can repair only the infeasible solutions of the optimization problems with rectangular constraints.

Algorithm 2.2: Generating Gaussian sparks in the FWA

Inputs:

- X: a vector of m components. Note that X is a Gaussian firework (*see 2.2.1.2*).

Algorithm parameters:

- *mutateProb:* spark probability [0,1] *// user determined mutation probability.*

Output:

- $\check{X}$, a *spark, a vector of m components.*

Steps:

1. Compute offset displacement $e = Gaussian(1,1)$.
2. ***for*** q = 1 to m *// m is number of components in X*
3. ***if*** *rand ()* < *mutateProb*
4. $\widetilde{X_q} = X_q \times e$ *// perturbing the q^{th} component (see 2.2.1.2)*
5. ***end if***
6. ***if*** $\widetilde{X_q^i}$ is out of feasible search space
7. $\widetilde{X_q} = X_q^{min} + |X_q|\% \, (X_q^{max} - X_q^{min})$ *// Repair mechanism (see 2.2.1.3)*
8. ***end if***
9. ***end for***

Example: A well-known benchmark function '*Generalized Rastrigin*' has upper and lower bounds in the interval $[-5.12, 5.12]$, where $X_q^{max} = 5.12$ and $X_q^{min} = -5.12$. In the explosion operation using (2.4) and (2.5), if any of the probabilistically selected components of the i^{th} firework X_q^i, for each of the $i = 1,2, \dots, N$, is updated beyond the upper and lower bound X_q^{min} and X_q^{max}, that component will be considered an infeasible component. The simple constraint in the benchmark function *Generalized Rastrigin* is that any of its probabilistically selected components of a firework should not be updated beyond the upper and lower bound X_q^{min} and X_q^{max}. A probabilistically selected infeasible component X_q^i of a candidate solution is repaired using the operator in (2.7) because the *Generalized Rastrigin* function has rectangular constraints.

The FWA operator in (2.7) cannot be used for optimization problems that have nonrectangular constraints. Therefore, the FWA needs a problem specific repair mechanism to repair infeasible solutions of the optimization problem with nonrectangular constraints.

2.2.1.4. Selection operation

After applying the explosion operator and Gaussian mutation operator, the total number of candidate solutions is greater than the *N* fireworks in the population. Therefore, a choice is made about which candidate solutions will be allowed in the next algorithm generation. Here, we denote h as the total number of candidate solutions that include fireworks, explosion sparks, and Gaussian mutation sparks. In the FWA, a distance-based selection operator [22] is used to select the *N* fireworks for the next algorithm generation. First, the best candidate solution is selected, then (*N−1*) candidate solutions are selected from the remaining candidate solutions by using the distance-based selection operation [22]. In the distance-based selection operation, the Euclidean distance $d(X^i, X^j)$ is the distance between a candidate solution X^i and all other candidate solutions X^j, where $j = 1,2, \dots h$. Note that $R(X^i)$ denotes the sum of the distances between a candidate solution X^i and all other candidate solutions X^j, where $j = 1,2, \dots h$, as shown below [22]:

$$R(X^i) = \sum_{j=1}^{h} d(X^i, X^j) = \sum_{j=1}^{h} \|X^i - X^j\|. \tag{2.8}$$

The roulette wheel probability of selection is computed as follows:

$$p(X^i) = \frac{R(X^i)}{\sum_{j=1}^{\hbar} R(X^j)}. \quad (2.9)$$

One can see that the candidate solutions with larger distances will have more chances to be selected for next algorithm generation [22]. As a result, candidate solutions in the less crowded regions will have more probability than candidate solutions in the crowded regions of being selected for the next algorithm generation. This will ensure diversity in the population computed by the next algorithm generation [23].

2.2.1.5. FWA operation

A pseudo code for the FWA is presented in Table 2.1 using Algorithm 2.1 and Algorithm 2.2. Initially, a population of *N* fireworks is generated randomly, and parameters for the FWA are initialized. After computing the cost value of the fireworks, the sparks s_i, and the amplitudes, A_i, are computed using (2.1) and (2.5), respectively, for each of the *N* fireworks. In the FWA, s_i refers to the number of sparks generated in the i^{th} firework and A_i refers to the amplitudes of sparks generated by the i^{th} firework. For each spark, the offset displacement, (2.3)–(2.4), is added probabilistically to the selected component of the firework X^i, for each of $i = 1,2,\dots,N$, with the user determined '*sparkProb*' probability. If the displacement operator maps a candidate solution outside the search space, the solution is updated to the feasible search space using the operator in (2.7). In the FWA, each firework is perturbed probabilistically to generate sparks around that firework using algorithm 1. All explosion sparks are evaluated using the cost function of optimization.

Now, a set $\mathcal{Z}$ of fireworks is randomly selected (for Gaussian explosion) from a population of *N* fireworks to execute the exploration process, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of set $\mathcal{Z}$. For each firework X in $\mathcal{Z}$, the Gaussian mutation operator (2.6) is used to map the value to each probabilistically selected component of a firework with the user determined '*mutateProb*' probability using Algorithm 2.2. After applying the Gaussian mutation operation to the X in $\mathcal{Z}$ firework, the Gaussian mutation sparks are evaluated using the cost function of the optimization problem. Now, the FWA selects a

population of the *N* fireworks from the total number of *h* candidate solutions that includes fireworks, explosion sparks, and Gaussian mutation sparks. In the FWA, first the best solution in the current generation of the algorithm is selected. Then, (*N−1*) fireworks are selected from the rest of the candidate solutions using a distance-based strategy (2.8)–(2.9).

Table 2.1 FWA pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$. 2. Initialize the *sparkProb* and *mutateProb*. 3. Declare *S* as an empty set of sparks.
B. Execution	4. ***while*** (stopping criteria not satisfied) 5. ***for*** $i = 1,2, \dots, N$ 6. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (2.1) and (2.5) respectively. 7. ***for*** $j = 1, 2, \dots, s_i$ 8. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 2.1**. 9. Add generated sparks in *S*. 10. ***end for*** 11. ***end for*** 12. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 2.2.1.2*) from a population of *N* fireworks. 13. ***for*** each firework X in $\mathcal{Z}$ 14. Generate mutation spark $\check{X}$ using **Algorithm 2.2**. 15. Add generated sparks in *S*. 16. ***end for*** 17. Select the best solution and the (*N−1*) solutions using (2.8)−(2.9) selection operation to make new population of the *N* fireworks for next algorithm generation. 18. ***end while***
C. Output	19. ***return*** the best solution found so far.

For the FWA and its variants in this thesis, we consider minimization as an optimization objective in our discussion unless stated otherwise. The term *'cost'* is used for the objective function value of the optimization function. The cost of the fireworks and control parameters are used to calculate the number of sparks s_i and the amplitudes A_i using (2.1) and (2.5), for $i = 1,2, \dots, N$. However, when maximization is an optimization objective, the FWA uses fitness values of fireworks and control parameters to calculate the number of sparks s_i and the amplitudes A_i as follows:

$$s_i = \text{round}\left(M_e \times \frac{f(X^i) - \text{Y}_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X^i) - \text{Y}_{min}) + \varepsilon}\right), \text{ where } i = 1,2,\dots,N, \quad (2.10)$$

$$A_i = \hat{a} \times \frac{\text{Y}_{max} - f(X^i) + \varepsilon}{\sum_{i=1}^{N}\left(\text{Y}_{max} - f(X^i)\right) + \varepsilon}, \text{ where } i = 1,2,\dots,N, \quad (2.11)$$

where s_i is the number of sparks, A_i is the amplitudes associated with the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{min} and Y_{max} are minimum and maximum costs, respectively, among the population of the N fireworks, $f(X^i)$ represents the cost of the i^{th} firework, M_e and $\hat{a}$ are constants used to control the number of sparks and the spark amplitudes, respectively, and ε is a small constant to avoid division by zero in (2.10) and (2.11).

2.2.2. Enhanced fireworks algorithm

An enhanced version of the FWA, the enhanced fireworks algorithm (EFWA) is a relatively recent development in swarm intelligence based evolutionary algorithms (EAs) [22], [23]. Several drawbacks were observed when the FWA was applied to some well-known benchmark problems [22], [23], as shown in Table 2.3 of this chapter. The EFWA is structurally similar to the FWA but includes enhancements and modifications of FWA operators. The FWA has the following drawbacks [23]:

1. The FWA does not perform well for functions that have optimal locations far from the origin, although it performs well for functions that are close to the origin [22] . Poor performance of the FWA, for the functions that have optimal locations far from the origin, is mainly caused by the following two operators:
   (i) The Gaussian mutation operator
   (ii) The repair mechanism
2. The distance-based selection operator used in the FWA has a high computational cost per generation.

Like the FWA, the EFWA has four operations: an explosion operator, a Gaussian mutation operator, a repair mechanism, and a selection operator. The development of the EFWA was undertaken to improve FWA operators and to mitigate FWA drawbacks [23]. FWA drawbacks and EFWA enhancements are discussed in the following subsections.

2.2.2.1. Explosion operator

As in the FWA, the EFWA explosion operator is used to perturb a firework to generate sparks using offset displacement and two parameters: explosion strength and explosion amplitude [23].

A. Explosion strength

The explosion strength determines the number of sparks generated in a firework explosion. Similar to the FWA, the EFWA uses the formula in (2.2) to determine explosion strength and the bound in (2.3) for the number of sparks.

B. Offset displacement

After computing the explosion amplitude, the EFWA determines the displacement within the explosion amplitude A_i of the i^{th} firework, where $i = 1,2, \dots, N$. The EFWA uses a displacement operator that is different from the displacement operator in the FWA. The drawbacks of the displacement operator in the FWA and the improvements in the displacement operator in the EFWA are described below.

Drawback in the FWA offset displacement

In the FWA, offset displacement ΔX^i (as in 2.4) is calculated once in each generation of the FWA for each of the N fireworks. Then, the ΔX^i is added to the probabilistically selected components of the i^{th} firework with user determined probability, where $i = 1,2, \dots, N$. Clearly, adding the same displacement value to the probabilistically selected components of a firework in one algorithm generation compromises the diversity of the local search in the FWA. Addition of the same offset displacement ΔX^i to probabilistically selected components of the firework severely affected the FWA's progress [23], but this obvious loophole was not properly addressed in the initial version of the FWA [21].

New offset displacement

In each EFWA generation, an offset displacement, ΔX_q^i, is calculated for each probabilistically selected component q of the i^{th} firework with a user determined probability for each of the $i = 1,2, \dots, N$ fireworks. Then, the new ΔX_q^i is added to the probabilistically selected component X_q^i of the i^{th} firework to ensure diversity in the sparks. Diversity in the local search is also improved in the EFWA compared to the FWA [23]. The offset displacement is computed as follows:

$$\Delta X_q^i = A_i \times rand\ (-1, 1), \text{ where } i = 1,2, \dots, N. \quad (2.12)$$

A component, X_q^i, from the i^{th} firework is probabilistically selected with user determined probability, and is updated with offset displacement as:

$$\widetilde{X_q^i} = X_q^i + \Delta X_q^i, \text{ where } i = 1,2, \dots, N, \quad (2.13)$$

where $\widetilde{X_q^i}$ is the value of the q^{th} component of the newly generated spark. Pseudo code of the Algorithm 2.3 is run once to generate an explosion spark $\widetilde{X^i}$.

C. Explosion amplitude

The explosion amplitude determines the range of displacement that is added probabilistically in one or more components of a firework to generate a spark. The cost of a firework and parameters are used to determine the explosion amplitude for that firework. Like the FWA, the EFWA uses equation (2.5) to determine the amplitude of a firework. In the FWA, A_i is the amplitude of the i^{th} firework, where $i = 1,2, \dots, N$, is used to determine the displacement of the newly generated sparks with user determined probability. However, there is a constant lower bound on amplitude that renders some drawbacks in the FWA. An adaptive lower bound A^{min} is introduced in the EFWA to mitigate the drawbacks in the FWA.

Algorithm 2.3: Generating explosion sparks in the EFWA

Inputs:

- X: a vector of m component. Note that X is a firework (a candidate solution).

Algorithm parameters:

- *sparkProb:* spark probability [0,1] *// user determined explosion probability*

- *A:* Explosion amplitude *(see 2.2.2.1-C)*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. ***for*** q = 1 to m *// m is number of components in* X^i
2. ***if*** *rand* < *sparkProb*
3. Calculate the offset displacement: $\Delta X_q = A \times rand(-1,1)$
4. $\widetilde{X_q} = X_q + \Delta X_q$ *// perturbing the* q^{th} *component (see 2.2.2.1-B)*
5. ***end if***
6. ***if*** $\widetilde{X_q}$ is out of feasible search space
7. $\widetilde{X_q} = X_q^{min} + rand\,(X_q^{max} - X_q^{min})$ *// Repair mechanism (see 2.2.2.3)*
8. ***end if***
9. ***end for***

A new minimal explosion amplitude check in the EFWA

A firework with a lower cost will have a smaller explosion amplitude, i.e., close to 0, while a firework with a larger cost has a larger explosion amplitude [23], as calculated in (2.5); If the explosion amplitude is close to zero, the explosion sparks will be located at (almost) the same location as the firework itself. To avoid this problem, a lower bound which we denote as A^{min} of the explosion amplitude is introduced based on the progress of the algorithm. During the early phase of the search, A^{min} is set to a higher value of explosion amplitude for more exploration. However, with an increasing number of objective function evaluations, the value of A^{min} is decreased for more exploitation. For each component of the i^{th} firework, the explosion amplitude A_i is defined as follows [22], [23]:

$$A_i = \begin{cases} A_i^{min} & if \;\; A_i < A_i^{min} \\ A_i, & otherwise \end{cases}, \text{ where } i = 1,2,\dots,N. \qquad (2.14)$$

A new value A_i of the i^{th} firework, for each of $i = 1,2,\dots,N$, is calculated in each algorithm generation and two different formulas can be used to calculate A_i^{min} in the EFWA. In the first approach A_i^{min} linearly decreases with the progress of the EFWA and in the second approach, A_i^{min} nonlinearly decreases with the progress of the EFWA. At the initial stage of the EFWA, A^{min} is set to a higher value to foster more exploration to find a promising region in the search space of the optimization problem. However, A^{min}

decreases with the progress of the algorithm to boost the exploitation of the good firework in the EFWA. Unlike the FWA, the EFWA uses linear and nonlinear decreases in A_i^{min}, as calculated below [22], [23]:

$$A_i^{min} = A^U - \frac{A^U - A^L}{t^{max}} \times \mathrm{t}, \tag{2.15}$$

$$A_i^{min} = A^U - \frac{A^U - A^L}{t^{max}} \times \sqrt{(2 \times t^{max} - t) \times t}, \tag{2.16}$$

where A^U and A^L are highest and lowest points of the minimum explosion amplitudes, t is the current number of function evaluations in a generation, and t^{max} is the maximum number of function evaluations (as a stopping criteria) for the EFWA.

2.2.2.2. Gaussian mutation operator

A set $\mathcal{Z}$ of fireworks (for Gaussian mutation) are randomly selected from the population of N fireworks in an FWA, where $|\mathcal{Z}| < N$, and $|\mathcal{Z}|$ is cardinality of (or number of elements in) the set $\mathcal{Z}$. Unlike the explosion operator, each of the fireworks $X^i \in \mathcal{Z}$ can generate only one spark using the Gaussian mutation operation. In the FWA after extensive experimentation, some drawbacks were observed in the Gaussian mutation operator [23]. Here, the drawbacks observed in the Gaussian mutation operator of the FWA are discussed and the new Gaussian mutation operator that is adopted for the EFWA is explained [21].

Drawbacks in the FWA Gaussian mutation operator

FWA performance varies with changes in characteristics of the objective functions. One such function is the well-known bench mark two-dimensional Ackley function with an optimal value at the origin (i.e., [*0, 0*]). Experimental results show that the Gaussian mutation operator is the main reason why the FWA works significantly better than other classic optimization algorithms for the Ackley function. In [23], the FWA is used after shifting the optimal value from the origin of the Ackley function. However, after shifting the origin of the FWA, the Gaussian mutation sparks generated are still close to the origin, even though the optimal value is now far away from the origin. These facts reveal that the

Gaussian sparks generated at the origin are not the result of FWA intelligence, rather FWA intelligence has no influence on the location of the function.

In the FWA, Gaussian sparks are generated close to the origin due to the Gaussian mutation operator (2.6), where mean and variance are both set to 1 for the Gaussian function. In cases where the value of the Gaussian function is close to 0, the new component value (i.e., $\widetilde{X_q^i}$) will be close to 0 as well. As a result, many Gaussian sparks will be located close to the origin of the search space in dimension q. Moreover, for large Gaussian values, many Gaussian sparks are created at locations that are outside the search space. Another potential problem with the Gaussian mutation operator is that the fireworks that are already located close to the origin of the search space cannot escape from that location due to (2.6). Apparently, in the first version of the FWA, parameters were not properly tuned, such as setting different values for mean μ and variance σ for random values of a normal distribution [21].

New Gaussian mutation operator for the EFWA

To overcome the drawbacks observed in the FWA, a modified Gaussian mutation operator is introduced in the EFWA. The EFWA adopts a Gaussian mutation operation to ensure diversity in the generated spark. In the EFWA, new sparks are generated between the best fireworks among the population of *N* fireworks, and a firework from set $\mathcal{Z}$ of the fireworks, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of set $\mathcal{Z}$. The probabilistically selected component X_q^i of the firework $X^i \in \mathcal{Z}$ with user determined probability is perturbed using the Gaussian distribution as follows:

$$\widetilde{X_q^i} = X_q^i + \left(X_q^b - X_q^i\right) \times Gaussian\,(0,1), \tag{2.17}$$

where $\widetilde{X_q^i}$ is a component of the newly generated spark, and X_q^b is a component of the best solution in the current algorithm generation, where the $Gaussian\,(\mu, \sigma)$ is the Gaussian distribution with mean $\mu = 0$ and variance $\sigma = 1$. Pseudo code of the Algorithm 2.4 is run once to generate a Gaussian explosion spark $\widetilde{X^i}$.

Shift function: In [23], seven different shift values are used to analyze the influence of shift values on the performance of the EFWA. A well-known benchmark function 'Ackley' with two dimensions (2D) is expressed as follows:

$$f(x,y) = -20\exp\left[-0.2 \times \sqrt{0.5 \times (x^2 + y^2)}\right] - \exp[0.5(cos2\pi x + cos2\pi y)] + e + 20.$$

The optimal value of the 2D *'Ackley'* function at $x = 0$ and $y = 0$ is $f(0,0) = 0$.

We can shift the 2D *'Ackley'* function by using a shift value (SV), also known as a displacement value. For example, if we want to displace/shift the function $f(x,y)$ along x- and y-axes, we can add -10 to *x* (for displacement along the x-axis) and we can add -20 to y (for displacement along the y-axis), then the displaced/shifted function $g(x,y) = f(x-10, y-20)$ can be expressed as follows:

$$g(x,y) = -20\,exp\left[-0.2\sqrt{0.5((x-10)^2 + (y-20)^2)}\right] - exp[0.5(cos2\pi(x-10) + cos2\pi(y-20))] + e + 20.$$

The optimal value of the 2D *'Ackley'* function $g(x,y)$ at $x = 10$, and $y = 20$ is $g(10,20) = 0$.

Algorithm 2.4: Generating Gaussian sparks in the EFWA

Inputs:

- X: a vector of *m* components. Note that X is a mutation firework (*see 2.2.1.2*).
- X^b: a vector of *m* components. Note that X^b is the best solution amongst *N* fireworks.

Algorithm parameters:

- *mutateProb:* spark probability [0,1] *// user determined mutation probability.*

Output:

- $\breve{X}$, a *spark, a vector of m components.*

Steps:

1. Calculate the offset displacement: $e = Gaussian(0,1)$
2. ***for*** q = 1 to *m // m is number of components in X*
3. ***if*** *rand ()* < *mutateProb*
4. $\breve{X}_q = X_q + (X_q^b - X_q) \times e$ *// perturbing the q[th] component (see 2.2.2.2)*
5. ***end if***
6. ***if*** $\breve{X}_q^i$ is out of feasible search space
7. $\breve{X}_q = X_q^{min} + rand\,(X_q^{max} - X_q^{min})$ *// Repair mechanism (see 2.2.2.3)*

8. ***end if***
9. ***end for***

2.2.2.3. Repair mechanism

In the EFWA, fireworks and sparks may fall in the infeasible space after executing the explosion and Gaussian mutation operators. The sparks in the infeasible space are called infeasible sparks and are useless for further evolution of the EFWA. Therefore, infeasible sparks need to be returned to the feasible space. The repair mechanism is used to deal with infeasible solutions. First, we present some drawbacks observed in [23] in the operators that are used to repair infeasible solutions in the FWA, then we describe the new operator for the EFWA.

Drawback of the FWA repair operator

The fireworks algorithm (FWA) uses an operator with a modulo operation (remainder of division), %, to update the component of an infeasible candidate solution [23] as shown in (2.7). In the FWA, when the location of a new spark exceeds the search space, the spark is infeasible. To make a spark feasible, the spark must be updated to another location using the operator in equation (2.7).

In the FWA, when the location of a new spark exceeds the search range in dimension *q*, the new spark will be mapped to another location using the repair mechanism in (2.7), i.e., $\widetilde{X_q^i} = X_q^{min} + |X_q^i|\%\ (X_q^{max} - X_q^{min})$. In many cases, a spark will go outside the search space only by a small value. Furthermore, as the search space is often equally distributed ($X_q^{max} \equiv -X_q^{min}$), the adjusted position of component $\widetilde{X_q^i}$ will be very close to the origin in many cases [23]. For example, consider an optimization problem within the search space [−20, 20]. If, in dimension *q*, a new spark is created at the point X^i = 21, it will be mapped to the location $\widetilde{X_q^i}$= −20 +|21| % (40). Since the result of the modulo operation 21 % (40) = 21, X^i will be mapped to the location X_q^i= 1, which is very close to the origin. In cases where $X_{min} \equiv -X_{max}$, this mapping operator is partly responsible for drawback (1) as mentioned at the beginning of section 2.2.2.

A new uniform random repair operator

To avoid the FWA drawbacks, the EFWA algorithm replaces $|X_q^i|\%$ in (2.9) with a uniform random operator *rand* to repair the infeasible solutions as follows:

$$\widetilde{X_q^i} = X_q^{min} + rand\,(X_q^{max} - X_q^{min}), \tag{2.18}$$

where $\widetilde{X_q^i}$ is the updated component of a firework or a spark and X_q^{max} and X_q^{min} refer to the lower and upper bounds of the search space in dimension q. However, the operator in (2.18) is still too general and can repair infeasible solutions of the optimization problems only with rectangular constraints.

Example: A well-known benchmark function '*Sphere*' has upper and lower bounds in the interval $[-100,\ 100]$, where $X_q^{max} = 100.0$ and $X_q^{min} = -100.0$. In the explosion operation using (2.14) and (2.15), if any of the probabilistically selected components of the i^{th} firework X_q^i, for each $i = 1,2, \dots, N$, is updated beyond the upper and lower bounds X_q^{min} and X_q^{max}, that component will be considered to be an infeasible component. The simple constraint in the benchmark function *Sphere* is that none of its probabilistically selected firework components should be updated beyond upper and lower bounds X_q^{min} and X_q^{max}. A probabilistically selected infeasible component X_q^i of a candidate solution is repaired using the operator (2.18) because the *Sphere* function has rectangular constraints. The EFWA needs a problem specific repair mechanism to repair infeasible solutions for the optimization problems with nonrectangular constraints.

2.2.2.4. Selection operation

The FWA and the EFWA use different selection operators to select a population of *N* fireworks for the next algorithm generation. The FWA uses a distance-based selection operator to select fireworks for the next algorithm generation. Drawbacks of the distance-based selection operation that was adopted for the FWA and an elitism random selection operation that is adopted for the EFWA are discussed below.

Drawback of the distance-based selection operation

The FWA uses a distance-based selection operator to select solutions from the less crowded regions of the search space in (2.8)–(2.9) [23]. Although, selecting solutions from the less crowded region with higher probability increases the diversity of the search, this process is computationally expensive. In [23], after computing the runtime for the FWA, it was observed that the selection operator was responsible for significant time consumption. Surprisingly, there were no specific/scientific reasons presented in the FWA to justify why the distance-based selection operation was used to select the population of *N* fireworks for the next algorithm generation [21]. In the EFWA, the simpler and less computationally expensive elitism-random selection strategy performs far better than the distance-based selection strategy [23].

A new elitism-random selection method

To speed up the selection process, the EFWA uses an elitism-random selection operation that significantly reduces the runtime of the EFWA. In the EFWA, the solution with the best cost is selected for the next algorithm generation. Then, $(N-1)$ candidate solutions are randomly selected from the remaining candidate solutions (i.e., fireworks and sparks) for the next algorithm generation.

2.2.2.5. EFWA operation

The pseudo code for the EFWA is presented in Table 2.2. Initially, a population of N fireworks is generated randomly, and algorithm parameters are initialized. After computing the cost of the population of the N fireworks, sparks s_i and amplitudes A_i are computed using (2.1) and (2.5). In the EFWA, each firework is associated with a number s_i, for each $i = 1,2,\dots,N$ spark it generates. The EFWA generates different random displacements of amplitude A_i, to ensure the diversity of sparks around the i^{th} firework, for each of $i = 1,2,\dots,N$. For each spark, the operator maps the displacement (2.12)–(2.13) of each probabilistically selected component of the firework X^i with a user determined '*sparkProb*' probability. If the displacement operator (2.12)–(2.13) maps the solution outside the search space, then the solution is updated to the search space using (2.18).

In the EFWA context, each firework is perturbed probabilistically to generate sparks around that firework by updating the displacement. This perturbation process exploits the existing small region (around a firework) and conducts a thorough search in a small region by generating sparks using algorithm 2.3. All the sparks are evaluated using the cost function of optimization.

Now, a set $\mathcal{Z}$ of Gaussian fireworks is randomly selected from the *N* fireworks, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework, $X^i \in \mathcal{Z}$, the Gaussian mutation explosion operator (2.17) is used to perturb the value of each probabilistically selected component with a user determined '*mutateProb*' probability using algorithm 2.4. After applying the mutation operator on Gaussian fireworks set $\mathcal{Z}$, the mutation sparks are evaluated using the cost function of optimization. Now, the EFWA selects a population of *N* fireworks from the total number of candidate solutions that include fireworks, explosion sparks, and Gaussian mutation sparks. In the EFWA, first, the best solution is selected in the current algorithm generation, then, the (*N–1*) fireworks are randomly selected from the remaining candidate solutions for the next algorithm generation.

Table 2.2 EFWA pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$. 2. Initialize the *sparkProb* and *mutateProb*. 3. Declare *S* as an empty set of sparks.
B. Execution	4. ***while*** (stopping criteria not satisfied) 5. ***for*** $i = 1,2, \dots, N$ 6. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (2.1) and (2.5) respectively. 7. ***for*** $j = 1, 2, \dots, s_i$ 8. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 2.3**. 9. Add generated sparks in *S*. 10. ***end for*** 11. ***end for*** 12. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 2.2.2.2*) from a population of *N* fireworks. 13. ***for*** each firework *X* in $\mathcal{Z}$ 14. Generate mutation spark $\check{X}$ using **Algorithm 2.4**. 15. Add generated sparks in *S*.

	16. ***end for*** 17. Select the solution with the best cost, and ($N-1$) solutions are randomly selected from the remaining candidate solutions (i.e., fireworks and sparks) for next algorithm generation. 18. ***end while***
C. Output	19. ***return*** the best solution found so far.

2.2.3. Biogeography-based optimization algorithm

The biogeography theory describes the migration, extinction, and geographical distribution of species among various islands, as illustrated in Figure 2.3. The term island in biogeography is used in descriptive meaning rather than literal meaning. Any habitat that is geographically isolated from other habitats is considered an island. Geographical areas can be suitable or unsuitable for species due to various factors. In biogeography, each habitat or island can be assigned a habitat suitability index (HSI), which is a measurement of the quality of life in a certain habitat. Some factors that vary the degree of the HSI are rainfall, temperature, vegetation, land area, etc. The variables that characterize the habitability of the habitat are known as suitability index variables (SIVs). Therefore, variations in independent variables (or SIVs) change the value of the HSI (a dependent variable) [20]. For example, a low HSI for a certain habitat reflects a low quality of life in that habitat, and vice versa.

Species travel from one island to another island in search of a favorable environment or a better quality of life. If the number of species grows on an island, then it becomes crowded due to a shortage of resources such as food, water, shelter, land area, etc. The scarcity of resources forces some individuals to emigrate from their island. Also, species from other habitats are less likely to immigrate to crowded islands. Similarly, when there is no species on an island and it has plenty of food, water, land area, and other supplies, species from neighboring islands are likely to immigrate up to the maximum immigration rate. Figure 2.4 shows a migration model in the biogeography, where S_{max} is the largest possible number of species that can be accommodated on an island at which point immigration rate is zero. Emigration rates from the resource filled islands would be

zero as shown in Figure 2.4 [43]. In biogeography, each island has certain immigration rate λ and emigration rate μ.

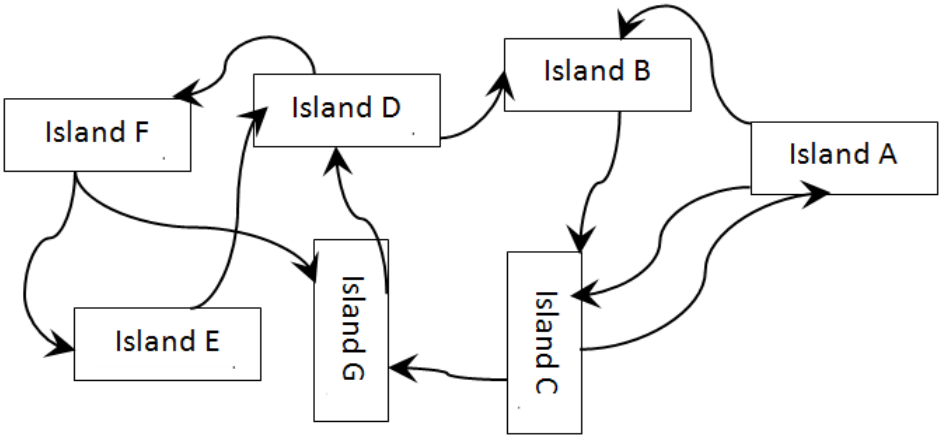


Figure 2.3 Species migration among islands.

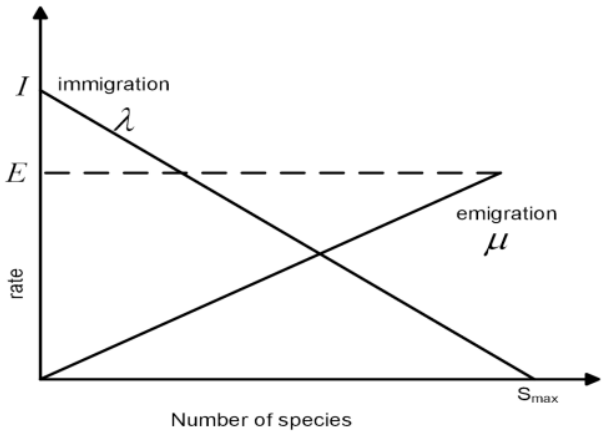


Figure 2.4 Typical BBO migration model [45].

These biogeography-based notions are used to develop the biogeography-based optimization (BBO) algorithm. In the BBO algorithm, each candidate solution (or individual) is considered as a habitat or an island, with a Habitat Suitability Index (HSI) representing its corresponding fitness function value. Each candidate solution (or island) consists of variables (or genes) that characterize the habitability, which is referred to as a Suitability Index Variables (SIVs). Good tutorial materials of the BBO algorithm are found in [20], [26], [44], [45].

2.2.3.1. Low-complexity BBO algorithm

Different algorithms have been developed for the conventional BBO based on different migration models. In this thesis, we use a simplified version of the low-complexity BBO (LC-BBO) algorithm, which is a special case of conventional BBO algorithm [26]. In comparison to the conventional BBO algorithm, in LC-BBO, the emigrating solution is uniformly selected from the population of size *N*. Except this change in migration procedure, rest of migration and mutation procedure in the LC-BBO algorithm is same as the conventional BBO algorithm. The pseudo code for the LC-BBO migration model is presented in the algorithm 2.5.

Algorithm 2.5: LC-BBO migration pseudo code

Inputs:

- X: an island of *m* components. Note that X is a candidate solution.

Algorithm parameters:

- I: immigration rate (*see 2.2.3*).

Output:

- X, *an island, a vector of m components*

Steps:

1. ***for*** q = 1 to m // *m is number of components in an island X*
2. ***if*** $rand\ () < \lambda$ // X_q *accepts immigration* (*see 2.2.3*)
3. Uniformly select an island $\check{X}$ that emigrates SIV to X (i.e.$X \neq \check{X}$)
4. $X_q = \check{X}_q$ // $\check{X}_q$migrates to X_q
5. ***end if***
6. ***end for***

2.2.4. Discrete artificial bee colony algorithm

Societies of insect can be viewed as a complex system of interacting individuals. Individuals in these societies collectively perform decision-making by exploiting the physical constraints of the system. In literature, generally a term swarm is used to refer to any restrained collection of interacting individuals (or agents). Honeybee swarming at their hives is a classic example of such swarm behavior. A model of honeybee forage selection consists of three components: (1) food sources, (2) employed bees and (3) unemployed bees. This model also consists of two leading honey bee behaviors: recruitment to a nectar source, and abandonment of a nectar source [39].

An employed bee is a bee that has been assigned a food source with a certain nectar quality. Unemployed bees are bees that have not been assigned a food source. Onlooker and scout bees are categorized as unemployed bees. An onlooker bee follows the dances (a.k.a. wiggle dance) of employed bees and uses (the dance) information as a guideline to locate a food source. In other words, employed bees dance conveyed nectar information of a food source to onlooker bees. Onlooker bees become employed bees as soon as they select information of food source from one of the employed bees. An employed bee becomes an unemployed bee after abandoning its food source due to poor nectar quality and it becomes a scout bee. The scout bee moves around randomly to discover a fresh food source. Once a new food source is found, the scout bee becomes an employed bee once again [40].

The metaphor of honeybee colony was extended to develop Artificial Bee Colony (ABC) and discrete ABC Algorithms [39]. In ABC algorithm, a candidate solution is referred to as a food source (or food source position). Each food source has a certain nectar quality, which is analogically corresponds to the fitness function value of target optimization problem [24], [45]. The nectar quality of a food source depends on different factors like the richness of the food, the ease of extracting the food, the closeness of the food to the hive, etc. Good tutorial materials of the ABC (or DABC) algorithm are found in [24], [25], [39], [40], [45].

2.3. Genetic algorithm

In evolutionary computation, the genetic algorithm (GA) is the first evolutionary algorithm (EA) and is inspired by science of genetics. GA is a class of EA that is inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (a.k.a. recombination). A GA is a search technique used in computing to find approximate solutions to various types of optimization problems. GA is a global search heuristic using an abstract representation of genetics such as chromosomes and individuals (or candidate solutions) to solve an optimization problem. In GA search operation, individuals evolve toward better solutions. Each candidate solution in GA is evolved by the cost function of the target optimization problem. The cost of the individual represents the quality of the

solution. A GA maintains a large number of individuals [20]. Typically, the GA may have dozens or hundreds of individuals that are commonly known as the population.

The GA operates on a population with various operators to maintain genetic diversity. The process of evolution in GA is a result of genetic variation. Various types of genetic operators are used in GA, which are analogous to real world natural phenomenon such as selection, reproduction (a.k.a. crossover or recombination) and mutation. In subsequent subsections, we briefly discuss the GA operators.

2.3.1. Selection

Some individuals have high fitness value while others have low fitness in a population, and this piece of information can be used for a selection mechanism in GA. In general, low-fitness individuals have a high probability of dying in their generation and vice versa. High probability of dying of species due to low-fitness means the species are unable to survive longer in a lesser fit environment. On the other hand, the species can survive longer in the better fit (high fitness) environment and have relatively low probability of dying. Therefore, low-fitness individuals are removed while the high-fitness individuals produce a new generation of individuals in the GA. This process is continued until the GA finds an acceptable solution to the optimization problem. In literature, many selection procedures are presented. Typically, the GA uses fitness-proportional selection mechanism such as is roulette-wheel selection [20].

2.3.2. Crossover

In GAs' terminology, a pair of individuals selected from a population are called parents. The two parents can mate, just like the individuals in biological populations. To mate two parents, we let them to '*crossover,*' which means that each individual share some of its genetic information with its offspring. A candidate solution in GA can be mathematically represented by a vector of *m* components and each component comprises of some genetic information. By using user-determined crossover probability, the two parents swap their genetic information. The crossover range of genetic information can be

from a single component (or gene) to multiple components (or genes). In other words, two parents have mated (i.e., crossover) to produce two offspring. Each offspring receives some genetic information from one parent, and rest of the genetic information from the other parent. The parents die, and the offspring survive to continue the evolutionary process [20].

2.3.3. Mutation

In GA, mutation is a genetic operator used to maintain genetic diversity from one generation of a population to the next generation. Analogically, mutation in GA is like the biological mutation. Usually in GA, mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. In addition, if some genetic information is missing from the population, mutation provides the possibility of injecting that new information into the population. As a result, GA may come to a better solution by using mutation. Typically, mutation probability in the GA is set to very low value, say 1 percent. This means that after the crossover process produces offspring, each component in each child has a 1 percent probability of altering the value. In case mutation probability is set too high, the search will turn into a random search.

2.4. Computational complexity

Typically, the computational complexity of population-based evolutionary algorithms (e.g., Genetic algorithm (GA)) is analyzed in terms of the number of cost function evaluations [45]. However, the computational complexity is highly dependent on the coding efficiency. In our experimental algorithms, the low complexity biogeography-based optimization (LC-BBO) and the GA, the cost function evaluations are equal to GN, where G is the total number of algorithm generation and N is the population size [45]. As in the LC-BBO algorithm [26] and the GA [20], the cost function is usually evaluated for a candidate solution at least once in an algorithm generation. However, the cost function evaluation may made more than once for a candidate solution such as DABC algorithm and FWA and EFWA.

2.4.1. Discrete artificial bee colony algorithm

A generation of the DABC algorithm consists of three phases, i.e. employed bees, onlooker bees and scout bee phase. Generally, in the employed bee and onlooker bee phases, the function evaluation procedure for the whole population runs twice, i.e., $2N$. In the scout bees' phase, the DABC selects food sources (or candidate solutions) that have not improved their nectar quality after t trials [45]. Then, the DABC algorithm replaces its associated employed bee with a scout that randomly selects a new food source location and keeps its nectar quality in her memory. The trial counter $trial$ is reinitialized to zero if the nectar quality of a candidate solution is improved, and the trial counter $trial$ is incremented if the nectar quality of a candidate solution is not improved. Therefore, the first individual to exceed the trials would be at the $t/2^{nd}$ generation. After the $t/2^{nd}$ generation in the worst-case scenario, every generation sends one scout that runs the function evaluation procedure. The total number of fitness function evaluations for the DABC algorithm in G generations would be [45]:

$$2GN + \left(G - \frac{t}{2}\right). \tag{2.19}$$

The complexity of the DABC algorithm is higher than the complexity of the BBO/LC-BBO algorithms and the GA. In the scout bee phase, a food source is abandoned when $trial > t$ and replaces their associated bees with a scout. The number of these replacements is unknown due to the stochastic nature of the algorithm.

2.4.2. Discrete FWA and its variants

Like the DABC algorithm, the FWA (or EFWA) run cost function evaluations more than the population of firework. Initially in the FWA/EFWA, the population of N fireworks is evaluated by using the cost function. Then, for each firework $i = 1,2, \dots, N$, the number of sparks, s_i, are generated by using explosion operations and is evaluated by using the cost function. We denote M_e as the total number of sparks (or candidate solutions) generated during the explosion operation as follows [46]:

$$M_e = \sum_{i=1}^{N} s_i. \tag{2.20}$$

The FWA/EFWA selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of N fireworks to generate sparks by the mutation explosion, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework, $X^i \in \mathcal{Z}$, the total number of $|\mathcal{Z}|$ sparks are generated and are evaluated using the cost function. After the N (i.e., fireworks) function evaluations, M_e and $|\mathcal{Z}|$ are the number of function evaluations in each FWA (or EFWA) generation. Then, the total number of function evaluations for the FWA/EFWA in G generations would be [46]:

$$N + G(M_e + |\mathcal{Z}|). \tag{2.21}$$

Note that the number of function evaluations in G generations of the modified fireworks algorithms proposed in the chapter 2 to chapter 5 are same as in (2.25).

2.5. Summary of the review

In this chapter, entities and operations of evolutionary algorithms (EAs) are discussed in general. These population-based heuristic algorithms are considered intelligent tools to solve challenging optimization problems. We discuss some typical characteristics of intelligence that are adopted by the EAs according to its nature. Most of the proposed algorithms in this thesis are the modification/enhancement in the swarm intelligence-based fireworks algorithm (FWA). Therefore, FWA and enhanced FWA (EFWA) is discussed with drawbacks/shortcomings in the FWA. In this thesis, the performance of the proposed algorithms is compared against two swarm intelligence-based algorithms such as low-complexity Biogeography-based Optimization (LC-BBO) and Discrete Artificial Bee Colony (DABC). The BBO, LC-BBO, DABC and classic Genetic algorithm (GA) are also briefly discussed.

Chapter 3. Optimizing power for virtual machine placement in datacenters

3.1. Introduction

Modern datacenters are challenged to provide crucial infrastructure for ever-growing Internet applications. Large companies like Facebook, Google, Amazon, and Alibaba use datacenters for storage, Web search, computing services, and cloud services, and operate around the clock to facilitate client requirements [7], [47], [48]. A datacenter consists of hardware and virtualization technologies for servers, network protocols, and environment control.

A typical datacenter consists of computer systems, telecommunications equipment, storage, variety of software, etc. Datacenter operation requires power supplies and environmental controls such as air conditioning, fire safety, and security systems. The ampleness of a datacenter is reflected in the amount of electricity it is using. A large datacenter can use as much electricity as a small town [49].

An important component of a datacenter is a physical machine (PM or server). Like any other physical computer on which an operating system such as Windows or Linux runs, a PM in a datacenter can have two or more CPUs, each with multiple cores. Traditional datacenters use dedicated servers to run dedicated applications, and this results in poor server utilization and high operational (power) costs [7].

Virtualization technology was introduced to overcome server underutilization and waste of costly resources like power. In virtualization technology, an operating system, or software within the operating system, simulates a computer environment where virtual machines (VMs) are created. Like any other computer, one can power on a VM and load an operating system. Each VM has its own virtual hardware such as a CPU, hard disks, and network interfaces. Using virtualization technologies (e.g., VMware, Xen), multiple VMs can be located on a single PM [7].

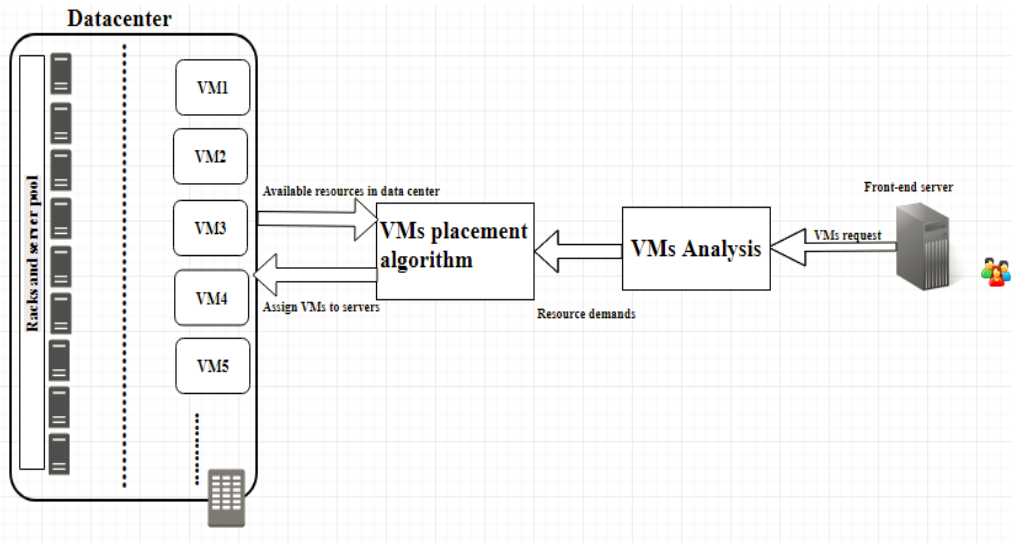


Figure 3.1 Overview of a datacenter.

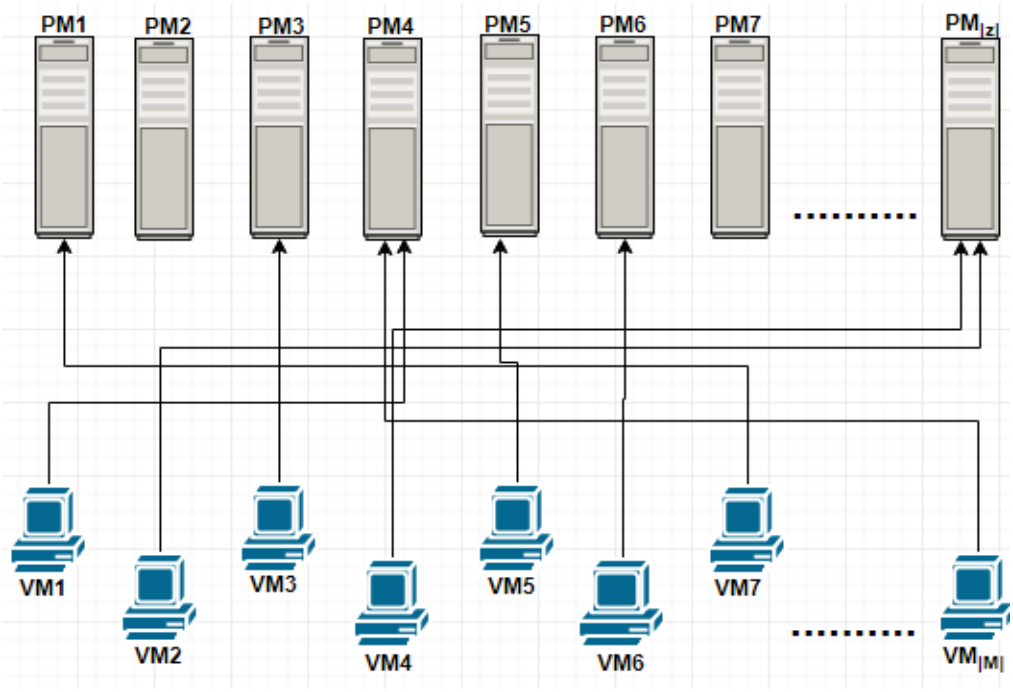


Figure 3.2 Assignment of VMs to PMs.

Figure 3.1 is an overview of a datacenter. The front-end server is an interface between a datacenter and a client (i.e., a VM). Input to the VM placement (VMP) algorithm is provided by analyzing the VM's demand, such as a CPU, memory, and bandwidth. A datacenter can provide its status of available resources, such as CPU, memory, and bandwidth to the VM's placement algorithm. Then the VMP algorithm is used to assign

VMs to PMs in a datacenter. VMP in a datacenter is a computationally challenging problem with a variety of objectives such as optimal power consumption, routing, and latency.

The goal of virtual technology is to assign VMs to PMs, as shown in Figure 3.2, in such a way that the total power consumed in a datacenter is minimized. The major contributing factors to power waste in a datacenter are: power dissipation in PMs, cooling systems, and inefficient allocation of computing resources (e.g., hardware, CPUs, memory) to clients. Therefore, power efficient hardware and efficient resource allocation algorithms are chosen to mitigate the overall power consumption in a datacenter.

3.2. Related work

To fulfill the demands of ever-growing Internet applications at minimal power is a computationally challenging optimization problem similar to some classic optimization problems like bin packing and knapsack problems [8], [50], [51]. These multi-objective or single objective optimization problems are solved using a variety of algorithms. Virtual machine placement problem (VMP) can be formulated with diverse objectives like type-aware [52] data latency optimization [53], load balance maximization, resource utilization [54], bandwidth guarantee [55], and power consumption [56]–[67] [68]–[73].

In [56], a VMP was formulated for a cloud datacenter with the objectives of minimizing energy consumption and maximizing load balance. An improved energy-efficient knee point-driven evolutionary algorithm (EEKnEA) was proposed to solve this problem. Experimental results showed that the EEKnEA outperforms other classic algorithms in terms of energy consumption and load balance. A power-aware and performance-guaranteed bi-objective VMP is formulated in [51]. The goal was to minimize power consumption in PMs and guaranteed VM performance using the ant colony optimization (ACO) algorithm. Other VMP goals were to maximize resource utilization and reduce the number of operating PMs. To maximize resource utilization and minimize the number of active PMs in VMP, the authors in [60] assigned ranks to VMs and place VMs in PMs based on these ranks. In [61], to minimize resource usage and power consumption in a datacenter, multilevel joint VMP and migration (MJPM) algorithms

based on a relaxed convex optimization framework were used for an approximate solution. In [62], the discrete firefly algorithm was used to optimize energy consumption and resource usage in VMP.

In [63], a topology-aware algorithm was presented to place groups of communicating VMs in a datacenter. The goal was to use small regions of a datacenter and consolidate network flows produced by the VMs. Idle servers and network switches were switched off during datacenter operation to minimize energy consumption. In [64], an energy-aware VMP (EVP) was formulated to schedule VMs that can reduce power consumption with lower time complexity. To minimize the number of active PMs and thus to save energy, the authors in [65] proposed an energy efficient statistical live VM placement scheme. The proposed VM placement scheme incorporated dynamic migration and considered factors that cause energy consumption. In [67], VMs were assigned to the most suitable PMs in a datacenter to optimize performance, resource utilization, and energy consumption without compromising the level of service. In [66], a modified intelligent water drops (MIWD) algorithm was presented to minimize the total energy consumption in a cloud computing environment.

In [67], a holistic VMP was proposed with conflicting performance metrics such as the energetic footprint, hardware or software outages, and security policies. Due to the nonexistence of a trivial VMP strategy, a predictive control model was proposed to devise optimal maps between VMs and PMs. In [58], a power-aware dynamic resource allocator was proposed for a datacenter. Each VM demanded four resources: (1) a CPU, (2) RAM, (3) a disk, and (4) bandwidth. The VMs were assigned PMs in such a way that the power consumption of active network devices was reduced. Ten different resource allocation strategies were introduced and were compared against heuristics like first fit, best fit, worst fit, joint/disjoint selection of IT, and network resources. Experimental results showed that joint approaches outperform disjointed ones. In [59], a profile-based VM placement approach was proposed to improve energy efficiency in datacenters. First, a profile-based optimization problem was formulated with the objective of minimizing energy consumption. Second, the problem was decomposed into multiple smaller problems, or intervals. In each interval, several VMs and PMs were sorted in terms of resource

requirements and energy efficiency, respectively. A heuristic first fit-decreasing (FFD) algorithm was used to place the sorted VMs to the sorted PMs. Experimental results showed that the second approach can reduce more energy consumption than the original FFD algorithm.

Due to their combinatorial nature, evolutionary algorithms (EAs) were often used to solve VMP using moderate computing resources. In [68], a server consolidation scheme was proposed in which all VMs were assigned to PMs in such a way that the maximum number of unused PMs were turned off to save energy. A genetic algorithm (GA) was used to find an optimal or near-optimal solution to server consolidation. The authors also proposed a decrease-and-conquer genetic algorithm (DCGA) to decrease the problem size and to decrease the number of VM migrations without significantly compromising the quality of solutions. The DCGA was compared against the classic GA and the FFD algorithm. Other EAs such as ant colony optimization (ACO) and its variants were proposed for VMP in [57], [69], [71]. A relatively new EA—a glowworm swarm optimization (GSO) algorithm—was used to solve VMP [70]. Hybrid EAs—GA/ACO and GA/simulated annealing algorithms—were also proposed to solve VMP [72], [73]. A simple heuristic FFD was used to solve VMP and was also used as a benchmark for the newly proposed algorithms [8], [59], [68], [72].

We propose swarm intelligence based EAs to solve Virtual Machine Placement and experimentally compare the performance of the newly proposed EAs with the performance of some classic EAs and the FFD algorithm.

3.3. Problem formulation

We present a formulation for virtual machine (VM) placement. The objective of VM placement is to minimize the overall power consumption in a datacenter. VM placement formulation and the power formulas presented are taken from [8]. Table 3-1 presents definitions for symbols used in this chapter.

Table 3.1 Notations used in chapter 3

Symbol	Definition
Z	set of virtual machines (VMs)
M	set of physical machines (PMs)
v_i	denotes a VM, where $i = 1,2, \dots, \lvert Z \rvert$
p_j	denotes a PM, where $j = 1,2, \dots, \lvert M \rvert$
u_j	percentage of CPU utilization of p_j
e_j	power consumption of p_j
e_{max}^j	maximum power consumption of p_j when $u_j = 100\%$
e_{idle}^j	power consumption of p_j in idle status
v_{cpu}^i	CPU demand of v_i
v_{mem}^i	memory (RAM) demand of v_i
v_{net}^i	network bandwidth demand of v_i
p_{cpu}^j	CPU capacity of p_j
p_{mem}^j	memory (RAM) capacity of p_j
p_{net}^j	network bandwidth capacity of p_j

The assignment of a virtual machine (VM) v_i to a physical machine (PM) p_j is indicated by the decision variable x_{ij}. If the i^{th} VM is assigned to the j^{th} PM, x_{ij} = 1 otherwise, x_{ij} = 0; that is,

$$x_{ij} = \begin{cases} 1, & \text{if VM } v_i \text{ is assigned to PM } p_j \\ 0, & \text{otherwise} \end{cases} \qquad 1 \le i \le |Z|, 1 \le j \le |M|. \tag{3.1}$$

VM assignments to PMs can be written in matrix notation *VP*, as follows:

$$VP = \begin{pmatrix} x_{11} & \cdots & x_{1|\mathrm{M}|} \\ \vdots & \ddots & \vdots \\ x_{|\mathrm{Z}|1} & \cdots & x_{|\mathrm{Z}||\mathrm{M}|} \end{pmatrix}. \tag{3.2}$$

where in matrix *VP*, j^{th} column VP_j represents connection(s) of VMs with j^{th} PM and i^{th} row represents a VM. The decision variable x_{ij}=1 if the i^{th} VM is assigned to the j^{th} PM, otherwise, x_{ij}=0.

For a given assignment VP, the CPU utilization of PM p_j can be calculated as follows:

$$u_j = \frac{\sum_{i \in Z} v_{cpu}^i \times x_{ij}}{p_{cpu}^j}. \tag{3.3}$$

The total power consumption e_j of a PM (i.e., p_j) includes the overhead power e_{idle}^j and the operational power. When the power of a PM is turned on, it consumes power even if it does not serve a virtual machine (VM); this is called overhead power e_{idle}^j. Total power of the j^{th} PM can be computed as follows:

$$e_j(VP_j) = e_j\left(\begin{pmatrix} x_{1j} \\ \vdots \\ x_{|Z|j} \end{pmatrix}\right) = \begin{cases} 0, & \text{if } VP_j \text{ is null vector} \\ (e_{max}^j - e_{idle}^j) \times u_j + e_{idle}^j, & \text{otherwise} \end{cases} \tag{3.4}$$

The maximum power e_{max}^j of a PM (p_j) is computed as follows [8]:

$$e_{max}^j = \left(1 - log\left(\frac{p_{cpu}^j}{1000}\right) \times 0.4\right) \times E \times \left(\frac{p_{cpu}^j}{1000}\right). \tag{3.5}$$

Note that *E* is a constant in (3.5) and is set to 100 (Watt). *E* represents the base power that is consumed by the smallest PM in terms of CPU, when $p_{cpu}^j = 1000$. Also, note that in our experiment, every PM has CPU capacity p_{cpu}^j greater than 1000. When no VM is assigned to a PM (i. e. , $\sum_{i \in Z} x_{ij} = 0$), this formulation assumes that the PM can be turned off and consumes no power. The objective of this problem is the assignment of VMs to PMs that minimizes the power consumption in a datacenter. The cost function and constraints for VM placement are as follows:

$$\min_{x_{ij} \in \{0,1\}, \forall i \in Z, j \in M} \quad \sum_{j \in M} e_j(VP_j) \tag{3.6}$$

subject to:

$$\sum_{j\in M} x_{ij} = 1, \forall i \in Z \text{ where } x_{ij} \in \{0,1\}, \quad (3.7)$$

$$\sum_{i\in Z} v_{cpu}^{i} \times x_{ij} \leq p_{cpu}^{j}, \forall j \in M, \quad (3.8)$$

$$\sum_{i\in Z} v_{mem}^{i} \times x_{ij} \leq p_{mem}^{j}, \forall j \in M, \quad (3.9)$$

$$\sum_{i\in Z} v_{net}^{i} \times x_{ij} \leq p_{net}^{j}, \forall j \in M. \quad (3.10)$$

where constraint (3.7) ensures that each VM is assigned to only one PM, constraints (3.8)–(3.10) ensure that the sum of the total CPU, the memory, and the network bandwidth demand of VMs assigned to a PM must not exceed the total CPU, memory, and bandwidth capacity of that PM. This study implicitly assumes that the overall resources of the datacenter can accommodate all VM assignments. In our computational experiments, we generate problems in such a way that the total resource capacity of PMs exceeds the total resource demand of VMs.

The total number of distinct VM to PM assignments is $|M|^{|Z|}$, and the number of placements increases with an increase in PMs or VMs that increase the search space exponentially. Therefore, finding an exact solution for VM placement (3.6) – (3.10) is impractical using the exhaustive search due to high computing demand. A practical approach is to use an approximate algorithm such as an evolutionary algorithm (EA) that can provide a solution of good quality using reasonable computing resources. In the next section, we present VM placement with the proposed evolutionary algorithms (EAs).

3.4. Problem Reformulation

3.4.1. Redefining the decision variables

The motivation is to implement the enhanced fireworks algorithm (EFWA) to the VM placement, which is developed in the field of swarm intelligence in 2013 [23]. The

limitation of the EFWA is that it is designed for continuous space optimization problems and its operators cannot operate on the nonbinary decision variables. In (3.1)–(3.2), VM placement is represented by the binary decision variables x_{ij}. The decision variable x_{ij}=1 if the i^{th} VM is assigned to the j^{th} PM, otherwise, x_{ij} = 0. These limitations of the EFWA prevent it from being implemented in the VM placement.

This thesis represents VM placement as a nonbinary integer space problem to reduce the constraint checks. An additional advantage of representing VM placement as a nonbinary integer is that constraint (3.7) is not required to be explicitly enforced when implementing the proposed algorithms. Constraint (3.7) states that a VM can be assigned to one PM only. The VM assignment x_{ij} in (3.1) can be represented as a vector of nonbinary integers (also known as a candidate solution):

$$X = \left(X_1, X_2, X_3, \dots, X_{|Z|}\right), \tag{3.11}$$

where each component of the vector *X* refers to a physical machine (PM) and indices of the vector represent an assigned VM number. Suppose we have ten VMs and three PMs. The vector $X = (1, 2, 1, 3, 2, 1, 3, 2, 1, 2)$ indicates that VMs $v_1,\ v_3, v_6, v_9$ are placed in PM p_1, VMs v_2, v_5, v_8, v_{10} are placed in PM p_2, and VMs $v_4,\ v_7$ are placed in PM p_3.

3.4.2. Reformulating the VM placement

In the terminology of evolutionary algorithms (EAs), an "individual" refers to a candidate solution to the optimization problem. Vector *X* is a candidate configuration of VM placement that represents VM assignments to PMs. Making use of constraint (3.7) to reduce the constraint checks, we use a decision vector of nonbinary integers $X = (X_1, X_2, X_3, \dots, X_{|Z|})$, where |Z| is the cardinality of set Z. In *X*, integer variable X_i, where $i = 1,2, \dots, |Z|$, takes values in the set $\{1, 2, \dots, |M|\}$, where elements 1, 2… |M| represent physical machines (PMs). Note that in *X*, the integer variable X_i represents the i^{th} VM assigned to PM *j*. Similarly, if no component of *X* (X_i, where $i = 1,2, \dots, |Z|$) takes a value in the set $\{1, 2, \dots, |M|\}$, then this candidate configuration assumes that PM *j* can be turned off and consumes no power. After representing VM placements as the nonbinary

integer vector X, we use $X = (X_1, X_2, X_3, \dots, X_{|Z|})$ as a decision variable for the VM placement.

The CPU utilization of a p_j can be calculated as:

$$u_j = \frac{\sum_{\{i:1 \le i \le |Z| \wedge X_i = j\}} v_{cpu}^i}{p_{cpu}^j}. \quad (3.12)$$

The power consumption and the maximum power consumption of p_j are computed by (3.4) and (3.5), respectively. However, after representing VM placement as a nonbinary integer vector X, the power consumption and the maximum power consumption of p_j can be computed by (3.5) and (3.12), respectively. The reformulated cost function and constraints for the VM placement are:

$$\min_X \quad \sum_{j \in M} e_j, \quad (3.13)$$

subject to:

$$\sum_{\{i:1 \le i \le |Z| \wedge X_i = j\}} v_{cpu}^i \le p_{cpu}^j, \forall j = 1,2,3, \dots, |M|, \quad (3.14)$$

$$\sum_{\{i:1 \le i \le |Z| \wedge X_i = j\}} v_{mem}^i \le p_{mem}^j, \forall j = 1,2,3, \dots, |M|, \quad (3.15)$$

$$\sum_{\{i:1 \le i \le |Z| \wedge X_i = j\}} v_{net}^i \le p_{net}^j, \forall j = 1,2,3, \dots, |M|. \quad (3.16)$$

The cost function (3.13) minimizes the overall power consumption in the datacenter, constraints (3.14)– (3.16) ensure that the sum of the CPU, memory, and network bandwidth demand of the VMs assigned to a PM must not exceed the total CPU, memory, and bandwidth capacity of that PM.

3.5. Proposed evolutionary algorithms

We propose three new evolutionary algorithms (EAs) to solve VM placement as formulated in (3.13) – (3.16) and represented in (3.11): a discrete fireworks algorithm (DFWA), a problem specific information-based DFWA (IDFWA), and a hybrid of the IDFWA and the low-complexity biogeography-based optimization algorithm (Hybrid IDFWA/LC-BBO). Every individual, as defined in (3.11), is a candidate solution of VM placement. The cost of a candidate solution is computed using the cost function defined in (3.13).

3.5.1. Discrete fireworks algorithm

We propose a discrete fireworks algorithm (DFWA) for VM placement. Operators in the enhanced fireworks algorithm (EFWA) are designed in [22] and [23] for continuous space optimization problems, and the operators cannot operate in discrete space problems without modifications. The proposed DFWA is a modified version of the EFWA, and the operators are modified to operate on VM placement. Like the EFWA, the DFWA has four operations: an explosion operator, a mutation operator, a repair mechanism, and a selection operator.

3.5.1.1. Explosion operator

The explosion operator in the DFWA generates sparks from a firework using offset displacement and two parameters: explosion strength and explosion amplitude.

A. Explosion strength

In the DFWA we adopt the same explosion strength formula that was used for the EFWA [22], [23]. The cost values of a firework and algorithm parameters determine the number of sparks that a firework can generate. Like the FWA/EFWA in the sections 2.2.1.1-A and 2.2.2.1-A, the DFWA computes the number of sparks s_i for the i^{th} firework:

$$s_i = \text{round}\left(M_e \times \frac{Y_{max} - f(X^i) + \varepsilon}{\sum_{i=1}^{N}\left(Y_{max} - f(X^i)\right) + \varepsilon}\right), \text{ where } i = 1,2,\dots,N, \tag{3.17}$$

where s_i is the number of sparks from the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{max} is the maximum cost of N fireworks in the current algorithm generation, $f(X^i)$ represents the cost of the i^{th} firework, M_e is a constant that controls the total number of sparks generated by N fireworks, and ε is a small constant used to avoid division by zero in (3.17).

B. Offset displacement

After computing the number of explosion sparks s_i for the i^{th} firework, where $i = 1,2,\dots,N$, the DFWA determines the offset displacements for the probabilistically selected component of the firework within the explosion amplitude. For the i^{th} firework, the DFWA

uses the random function *'rand'* to generate offset displacements to perturb the probabilistically selected components of the firework within explosion amplitude A_i, where $i = 1,2, \dots, N$; *'rand'* generates uniformly distributed random values between *0* and *1*. Similar to offset displacement in the EFWA, offset displacement in the DFWA is calculated for each probabilistically selected component of a firework with user-determined probability to ensure the diversity of sparks generated around that firework. To ensure the nonzero value as a result of perturbation, we use the *ceil* function in the DFWA to convert the '$ceil(A_i \times rand\ (0,1))$' value to an integer for nonbinary integer space problems. The *ceil* function rounds a real value to the nearest integer in the direction of positive infinity whereas the *round* function rounds a real value to the nearest integer.

$$\widetilde{X_q^i} = ceil(X_q^i + A_i \times rand(0,1)), \quad \text{for } q = 1,2, \dots, m \tag{3.18}$$

Algorithm 3.1: Generating explosion sparks in the DFWA

Inputs:

- X: a vector of *m* component. Note that X is a firework (a candidate solution).

Algorithm parameters:

- *sparkProb:* spark probability [0,1] *// user determined explosion probability*
- *A:* Explosion amplitude *(see 3.5.1.1-C)*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. ***for*** q = 1 to *m* *// m is number of components in X*
2. ***if*** $rand() < sparkProb$
3. Calculate the offset displacement: $\Delta X_q = A \times rand()$
4. $\widetilde{X_q} = ceil(X_q + \Delta X_q)$ *// perturbing the q^{th} component (see 3.5.1.1-B)*
5. ***end if***
6. ***end for***

where $\widetilde{X_q^i}$ is the spark component after adding the displacement '$A_i \times rand(0,1)$' in the X_q^i component of the i^{th} firework, for each of $i = 1,2, \dots, N$. Pseudo code of the Algorithm 3.1 is run once to generate an explosion spark.

C. Explosion amplitude

The explosion amplitude quantifies the range of the displacement that is used to perturb one or more components of a firework. In a population of *N* fireworks, a firework

with a lower cost is considered a good firework and a firework with a larger cost is considered a bad firework. In the DFWA, the amplitude formula (of the FWA/EFWA in the sections 2.2.1.1-C and 2.2.2.1-C) is modified by using '*round*' function to optimize nonbinary discrete space as:

$$A_i = \text{round}\left(\hat{a} \times \frac{f(X^i) - Y_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X^i) - Y_{min}) + \varepsilon}\right), \text{where } i = 1,2,\dots,N, \quad (3.19)$$

where A_i is the amplitude associated with the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{min} is the minimum cost among the N fireworks in the current algorithm generation, $f(X^i)$ represents the cost of the i^{th} firework, â is a constant used to control the amplitude, and ε is a small constant used to avoid division by zero in (3.19).

3.5.1.2. Mutation operator

In the VM placement as formulated in *section 3.4.3*, the fireworks and the sparks set up by those fireworks have positive integers as their components, and we use a random integer function '*randi ()*' and absolute function '*abs ()*' to ensure a spark has a positive value. The DFWA selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of N fireworks to set up sparks by the mutation explosion, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is cardinality of the set $\mathcal{Z}$. One spark is generated for each firework. The mutation explosion operator is represented in (3.20):

$$\widetilde{X_q^i} = X_q^i + (X_q^b - X_q^i) \times randi\,(X_q^{min}, X_q^{max}), \quad \text{for } q = 1,2,\dots,m \quad (3.20)$$

where $\widetilde{X_q^i}$, X_q^b, and X_q^i are the q^{th} component of a newly generated spark, component of the best solution up to the current algorithm generation and component of i^{th} firework (for each of $i = 1,2,\dots,N$) to be mutated respectively. Note that X_q^i is the probabilistically selected component of $X^i \in \mathcal{Z}$ (with corresponding component X_q^b) by the user-determined probability *mutateProb*, where $i = 1,2,\dots,N$; X_q^{min} and X_q^{max} are lower and upper bounds of the search space in dimension q. Pseudo code of the Algorithm 3.2 is run once to generate a mutation spark.

Algorithm 3.2: Generating Mutation sparks in the DFWA

Inputs:

- X: a vector of *m* components. Note that X is a firework to be mutated (*see 3.5.1.2*).
- X^b: a vector of *m* components. Note that X^b is a best solution amongst *N* fireworks.

Algorithm parameters:

- *mutateProb:* spark probability [0,1] *// user determined mutation probability.*

Output:

- $\check{X}$, a *spark, a vector of m components.*

Steps:

1. ***for*** q = 1 to *m* *// m is number of components in X*
2. ***if*** *rand ()* < *mutateProb*
3. $\check{X}_q = X_q + \left(X_q^b - X_q\right) \times randi\left(X_q^{min}, X_q^{max}\right)$
   // perturbing the q^{th} component (see 3.5.1.2)
   //note that *randi() returns integer between* X_q^{min} *and* X_q^{max}
4. ***end if***
5. ***end for***

3.5.1.3. Repair mechanism

Randomly generated fireworks, sparks, and mutation sparks (i.e., candidate solutions) may fall in the infeasible space after executing DFWA operations. The candidate solutions may become infeasible due to violation of rectangular or nonrectangular constraints. These infeasible solutions are useless for further evolution in any evolutionary algorithm (EA). Therefore, infeasible candidate solutions need to be repaired. The proposed repair algorithm for the VMP checks the feasibility of candidate solution as defined in (3.11) and repairs the infeasible one.

A. Repair algorithm

The implementation of repair algorithm with detailed pseudo code is discussed in appendix of this chapter. In this section, pseudo code and repair algorithm are concisely presented. A candidate solution for the VM placement, as defined in (3.11), either generated randomly or evolved by the experimented evolutionary algorithm (EA), may violate one or more constraints of the VM placement, and therefore becomes infeasible. The proposed repair algorithm is used to check feasibility and repaired infeasible candidate solutions.

The system parameters, as defined in the *section 3.2*, and a candidate solution *X*, *section 3.4.1*, is the input to the repair algorithm. A PM is considered overloaded in *X*, if

load (i.e., CPU, memory, and bandwidth) on a PM exceeds the capacity (i.e., CPU, memory, and bandwidth) of that PM. The load on a PM is the sum of VMs' CPU, memory and bandwidth connected to that PM. A candidate soliton *X* is considered infeasible, if one or more PMs are overloaded. In contrast, a PM is considered underloaded in *X*, if current load does not exceed the capacity of that PM.

The repair algorithm computes the load of PMs in terms of CPU, Memory and Bandwidth, and checks the feasibility of a candidate solution *X*. For an infeasible candidate solution *X*, the repair algorithm disconnects VMs one by one from the overloaded PMs. A disconnected VM from overloaded PM need to be reconnected to any of the underloaded PMs in *X*. Here, repair algorithm checks whether the disconnected VM can be legitimately reconnected to any of the underloaded PMs. In case the reconnection is feasible, the VM is assigned to that PM. In other words, the VM placement in an underloaded PM is considered legitimate if load remains less or equal to the capacity of that PM. The repair algorithm iteratively disconnects VMs from overloaded PMs until load become less or equal to its capacity and reconnects disconnected VMs to underloaded PMs.

Table 3.2 Repair algorithm for infeasible solutions

A. Inputs	**Steps:** 1. (a) System parameters such as VMs CPU, memory, and bwd demand of VMs, and PMs CPU, memory, and bwd capacity of PMs, etc. (b) Candidate solution *X*.
B. Execution	**Steps:** 2. Calculate load demand of all VMs to the corresponding PMs. 3. Overloaded information for each PM is checked. 4. ***if*** (Candidate solution *X* is infeasible) 5. VMs are disconnected one by one from the overloaded PMs until overloaded PMs become less or equal to its maximum capacity. 6. After checking feasible load on PMs, each disconnected VM is reconnected to the first available PM. 7. Calculate load demand of all VMs to the corresponding PMs. 8. Overloaded information for each PM is checked. 9. ***end if*** 10. ***while*** (*X* is infeasible) 11 Randomly generate a candidate solution *X*. 12. Repeat steps 2 to 9.

	13. ***end while*** *// A solution is repaired*
C. Output	14. ***return*** feasible solution *X*.

The pseudo code steps 2–9 in the Table 3-3 for the proposed repair algorithm does not guarantee that the repairable (or infeasible) solution will become feasible after going through the repair mechanism. The reason is that the proposed repair algorithm is not checking each VM to PM feasible connections exhaustively. In other words, the repair algorithm only checks for the first available feasible connection from a VM to a PM to replace an infeasible connection. If a candidate solution is not repairable (or no feasible VM to PM connection is available), the proposed repair algorithm randomly generates a new candidate solution *X* in steps 10–13 and checks its feasibility using steps 2–9 of pseudo code in the Table 3-2.

3.5.1.4. Selection operation

Each generation of the DFWA produces a number of candidate solutions greater than the *N* fireworks population. After applying all the DFWA operators, a new population of *N* fireworks is selected from the current candidate solutions. The DFWA algorithm adopts the same elitism-random selection operation laid down in the enhanced fireworks algorithm (EFWA) [22], [23]. In the DFWA, the solution with the minimum cost value is selected, then $(N-1)$ candidate solutions are randomly selected from the remaining candidate solutions for the next algorithm generation.

3.5.1.5. DFWA algorithm operation

The pseudo code for the DFWA algorithm is presented in Table 3-3. Initially, a population *F* of the *N* fireworks is randomly generated, and algorithm parameters are initialized. After computing the cost of the *N* fireworks using (3.13) – (3.16), the sparks s_i and amplitude values A_i are computed using (3.17) and (3.19) for each of the $i = 1,2,\dots,N$ fireworks. For each spark, the offset displacement, (3.18), is added probabilistically to the selected components of the firework X^i, for each of $i = 1,2,\dots,N$, with the user determined '*sparkProb*' probability. In the context of the fireworks algorithm, each firework is perturbed probabilistically by adding a displacement to generate sparks around that firework. This perturbation process exploits the existing small region (around a firework)

and a thorough search is conducted in a small region by generating sparks. All the sparks generated from the *N* fireworks are evaluated using the cost function (3.13).

The DFWA selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of *N* fireworks to execute the exploration process. For each firework $X^i \in \mathcal{Z}$, the mutation operator (3.20) is used to generate mutation sparks with a user-determined *mutateProb* probability. After executing the exploration process on the $X^i \in \mathcal{Z}$ fireworks, the mutation sparks are evaluated.

Table 3.3 DFWA pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$. 2. Initialize the *sparkProb* and *mutateProb*. 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table 3-2 and evaluate using the cost function in (3.13). 5. ***while*** (stopping criteria not satisfied) 6. ***for*** $i = 1,2, \dots, N$ 7. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (3.17) and (3.19), respectively. 8. ***for*** $j = 1, 2, \dots, s_i$ 9. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 3.1**. 10. Add generated spark in the set *S*. 11. ***end for*** 12. ***end for*** 13. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 3.4.1.2*) from a population of *N* fireworks. 14. ***for*** each firework X in $\mathcal{Z}$ 15. Generate mutation spark $\breve{X}$ using **Algorithm 3.2**. 16. Add generated spark in the set *S*. 17. ***end for*** 18. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table 3-2 and evaluate using the cost function in (3.13). 19. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 20. ***end while***
C. Output	21. ***return*** the best solution found so far.

After performing the explosion operation and mutation operation for one algorithm generation, like the EFWA, the DFWA selects a new population of the *N* fireworks. In the

DFWA, first the solution with the minimum cost is selected for the next algorithm generation, then (*N-1*) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation [22], [23].

3.5.2. Problem specific information-based DFWA

Generally, evolutionary algorithms (EAs) are model-free and do not need any problem specific information or domain-knowledge [74] during their operations. However, incorporating problem specific information in EAs can improve their overall efficiency. In this subsection, we introduce a new algorithm that utilizes domain-knowledge for virtual machine placement.

3.5.2.1. Domain-knowledge for VM placement

Some type of domain knowledge can be extracted from computationally challenging problems that can be used in evolutionary algorithms (EAs) to optimize their solutions. However, there is no guarantee that useful information is accessible or that the information can be used in the EA to solve an optimization problem. Some domain knowledge in the VM placement problem is easily accessed. According to definition (3.4), any physical machine (PM) that is turned on spends overhead power e_{idle}^{j}. After being turned on, a PM consumes 100 percent power (i.e., e_{max}^{j}) if all its resources are utilized (by VMs). On the other hand, a PM after turning its power on consumes 70 percent [8] overhead power e_{idle}^{j} if none of its resources are utilized. In other words, overhead power is consumed in a PM after turning power on if no VM is connected to that PM. The objective of VM placement in (3.13), is to minimize the power consumption in datacenters. The overhead power e_{idle}^{j} required by a PM that is turned on can be better utilized if that PM's resource utilization is high. Thus, to minimize the overall power consumption in a datacenter, efficient VM assignment to PMs, plausibly, will tend to allow many PMs to be turned off, while satisfying the demands of VMs. The proposed IDFWA tries to incorporate such domain knowledge in assigning VMs to PMs.

3.5.2.2. Obtaining domain-knowledge for VM placement

Each component of the integer vector X in (3.11) represents a VM placement, and the value of the vector component specifies the physical machine (PM) serving the VM corresponding to the component. Useful information can be collected from the integer vector X by counting the number of VMs served by each PM. A PM is likely to be efficiently utilized if it serves many VMs, subject to fulfilling constraints (3.13) – (3.16). We can collect information from X and apply domain knowledge to guide the exploitation of the DFWA. According to the definition of X in (3.11), components values with high frequency (PMs serving many VMs) are considered good components and components values with low frequency (PMs serving comparatively fewer VMs) are considered bad components.

Algorithm 3.3: Generating explosion sparks in the IDFWA

Inputs:

- X: a firework (a candidate solution, m dimensional vector). Note that q^{th} component X_q represents a PM (in set M), where index q is representing a VM.

Algorithm parameters:

- *sparkProb:* spark probability [0,1] *// user determined explosion probability*
- *A:* Explosion amplitude *(see 3.5.1.1-C)*
- Δ*:* user-defined fraction *// to choose portion of the m components in X.*

Output:

- $\check{X}$ *,* a *spark, a vector of m components*

Steps:

1.From X, select a set T of *round* ($\Delta|M|$) components (*see 3.4.2*).
// a set, T, of $\Delta|M|$ PMs that serve the smallest number of VMs
2. ***for*** each component q in T
3. ***if*** *rand () < sparkProb*
4. Calculate the offset displacement: $\Delta X_q = A \times rand()$
5. $\widehat{X_q} = ceil(X_q + \Delta X_q)$ *// perturbation of q^{th} component (see 3.5.1.1-B)*
6. ***end if***
7. ***end for***

3.5.2.3. Incorporating domain knowledge in the DFWA

In the IDFWA, in choosing the components of a firework for displacement operation in (3.18), we try to avoid having much overhead power ($i.e., e^{j}_{idle}$) rather than choosing/updating VMs to PMs (i.e., components) randomly. To that end, we choose some number of PMs that serve many VMs and then perturb the placement of the VMs currently

assigned to those PMs. More specifically, the IDFWA uses some user-defined fraction, Δ, to determine the number of such PMs to be chosen and choose a set, *T*, of $\Delta|M|$ PMs that serve the smallest number of VMs currently. Then, we perturb those VMs that are currently assigned to the PMs in the set *T*. Pseudo code of the Algorithm 3.3 is run once to generate an explosion spark.

A. Example of using domain knowledge in VM placement

Suppose we have the i^{th} firework $X^i = (1, 2, 2, 3, 2, 1, 3, 2, 1, 2)$, for each of $i = 1,2, \ldots, N$, and Δ= 2/3. In the i^{th} firework, three PMs (p_1, p_2 and p_3) serve ten VMs. In accordance with the firework X^i, p_1 serves three VMs, p_2 serves five VMs, and p_3 serves two VMs. The number of VMs served by p_1, p_2, and p_3 are 3, 5, and 2, respectively. In this example, we have 3×(2/3) = 2, so two PMs are considered that are currently serving the smallest number of VMs. The two PMs are p_1 and p_3 in this example. The set of VMs served by p_1 and p_3 is *T*= $\{v_1, v_4, v_6, v_7, v_9\}$. Now, the offset displacements are added in those components of *X* that are associated with set of VMs in *T* with the user-determined probability *sparkProb* to construct a new spark. Except for the incorporation of domain knowledge in the DFWA algorithm, the IDFWA algorithm operation is the same as that of the DFWA algorithm [20].

3.5.2.4. IDFWA algorithm operation

The pseudo code for the IDFWA is presented in Table 3-4. Initially, a population *F* of *N* fireworks is generated randomly, and algorithm parameters are initialized. After computing the objective function values of *F* fireworks using (3.13) – (3.16), the sparks s_i and the amplitudes A_i are computed using (3.17) and (3.19), respectively, for each firework. Now, s_i sparks are generated for each of the *N* firework. The offset displacement (3.18) is added to the set of *T* components of fireworks X^i with user-determined *sparkProb* probability. The set of *T* components of a firework is determined by using the domain-knowledge of VM placement. This process is a local search and is also called "exploitation." In the context of the fireworks algorithm, each firework is perturbed probabilistically by adding an offset displacement within amplitude A_i to generate sparks around that firework. This controlled perturbation (by selecting *T* components) exploits a

small region around a firework, and a thorough search is conducted over this small region by generating sparks. All the sparks generated from the *N* fireworks are evaluated using the cost function (3.13).

Now, we select a set $\mathcal{Z}$ of fireworks randomly to be mutated from the population of *N* fireworks to execute the exploration, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework $X^i \in \mathcal{Z}$, the mutation operator (3.20) is used to generate one mutation spark with user-determined *mutateProb* probability. After executing the exploration process on the set of $\mathcal{Z}$ fireworks, the mutation sparks are evaluated using the cost function (3.13).

In one IDFWA generation, the total number *h* of candidate solutions that includes fireworks, explosion sparks, and mutation sparks, where $h > N$. For the next algorithm generation, we need to select a population of *N* fireworks from number *h* of candidate solutions. In the IDFWA, first the solution with the minimum cost is selected, then (*N-1*) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation.

Table 3.4 IDFWA pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$. 2. Initialize the *sparkProb*, *mutateProb*, and Δ (user-defined fraction). 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table 3-2 and evaluate using the cost function in (3.13). 5. ***while*** (stopping criteria not satisfied) 6. ***for*** $i = 1,2, \dots, N$ 7. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (3.17) and (3.19), respectively. 8. ***for*** $j = 1, 2, \dots, s_i$ 9. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 3.3**. 10. Add generated spark in *S*. 11. ***end for*** 12. ***end for*** 13. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 3.4.1.2*) from a population of *N* fireworks.

	14. ***for*** each firework X in $\mathcal{Z}$ 15. Generate mutation spark $\check{X}$ using **Algorithm 3.2**. 16. Add generated spark in *S*. 17. ***end for*** 18. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table 3-2 and evaluate using the cost function in (3.13). 19. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 20. ***end while***
C. Output	21. ***return*** the best solution found so far.

3.5.3. Hybrid IDFWA/LC-BBO algorithm

IDFWA is presented in the *section 3.4.2* and the low-complexity BBO (LC-BBO) algorithm is discussed in the *section 2.2.3.1* of chapter 2. In next section, hybrid of LC-BBO algorithm and the IDFWA (Hybrid IDFWA/LC-BBO) for the VM placement problem is presented.

3.5.3.1. Hybrid IDFWA/LC-BBO algorithm operation

The pseudo code for the Hybrid IDFWA/BBO algorithm is presented in Table 3-5. Initially, a population of *N* fireworks is generated randomly, and algorithm parameters are initialized. After computing the cost function, for *N* fireworks using (3.13) – (3.16), values for sparks s_i and amplitudes A_i are computed using (3.17) and (3.19), respectively, for each i^{th} firework, where $i = 1,2, \dots, N$. In the Hybrid IDFWA/LC-BBO algorithm, either the migration procedure of the LC-BBO algorithm or the explosion procedure of the IDFWA is selected with user-determined probability θ to generate spark(s) for each firework. If the LC-BBO algorithm migration procedure [74] is selected as an exploitation process, emigrating solution $\widetilde{X^j}$ is selected from the population of *N* fireworks. The possibility of immigrating a feature from $\widetilde{X^j}$ to X^i is decided using probability λ. Alternately, if the explosion procedure of the IDFWA is selected as an exploitation process with user-determined probability θ, s_i sparks are generated for the firework. In the IDFWA, the offset displacement (3.18) is added to the set of T components of fireworks X^i with user-determined *sparkProb* probability. The set of *T* components of a firework is determined by using the domain knowledge in VM placement. In the Hybrid IDFWA/LC-BBO algorithm,

migration and explosion are exploitation processes. In a generation of the IDFWA/LC-BBO algorithm, total number of candidate solutions includes fireworks, explosion sparks, islands/habitats, and mutation sparks from the IDFWA. All the sparks/islands generated from the *N* fireworks are evaluated using the cost function (3.13).

Table 3.5 Hybrid IDFWA/LC-BBO algorithm pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$. 2. Initialize the *sparkProb*, *mutateProb*, Δ (user-defined fraction), I (user-determined immigration rate), and θ (user-determined probability). 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table 3.2 and evaluate using the cost function in (3.13). 5. ***while*** (stopping criteria is not satisfied) 6. ***for*** $i = 1,2, \dots, N$ 7. ***if*** *rand ()* < θ 8. Use LC-BBO algorithm in **Algorithm 2.5.** *// Chapter 2, section 2.2.3.1* 9. Add generated islands in the set *S.* 10. ***else*** 11. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (3.17) and (3.19), respectively. 12. ***for*** *j* = 1 to s_i 13. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 3.3**. 14. Add generated sparks in *S* 15. ***end for*** 16. ***end if*** 17. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 3.5.1.2*) from a population of *N* fireworks. 18. ***for*** each firework X in $\mathcal{Z}$ 19. Generate mutation spark $\check{X}$ using **Algorithm 3.2**. 20. Add generated spark in *S.* 21. ***end for*** 22. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table 3.2 and evaluate using the cost function in (3.13). 23. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 24. ***end while***
C. Output	25. ***return*** the best solution found so far.

Now, we select a set $\mathcal{Z}$ of fireworks to be mutated from the population of N fireworks to execute the exploration process, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework $X^i \in \mathcal{Z}$, the mutation operator (3.20) is used to generate mutation sparks using a user determined *mutateProb* probability. After executing the exploration process on the $|\mathcal{Z}|$ fireworks, the mutation sparks are evaluated using the cost function (3.13).

After one algorithm generation, the IDFWA/LC-BBO algorithm selects a new population of *N* fireworks. Like the EFWA, in the IDFWA/LC-BBO, first the solution with the minimum cost is selected, then (*N-1*) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation.

3.6. Results and discussion

In computational experiment to assign virtual machines (VMs) to physical machines (PMs), the set of VMs Z was 20, 50, 100, and 200. We randomly generate different test problems with different computing resources for the different types of VMs (e.g., v^i_{cpu}, v^i_{mem}, v^i_{net}) that are assigned to PMs (e.g., p^j_{cpu}, p^j_{mem}, p^j_{net}). In VMs placement experimentation, the capacity of PMs p^j_{cpu} and demand of the VMs v^i_{cpu} are randomly generated between [1000−3000] and [1−2000], respectively. For various VMs placement problem instances, we scaled up the computational capacity of PMs $round\left(\frac{VMs}{PMs}\right) \times p^j_{cpu}$, to ensure that there are enough computing resources available for the VMs with the change in problem size. The memory and bandwidth demand of VMs and the capacity of PMs are randomly generated in the same way as those of the CPU [8]. For each PM, the idle status power is set to 70% of the maximum power as follows [8]:

$$e^j_{idle} = e^j_{max} * 0.7. \tag{3.21}$$

3.6.1. VM Placement performance

We compared the VM placement performance of the low-complexity BBO (LC-BBO) algorithm, the DFWA, the IDFWA, the Hybrid IDFWA/LC-BBO algorithm, the Discrete ABC (DABC) algorithm (in chapter 2), and the GA (in chapter 2) with the first fit decreasing (FFD) algorithm; that is, we used the FFD algorithm as a benchmark for the algorithms listed above for VM placement as defined in (3.13)–(3.16). In the FFD algorithm, we sort the VMs in decreasing order of their CPU demand and assign the VMs one by one to the PMs (in the given order). The number of objective function evaluations is the stopping criteria for the experimented algorithms as mentioned in the 3rd column of the Tables 3.7 and 3.8.

Parameters for the experimental algorithms are listed in Table 3-6. We divided our experiments into four groups based on the number of VMs and PMs to be linked. In each group, the number of PMs ranges from a relatively small number to the number of VMs. In total, 20 VM placement problem instances (i.e., five instances for each group) are tested using various proposed algorithms. The results presented in Table 3-7 and Table 3-8 represent the average of 100 independent trails to measure the VM placement performance of each algorithm.

Table 3.6 Parameters for the experimental algorithms

Algorithms	Algorithm specific parameters	Common parameters
Discrete ABC	t = 1.2×Population size	Population size: 30
Low-complexity BBO	λ is defined as in [44] Emigrating method is taken from [44] Probability of mutation = 0.01	
GA	Probability of crossover = 0.9 Probability of selection = 0.5 Probability of mutation = 0.01	
Hybrid IDFWA/LC-BBO	λ is defined in chapter 2 mutationProb = sparkProb = 0.5 Migration probability θ = 0.5 Emigrating method is in Table 3-5 Least frequent PMs indices Δ = 1/2	# of Fireworks:10 # of mutation Fireworks: 5
IDFWA and DFWA	mutationProb = sparkProb = 0.5 Least frequent PMs indices Δ = 1/2	

We used four metrics to record the results of experiments in this chapter: "average power consumed," "standard deviation (Std.)," "percentage of power saved," and "average CPU time" (sec.). The "percentage of power saved" in VMs placement is computed using a proposed algorithm against the FFD algorithm (as a benchmark). We computed the percentage of power saved against the FFD algorithm for each of the other algorithms using the formula:

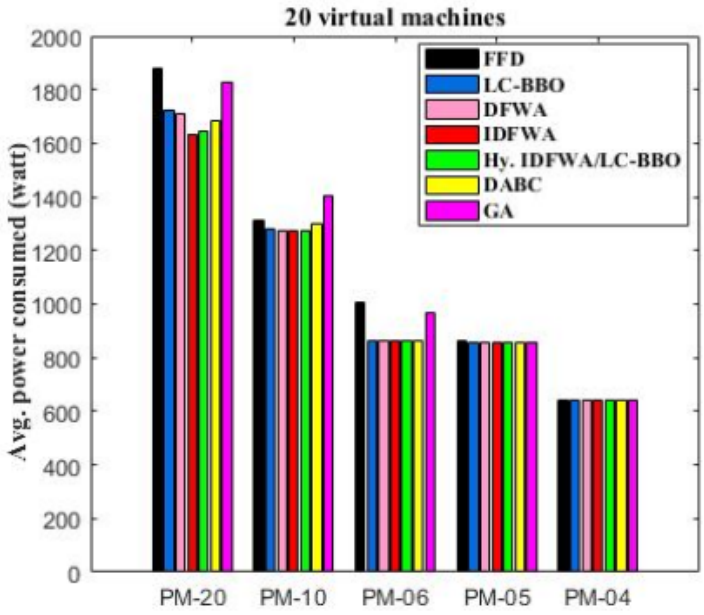
$$\left(1 - \frac{Avrage\ (power\ consumed\ by\ the\ algorithm)}{Power\ consumed\ by\ the\ FFD\ algorithm}\right) \times 100. \tag{3.22}$$

In Figures 3.3 and 3.4, the "percentage of power saved" in VM placements is shown for five problem instances. The "average power consumed" for all the VM placements is shown for the first fit decreasing (FFD) algorithm, the low-complexity BBO (LC-BBO) algorithm, the discrete fireworks algorithm (DFWA), the problem specific information-based DFWA (IDFWA), the Hybrid IDFWA/LC-BBO algorithm, the Discrete ABC (DABC) algorithm, and the GA.

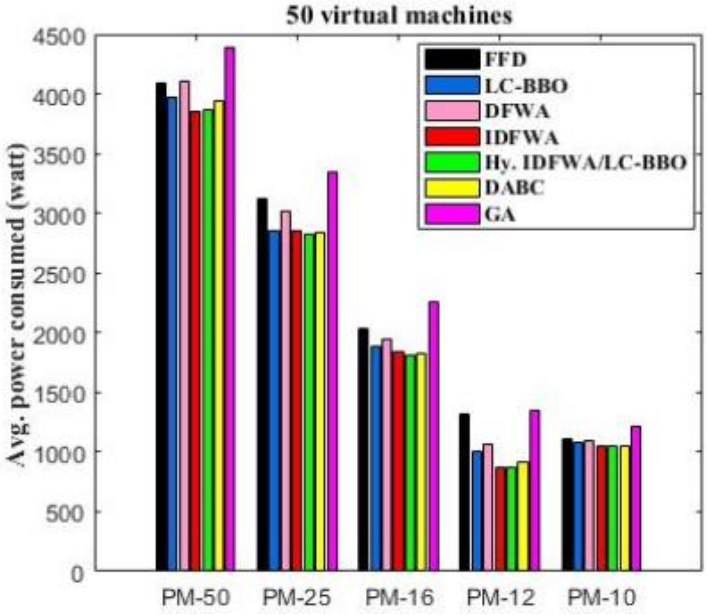
Figures 3.3 and 3.4 show that there is a general trend toward increasing power consumption when the number of PMs becomes larger. This trend is similar for all the tests. Power consumption in all VM placements using various algorithms are similar to each other and consistent. In most of the VM placements, the Hybrid IDFWA/LC-BBO algorithm had the best performance, followed by the IDFWA, the LC-BBO algorithm, the DFWA, the DABC algorithm, the GA, and the FFD algorithm in terms of average power consumption. The FFD algorithm, the GA, and the DABC algorithm performed poorly compared to the Hybrid IDFWA/LC-BBO algorithm, the IDFWA, and the LC-BBO algorithm, especially when the number of VM placements become large (i.e. ≥ 50) (Figure 2.4, a-b). Thus, our proposed algorithms improved the performance in terms of the power consumption, especially when VM placements problem size increased. Our work also shows that the proposed algorithms are scalable and applicable to real-world VM placement problems.

We compared all the algorithms with the FFD which was used as the benchmark in terms of power consumption. We depict using metric the "percentage of power saved" with

respect to the FFD algorithm in Figures 3.5 and 3.6 and Tables 3-7 and 3-8. Power savings of the IDFWA and the Hybrid IDFWA/LC-BBO algorithm were approximately 1% as in (20, 4) to 53% as in (200, 40). For most VM placements, we obtained more than 10% of power savings when applying the IDFWA and the Hybrid IDFWA/LC-BBO algorithm to a large number ($\geq$ 100) of VMs for all PM variations. The IDFWA and the Hybrid IDFWA/LC-BBO algorithm had comparable power saving performance and generally outperformed the other algorithms in this respect.

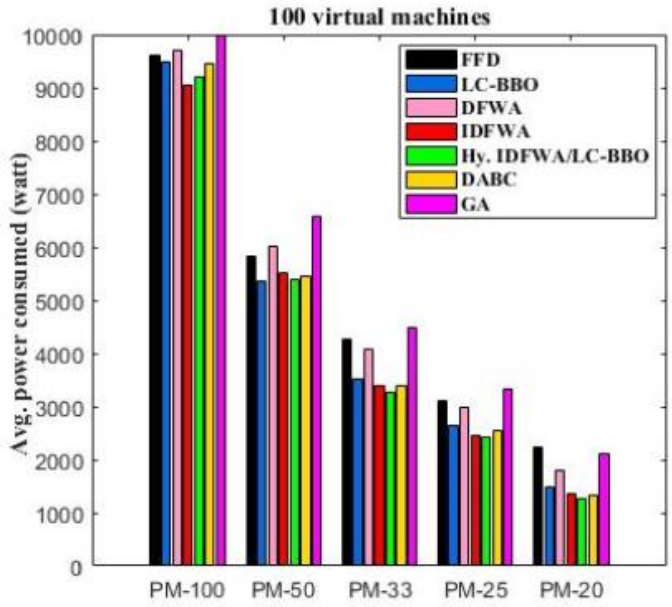


(a)

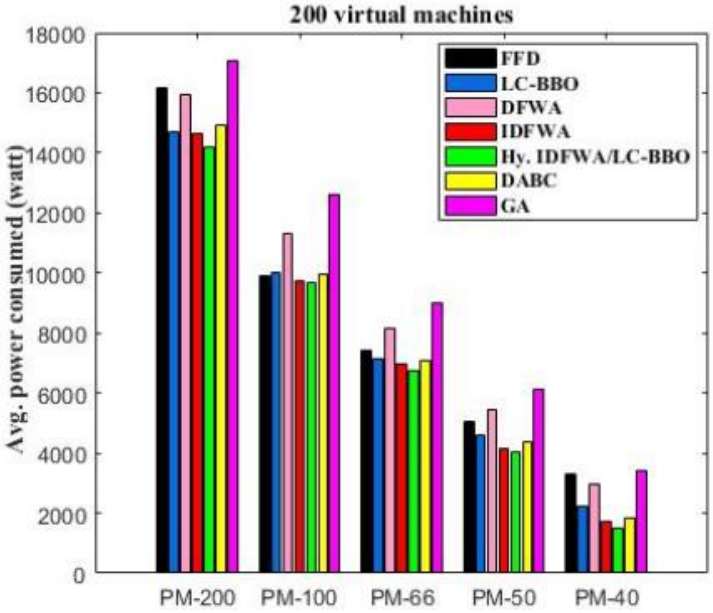


(b)

Figure 3.3 Average power consumed for 20 and 50 VMs.

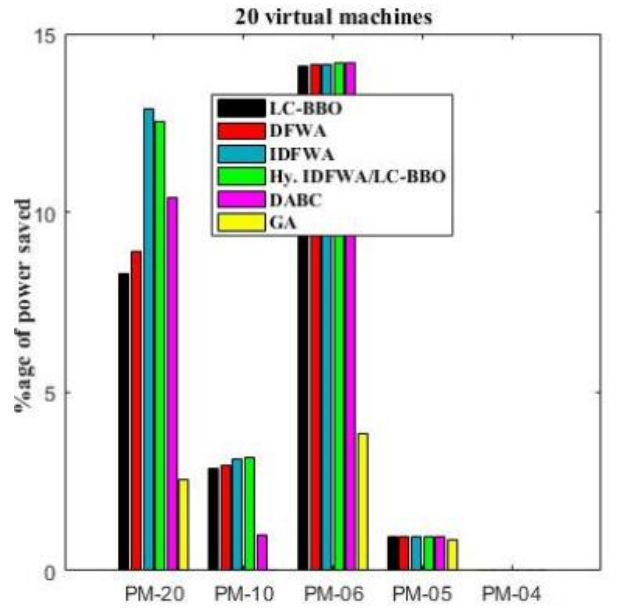


(a)

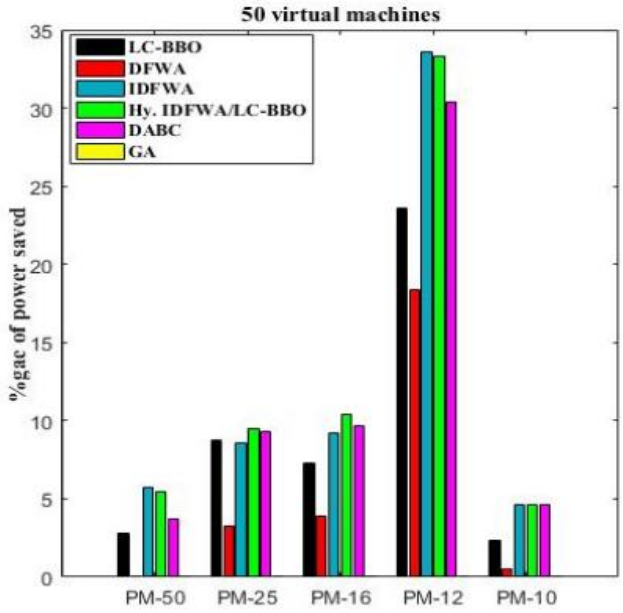


(b)

Figure 3.4 Average power consumed for 100 and 200 VMs.

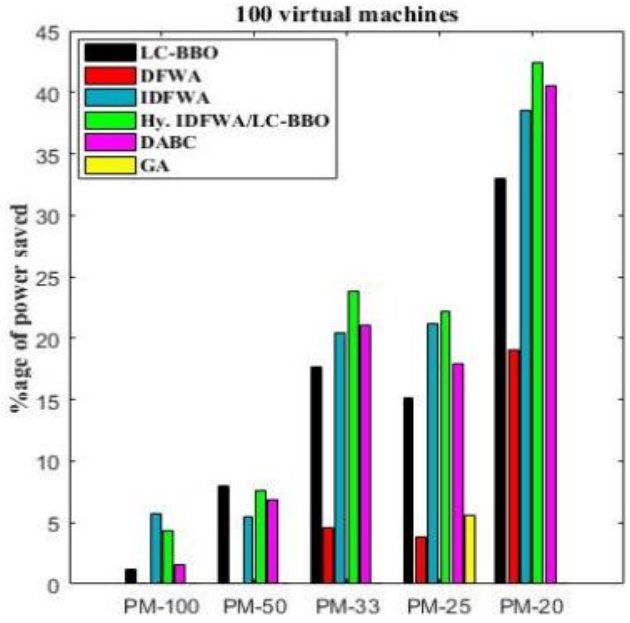


(a)

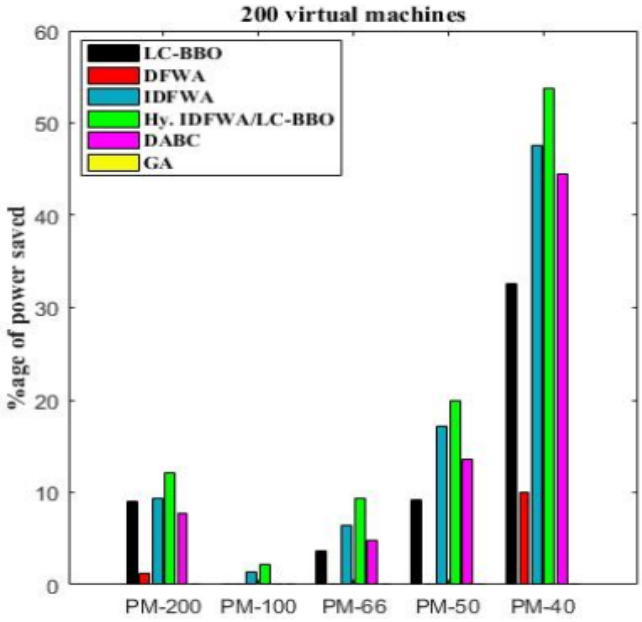


(b)

Figure 3.5 Percentage of power saved by 20 and 50 VMs.

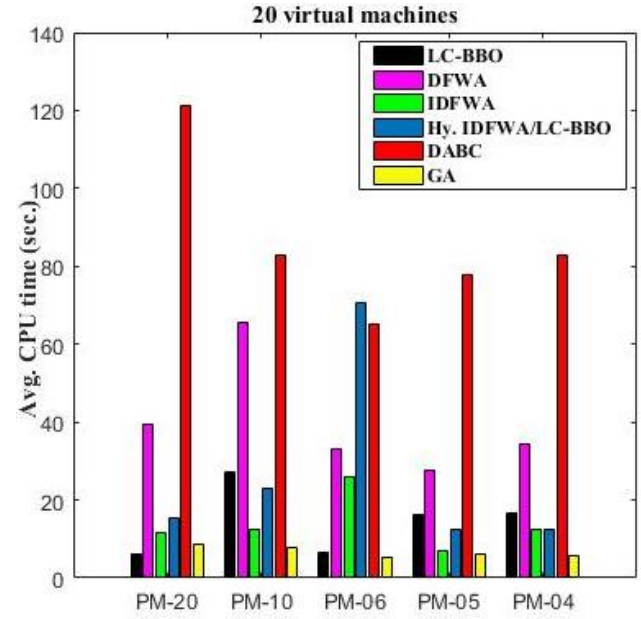


(a)

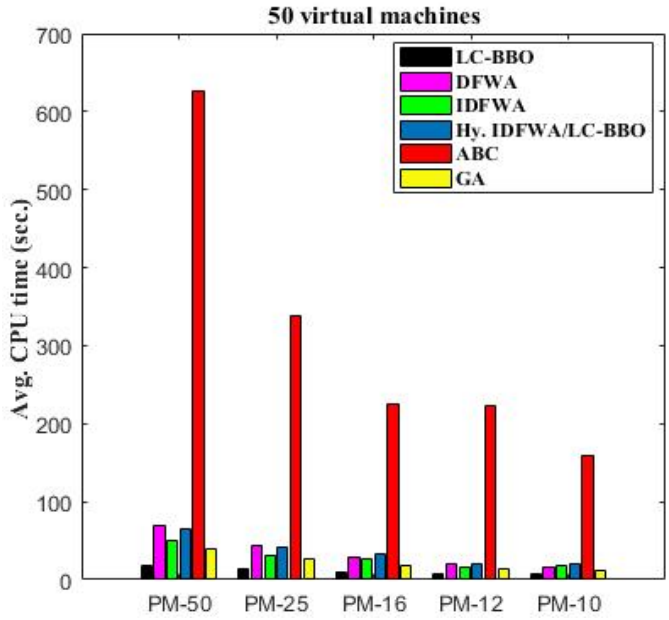


(b)

Figure 3.6 Percentage of power saved by 100 and 200 VMs.

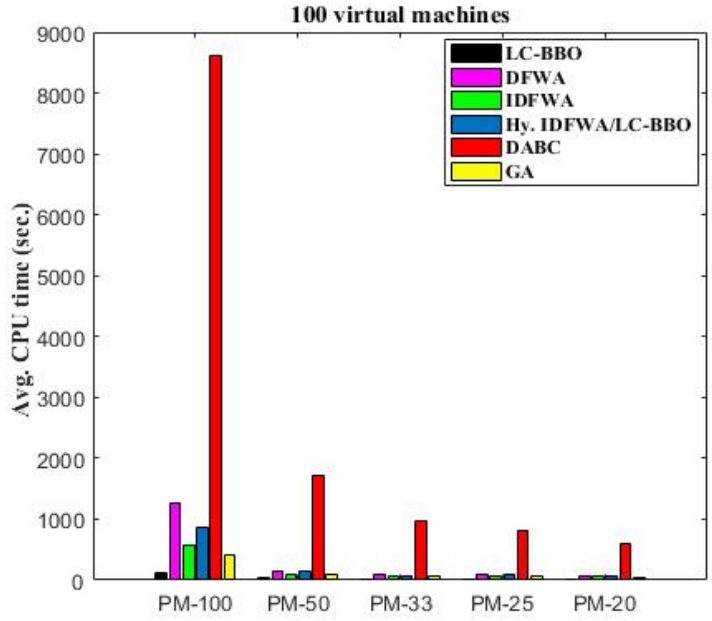


(a)

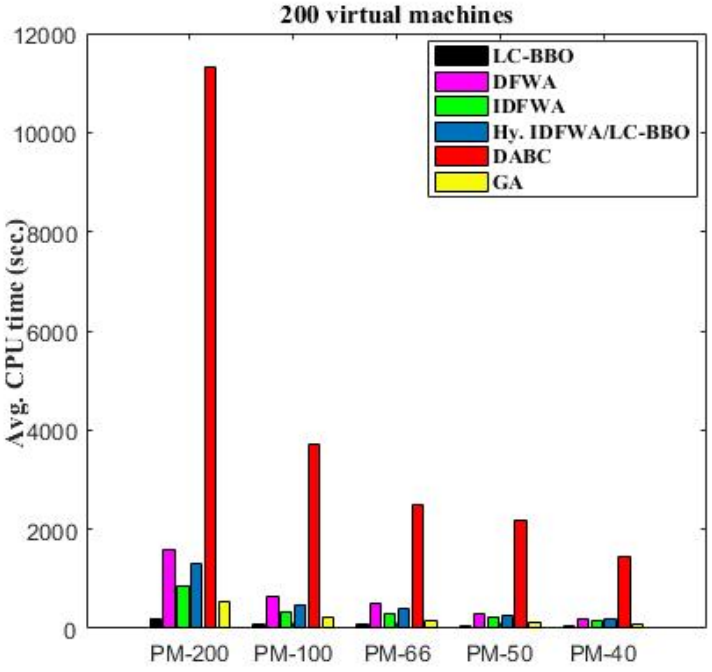


(b)

Figure 3.7 Avg. Matlab CPU time (sec.) consumed by 20 and 50 VMs.

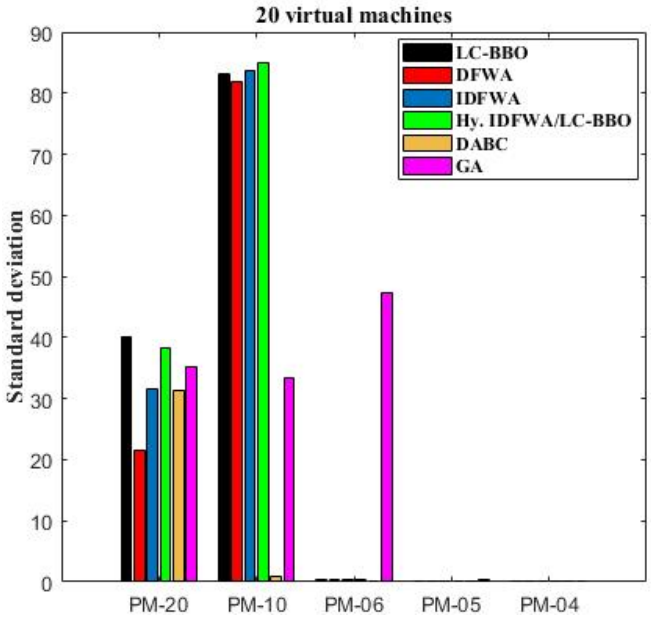


(a)

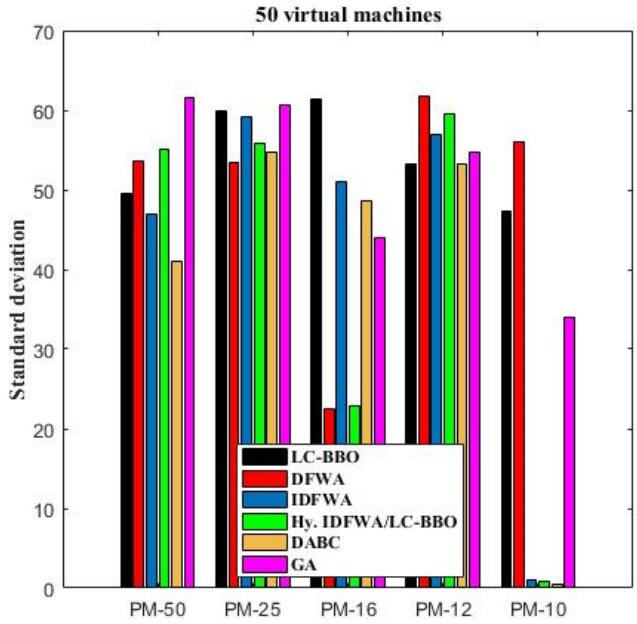


(b)

Figure 3.8 Avg. Matlab CPU time (sec.) consumed by 100 and 200 VMs.

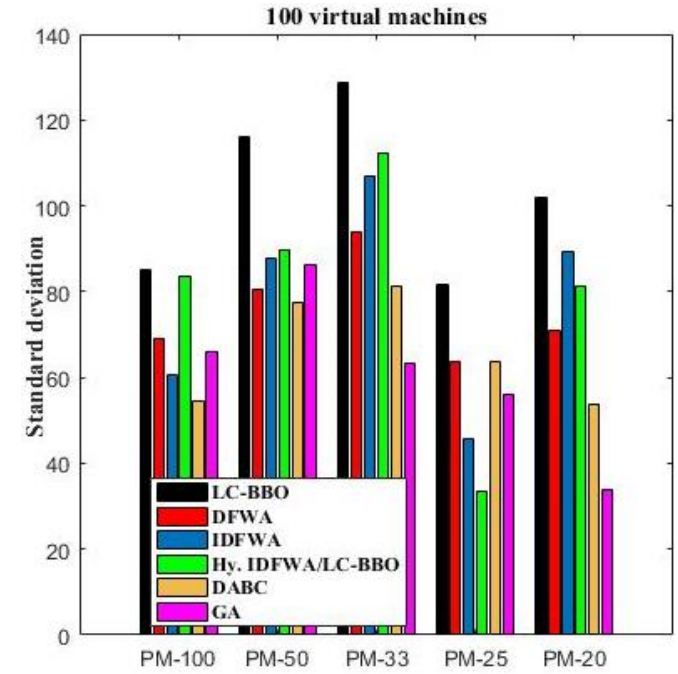


(a)

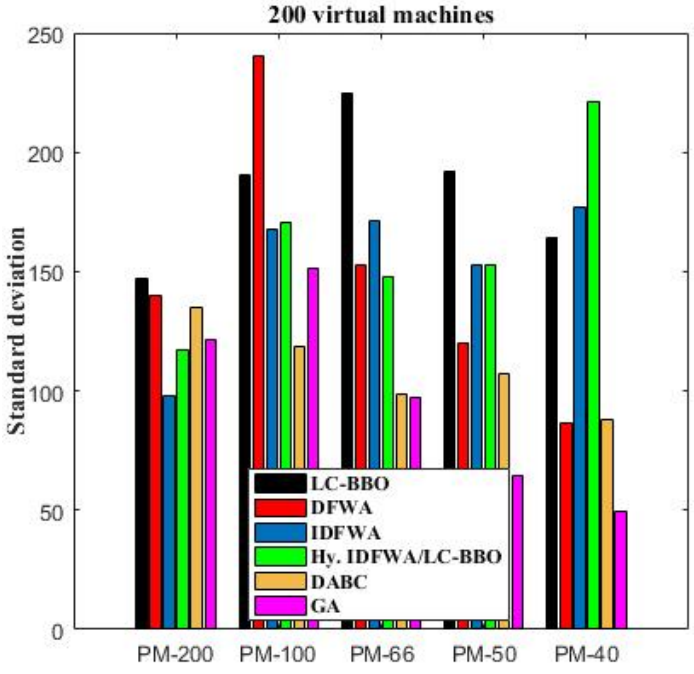


(b)

Figure 3.9 Standard deviation for 20 and 50 VMs.



(a)



(b)

Figure 3.10 Standard deviation for 100 and 200 VMs.

CPU time can differ among algorithms, and a designer can avoid using algorithms with undesirable CPU time performance. Figures 3.7 and 3.8 indicate that CPU time increases when the number of VMs increases for all the algorithms, but the CPU time

increase is much more sensitive to the number of VM for the DABC algorithm. Comparing performance of the algorithms based on CPU time can help algorithm designer to differentiate between them from another angle. The GA performs better than the other algorithms in terms of consuming CPU time for 20, and 50 VMs problem instances. However, LC-BBO algorithm is the fastest, followed by the IDFWA, the DFWA, the Hybrid IDFWA/LC-BBO algorithm, the GA, and the DBAC algorithm for the 100 and 200 VMs problem instances.

The standard deviations of cost of all algorithms with respect to multiple VM placements are plotted in Figures 3.9 and 3.10. As conclusive information is not obvious from these results, in next subsection we present these statistics in a different way to obtain further insight into VM placement.

Table 3.7 Simulation results (LC-BBO, DFWA, and IDFWA)

# of VMs	# of PMs	Max. # of function evaluations	Power consumed (watt) by FFD	Low-complexity BBO			DFWA			IDFWA		
				Avg. power consumed (watt) by LC-BBO (Std.)	Power saved by LC-BBO (%)	Avg. Matlab CPU Time by LC-BBO (Sec.)	Avg. power consumed (watt) by DFWA (Std.)	Power saved by DFWA (%)	Avg. Matlab CPU Time by DFWA (Sec.)	Avg. power consumed (watt) by IDFWA (Std.)	Power saved by IDFWA (%)	Avg. Matlab CPU Time by IDFWA (Sec.)
20	20	8000	1877.0	1721.07(40.2)	8.31	6.00	1709.37(21.5)	8.93	39.61	1634.63(31.7)	12.91	11.49
20	10		1314.46	1276.91(83.2)	2.86	27.21	1275.84(81.9)	2.94	65.57	1273.60(83.8)	3.11	12.57
20	6		1007.08	865.20(0.5)	14.09	6.70	864.67(0.4)	14.14	33.25	864.60(0.5)	14.15	25.87
20	5		864.37	856.23(0.1)	0.94	16.09	856.23(0.1)	0.94	27.45	856.26(0.1)	0.94	6.91
50	50	12000	4087.04	3974.12(49.6)	2.76	17.59	4100.11(53.6)	--	69.99	3852.96(47.0)	5.73	50.25
50	25		3125.60	2852.39(59.9)	8.74	14.37	3023.20(53.5)	3.28	44.21	2858.56(59.1)	8.54	31.87
50	16		2028.46	1881.20(61.4)	7.26	10.56	1949.82(22.5)	3.88	28.50	1841.00(51.1)	9.24	25.81
50	12		1311.39	1001.95(53.3)	23.60	8.24	1070.58(61.8)	18.36	19.37	870.71(56.9)	33.60	15.87
50	10		1105.03	1079.09(47.3)	2.35	7.93	1099.14(56.0)	0.53	16.96	1054.30(1.1)	4.59	17.58
100	100	18000	9601.95	9489.20(85.0)	1.17	124.61	9696.82(68.9)	--	1272.35	9058.02(60.7)	5.66	582.22
100	50		5834.19	5369.82(116.1)	7.96	31.40	6027.78(80.7)	---	148.74	5513.20(87.8)	5.50	99.49
100	33		4280.66	3522.10(128.6)	17.72	21.25	4081.58(93.9)	4.65	91.34	3405.68(107.1)	20.44	64.78
100	25		3115.05	2643.41(81.6)	15.14	22.58	2996.96(63.5)	3.79	97.20	2456.96(45.9)	21.13	69.46
100	20		2224.85	1491.23(101.9)	32.97	18.04	1800.99(71.1)	19.05	69.75	1366.90(89.5)	38.56	61.90
200	200	20000	16167.76	14719.39(147.4)	8.96	206.03	15961.04(140.3)	1.28	1599.35	14668.38(97.9)	9.27	858.90
200	100		9879.88	10005.90(190.3)	--	101.90	11287.70(240.4)	--	630.86	9744.06(167.6)	1.37	344.16
200	66		7419.70	7151.74(225.0)	3.61	81.14	8146.97(152.7)	--	490.05	6943.34(171.4)	6.42	313.38
200	50		5040.06	4576.50(192.1)	9.20	49.60	5429.02(120.2)	--	281.33	4172.10(153.0)	17.22	212.49
200	40		3281.66	2209.79(164.5)	32.66	39.98	2955.71(86.8)	9.93	203.07	1718.50(176.9)	47.63	162.70

Table 3.8 Simulation results (Hybrid IDFWA/LC-BBO, Discrete ABC, and GA)

# of VMs	# of PMs	Max. # of function evaluations	Power consumed (watt) by FFD	Hybrid IDFWA/BBO			Discrete ABC			GA		
				Avg. power consumed (watt) by Hy. IDFWA/LC-BBO (Std.)	Power saved by Hy. IDFWA/BBO (%)	Avg. Matlab CPU Time by IDFWA/LC-BBO (Sec.)	Avg. power consumed (watt) by Dis. ABC (Std.)	Power saved by Dis. ABC (%)	Avg. Matlab CPU Time by Dis. ABC (Sec.)	Avg. power consumed (watt) by GA (Std.)	Power saved by GA (%)	Avg. Matlab CPU Time by GA (Sec.)
20	20	8000	1877.0	1641.77(38.3)	12.53	15.28	1681.36(31.3)	10.42	121.20	1829.37(35.1)	2.54	8.45
20	10		1314.46	1273.08(85.0)	3.15	22.90	1301.36(0.8)	1.00	82.86	1407.07(33.3)	--	7.62
20	6		1007.08	864.38(0.3)	14.17	70.82	864.22(0.1)	14.18	65.32	968.38(47.4)	3.84	5.42
20	5		864.37	856.22(0.1)	0.94	12.30	856.14(0.1)	0.95	77.75	857.00(0.3)	0.85	6.00
50	50	12000	4087.04	3863.30(55.1)	5.47	65.26	3936.59(41.0)	3.68	626.26	4385.41(61.5)	--	39.03
50	25		3125.60	2829.73(55.8)	9.47	42.04	2834.81(54.7)	9.30	337.50	3345.99(60.6)	--	27.00
50	16		2028.46	1817.17(22.9)	10.42	32.08	1831.74(48.7)	9.70	225.51	2257.26(43.9)	--	18.57
50	12		1311.39	874.24(59.5)	33.34	19.32	912.72(53.3)	30.40	222.54	1345.81(54.8)	--	13.60
50	10		1105.03	1053.99(0.9)	4.62	21.07	1053.69(0.4)	4.65	158.01	1218.90(33.9)	--	12.57
100	100	18000	9601.95	9191.01(83.7)	4.28	873.40	9445.41(54.4)	1.63	8601.21	9974.48(66.0)	--	400.59
100	50		5834.19	5393.62(89.6)	7.55	134.00	5437.96(77.6)	6.79	1725.10	6571.45(86.3)	--	101.30
100	33		4280.66	3259.54(112.4)	23.85	79.60	3377.83(81.4)	21.09	967.91	4479.81(63.2)	--	62.50
100	25		3115.05	2425.80(33.4)	22.13	85.53	2558.94(63.5)	17.85	822.93	3334.02(56.1)	--	53.54
100	20		2224.85	1280.27(81.3)	42.46	75.20	1321.85(53.9)	40.59	600.83	2099.90(33.8)	5.62	39.40
200	200	20000	16167.76	14200.79(116.9)	12.17	1309.36	14914.06(135.0)	7.75	11324.49	17069.47(121.5)	--	553.58
200	100		9879.88	9666.21(170.4)	2.16	478.37	9982.98(118.6)	--	3710.51	12619.27(151.0)	--	215.52
200	66		7419.70	6728.78(148.1)	9.31	403.47	7066.25(98.8)	4.76	2495.82	9023.07(97.4)	--	157.43
200	50		5040.06	4035.83(152.8)	19.92	268.36	4355.64(107.0)	13.58	2189.91	6121.30(64.5)	--	138.63
200	40		3281.66	1520.07(220.8)	53.68	198.73	1820.77(88.1)	44.52	1450.06	3405.73(49.3)	--	90.57

3.6.2. Performance significance of the Hybrid IDFWA/BBO algorithm

We applied the statistical T-test to compare the performance of our proposed Hybrid IDFWA/LC-BBO algorithm with the performance of the other experimental algorithms. A p-value (Table 3-9) was obtained between the Hybrid IDFWA/LC-BBO algorithm and each of the other experimental algorithms. For each (VM placement) problem instance and each algorithm comparison, the null hypothesis H_0 states that both algorithms produce the same average cost. Also, we performed the t-test of an alternative hypothesis H_1 which states that the Hybrid IDFWA/LC-BBO algorithm produces lower average cost. The p-values can be compared against the generally acceptable level of significance $\alpha = 0.05$ to decide whether hypothesis H_1 is accepted. If the average power consumed by the VM placement using the Hybrid IDFWA/LC-BBO algorithm is lower than any compared algorithm and $p \leq \alpha$, then we conclude that there is a statistically significant difference between the Hybrid IDFWA/LC-BBO algorithm and the other experimental algorithms. Otherwise, we conclude that the observed difference is not statistically significant.

For half of the 20 VM placements, the p-values shown in Table 3-9 indicate that the Hybrid IDFWA/LC-BBO algorithm performed significantly better than the IDFWA, the LC-BBO algorithm, the DFWA, the DABC algorithm, and the GA. However, the Hybrid IDFWA/LC-BBO algorithm did not perform significantly better than the IDFWA for (20, 20), (20, 10), and (20, 04); better than the DFWA for (20, 10), (20, 05), and (20, 04); better than the LC-BBO algorithm for (20, 10) and (20, 05); or better than the GA for (20, 05) VM placements. Similarly, the Hybrid IDFWA/LC-BBO algorithm significantly outperformed the IDFWA, the LC-BBO algorithm, the DFWA, the DABC algorithm, and the GA in most of the 50 VM placements; the exception was the IDFWA for (50, 50), (50, 12), and (50, 10), and the DABC algorithm for (50, 25) VM placements, where the p-values > 0.05. A significant performance difference is observed between the Hybrid IDFWA/LC-BBO algorithm and the other experimental algorithms for most of the 100 and 200 VM placements, as $p \leq 0.05$, except for the LC-BBO algorithm for (100, 50). Overall,

the Hybrid IDFWA/LC-BBO algorithm or the IDFWA would be preferred over the other experimental algorithms for 20 and 50 VM placements, and the Hybrid IDFWA/LC-BBO algorithm would be a better choice than all of the other experimental algorithms for the cases of 100 and 200 VM placements.

Table 3-9 shows the p-values associated with null hypothesis and the significance of the results for the experimental algorithms, but it does not show the median, minimum, maximum, and the spread of power consumption values for the different algorithms. We used a box-plot to graphically represent the results and present more meaningful illustrations for the same groups of algorithms that we listed in Table 3-9. We depict the results using VM placements 50, 100, and 200 to show the trend from small to large numbers of VM placements with different numbers of PMs. We ignore the box-plot for 20 VM placements, as there was not a reasonable spread observed for the power consumed.

A reasonable variation in the average power consumed can be seen for VM placement using all the experimental algorithms in Figure 3.11 (a), (b), and (d) for the (50, 50), (50, 25), and (50, 12) VM placements compared to the (50, 16) and (50, 10) VM placements. The Hybrid IDFWA/LC-BBO algorithm achieved better performance and more agreement (in terms of less variance) in most of the cases, except for the (50, 12) VM placements as shown in Figure 3.11 (d). Unlike the 50 VM placements, the variability and symmetry improved in the 100 VM placements, as shown in Figure 3.11. However, the Hybrid IDFWA/LC-BBO algorithm is the best performer of the experimental algorithms in terms of power consumption for 100 VM placements. In Figure 3.12, where there are a relatively large number of VM placements, the Hybrid IDFWA/LC-BBO algorithm outperformed the other experimental algorithms in terms of power minimization. Therefore, our proposed algorithm is effectively minimizing power consumption, which is critical in assigning VMs to PMs. Our proposed Hybrid IDFWA/LC-BBO algorithm shows better results when the number of VM placements increases (Figures 3.11 to 3.13), which demonstrates the effectiveness of our algorithms to achieve better power consumption.

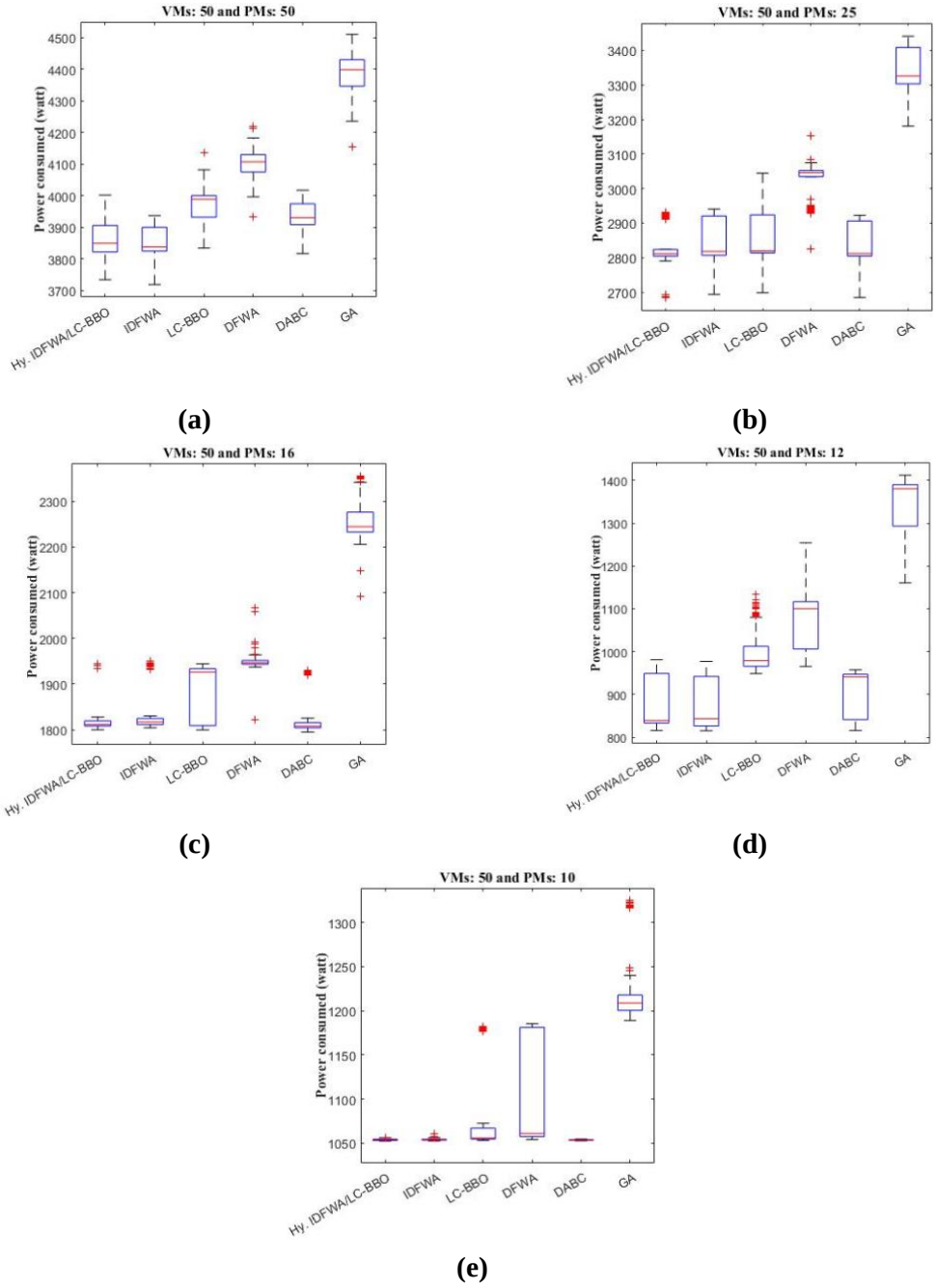


Figure 3.11 Power consumption of VMs is 50 placements to 50, 25, 16, 12 and 10 PMs, respectively, using different algorithms.

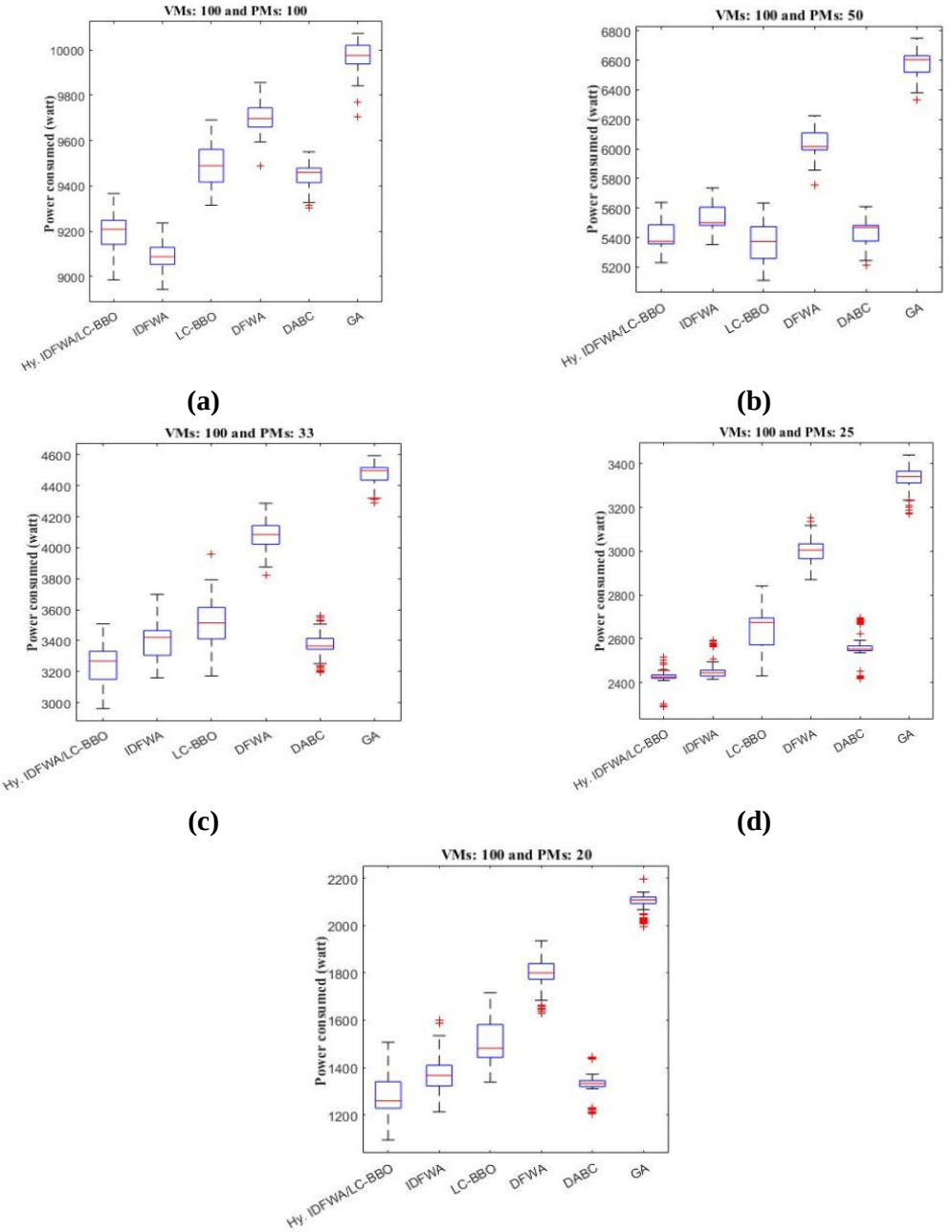
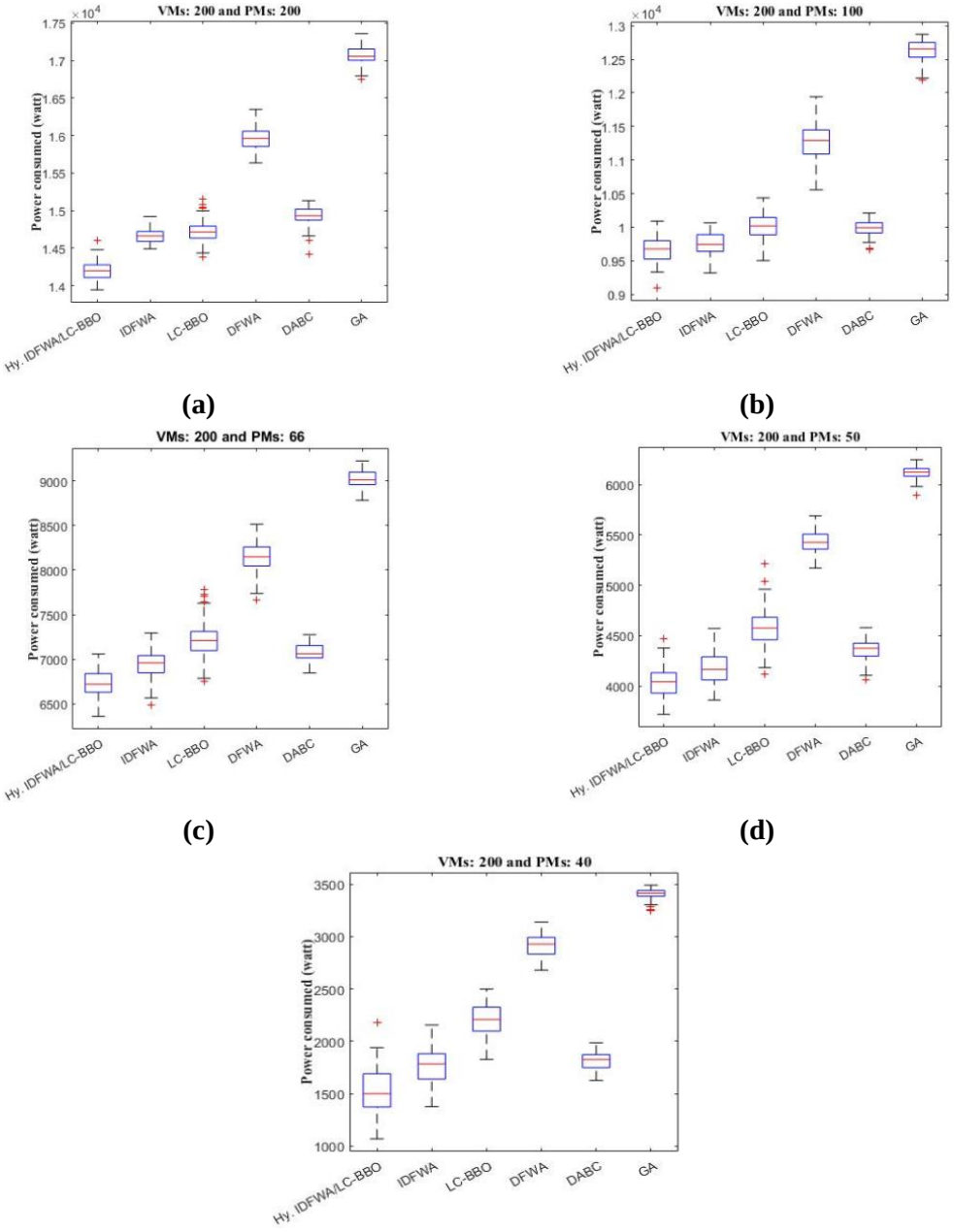


Figure 3.12 Power consumption of 100 VM placements to 100, 50, 33, 25 and 20 PMs, respectively, using different algorithms.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.13 Power consumption of 200 VM placements to 200, 100, 66, 50 and 40 PMs, respectively, using different algorithms.

Table 3.9 T-test for the VM placement problem

# of VMs	# of PMs	Max. # of function evaluations	Algorithms				
			p-value for Hybrid IDFWA/LC-BBO vs. IDFWA	p-value for Hybrid IDFWA/LC-BBO vs. LC-BBO	p-value for Hybrid IDFWA/LC-BBO vs. DFWA	p-value for Hybrid IDFWA/LC-BBO vs. DABC	p-value for Hybrid IDFWA/LC-BBO vs. GA
20	20	8000	0.152	0.001	0.001	0.001	0.001
20	10		0.965	0.747	0.816	0.0010	0.001
20	6		0.000	0.000	0.001	0.001	0.001
20	5		0.014	0.590	0.513	0.001	0.001
20	4		0.116	0.003	0.809	0.0040	0.803
50	50	12000	0.155	0.001	0.001	0.001	0.001
50	25		0.000	0.0060	0.001	0.516	0.001
50	16		0.001	0.001	0.001	0.007	0.001
50	12		0.669	0.001	0.001	0.001	0.001
50	10		0.035	0.001	0.001	0.003	0.001
100	100	18000	0.001	0.001	0.001	0.001	0.001
100	50		0.001	0.106	0.001	0.001	0.001
100	33		0.001	0.001	0.001	0.001	0.001
100	25		0.001	0.001	0.001	0.001	0.001
100	20		0.001	0.001	0.001	0.001	0.001
200	200	20000	0.001	0.001	0.001	0.001	0.001
200	100		0.001	0.001	0.001	0.001	0.001
200	66		0.001	0.001	0.001	0.001	0.001
200	50		0.001	0.001	0.001	0.001	0.001
200	40		0.001	0.001	0.001	0.0070	0.001

3.7. Conclusion

We proposed discrete FWA (DFWA), problem-specific information based DFWA (IDFWA) and hybrid IDFWA/LC-BBO algorithms for integer space VM placement

problems in a datacenter. We proposed modifications to EFWA operators to solve VM placements and the modified algorithm is called the DFWA. We collected problem-specific information based on domain-knowledge of the VM placement problem and incorporate the domain knowledge into the DFWA (IDFWA), then hybridized the IDFWA with the LC-BBO algorithm to develop a Hybrid IDFWA/LC-BBO. In the Hybrid IDFWA/LC-BBO, the exploitation operator uses either a low-complexity BBO migration operator or an IDFWA explosion operator with user determined probability. The new algorithms were tested for VM placement to PMs with the objective of minimizing the power consumption in datacenters.

Our experimental results highlight three findings. (1) The Hybrid IDFWA/LC-BBO and the IDFWA saved power (in VMs to PMs Placement) of approximately 53 percent and 47 percent, respectively as compared to the FFD (first fit decreasing) algorithm. (2) The Hybrid IDFWA/LC-BBO algorithm and the IDFWA consume less average CPU time than the DFWA and the DABC algorithm. (3) Statistical analysis showed that the Hybrid IDFWA/LC-BBO and the IDFWA perform significantly better than the other algorithms tested.

Our results demonstrate that the metrics 'average power consumed,' 'percentage of power saved,' and 'average CPU time (sec.),' can be used to select an appropriate algorithm for VM placement in a datacenter. In other words, the above metrics are design trade-offs that can be used to select an algorithm for VM placement to PMs in a datacenter. For example, the LC-BBO algorithm is very fast (in terms of CPU time) with a relatively good performance in terms of power consumption. However, for a relatively fast algorithm and very good performance in terms of power consumption, one can also select the Hybrid IDFWA/LC-BBO algorithm or the IDFWA.

Chapter 4. Optimizing power for emerging IoT applications

4.1. Introduction

The Internet of Things (IoT) is an emerging technology that consists of physical devices (e.g., vehicles, home appliances) and other items embedded with electronics that can be wirelessly connected to the Internet. These wireless devices use various configurations to exchange data [75]. Emerging IoT applications range from town planning, smart parking, traffic routing, and robotics. IoT is also used to improve efficiency in agriculture and the retail industry and has been employed in various types of forecasting [11], [76]–[85]. Typically, IoT networks are resource (i.e., computation, memory) and power constrained. Although IoT services can decrease human labour, they contribute to an increase in global energy consumption, which is a threat to the environment as it is indirectly linked to greenhouse gas emissions [11], [76].

Excessive power consumption is costly and a point of concern in remote outdoor environments, particularly where electrical power is not easily accessible. Therefore, in designing an IoT network, power-efficient resource assignment algorithms are developed. Recent research in IoT applications is intended to reduce the power consumption and minimize the carbon footprint.

Data transmissions in IoT network from source to sink consumes significant power [86]. Therefore, paths (i.e., routes) are selected that have a small number of hops, and the status of the battery power in IoT nodes is considered during the routing of data from source to sink. Another challenging task in the wireless sensor network (WSN) is to select/elect the cluster head (CH) to coordinate among the IoT nodes and to route data from IoT nodes to the sink. Selecting CH locations and estimating their overall impact on the lifetime of the WSN of an IoT system is a challenge in designing energy efficient routing algorithms. Typically, WSN nodes have limited computing resources such as processing and memory. Therefore, most WSN models consider only power consumption during radio communication and ignore computational power consumption [86]. However, IoT

networks face challenges of latency in the communication for IoT nodes [11]. Fog computing can be a potential solution to minimize the latency in IoT network, but gateways act as CHs can be battery powered [87].

WSNs for IoT systems are often formulated as nonconvex optimization or combinatorial optimization problems [88], [89]. Finding exact solutions for optimization problems is infeasible in reasonable time due to the nonexistence of polynomial time algorithms. Therefore, approximate algorithms such as the genetic algorithm (GA) and particle swam optimization (PSO) algorithms are used to solve these problems to find quality solutions in moderate time [90]–[92].

In this chapter we model an IoT network in which IoT nodes require real-time communication. In real-time communication, IoT nodes need real-time feedback from the CHs and CHs need reasonable computing resources for a real-time response. The proposed cluster-assisted IoT network contains battery powered core cluster nodes (CCNs) as CHs with computing resources such as a central processing unit (CPU) and memory. The prolonged life of the proposed IoT network is critically dependent on the better utilization of computing resources of the CCNs, which route data to the base stations (BSs) or sinks. The goal of the proposed model is to minimize the transmission (IoT-CCN and CCN-BS) and computational power at CCNs. We formulate IoTs-CCNs and CCNs-BSs assignments as an integer programming problem. We propose fireworks based evolutionary algorithms (EAs) to solve IoTs-CCNs and CCNs-BSs assignments.

4.2. Related work

In Internet of Things (IoT) network, billions of physical objects connect to the Internet and generate huge amount of data–a.k.a. Big Data–which required smart computation, storage, memory, bandwidth and reliability. Big Data can be processed using centralized datacentres by moving computing, control, and data storage into clouds. However, scattered nature, latency, power sensitivity, and unreliable transmission are the challenges for traditional cloud computing to meet the requirements of IoT networks. Fog computing provides a bridge between IoT nodes and classic cloud computing. The idea

behind fog computing is to bring the cloud closer to IoT nodes to mitigate the latency and unreliability of data transfer. Each fog node hosts local computation, networking, and storage capabilities. The research community has taken significant interest in designing algorithms that can efficiently assign computing resources (e.g., CPU memory) and reduce power consumption in WSNs. The main goal of resource assignment in WSNs is to reduce the overall cost of power consumption.

In [90] a sensor genetic algorithm (SGA) and a base station genetic algorithm (BGA) were presented. These new algorithms were used to solve the energy constraint in a mission-critical WSN. In the mission-critical WSN, each sensor satisfies its own mission depending on its location. The goal of the SGA was to place each sensor in the best position relative to the degree of mission and quality of communication among nodes. The goal of the BGA was to place a BS with respect to the available resources in the network.

In [91], some of the diverse aspects that cause an energy deficiency in a WSN were considered. One such aspect was energy exhaustion while transmitting data because the energy absorbed in transmitting the data was twice the energy employed in transforming the data. The harmful impact of energy exhaustion highlights the need to adopt an efficient route to transmit data to a sink. Therefore, the transmission route should be selected in such a way that it drains minimal energy while successfully transmitting data. A nature inspired approach was presented to acquire an energy efficient route from source to destination to reduce the energy consumption and to raise the network lifetime.

In [92], a clustering design for a WSN was presented as an efficient way to reduce the consumed power during the transmission of sensed data to a sink/BS. Like LEACH (low-energy adaptive clustering hierarchy) [93], an intelligent clustering protocol was presented to prolong network lifetime and minimize energy consumption. The proposed protocol performs clustering with a dynamic number of clusters depending on the node distribution and the field dimension. The modified genetic algorithm (MGA) was used to select an optimum number of clusters and elect suitable cluster heads (CHs). The goal of the WSN was to minimize the total energy consumed by all nodes. The simulation result showed that the MGA outperforms the classic clustering protocol in terms of network

lifetime and energy consumption. In [94] a clustering model of a WSN was studied and it was noted that this model lead to heavy traffic and a faster depletion of energy in the nodes that were closer to the sink. A fuzzy logic-based energy conserved unequal clusters with fuzzy logic (ECUCF) algorithm was presented to conserve energy, suppress the hot spot problem, and achieve a load balance. The CH clusters that were located closer to the BS/sink were designed to be smaller than the CH clusters that were situated far away from the sink.

The advantages of the proposed algorithm (MGA) were compared against the advantages of the Low-energy adaptive clustering hierarchy (LEACH) and the fuzzy based unequal clustering (FBUC) algorithms. The energy consumed in sensing versus the energies consumed in transmission and reception were analysed in [95]. The analysis showed that the sensing energy consumed in practical applications was either comparable or greater than the energies consumed in transmission and reception. The authors have investigated the effectiveness of compressed sensing and distributed compressed sensing using real datasets. However, compression might increase the computational energy consumption in the proposed techniques.

In [96], a system was presented that identifies energy consumption behaviour patterns in users' homes to promote more efficient energy usage. A context-aware framework for collaborative learning applications (CAFCLA) was used to develop the system for home users. However, the accuracy of the system was not satisfactory, and the system implementation was expensive for home users. Typically, sensor network (SN) data were routed from the sink to the Internet and acute energy was an important resource during the communication phase to prolong the lifetime of SN data. Switching off the nodes transceiver was a way to conserve energy when SNs were neither transmitting nor receiving packets.

In [86], a data caching algorithm (DCAL) was used to optimize the sleep/wake periods of sensor nodes (SNs) to save energy and reduce latency. The DCAL was used to analyse data to avoid continuously transmitting the same information from the SN to the sink. The DACL evaluated whether cached data were different from or the same as data

previously cached. In the case that data are different from previously cached data, the SN would wake up and transmit the data to a sink. Otherwise the newly sensed data would not be transmitted to the sink, thus saving transmission power. However, the DCAL added computational overhead while processing/evaluating cached data.

Authors of [97] conducted a detailed survey on the challenges and limitations of WSNs in the agricultural domain. A taxonomy was designed to classify the energy-efficient techniques that can be used in agricultural applications.

Unlike fog/edge computing [98], our proposed delay sensitive IoT network (IoTN) is an application in which critical but a limited computing is required during the operation of the network. For many resource allocation problems in virtual machine placement [28], in wireless network planning [99], and in IoT networks [100]–[102], computationally efficient algorithms for finding an exact solution are not known. Algorithms that provide well-performing, or high-quality, solutions were devised. For example, the problems discussed in [28] and [101], [102] have characteristics similar to the bin packing problem, a combinatorial optimization problem, and these problems can be solved by using simple heuristics such as first fit/best fit decreasing algorithms.

In Table (4-1), we compare some existing state-of-the-art WSN models, which either minimize the transmission power or minimize the transmission power and computational power. To the best of our knowledge, limited work has been reported in the literature that considers the objective of simultaneously minimizing the data transmission power and the computational power in an IoT network. The layout of the proposed IoT network in a remote area with limited power availability is shown in Figure 4.1. The proposed IoT network is comprised of three types of nodes: IoT, core cluster node (CCN), and base station (BS). IoTs may or may not be battery powered, but CCNs are battery powered and thus have limited power capacity.

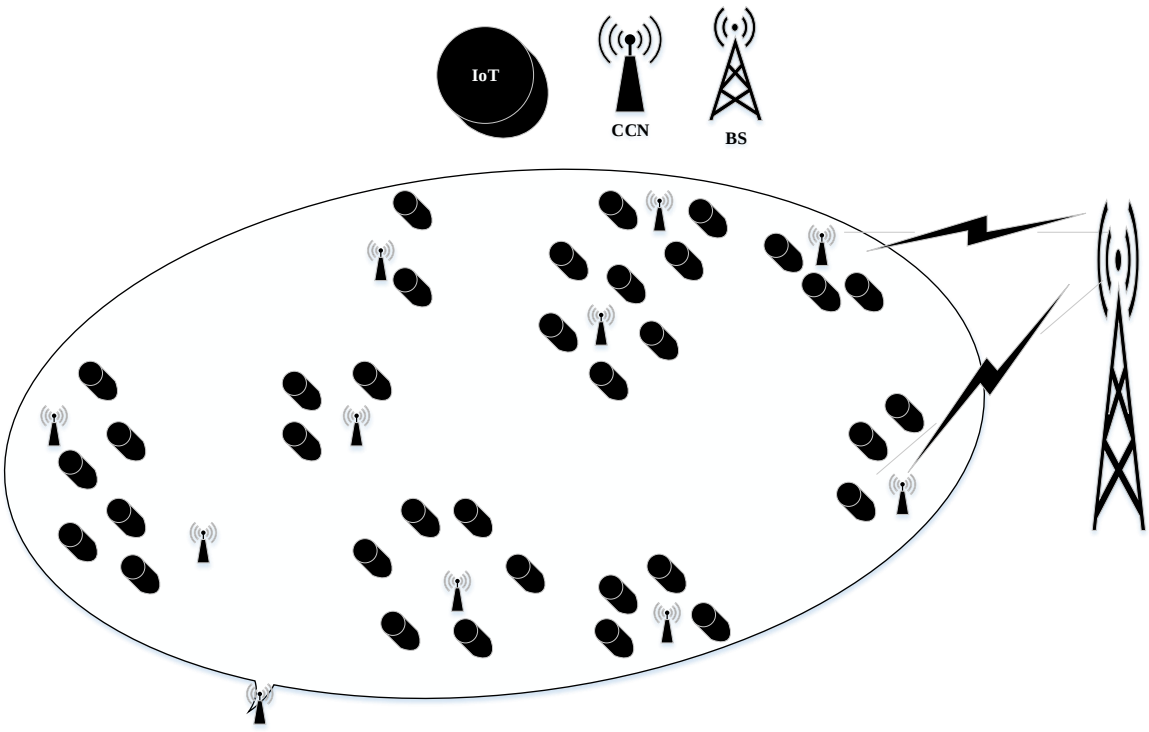


Figure 4.1 Proposed IoT network.

In section 4.3, we present a system model and mathematical framework for optimizing power in the proposed IoT network. In later sections algorithms that find well-performing solutions to the formulated assignment problem will be presented. We propose swarm intelligence based EAs for IoT assignments and experimentally compare the performance of the newly proposed EAs with the performance of some classic EAs and the heuristic First Fit Deceasing (FFD) algorithm.

Table 4.1 Transmission/computation power as an optimization objective in WSNs

Reference	Objective of WSN	Transmission power	Computational power	Algorithms/Methods	Remarks
[86]	Optimizing the energy in precision agriculture.	✓	✓	Energy Efficient Data Caching Algorithm (DCAL).	DCAL algorithm is proposed to optimize the energy consumption and reduce latency.
[88]	Energy efficiency maximization for WSNs	✓	×	Suboptimal iterative algorithms.	Energy efficiency maximization problem with constraints QoS, minimum harvested energy and maximum transmission power.
[89]	Balance the energy consumption in a WSN	✓	×	Dynamic hierarchical protocol based on combinatorial optimization (DHCO).	Optimal route is formulated as a combinatorial optimization problem.
[90]	Optimizing the energy and locations	✓	×	Genetic Algorithm	Optimal SNs and BSs placements that minimize energy.
[91]	Minimizing energy with optimal routing	✓	×	Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).	Optimal routing from SNs to sink that uses minimal energy.
[92]	Prolong lifetime with optimal energy	✓	×	Genetic Algorithm	Balancing the residual energy among the network nodes with an energy filter.
[94]	Prolong lifetime with load balancing and optimal energy	✓	×	Energy Conserved Unequal Clusters with Fuzzy logic (ECUCF) Algorithm.	Balancing the load among the clusters in such a way that WSN consumes minimum energy.
[95]	Optimizing the sensing energy	✓	✓	Compressed sensing and distributed compressed sensing methods.	Compressed and distributed compressed sensing show their potential for efficient utilization of sensing and overall energy costs in wireless sensor networks.

[97]	Optimizing the energy in WSNs for agriculture.	✓	×	Proposed: a precision agriculture management tool.	Presents the taxonomy of energy-efficient techniques for WSNs that can be used in agricultural monitoring systems.
[103]	Transmission power optimization with a minimum node degree	✓	×	Power optimization with a minimum node-degree.	Topology control and optimal transmission range according to node degree and node density.
[104]	Transmission power optimization algorithm	✓	×	Power-optimized cooperative MAC protocol.	Node cooperation mechanism is proposed involving one or multiple nodes with higher channel gain and sufficient residual energy.

4.3. System Model and Problem Formulation

4.3.1. IoT network model

In the proposed IoT network, the IoT node collects data and sends the data to a core cluster node (CCN). An IoT operates in either active or sleep mode. A CCN is a cluster head (CH) with reasonable computing resources. In the mission-specific IoT network [13], [105], the IoT requires real-time feedback. Real-time feedback may not be available due to delay if IoT data are processed at the sink (i.e., the BS) or beyond. Therefore, we assume data are partially processed at the CCN to provide real-time feedback to the IoT. After partial data processing at the CCN, reduced data are sent to a BS. A BS is a node with an uninterrupted main power supply and possesses better computing resources than a CCN. Therefore, we do not incorporate the computing power of a BS in our model of an IoT network. A BS is directly connected to the Internet. Unlike a CCN, only transmission power is considered for a CCN-BS radio link. One or more CCNs can be connected to a BS.

4.3.2. Problem formulation

We present a mathematical framework in which the data transmission and computational power consumption in an IoT network are to be minimized. We formulate IoTs-CCNs and CCNs-BSs resource (i.e. memory, CPU) assignments in the proposed IoT network. The notation/terminology used for the IoTs-CCNs and CCNs-BSs assignments are given in Table (4-2).

Table 4.2 Notations used in chapter 4

Symbol	Definition
$\mathcal{H}$	set of IoT nodes (IoTs).
$\mathcal{M}$	set of core cluster nodes (CCNs).
$\mathcal{G}$	set of base stations (BSs).
$\mathcal{S}_i$	denotes an IoT, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
c_j	denotes a CCN, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.
$\mathscr{b}_k$	denotes a BS, where $k = 1,2, \dots, \lvert\mathcal{G}\rvert$.
$\mathbb{u}_j$	represents the percentage of CPU utilization of a CCN c_j.
e_j	power consumption of a CCN c_j.
e_{max}^{j}	maximum power consumption of a CCN c_j, when $\mathbb{u}_j = 100\%$.
e_{idle}^{j}	power consumption of a CCN c_j in idle mode.
$\mathcal{S}_{cpu}^{i}$	CPU demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
$\mathcal{S}_{mem}^{i}$	memory (RAM) demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
$\mathcal{S}_{data}^{i}$	data transmission demand of an IoT $\mathcal{S}_i$, where $i = 1,2, \dots, \lvert\mathcal{H}\rvert$.
c_{cpu}^{j}	CPU capacity of a CCN c_j, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.
c_{mem}^{j}	memory (RAM) capacity of a CCN c_j, where $j = 1,2, \dots, \lvert\mathcal{M}\rvert$.
x_{ij}	binary value representing whether an IoT, $\mathcal{S}_i$, is assigned to a CCN c_j.
y_{jk}	binary value representing, whether a CCN c_j is assigned to a BS $\mathscr{b}_k$.

The objective of the optimization problem is to minimize the total computational and transmission power consumption in the IoT network. The IoTs-CCNs and CCNs-BSs

assignments are represented by the binary decision variables x_{ij} and y_{jk}, respectively, as follows:

$$x_{ij} = \begin{cases} 1, & \text{if } \mathcal{S}_i \text{ is assigned to } c_j \\ 0, & \text{otherwise} \end{cases} \quad 1 \le i \le |\mathcal{H}|, 1 \le j \le |\mathcal{M}|. \tag{4.1}$$

$$y_{jk} = \begin{cases} 1, & \text{if } c_j \text{is assigned to } \mathcal{b}_k \\ 0, & \text{otherwise} \end{cases} \quad 1 \le j \le |\mathcal{M}|, 1 \le k \le |\mathcal{G}|. \tag{4.2}$$

Note that in (4.1) x_{ij}= 1 if a radio link is established between an IoT and a CCN, and x_{ij}= 0 otherwise. Similarly, in (4.2) y_{jk}=1 if a radio link is established between a CCN and a BS, and y_{jk}=0 otherwise. We denote IoTs to CCNs assignments as a matrix $\mathcal{SC}$ in terms of binary decision variables x_{ij} as follows:

$$\mathcal{SC} = \begin{pmatrix} x_{11} & \cdots & x_{1|\mathcal{M}|} \\ \vdots & \ddots & \vdots \\ x_{|\mathcal{H}|1} & \cdots & x_{|\mathcal{H}||\mathcal{M}|} \end{pmatrix}. \tag{4.3}$$

Let $\mathcal{SC}_j$ is denoted as the j^{th} column of the matrix *SC* (4.3), where j^{th} column $\mathcal{SC}_j$ represents connection(s) of IoTs with j^{th} CCN and i^{th} row represents an IoT. The decision variable x_{ij}=1, if the i^{th} IoT is assigned to the j^{th} CCN, and x_{ij}=0 otherwise.

Similarly, we denote radio links between CCNs and BSs as a matrix $\mathcal{CB}$ in terms of binary decision variables y_{jk}. The matrix $\mathcal{CB}$ represents CCNs to BSs assignments in terms of binary decision variables y_{jk} as follows:

$$\mathcal{CB} = \begin{pmatrix} y_{11} & \cdots & y_{1|\mathcal{G}|} \\ \vdots & \ddots & \vdots \\ y_{|\mathcal{M}|1} & \cdots & y_{|\mathcal{M}||\mathcal{G}|} \end{pmatrix}. \tag{4.4}$$

where in (4.4) matrix $\mathcal{CB}$, each row represents a CCN and each column represents a BS. The decision variable y_{jk}=1, if the j^{th} CCN is assigned to the k^{th} BS, and y_{jk}=0 otherwise. In other words, only active/idle mode CCNs should be connected to BSs (i.e., y_{jk}=1), while CCNs in sleep mode are not connected to any BS (i.e., y_{jk}=0).

CPU utilization of a CCN is the ratio of the sum of the CPU demand of the IoTs connected to the j^{th} CCN to the CPU capacity of the j^{th} CCN. We define $\mathbb{u}_j$ as the CPU utilization of the j^{th} CCN. For a given assignment in SC, the CPU utilization $\mathbb{u}_j$ of the j^{th} CCN c_j is computed as follows:

$$\mathbb{u}_j = \frac{\sum_{i \in \mathcal{H}} \mathcal{S}^i_{cpu} \times x_{ij}}{c^j_{cpu}}, \tag{4.5}$$

where $\mathcal{S}^i_{cpu}$ is the CPU demand of the i^{th} IoT node in (4.5), and c^j_{cpu} is the CPU capacity of the j^{th} CCN. The computational power consumption e_j of a CCN, c_j, includes the overhead power e^j_{idle}. The overhead power e^j_{idle} is the power consumed by a CCN, c_j, in idle mode. Total power of a j^{th} CCN can be computed as follows:

$$e_j(SC_j) = e_j\left(\begin{pmatrix} x_{1j} \\ \vdots \\ x_{|\mathcal{H}|j} \end{pmatrix}\right) = \begin{cases} 0, & \text{if } SC_j \text{ is null vector} \\ (e^j_{max} - e^j_{idle}) \times \mathbb{u}_j + e^j_{idle}, & \text{otherwise} \end{cases} \tag{4.6}$$

where in (4.6) e^j_{max} is the maximum power of a CCN c_j. We assume that the power of the j^{th} CCN is e_j=0, when no IoT is assigned to the j^{th} CCN c_j. More specifically, e_j=0 when SC_j is a null vector. The maximum power e^j_{max} of a CCN c_j is a device dependent parameter.

When no IoT is assigned to c_j (i. e., $\sum_{i \in \mathcal{H}} x_{ij} = 0$), this formulation assumes that a CCN can be turned into sleep mode, and it consumes no power. In this work, the power of idle status a CCN is set to 70% of the maximum power as follows:

$$e^j_{idle} = e^j_{max} \times 0.7, \tag{4.7}$$

where in (4.7), e^j_{idle} is the overhead power of a CCN when the power is turned on. Using current formulation, one can set different value for the idle status of a CCN (e^j_{idle}).

In [99], [106], [107], various models of power consumption are presented for radio communication between two communicating nodes. We consider the channel gain to be a factor of the transmitting power for the radio links between IoTs to CCNs and between

CCNs to BSs. For transmission power for radio links from IoTs to CCNs and from CCNs to BSs, we define the gain as follows:

- Propagation factor of the radio link between IoT *i* and CCN *j*:
  - $0 < g_{ij} < 1, \forall i \in \mathcal{H}$ and $j \in \mathcal{M}$
- Propagation factor of the radio link between CCN *j* and BS *k*:
  - $0 < g_{jk} < 1, \forall j \in \mathcal{M}$ and $k \in \mathcal{G}$

The objective of the problem is to find appropriate IoTs-CCNs and CCNs-BSs assignments that minimize the computational and transmission power in the IoT network. The total computational power at CCNs is denoted by ϕ and is expressed as follows:

$$\phi = \sum_{j \in \mathcal{M}} e_j(SC_j).$$

We denote Φ as the transmission power between IoTs to CCNs and between CCNs to BSs. The transmission power Φ is a part of the optimization objective and it has two terms. These terms compute the transmission power between IoTs to CCNs and between CCNs to BSs, respectively. In this work, we assume that the data transmitted from a CCN to a BS will be halved after data processed at a CCN and is denoted as: $c_{data}^{j} = \frac{\sum_{i \in \mathcal{H}} s_{data}^{i} \times x_{ij}}{2}$. As propagation factor is in open interval (0,1), if it is closer to 1, transmission power between two transmitting nodes is lower and vice versa. The transmission power Φ is expressed as follows:

$$\Phi = \sum_{j \in \mathcal{M}} \sum_{i \in \mathcal{H}} \left(\frac{s_{data}^{i} \times x_{ij}}{g_{ij}} \right) + \sum_{k \in \mathcal{G}} \sum_{j \in \mathcal{M}} \left(\frac{c_{data}^{j} \times y_{jk}}{g_{jk}} \right).$$

The cost function and constraints for the IoTs-CCNs and CCNs-BSs assignments are as follows:

$$\min_{\substack{x_{ij} \in \{0,1\}, \forall\, i \in \mathcal{H}, j \in \mathcal{M} \\ y_{jk} \in \{0,1\}, \forall\, j \in \mathcal{M}, k \in \mathcal{G}}} \quad W_1 \times \phi + W_2 \times \Phi \tag{4.8}$$

subject to:

$\sum_{j\in\mathcal{M}} x_{ij} = 1, \forall i \in \mathcal{H}$, where $x_{ij} \in \{0,1\}$ (4.9)

$\sum_{k\in\mathcal{G}} y_{jk} \leq 1, \forall j \in \mathcal{M}$, where $y_{jk} \in \{0,1\}$ (4.10)

$\sum_{i\in\mathcal{H}} \mathcal{S}_{cpu}^{i} \times x_{ij} \leq c_{cpu}^{j}, \forall j \in \mathcal{M}$ (4.11)

$\sum_{i\in\mathcal{H}} \mathcal{S}_{mem}^{i} \times x_{ij} \leq c_{mem}^{j}, \forall j \in \mathcal{M}$ (4.12)

W_1 and W_2 are weight parameters for the computational and transmission power of the cost function in (4.8). Constraint (4.9) ensures that each IoT can be assigned to only one CCN. Constraint (4.10) confirms that each CCN in active mode can be assigned to only one BS. Constraints (4.11)−(4.12) ensure that the sum of the total CPU and memory demand of the IoTs assigned to a CCN does not exceed the total CPU and memory capacity of that CCN. We assume that the c_{data}^{j} transmitted from the IoTs is reduced to half of the total data after partial processing at a CCN. There are two advantages of partially processing data at CCNs:

- To provide real-time feedback to the communicating IoTs,
- To reduce the transmission power between a CCN-BS.

In this work, we assume that IoTs' demand of the resources (i.e., memory and CPU) and CCNs' capacity of the resources (i.e., memory and CPU) are enough to accommodate all IoTs. In other words, simulation parameters are generated in such a way that the total resource capacity of CCNs exceeds the total resource demand of IoTs. The number of feasible assignments of IoTs to CCNs and CCNs to BSs are increased with an increase in size of any of the sets $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{G}$. Therefore, it is impractical to try to find an exact solution through an exhaustive search for IoTs-CCNs and CCNs-BSs assignments (4.8)−(4.12). A practical approach is to use approximate algorithms for good-quality solutions with reasonable computing resources.

4.4. Problem Reformulation

The IoTs-CCNs and CCNs-BSs assignments are formulated in section 4.2.2 as a special case of discrete (binary) space optimization. Our proposed EAs are unable to operate on the current IoTs-CCNs and CCNs-BSs assignments formulation. In this work, we redefine the decision variables and reformulate the IoTs-CCNs and CCNs-BSs assignment problem.

4.4.1. Redefining the decision variables

In the IoTs-CCNs and CCNs-BSs assignments, $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{G}$ denote the sets of IoTs, CCNs, and BSs, respectively. We define a candidate solution as a vector of nonnegative integers $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$, where $|\mathcal{H}|$ and $|\mathcal{M}|$ are the cardinalities of sets $\mathcal{H}$ and $\mathcal{M}$. IoTs and CCNs are randomly assigned the indices in the vector X. Note that in X, X_i $(i = 1,2, \dots, |\mathcal{H}|)$ represents the i^{th} IoT connected to some CCN j, and X_j $(j = |\mathcal{H}| + 1, |\mathcal{H}| + 2, \dots, |\mathcal{H}| + |\mathcal{M}|)$ represents the j^{th} CCN that can be connected to some BS k. In X, X_j is zero when the j^{th} CCN is in sleep mode and is not connected to any BSs. Each IoT can be connected to any one of the CCNs, and an active mode CCN can be connected to any one of the BSs. Therefore, in X, we represent BSs followed by CCNs in the consecutive order of positive integers. Suppose we have $|\mathcal{G}|$ BSs and $|\mathcal{M}|$ CCNs in a candidate solution X; then, in the candidate solution X the representations for BSs are 1, 2, 3…, $|\mathcal{G}|$ and representations for CCNs are $1+|\mathcal{G}|$, $2+|\mathcal{G}|$, $3+|\mathcal{G}|$…, $|\mathcal{M}|+|\mathcal{G}|$. For example, let us consider three BSs, i.e., $\mathcal{G}$= {1,2,3}, three CCNs, i.e., $\mathcal{M}$= {1,2,3}, and four IoTs, i.e., $\mathcal{H}$= {1,2,3,4}, and a candidate solution is $X = (5,4,4,5,1,2,0)$. In X, we represent three BSs as 1, 2, and 3 and three CCNs as $1+|\mathcal{G}|$ (i.e., 4), $2+|\mathcal{G}|$ (i.e., 5), and $3+|\mathcal{G}|$ (i.e., 6), where $|\mathcal{G}|$=3 is the cardinality of the set $\mathcal{G}$. Here in X, the first four indices represent IoTs (connected to some CCNs) and the last three indices represent CCNs (connected to some BSs). Clearly we can see that IoT X_1 is connected to the 2nd CCN *'2+ $|\mathcal{G}|$'* (i.e., 5), X_2 is connected to the 1st CCN *'1+$|\mathcal{G}|$'* (i.e., 4), X_3 is connected to the 1st CCN *'1+$|\mathcal{G}|$'* (i.e., 4), and X_4 is connected to the 2nd CCN *'2+ $|\mathcal{G}|$'* (i.e., 5). Similarly, in the active mode, CCNs X_5 and X_6 are connected to the 1st and the 2nd BSs,

respectively, while CCN X_7 is assumed to be in the sleep mode and is not connected to any BS (i.e., $X_7 = 0$). The candidate solution X is represented as follows:

$$X = \left(X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|}\right). \tag{4.13}$$

where $|\mathcal{H}|$and $|\mathcal{M}|$ are the cardinalities of the sets $\mathcal{H}$ and $\mathcal{M}$, respectively.

4.4.2. Reformulating the IoTs assignments

With the active CCNs, IoTs-CCNs and CCNs-BSs connections are the configuration of the proposed IoT assignments. Implicitly enforcing constraints (3.10) and (3.11), we use the decision vector of nonnegative integers $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$, where $|\mathcal{H}|$ and $|\mathcal{M}|$ are the cardinalities of sets $\mathcal{H}$ and $\mathcal{M}$, respectively. Here each X is a candidate configuration of the IoTs-CCNs and CCNs-BSs assignments. We use candidate solution $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$ to implement the IoTs-CCNs and CCNs-BSs assignments. The CPU utilization of CCN c_j can be calculated as follows:

$$\mathbb{u}_j = \frac{\sum_{\{i:1 \le i \le |\mathcal{H}| \wedge X_i = j\}} \mathcal{S}^i_{cpu}}{c^j_{cpu}}. \tag{4.14}$$

The CPU utilization $\mathbb{u}_j$ of a CCN is the ratio of the sum of the CPU demand of the IoTs connected to the j^{th} CCN and the CPU capacity of the j^{th} CCN. The total computational power ϕ' at CCNs, for $j = 1,2,3,\dots|\mathcal{M}|$, is as follows:

$\phi' = \sum_{j\in\mathcal{M}}\left((e^j_{max} - e^j_{idle}) \times \mathbb{u}_j + e^j_{idle}\right)$, where $\mathbb{u}_j$ is CPU utilization of the j^{th} CCN as defined in (4.14).

The total transmission power Φ' between IoTs to CCNs and between CCNs to BSs is as follows:

$$\Phi' = \sum_{j\in\mathcal{M}} \sum_{i\in\mathcal{H}} \left(\frac{\sum_{i:1\le i\le|\mathcal{H}| \wedge X_i = j} \mathcal{S}^i_{data}}{g_{ij}}\right) + \sum_{j\in\mathcal{M}} \sum_{k\in\mathcal{G}} \left(\frac{\sum_{j:1+|\mathcal{H}|\le j\le|\mathcal{H}|+|\mathcal{M}| \wedge X_j = k} c^j_{data}}{g_{jk}}\right)$$

The reformulated cost function and constraints for IoTs-CCNs and CCNs-BSs assignments are as follows:

$$\min_{X} \quad W_1 \times \phi' + W_2 \times \Phi' \tag{4.15}$$

subject to:

$$\sum_{\{i:1\leq i\leq|\mathcal{H}|\,\wedge\, X_i=j\}} \mathcal{S}_{cpu}^{i} \leq c_{cpu}^{j}, \forall j = 1, 2, 3, \ldots . |\mathcal{M}| \tag{4.16}$$

$$\sum_{\{i:1\leq i\leq|\mathcal{H}|\,\wedge\, X_i=j\}} \mathcal{S}_{mem}^{i} \leq c_{mem}^{j}, \forall j = 1, 2, 3 \ldots |\mathcal{M}| \tag{4.17}$$

The cost function in (4.15) minimizes the overall (i.e., data computation and transmission) power consumed in the IoT network. Weight parameters W_1 and W_2 are used to assign weights to computation and data transmission power in the cost function in (4.15). The first term in (4.15) computes the computational power for the CCNs in active/idle modes, while the second term computes the transmission power from IoTs-CCNs and CCNs-BSs, respectively. Constraints (4.16) – (4.17) ensure that the sum of the total CPU and memory demand of the IoTs assigned to a CCN does not exceed the total CPU and memory capacity of that CCN. Equations (4.6), (4.7), and (4.14) are used as computational power formulas for IoTs-CCNs and CCNs-BSs assignments.

4.5. Proposed evolutionary algorithms

IoTs-CCNs and CCNs-BSs assignment appears to be computationally challenging, and no polynomial-time algorithm is in sight to solve this type of problems. In this chapter, we propose relatively new swarm intelligence (SI)-based EAs. The SI-based EAs are population-based metaheuristics algorithms with features such as adaptation, randomness, communication, feedback, exploration, and exploitation [20], [22]. EAs use these features in their operations to evolve the population of candidate solutions.

In this section, we experiment with EAs for the IoTs-CCNs and CCNs-BSs assignments as formulated in (4.15) – (4.17). The EAs of our choice are discrete fireworks algorithm (DFWA), problem specific information-based DFWA (IDFWA), Hybrid of the

IDFWA, and low-complexity biogeography-based optimization (LC-BBO) algorithms. These algorithms are also used for VM placement in chapter 3.

4.5.1. Discrete fireworks algorithm

Exploitation and exploration are the basic features of search operation in any EA. In EAs, exploitation refers to using better solutions (i.e., solutions with a lower value) for thorough search in a small region of a search space, while exploration refers to exploring various promising regions in the whole search space. Originally, the operators of enhanced fireworks algorithm (EFWA) are designed [23] for continuous space optimization problems, and these operators cannot operate for discrete space problems without modifications. In the subsequent subsections, we modify the operators of the EFWA algorithm to operate on integer space optimization problems. Hereafter, the new algorithm (also discussed in chapter 3) is called discrete fireworks algorithm (DFWA). Like the EFWA [23], the DFWA has operators like the explosion operator, the mutation operator, the repair mechanism and the selection operation.

4.5.1.1. Explosion operator

The explosion operator in the DFWA generates sparks from a firework using offset displacement and two parameters: explosion strength and explosion amplitude.

A. Explosion strength

In the DFWA we adopt the same explosion strength formula that was used for the EFWA [22], [23]. The cost values of a firework and parameters determine the number of sparks that a firework can generate. Like the DFWA (in Chapter 3), the DFWA computes the number of sparks s_i for the i^{th} firework:

$$s_i = \text{round}\left(M_e \times \frac{\text{Y}_{max} - f(X^i) + \varepsilon}{\sum_{i=1}^{N}\left(\text{Y}_{max} - f(X^i)\right) + \varepsilon}\right), \text{ where } i = 1,2,\dots,N, \tag{4.18}$$

where s_i is the number of sparks from the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{max} is the maximum cost of N fireworks in the current algorithm generation, $f\left(X^i\right)$ represents the

cost of the i^{th} firework, M_e is a constant that controls the total number of sparks generated by N fireworks, and ε is a small constant used to avoid division by zero in (4.18).

B. Offset displacement

After computing the number of explosion sparks s_i for the i^{th} firework, where $i = 1,2,\dots,N$, the DFWA (as in Chapter 3) determines the offset displacements for the probabilistically selected component of the firework within the explosion amplitude.

$$\widetilde{X_q^i} = ceil(X_q^i + A_i \times rand(0,1)), \tag{4.19}$$

where $\widetilde{X_q^i}$ is the spark component after adding the displacement '$A_i \times rand(0,1)$' in the X_q^i component of the i^{th} firework, for each of $i = 1,2,\dots,N$. Pseudo code of the Algorithm 4.1 is run once to generate an explosion spark.

Algorithm 4.1: Generating explosion sparks in the DFWA

Inputs:

- $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$ *// a firework (a candidate solution)*

Algorithm parameters:

- *sparkProb:* spark probability [0,1] *// user determined explosion probability*
- *A:* Explosion amplitude *(see 4.5.1.1-C)*

Output:

- $\check{X}$, a *spark, a vector of* $|\mathcal{H}| + |\mathcal{M}|$ *components*

Steps:

1. ***for*** q = 1 to $|\mathcal{H}| + |\mathcal{M}|$ *// m is number of components in X*
2. ***if*** $rand()$ < *sparkProb*
3. Calculate the offset displacement: $\Delta X_q = A \times rand()$
4. $\widetilde{X_q} = ceil(X_q + \Delta X_q)$ *// perturbing the q^{th} component (see 4.5.1.1-B)*
5. ***end if***
6. ***end for***

C. Explosion amplitude

The explosion amplitude quantifies the range of the displacement that is used to perturb one or more components of a firework. In the DFWA (as in Chapter 3), the amplitude formula is modified to optimize discrete (integer) space:

$$A_i = \text{round}\left(\hat{a} \times \frac{f(X^i) - Y_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X^i) - Y_{min}) + \varepsilon}\right), \text{where } i = 1,2,\dots,N, \quad (4.20)$$

where A_i is the amplitude associated with the i^{th} firework (for each of $i = 1,2,\dots,N$), Y_{min} is the minimum cost among the N fireworks in the current algorithm generation, $f(X^i)$ represents the cost of the i^{th} firework, $\hat{a}$ is a constant used to control the amplitude, and ε is a small constant used to avoid division by zero in (4.20).

4.5.1.2. Mutation operator

We adopt a modified mutation operator for the DFWA (as in Chapter 3) that uses the random integer function *randi* for the mutation explosion. The DFWA selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of N fireworks to set up sparks by the mutation explosion, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. One spark is generated for each mutation firework $X^i \in \mathcal{Z}$ using the best firework among the N fireworks. The mutation explosion operator is represented as:

$$\widetilde{X_q^i} = X_q^i + (X_q^b - X_q^i) \times randi\,(X_q^{min}, X_q^{max}), \quad (4.21)$$

where $\widetilde{X_q^i}$ is the component of a newly generated spark and the X_q^b is the component of the best solution in the current algorithm generation. Note that X_q^i is the probabilistically selected component of $X^i \in \mathcal{Z}$ by the user-determined probability *mutateProb*, where $i = 1,2,\dots,N$; X_q^{min} and X_q^{max} are lower and upper bounds of the search space in dimension q. Pseudo code of the Algorithm 4.2 is run once to generate a mutation spark.

Algorithm 4.2: Generating Mutation sparks in the DFWA

Inputs:

- $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$ *// a firework (a candidate solution)*
- X^b: a vector of $|\mathcal{H}| + |\mathcal{M}|$ components. Note that X^b is the best solution amongst N fireworks.

Algorithm parameters:

- *mutateProb:* spark probability [0,1] *// user determined mutation probability.*

Output:

- $\check{X}$, a *spark, a vector of m components.*

Steps:

1. ***for*** q = 1 to $|\mathcal{H}| + |\mathcal{M}|$ *// $|\mathcal{H}| + |\mathcal{M}|$ is number of components in X*

2. ***if*** *rand () < mutateProb*
3. $\check{X}_q = X_q + \left(X_q^b - X_q\right) \times randi\left(X_q^{min}, X_q^{max}\right)$
   // perturbing the q[th] component (see 4.5.1.2)
   // note that *randi() returns integer between* X_q^{min} *and* X_q^{max}
4. ***end if***
5. ***end for***

4.5.1.3. Repair mechanism

Like any other evolutionary algorithm (EA), a candidate solution (e.g., firework, spark, or mutation spark) in the DFWA may violates one or more constraints during the operation and becomes infeasible solution. An infeasible solution is useless for further evolution in any EA. Our proposed IoTs-CCNs and CCNs-BSs assignment problem has rectangular and nonrectangular constraints. The proposed repair algorithm, either checks feasibility or repairs an infeasible candidate solution for the IoTs-CCNs and CCNs-BSs assignment problem.

A. Repair algorithm

The implementation details and pseudo code of the repair algorithm for the IoTs-CCNs and CCNs-BSs assignment is presented in the appendix of the chapter. In this section, pseudo code (in the Table 4-3) and repair algorithm are concisely discussed. The proposed repair algorithm checks the feasibility or repair the infeasible candidate solution, which is either randomly generated or evolved by the experimented EAs.

The system parameters, as defined in the *section 4.2*, and a candidate solution *X* to repair is input to the repair algorithm. The proposed IoT network comprises of two levels of resource assignments: between IoTs and CCNs, and between CCNs and BSs as discussed in the *section 4.2*. In the repair algorithm, candidate solution *X* splits into two vectors, $\dot{X} = (X_1, X_2, \ldots, X_{|\mathcal{H}|})$ and $\ddot{X} = (X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \ldots, X_{|\mathcal{H}|+|\mathcal{M}|})$. The repair algorithm checks feasibility or repairs the infeasible vectors $\dot{X}$ and $\ddot{X}$ separately and concatenates both $\dot{X}$ and $\ddot{X}$ vectors as an *X* vector to return as a feasible candidate solution.

Using vector $\dot{X}$, repair algorithm computes the load on CCNs in terms of CPU and Memory. Note that a CCN is considered overloaded, if current load of a CCN exceeds the

capacity of that CCN. The load on a CCN is the sum of IoTs' CPU and memory demands connected to that CCN. A candidate soliton X is considered infeasible, if one or more CCNs are overloaded in $\dot{X}$. In contrast, a CCN is considered underloaded in $\dot{X}$, if current load does not exceed the capacity of that CCN. If candidate solution X is infeasible, the repair algorithm checks overloaded information about each CCN and disconnects IoTs one by one from the overloaded CCNs in $\dot{X}$. A disconnected IoT from an overloaded CCN need to be reconnected to an underloaded CCN. The repair algorithm checks feasibility of an underloaded CCN before reconnecting a disconnected IoT to that CCN. In case this reconnection is feasible, the disconnected IoT is assigned to the underloaded CCN. The repair algorithm continues disconnecting IoT from the overloaded CCN until load on the overloaded CCN becomes less or equal to the capacity of that CCN. The repair algorithm runs to repair each overloaded CCN in the vector $\dot{X}$.

The proposed repair algorithm to repair $\dot{X}$ does not guarantee that each of the repairable (or infeasible) solutions will become feasible solution. The reason is that the repair algorithm is not checking each IoT connection to each CCN exhaustively. In other words, the repair algorithm only checks for the first available feasible connection between an IoT to a CCN to replace the infeasible connection. If a candidate solution is not repairable (or no feasible IoT to CCN connection is available), the repair algorithm randomly generates a new $\dot{X}$ and checks its feasibility.

After checking feasibility or repairing infeasible $\dot{X}$, the repair algorithm checks feasibility of the vector $\ddot{X}$. Note that indices of $\ddot{X}$ represent CCNs and values of components of $\ddot{X}$ are base stations (BSs) connected to the corresponding CCNs. The repair algorithm checks the operational/nonoperational status of CCNs in the $\dot{X}$. If a CCN is not serving any IoT in the vector $\dot{X}$, assign a '*0*' value to the corresponding CCN in $\ddot{X}$ (*see section 4.3.1*). Note that '*0*' value in $\ddot{X}$ means the corresponding CCN is not in use. On the other hand, if a CCN is serving IoT(s) in $\dot{X}$ and the corresponding CCN is a '*0*' value in $\ddot{X}$, then assign a BS randomly (from $k = 1,2, \dots, |\mathcal{G}|$) to the corresponding component in the vector $\ddot{X}$. Note that any nonzero value in $\ddot{X}$ means the corresponding CCN is in use. Finally, the repair

algorithm concatenates $\dot{X}$ and $\ddot{X}$ vectors to the vector X and returns as a feasible candidate solution X.

Table 4.3 Repair algorithm for infeasible solutions

A. Inputs	**Steps:** 1. (a) System parameters such as IoTs: CPU and memory demand, CCNs: CPU and memory capacity, etc. (b) Candidate solution X.
B. Execution	**Steps:** 2. Split candidate solution X into two vectors, $\dot{X} = (X_1, X_2, \dots, X_{\lvert\mathcal{H}\rvert})$ and $\ddot{X} = (X_{\lvert\mathcal{H}\rvert+1}, X_{\lvert\mathcal{H}\rvert+2}, \dots, X_{\lvert\mathcal{H}\rvert+\lvert\mathcal{M}\rvert})$ *// see section 4.3.1.* 3. Calculate load demand of all IoTs to the corresponding CCNs in $\dot{X}$. 4. Overloaded information for each CCN is checked in $\dot{X}$. 5. ***if*** ($\dot{X}$ is infeasible) 6. IoTs are disconnected one by one from the overloaded CCNs until overloaded CCNs become less or equal to its maximum capacity. 7. After checking feasible load on CCNs, each disconnected IoT is reconnected to the first available CCN. 8. Calculate load demand of all IoTs to the corresponding CCNs in $\dot{X}$. 9. Overloaded information for each CCN is checked in $\dot{X}$. 10. ***end if*** 11. ***while*** ($\dot{X}$ is infeasible) 12 Randomly generate a vector $\dot{X}$. 13. Repeat steps 3 to 10. 14. ***end while*** *// $\dot{X}$ is finally repaired* 15. ***if*** ($\ddot{X}$ is infeasible) *// Indices of $\ddot{X}$ represent CCNs* 16. Check the operational/nonoperational CCNs in $\dot{X}$. Assign a '*0*' value to the nonoperational CCN in $\ddot{X}$. 17. Replace a '*0*' value with a randomly selected BS for the operational CCN in $\ddot{X}$. *// see 4.3.1 for further clarification on X, $\dot{X}$, and $\ddot{X}$.* *// Note that nonzero value in $\ddot{X}$ means CCN is in operation* *// Note that '0' value in $\ddot{X}$ means CCN is nonoperational* 18. ***end if*** *// repaired $\ddot{X}$* 19. $X = \dot{X} + \ddot{X}$ *// Concatenate $\dot{X}$* and $\ddot{X}$
C. Output	20. ***return*** feasible solution X.

4.5.1.4. Selection operation

Each iteration of the DFWA (as in chapter 3) generates several sparks (i.e., candidate solutions) that are more than the population of the *N* fireworks. Therefore, after applying all the DFWA operators, a new population of the *N* fireworks need to be selected from the current group of candidate solutions. The DFWA algorithm adopts a random selection operator, which is laid down in the EFWA [23]. In DFWA, first, the solution with the minimum cost is selected, and then the $(N-1)$ candidate solutions are randomly selected from the remaining candidate solutions for the next algorithm iteration.

4.5.1.5. DFWA algorithm operation

The pseudo code for the DFWA algorithm is presented in Table (4-4). Initially, a population *F* of the *N* fireworks is generated randomly, and parameters are initialized. After computing the cost of the *N* fireworks using (4.15) − (4.17), the number of sparks s_i, and the amplitude values A_i, are computed using (4.18) and (4.20) for each firework, where $i = 1, 2, \dots N$. Now, s_i number of sparks are generated for each firework X^i in the population of *N* fireworks. For each spark, an offset displacement (4.19) is added in a probabilistically selected component of the firework X^i with user-determined '*sparkProb*' probability. All the sparks generated from the *N* fireworks are evaluated using the cost function (4.15).

Now, the DFWA selects a set, $\mathcal{Z}$, of fireworks to be mutated from population of the *N* fireworks to execute the exploration process, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework $X^i \in \mathcal{Z}$, the mutation operator (4.22) is used to generate mutation sparks with user-determined '*mutateProb*' probability. After executing the exploration process on the $|\mathcal{Z}|$ fireworks, the mutation sparks are also evaluated using the cost function (4.15). After performing the explosion operation and mutation operation for one EA generation, the DFWA selects a new population of the *N* fireworks. In the DFWA, first the solution with the lowest cost is selected for the next algorithm generation, then ($N-1$) fireworks are selected randomly from the remaining candidate solutions for the next EA generation.

Table 4.4 DFWA pseudo code

A. Initialization	1. Randomly generate a population of the N fireworks, $X^i, i = 1,2, \dots, N$ 2. Initialize the *sparkProb* and *mutateProb*. 3. Declare S as an empty set of sparks.
B. Execution	4. Check the feasibility of the N fireworks or repair the infeasible ones using the repair algorithm in Table 4-3 and evaluate using the cost function in (4.15). 5. ***while*** (stopping criteria not satisfied) 6. ***for*** $i = 1,2, \dots, N$ 7. Calculate the number of sparks s_i and the amplitude A_i for the i^{th} Firework X^i using (4.18) and (4.20), respectively. 8. ***for*** j = 1 to s_i 9. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 4.1**. 10. Add generated sparks in S 11. ***end for*** 12. ***end for*** 13. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 4.5.1.2*) from a population of N fireworks. 14. ***for*** each firework X in $\mathcal{Z}$ 15. Generate mutation spark $\check{X}$ using **Algorithm 4.2**. 16. Add generated spark in S. 17. ***end for*** 18. Check the feasibility of all the sparks in S or repair the infeasible ones using the repair algorithm in Table 4-3 and evaluate using the cost function in (4.15). 19. Select the best solution, and (N-1) solutions to make a new population of the N fireworks. 20. ***end while***
C. Output	21. ***return*** the best solution found so far.

4.5.2. Problem specific information-based DFWA

Generally, EAs are model-free and do not need any problem specific information [20]. However, incorporating problem specific information in EAs may improve the overall efficiency of EAs. In this subsection, we propose a DFWA algorithm that utilizes some domain knowledge of the IoTs-CCNs and CCNs-BSs assignments.

As discussed in *section 4.3.1*, we define IoTs-CCNs and CCNs-BSs assignments as a vector of nonnegative integers $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$, where $|\mathcal{H}|$ and $|\mathcal{M}|$ are the cardinalities of the sets $\mathcal{H}$ and $\mathcal{M}$. Note that in X, X_i (where $i =$

$1,2,\ldots,|\mathcal{H}|$) represents the i^{th} IoT connected to some CCN j, and X_j (where $j = |\mathcal{H}| + 1, |\mathcal{H}| + 2, \ldots, |\mathcal{H}| + |\mathcal{M}|$) represents the j^{th} CCN that can be connected to some BS k, where $k = 1,2,\ldots,|\mathcal{G}|$. In X, we represent BSs followed by CCNs as the consecutive order of positive integers. In the proposed IoT network, CCNs are battery powered, and their power utilization is considered sensitive to the life span of the IoT network. In contrast, BSs are main powered and do not affect the life span of the IoT network. In addition to partial data processing, CCNs make clusters and act as bridge between IoT nodes and BSs (sinks). Overall network life may be affected due to inefficient use of a CCN resources. Therefore, we are accessing domain knowledge of the IoT-CCN connections in X_i, where $i = 1,2,\ldots,|\mathcal{H}|$.

4.5.2.1. Domain-knowledge for IoTs assignments

Most computationally challenging problems have some type of domain knowledge that can be used in evolutionary algorithms for their optimization. However, there is no guarantee that useful information is accessible or that the information can be used in the evolutionary algorithm to solve an optimization problem. Some domain knowledge in the IoTs assignment problem is easily accessed. Note that we are accessing domain knowledge of the connections between IoTs and CCNs. In X, X_i is used to access the domain knowledge, where $i = 1,2,\ldots,|\mathcal{H}|$. In accordance with equation (4.5), any CCN that is in active mode spends computing power overhead e_{idle}^{j}. For example, a CCN, after being turned on, consumes 100 percent (i.e., e_{max}^{j}) of power if all its resources are utilized. The same CCN, after being turned on, consumes the e_{idle}^{j} (70 percent) overhead power even if none of its resources are utilized. The objective of the problem, as represented in the cost function in (4.15), is to minimize weighted sum of transmission and computational power consumption in the IoT network, and the overhead computation power e_{idle}^{j} in a CCN that is in active mode can be better utilized if that CCN's utilization is high. Thus, minimizing the transmission power in IoT network, efficient IoT to CCN, in general, will have tendency to reduce the number of CCNs in the active mode, while satisfying the IoT demand. The proposed problem specific information-based DFWA (IDFWA) algorithm takes advantage of such domain knowledge in assigning IoTs-CCNs assignments.

4.5.2.2. Obtaining domain-knowledge form IoTs assignments

In X (4.13), each component of X_i represents IoT-CCN assignment, where $i = 1,2,\dots,|\mathcal{H}|$. Each of the X_i components specify the CCN serving the corresponding IoT. Some useful information can be collected from the integer vector X_i by counting the number of IoTs served by each CCN. A CCN is likely to be efficiently utilized if it serves many IoTs subject to fulfilling the constraints (4.15) − (4.17). We collect such information from X_i and apply the domain knowledge to guide the exploitation operation in our IDFWA. In X_i, we considered the components with high frequency (CCNs serving many IoTs) as good components and the components with low frequency (CCNs serving fewer IoTs) as poor components, where $i = 1,2,\dots,|\mathcal{H}|$.

4.5.2.3. Incorporating domain knowledge in the DFWA algorithm

The main idea of IDFWA is to exploit this problem-specific information from X_i to not perturb good components of vector X_i (the CCNs that serve many IoTs), where $i = 1,2,\dots,|\mathcal{H}|$. To that end, in the IDFWA, we added two extra steps to the IDFWA in generating new sparks. In X (4.13), X_j represents the j^{th} CCN that can be connected to some BS k, where $j = |\mathcal{H}|+1, |\mathcal{H}|+2, \dots, |\mathcal{H}|+|\mathcal{M}|$. The offset displacement (4.19) is used to perturb the probabilistically selected components of the X_j within explosion amplitude A_p, where $p = 1,2,\dots,N$.

In the IDFWA, in choosing the components of an X_i for displacement operation in (4.19), we try to avoid having much overhead power e_{idle}^{j}, where $j = |\mathcal{H}|+1, |\mathcal{H}|+2, \dots, |\mathcal{H}|+|\mathcal{M}|$, rather than choosing those components randomly. To that end, we choose some number of CCNs that serve many IoTs and then perturb the assignment of the IoTs currently assigned to those CCNs. More specifically, we use some fraction, Δ, to determine the number of such CCNs to be chosen and choose a set, T, of $\Delta.|\mathcal{M}|$ CCNs that serve the smallest number of IoTs currently. Then, we perturb those IoTs that are currently assigned to the CCNs in the set T. Pseudo code of the Algorithm 4.3 is run once to generate an explosion spark.

Algorithm 4.3: Generating explosion sparks in the IDFWA

Inputs:

- $X = (X_1, X_2, \dots, X_{|\mathcal{H}|}, X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$ // *a firework (or a candidate solution)*

Algorithm parameters:

- *sparkProb:* spark probability [0,1] // *user determined explosion probability*
- *A:* Explosion amplitude *(see 4.5.1.1-C)*
- Δ*:* user-defined fraction //*to choose portion of the m components in* X.

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. Split X into two vectors, $\dot{X} = (X_1, X_2, \dots, X_{|\mathcal{H}|})$ and $\ddot{X} = (X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$
2. From $\dot{X}$, select a set T of *round* $(\Delta . |\mathcal{H}|)$ components *(see 4.5.2)*
3. ***for*** $m = 1, 2, \dots |\mathcal{H}| + |\mathcal{M}|$
4. ***if*** (*rand ()* < *sparkProb* ***AND*** $m \in T$)
5. Calculate the offset displacement: $\Delta X_q = A \times rand()$
6. $\overset{\frown}{X_q} = ceil(\dot{X}_q + \Delta X_q)$ // *perturbation of* q^{th} *component (see 4.5.2.3)*
7. ***end if***
8. ***if*** (*rand ()* < *sparkProb* ***AND*** $m > |\mathcal{H}|$)
9. Calculate the offset displacement: $\Delta X_q = A \times rand()$
10. $\overset{\frown}{\ddot{X}_q} = ceil(\ddot{X}_q + \Delta X_q)$ // *perturbation of* q^{th} *component (see 4.5.2.3)*
11. ***end if***
12. ***end for***
13. $\check{X} = \overset{\frown}{X_q} + \overset{\frown}{\ddot{X}_q}$. // *Concatenate* $\overset{\frown}{\ddot{X}_q}$ *with* $\overset{\frown}{\ddot{X}_q}$

A. Example of using domain knowledge for IoTs assignments

Let us consider an example. Suppose we have $|\mathcal{H}|$=10, $\mathcal{M}$ = { c_1,c_2, c_3}, X_i = ($X_1, X_2, \dots X_{10}$) = (1, 2, 2, 3, 2, 1, 3, 2, 1, 2) and the user-defined fraction Δ = 2/3, where i = 1,2, …,10. In this example, we have 2 ×(2/3) = 2, so two CCNs are considered that are currently serving the smallest number of IoTs. The two CCNs are c_1 and c_3 in this example. The set of IoTs served by c_1 and c_3 is T= {$\mathcal{S}_1$, $\mathcal{S}_4$, $\mathcal{S}_6$, $\mathcal{S}_7$, $\mathcal{S}_9$}. Now the offset displacements are added in those components of X_i that are associated with set of IoTs in T with the user-determined probability *sparkProb* to construct a new spark. Except for the incorporation of domain knowledge in the DFWA algorithm, the IDFWA algorithm operation is the same as that of the DFWA algorithm [20].

4.5.2.4. IDFWA algorithm operation

The pseudo code for the IDFWA algorithm is presented in Table (4-5). Initially, a population F of N fireworks is generated randomly, and algorithm parameters are initialized. After computing the objective function values of N fireworks using (4.15) – (4.17), the number of sparks s_p and the amplitudes A_p are computed using (4.18) and (4.20), respectively, for each firework, where $p = 1,2, \dots, N$. Now, s_p sparks are generated for each of the N firework. Note that we split each of the firework X^p, where $p = 1,2, \dots, N$ into two vectors X_i, where $i = 1,2, \dots, |\mathcal{H}|$ and X_j, where $j = |\mathcal{H}| + 1, |\mathcal{H}| + 2, \dots, |\mathcal{H}| + |\mathcal{M}|$ to generate sparks. The offset displacement (4.18) is added to the set of T components of X_i, where $i = 1,2, \dots, |\mathcal{H}|$, and all components of X_j, where $j = |\mathcal{H}| + 1, |\mathcal{H}| + 2, \dots, |\mathcal{H}| + |\mathcal{M}|$, with user-determined *sparkProb* probability. The set of T components of a firework is determined by using the domain-knowledge of IoTs-CCNs assignments. This process is a local search and is also called "exploitation". In the context of the fireworks algorithm, each firework is perturbed probabilistically by adding an offset displacement within amplitude A_p to generate sparks around that firework. This controlled perturbation (by selecting T components) exploits a small region around a firework, and a thorough search is conducted over this small region by generating sparks. All the sparks generated from the N fireworks are evaluated using the cost function (4.15).

Now, IDFWA selects a set $\mathcal{Z}$ of fireworks randomly to be mutated from the population of N fireworks to execute the exploration, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of the set $\mathcal{Z}$. For each firework $X^p \in \mathcal{Z}$, where $p = 1,2, \dots, N$, the mutation operator (4.21) is used to generate one mutation spark with user-determined *mutateProb* probability. After executing the exploration process on the set of $\mathcal{Z}$ fireworks, the mutation sparks are evaluated using the cost function (4.15).

In one IDFWA generation, the total number of candidate solutions h include fireworks, explosion sparks, and mutation sparks, where $h > N$. For the next algorithm generation, we need to select a population of N fireworks from h candidate solutions. In the IDFWA, first the solution with the minimum cost is selected, then ($N-1$) fireworks are

selected randomly from the remaining candidate solutions for the next algorithm generation.

Table 4.5 IDFWA pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^p, p = 1,2,\dots,N$ 2. Initialize the *sparkProb*, *mutateProb*, and Δ 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table (4-3) and evaluate using the cost function in (4.15). 5. ***while*** (stopping criteria not satisfied) 6. ***for*** $p = 1,2,\dots,N$ 7. Calculate the number of sparks s_p and the amplitude A_p for the p^{th} Firework X^p using (4.18) and (4.20), respectively. 8. ***for*** k = 1 to s_p 9. Generate k^{th} explosion spark $\widetilde{X^k}$ using **Algorithm 4.3**. 10. Accumulate sparks in the set *S*. 11. ***end for*** 12. ***end for*** 13. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 4.5.1.2*) from a population of *N* fireworks. 14. ***for*** each firework X in $\mathcal{Z}$ 15. Generate mutation spark $\check{X}$ using **Algorithm 4.2**. 16. Accumulate spark in the set *S*. 17. ***end for*** 18. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table (4-3) and evaluate using the cost function in (4.15). 19. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 20. ***end while***
C. Output	21. ***return*** the best solution found so far.

4.5.3. Hybrid IDFWA/LC-BBO algorithm

IDFWA is presented in the *section 4.4.2* and the low-complexity BBO (LC-BBO) algorithm is discussed in the *section 2.2.3.1* of chapter 2. In next section, the operation of the hybrid LC-BBO algorithm and the IDFWA (Hybrid IDFWA/LC-BBO) for the resource assignments problem in the IoT network is presented.

4.5.3.1. Hybrid IDFWA/LC-BBO algorithm operation

The pseudo code for the Hybrid IDFWA/LC-BBO algorithm is presented in Table (4-6). Initially, a population of *N* fireworks is generated randomly, and algorithm parameters are initialized. After computing the cost function for *N* fireworks using (4.15) – (4.17), the number of sparks s_p and amplitudes A_p are computed using (4.18) and (4.20), respectively, for each p^{th} firework, where $p = 1,2, \dots, N$. In the Hybrid IDFWA/LC-BBO algorithm, either the migration procedure of the LC-BBO algorithm or the explosion procedure of the IDFWA is selected with user-determined probability θ to generate spark(s) for each firework. If the LC-BBO algorithm migration procedure [74] is selected as an exploitation process, emigrating solution $\widetilde{X^q}$ is selected from the population of *N* fireworks. The possibility of immigrating a feature from $\widetilde{X_m^q}$ to X_m^p, where $m = 1, 2, \dots |\mathcal{H}| + |\mathcal{M}|$, is decided using immigrating probability λ. Alternately, if the explosion procedure of the IDFWA is selected as an exploitation process with user-determined probability θ, s_p sparks are generated for the firework, where $p = 1,2, \dots, N$. Note that the firework X^p splits into two vectors X_i, where $i = 1,2, \dots, |\mathcal{H}|$, to represent the connections between IoTs and CCNs and X_j, where $j = |\mathcal{H}| + 1, |\mathcal{H}| + 2, \dots, |\mathcal{H}| + |\mathcal{M}|$, to represent the connections between the CCNs and BSs. The set of *T* components of a firework is determined by using the domain knowledge in IoTs-CCNs connections in X_i, where $i = 1,2, \dots, |\mathcal{H}|$. The offset displacement (4.18) is added to the set of *T* components of X_i and all components of X_j with user-determined *sparkProb* probability. In a generation of the IDFWA/LC-BBO algorithm, the total number of candidate solutions *h* includes fireworks, explosion sparks, islands/habitats, and mutation sparks from the IDFWA. All the sparks/islands generated from the *N* fireworks are evaluated using the cost function (4.15).

After one algorithm generation, the IDFWA/LC-BBO algorithm (like the DFWA in chapter 3) selects a new population of *N* fireworks from the total number of *h* candidate solutions. In the IDFWA/LC-BBO algorithm, first the solution with the best fitness is selected, then (*N-1*) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation.

Table 4.6 Hybrid IDFWA/LC-BBO pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^p, p = 1,2, \dots, N$ 2. Initialize the *sparkProb*, *mutateProb*, and Δ 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table (4.3) and evaluate using the cost function in (4.15). 5. ***while*** (stopping criteria is not satisfied) 6. ***for*** $p = 1,2, \dots, N$ 7. ***if*** *rand ()* $< \theta$ 8. Use LC-BBO algorithm in **Algorithm 2.5**. 9. Accumulate the islands in the set *S*. 10. ***else*** 11. Calculate the number of sparks s_p and the amplitude A_p for the p^{th} Firework X^p using (4.18) and (4.20), respectively. 12. ***for*** $j = 1$ to s_p 13. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 4.3**. 14. Accumulate sparks in the set *S* 15. ***end for*** 16. ***end if*** 17. ***end for*** 18. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 4.5.1.2*) from a population of *N* fireworks. 19. ***for*** each firework X in $\mathcal{Z}$ 20. Generate mutation spark $\check{X}$ using **Algorithm 4.2**. 21. Accumulate spark in the *S*. 22. ***end for*** 23. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table (4.3) and evaluate using the cost function in (4.15). 24. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 25. ***end while***
C. Output	26. ***return*** the best solution found so far.

4.6. Results and discussion

In computational experiments to assign IoTs-CCNs and CCNs-BSs, the set of sensor nodes (IoTs) $\mathcal{H}$ was 20, 50, 100, and 200. Following [8], we randomly generate various test problems with different computing resources for the different types of IoTs

(e.g., $\mathcal{S}_{cpu}^{j}, \mathcal{S}_{mem}^{j}, \mathcal{S}_{data}^{i}$) that are assigned to CCNs (e.g.,c_{cpu}^{j}, c_{mem}^{j}, c_{data}^{j}). For the purpose of experimentation, the capacity of CCN c_{cpu}^{j} and the demands of IoT $\mathcal{S}_{cpu}^{j}$ are randomly generated within the intervals [100−3000] and [1−2000], respectively. We scaled up the computational capacity of CCNs, $round\ (\frac{SNs}{CCNs}) \times c_{cpu}^{j}$, to ensure that there are enough computing resources available for the IoTs with the change in problem size. The memory demand of IoTs and the capacity of CCNs are randomly generated in the same way as those of the CPU.

4.6.1. Simulation parameters for the experimental algorithms

In Table (4-7), we summarize algorithm specific parameters that are used for the experiments in this chapter. The population of candidate solutions (or individuals) is a common parameter for all the experimental evolutionary algorithms (EAs).

In the proposed DFWA, IDFWA, and Hybrid IDFWA/LC-BBO algorithm, the fireworks are considered as a population. The explosion and mutation probabilities are set to 0.5 in the DFWA and the IDFWA. In the Hybrid IDFWA/LC-BBO algorithm, one of the migration or explosion operations can be selected probabilistically with a user-determined probability of $\theta = 0.5$. Both in the IDFWA and the Hybrid IDFWA/LC-BBO algorithm, we use a user-defined fractional parameter Δ (e.g., $\Delta = 1/2$) to select a set, *T*, of components that were perturbed to generate new sparks. In this work, we use the classic GA to solve the IoTs-CCNs and CCNs-BSs assignments in which probability of crossover, selection, and mutation are set to 0.9, 0.5, and 0.01, respectively.

4.6.2. Performance

We experimentally compare the performance of the low-complexity BBO (LC-BBO) algorithm, the DFWA, the IDFWA, the Hybrid IDFWA/LC-BBO algorithm, the Discrete ABC (DABC) algorithm, and the GA against the first fit decreasing (FFD) algorithm. We use the FFD algorithm as a benchmark to differentiate the performance of different algorithms in IoTs-CCNs and CCNs-BSs assignments as defined in (4.15)–(4.17).

In the FFD algorithm, we sort the IoTs in decreasing order of their CPU demand and assign the IoTs one by one to the CCNs (in the given order). The FFD algorithm is a low-complexity heuristic that is often used as a benchmark. However, the FFD algorithm may overlook many potential solutions due to the oversimplification of ordering the IoTs in one dimension. The number of objective function evaluations is the stopping criteria for the experimented algorithms as mentioned in the 4th column of the Tables 4.8 and 4.9.

Table 4.7 Algorithm parameters

Algorithms	**Algorithm specific parameters**	**Common parameters**
Discrete ABC	t = 1.2×Population size	Population size: 30
Low-complexity BBO	λ is defined as in [44] Emigrating method is taken from [44] Probability of mutation = 0.01	
GA	Probability of crossover = 0.9 Probability of selection = 0.5 Probability of mutation = 0.01	
Hybrid IDFWA/LC-BBO	λ is defined in chapter 2 Emigrating method is in chapter 2 mutationProb = sparkProb = 0.5 Migration probability θ = 0.5 Least frequent CCNs indices Δ = 1/2	# of Fireworks:10 # of mutation Fireworks: 5
IDFWA and DFWA	mutationProb = sparkProb = 0.5 Least frequent CCNs indices Δ = 1/2	

We divided our experiments into four groups based on the number of IoTs and BSs. In total, 20 assignments problem instances (i.e., five instances for each group) are tested with the LC-BBO algorithm, the DFWA, the IDFWA, the Hybrid IDFWA/LC-BBO algorithm, the DABC algorithm, and the GA. The results presented in this chapter are the average of 100 independent trails of each problem instance. The number of IoTs in each group of (assignments problem) instances are: 20, 50, 100, and 200, that is, 10 times the number of BSs. Each group has five variations of CCN as: $round\left(\frac{Number\ of\ IoTs}{i}\right)$, where i=1, 2, …5. In the IoT network, cost—data transmission and computational power as defined in (4.15)—is minimized using the experimental algorithms. We used four metrics to record the results of experiments in this chapter: "average power consumed," "standard deviation (Std.)," "percentage of power saved," and "average CPU time" (sec.). The "percentage of power saved" in IoT network using a proposed algorithm is calculated

against the FFD algorithm. We computed the percentage of overall power saved against the FFD algorithm for each of the experimental EAs using the formula:

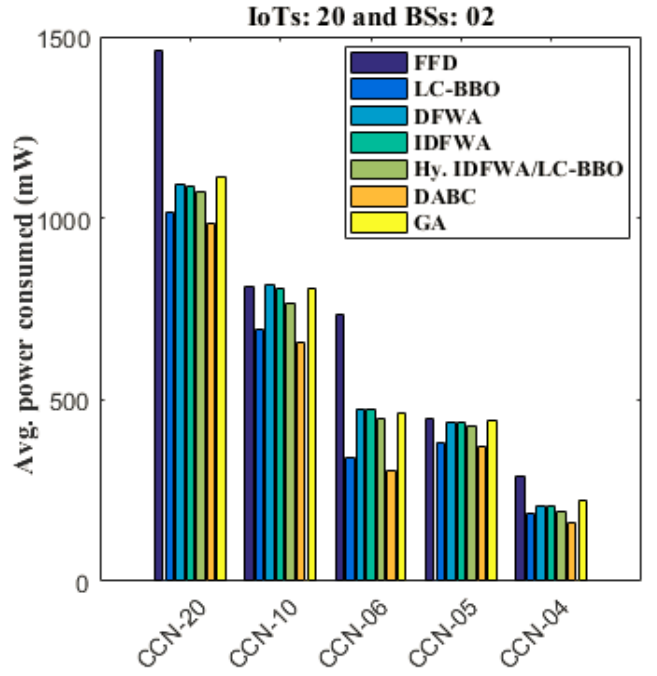
$$\left(1 - \frac{Avrage\ (power\ consumed\ using\ an\ experimented\ algorithm)}{Power\ cosumed\ by\ the\ FFD}\right) \times 100. \qquad (4.22)$$

Table 4.8 Simulation results (LC-BBO, DFWA, and IDFWA)

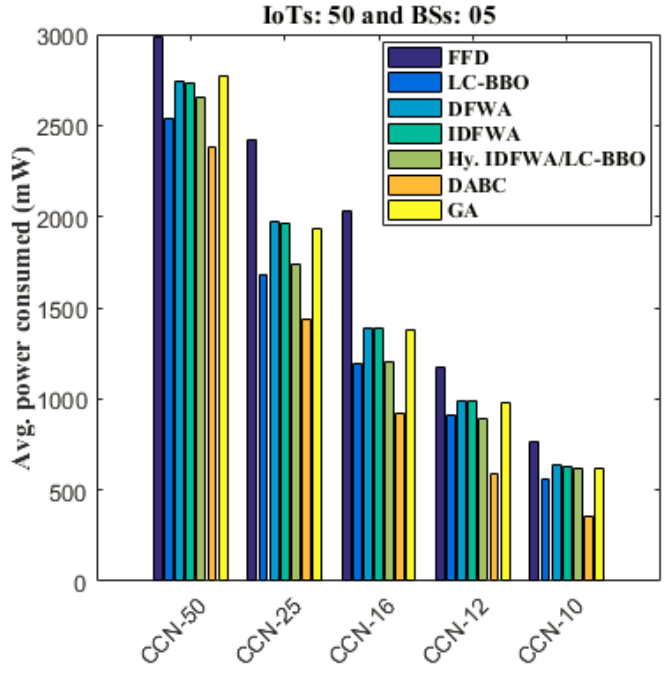
# of IoTs	# of CCNs	# of BSs	Max. # of function evaluations	Power consumed (milli watt) by FFD	Low-complexity BBO			DFWA			IDFWA		
					Avg. power consumed (milli watt) by LC-BBO (Std.)	Percentage of saved by LC-BBO (%)	Avg. Matlab CPU Time by LC-BBO (Sec.)	Avg. power consumed (milli watt) by DFWA (Std.)	Percentage of saved by DFWA (%)	Avg. Matlab CPU Time by DFWA (Sec.)	Avg. power consumed (milli watt) by IDFWA (Std.)	Percentage of saved by IDFWA (%)	Avg. Matlab CPU Time by IDFWA (Sec.)
20	20	02	8000	1459.12	1013.90(19.1)	30.51	4.94	1092.13(19.3)	25.15	7.74	1087.17(22.8)	25.49	8.26
20	10			813.01	692.27(27.9)	14.85	4.14	817.76(19.5)	--	5.60	803.92(21.6)	1.12	6.16
20	6			733.57	340.60(31.6)	53.57	3.43	475.20(20.8)	35.22	4.61	475.22(21.3)	35.22	5.01
20	5			446.09	383.29(16.8)	14.08	2.75	437.27(11.6)	1.98	3.37	435.20(16.0)	2.44	3.60
20	4			288.64	187.02(12.5)	35.21	2.32	206.43(11.1)	28.48	2.58	206.02(11.3)	28.62	2.87
50	50	05	12000	2986.49	2539.49(47.6)	14.97	25.09	2744.77(44.9)	8.09	43.84	2734.37(41.9)	8.44	45.49
50	25			2422.89	1682.18(65.6)	30.57	22.87	1974.47(35.3)	18.51	29.40	1961.85(43.0)	19.03	30.64
50	16			2030.18	1191.25(73.5)	41.32	17.92	1391.69(31.4)	31.45	21.50	1391.38(33.6)	31.47	22.36
50	12			1178.07	914.11(41.2)	22.41	10.45	990.61(25.0)	15.91	10.99	990.79(31.3)	15.90	11.34
50	10			768.64	557.03(31.2)	27.53	10.27	636.15(24.3)	17.24	9.87	631.35(21.9)	17.86	10.78
100	100	10	18000	7062.74	5318.43(118.6)	24.70	176.48	5514.42(62.7)	21.92	246.81	5510.08(41.1)	21.98	243.16
100	50			5395.00	4482.02(81.0)	16.92	96.68	4534.10(60.4)	15.96	115.14	4531.19(62.1)	16.01	111.99
100	33			3228.61	3193.22(61.0)	1.10	52.35	3197.34(48.8)	0.97	58.43	3201.86(56.0)	0.83	53.97
100	25			2956.69	2060.27(49.2)	30.32	29.77	2073.52(43.6)	29.87	34.85	2074.26(53.1)	29.85	33.69
100	20			1301.94	1184.35(42.7)	9.03	26.13	1202.50(35.1)	7.64	27.85	1190.20(34.7)	8.58	28.69
200	200	20	20000	13350.89	11407.29(143.7)	14.56	1104.58	11800.49(130.6)	11.61	1614.41	11769.01(128.9)	11.85	1631.60
200	100			9534.94	8678.76(147.1)	8.98	286.62	8806.95(109.9)	7.64	371.93	8757.03(125.6)	8.16	380.26
200	66			6718.05	6541.10(102.7)	2.63	250.46	6552.99(84.1)	2.46	298.73	6533.64(108.5)	2.74	299.33
200	50			5501.75	5002.17(84.4)	9.08	195.23	5012.30(88.7)	8.90	245.33	4998.57(99.5)	9.15	251.78
200	40			3655.04	3141.59(80.3)	14.05	101.02	3150.87(81.1)	13.79	112.20	3127.10(93.3)	14.44	106.26

Table 4.9 Simulation results (Hybrid IDFWA/LC-BBO, Discrete ABC, and GA)

# of IoTs	# of CCNs	# of BSs	Max. # of function evaluations	Power consumed (watt) by FFD	Hybrid IDFWA/LC-BBO			Discrete ABC			GA		
					Avg. power consumed (watt) by Hy. IDFWA/LC-BBO (Std.)	Percentage of saved by Hy. IDFWA/BBO (%)	Avg. Matlab CPU Time by IDFWA/LC-BBO (Sec.)	Avg. power consumed (watt) by Dis. ABC (Std.)	Percentage of saved by Dis. ABC (%)	Avg. Matlab CPU Time by Dis. ABC (Sec.)	Avg. power consumed (watt) by GA (Std.)	Percentage of saved by GA (%)	Avg. Matlab CPU Time by GA (Sec.)
20	20	02	8000	1459.12	1072.66(17.9)	26.49	11.92	983.61(11.5)	32.59	6.12	1112.65(30.4)	23.74	3.52
20	10			813.01	766.02(17.2)	5.78	8.67	657.18(6.9)	19.17	5.13	806.83(27.6)	0.76	3.04
20	6			733.57	446.39(21.0)	39.15	7.12	304.95(7.8)	53.43	5.26	462.14(30.6)	37.00	2.50
20	5			446.09	425.45(12.3)	4.63	5.28	370.78(9.2)	16.88	4.31	441.89(23.4)	0.94	1.91
50	50	05	12000	2986.49	2649.91(41.5)	11.27	61.46	2381.17(23.7)	20.27	14.64	2769.57(57.5)	7.26	10.81
50	25			2422.89	1739.39(46.5)	28.21	44.38	1436.17(27.6)	40.72	14.07	1936.76(64.0)	20.06	11.94
50	16			2030.18	1201.00(61.7)	40.84	32.39	921.50(28.2)	54.61	12.20	1373.72(57.5)	32.34	10.00
50	12			1178.07	890.87(50.3)	24.38	17.27	587.21(24.7)	50.15	8.55	983.04(47.1)	16.55	6.04
50	10			768.64	618.70(27.8)	19.51	15.99	357.79(21.9)	53.45	9.81	622.62(32.5)	19.00	6.94
100	100	10	18000	7062.74	5372.66(68.3)	23.93	342.11	4543.08(41.3)	35.68	47.01	5493.74(93.4)	22.22	43.18
100	50			5395.00	4406.68(102.2)	18.32	164.50	3495.45(55.6)	35.21	28.48	4472.97(100.4)	17.09	27.58
100	33			3228.61	3158.15(98.7)	2.18	82.92	2075.90(47.5)	35.70	20.90	3146.57(95.4)	2.54	19.77
100	25			2956.69	2044.70(48.7)	30.84	49.73	1293.07(36.8)	56.27	16.93	2051.42(72.2)	30.62	15.09
100	20			1301.94	1176.24(38.0)	9.66	42.28	625.23(25.5)	51.98	15.88	1179.01(51.2)	9.44	13.88
200	200	20	20000	13350.89	11303.02(160.3)	15.34	2176.98	9659.64(66.4)	27.65	211.98	11769.94(198.4)	11.84	220.52
200	100			7964.88	8123.91(321.5)	14.80	548.21	6447.09(97.0)	32.38	61.67	8587.22(189.6)	9.94	66.66
200	66			6718.05	6329.01(241.8)	5.79	462.35	4577.39(75.2)	31.86	60.03	6390.50(152.4)	4.88	70.43
200	50			5501.75	4989.41(74.1)	9.31	365.74	3334.81(65.5)	39.39	55.65	4914.31(142.7)	10.68	61.64
200	40			3655.04	3095.80(83.3)	15.30	166.26	1652.13(67.2)	54.80	36.33	3039.43(118.8)	16.84	37.72

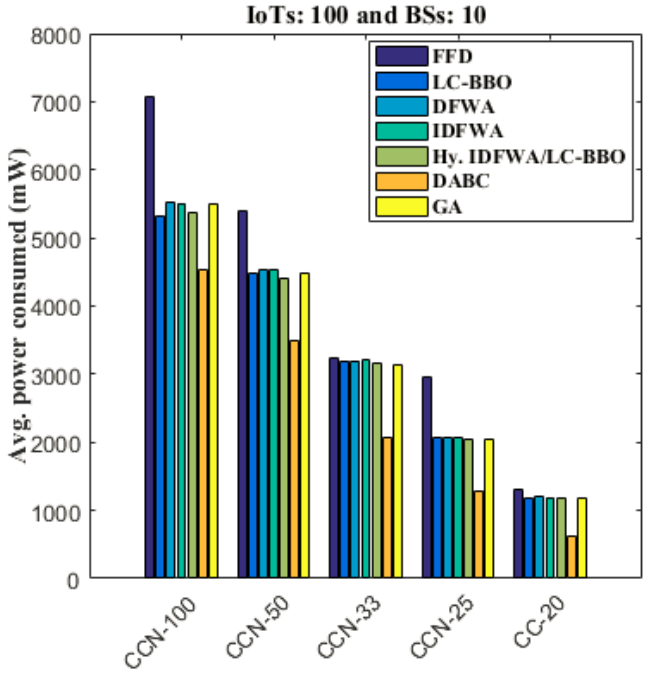


(a)

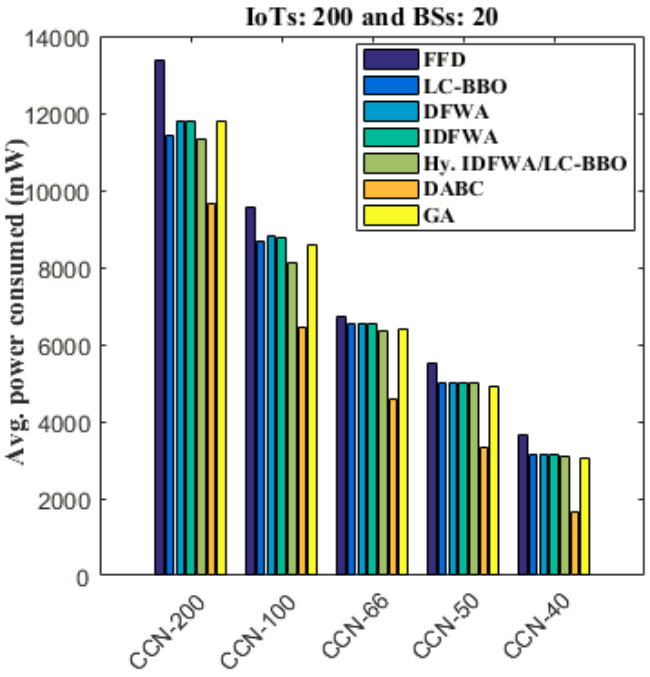


(b)

Figure 4.2 Average power consumed for 20 and 50 IoTs.

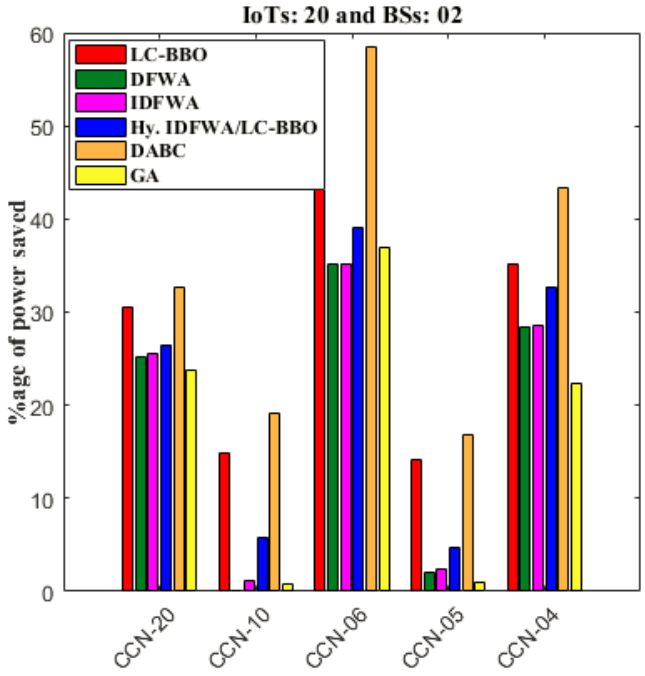


(a)

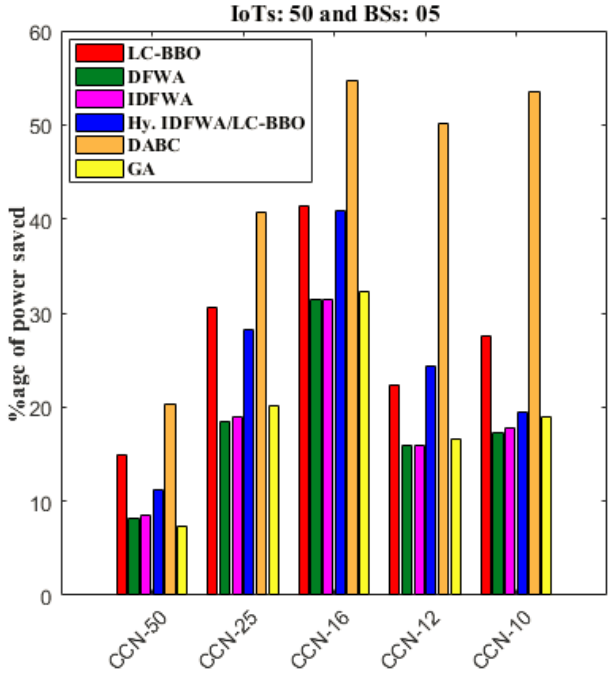


(b)

Figure 4.3 Average power consumed for 100 and 200 IoTs.

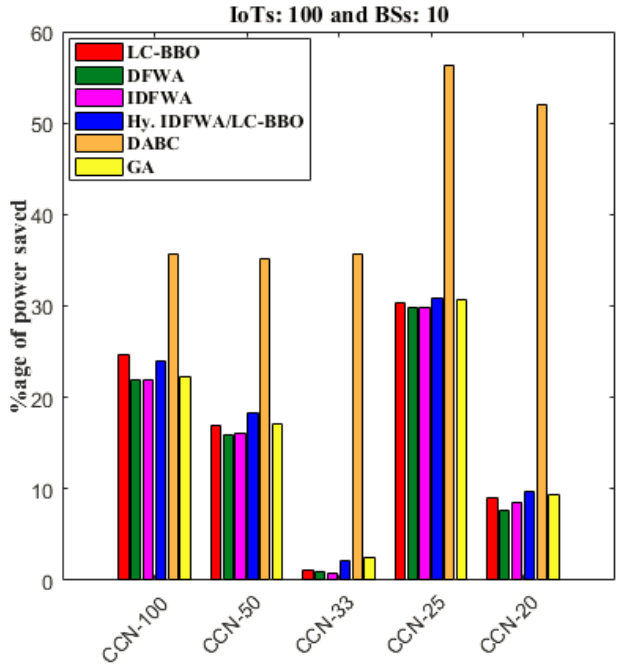


(a)

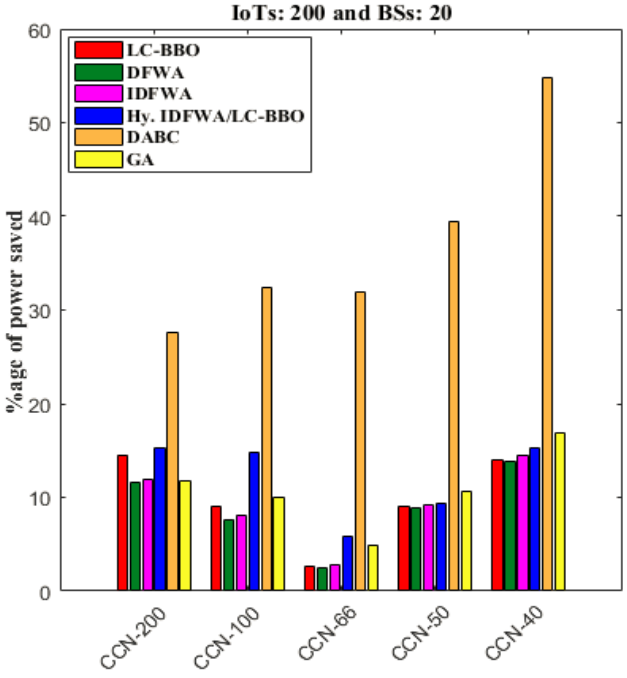


(b)

Figure 4.4 Percentage of power saved by 20 and 50 IoTs.

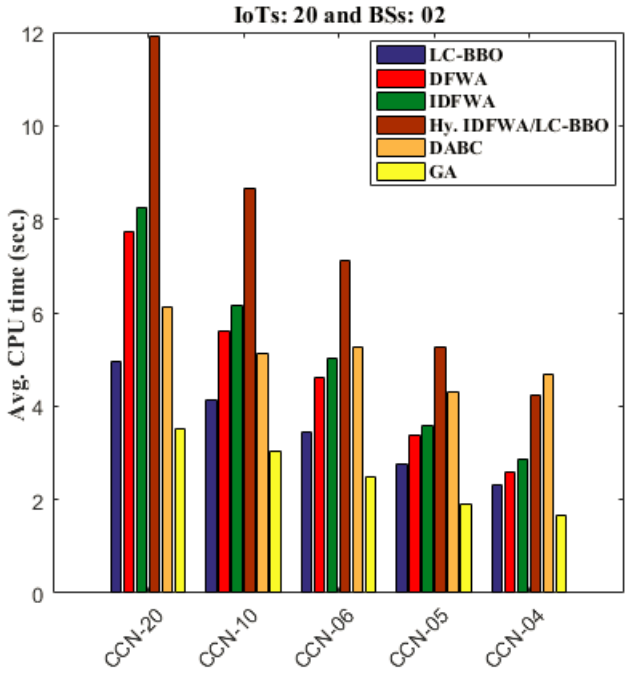


(a)

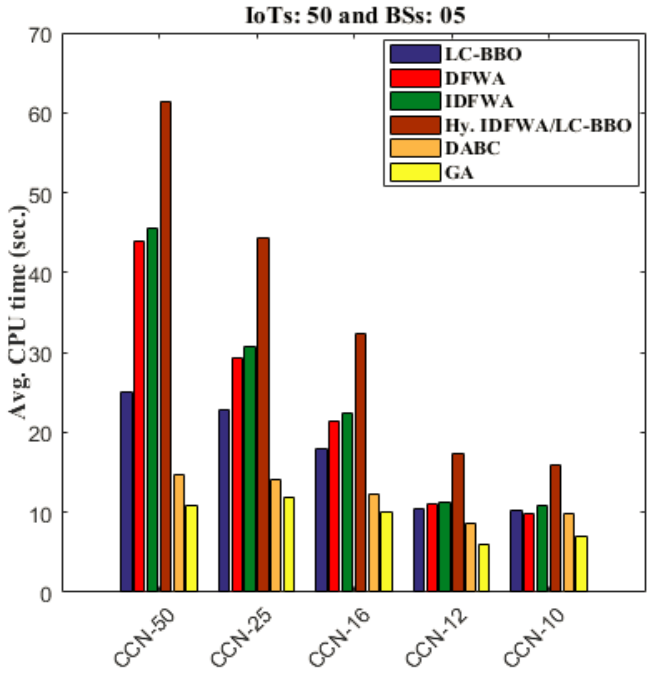


(b)

Figure 4.5 Percentage of power saved by 100 and 200 IoTs.

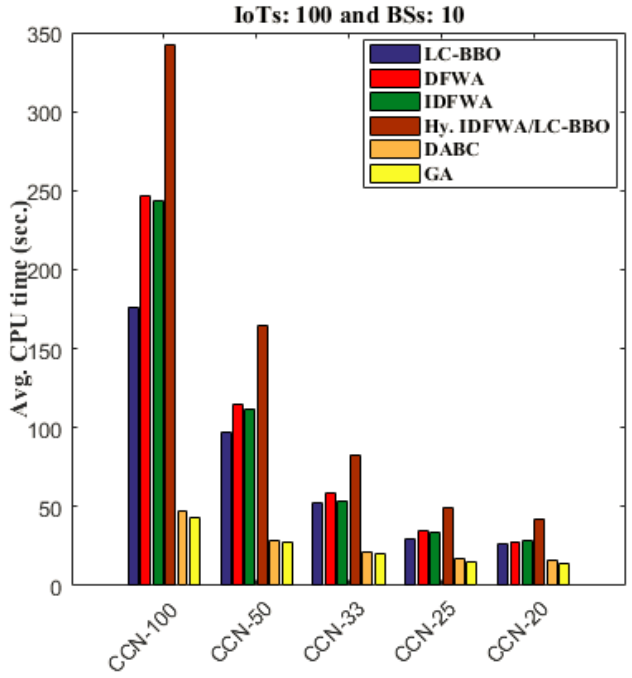


(a)

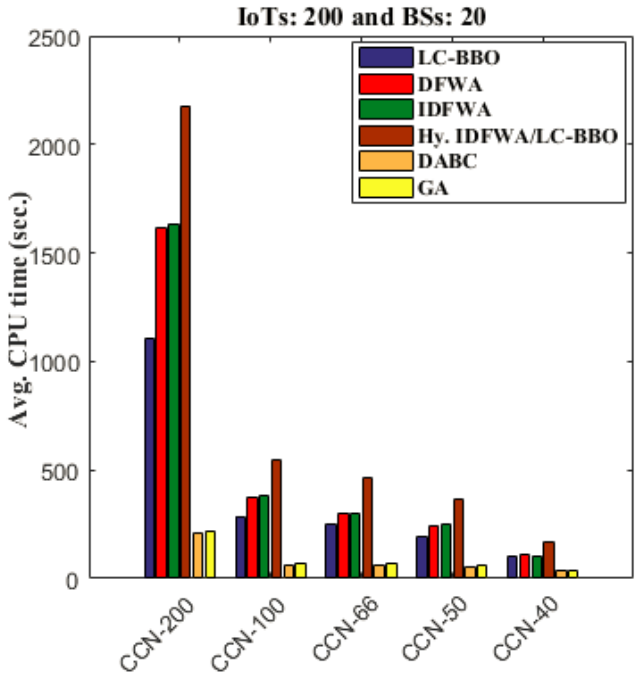


(b)

Figure 4.6 Avg. Matlab CPU time (sec.) consumed by 20 and 50 IoTs.

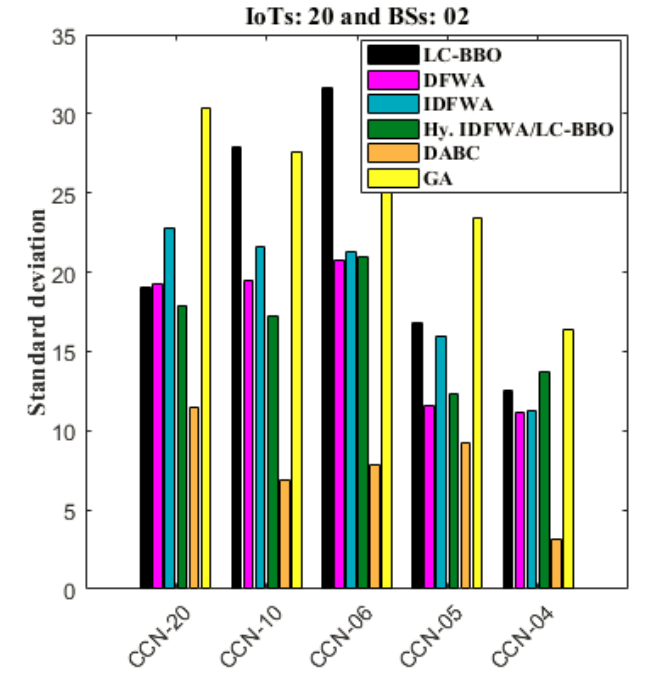


(a)

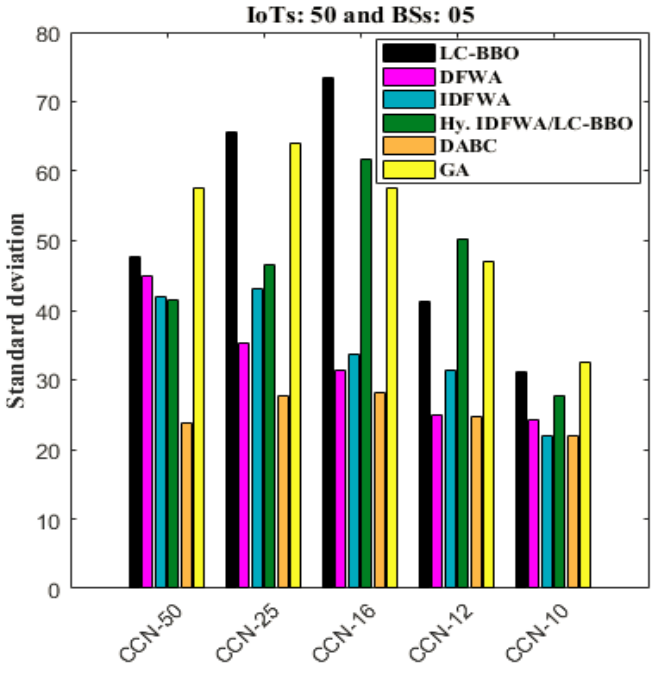


(b)

Figure 4.7 Avg. Matlab CPU time (sec.) consumed by 100 and 200 IoTs.

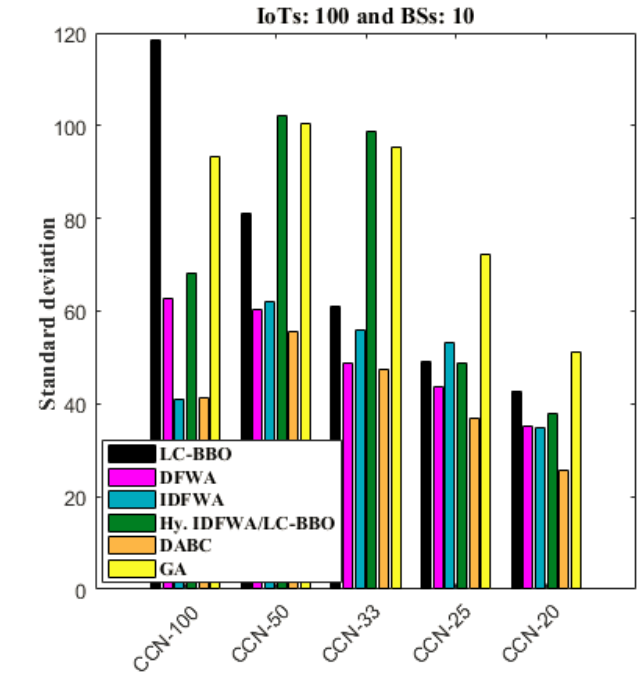


(a)

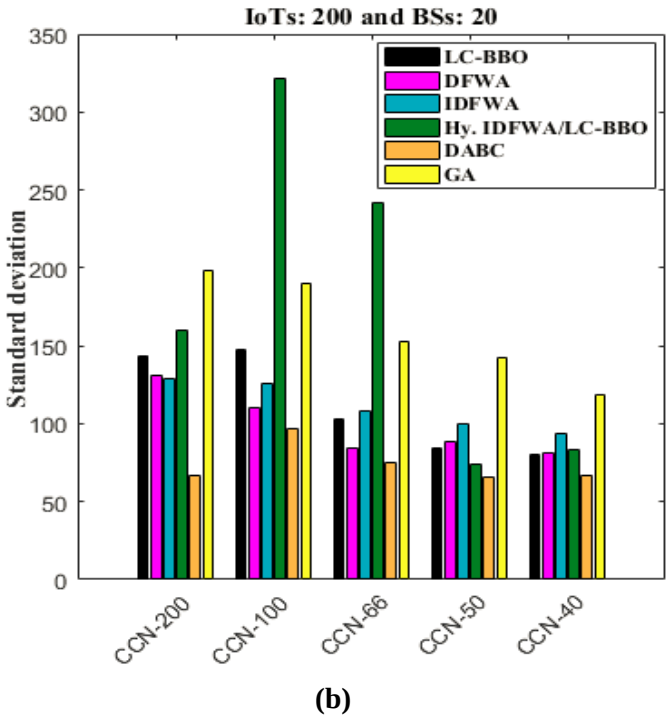


(b)

Figure 4.8 Standard deviation for 20 and 50 IoTs.



(a)



(b)

Figure 4.9 Standard deviation for 100 and 200 IoTs.

The corresponding data for the metrics “average power consumed,” “standard deviation (Std.),” “percentage of power saved,” and “average CPU time (sec.)” are recorded in Tables (4-8)−(4-9). In Figures 4.2 (a−b) and 4.3 (a−b), we plot the results for

the four group of (assignments problem) instances. In each subfigure, we show the average power consumed in Figures 4.2−4.3 and the percentage of the saved power against the FFD algorithm in Figures 4.4−4.5 for all four CCN variations. The average CPU time (sec.) is plotted in Figures 4.6−4.7 and the standard deviation of cost is shown in Figures 4.8−4.9.

For the sake of exposition, we refer to each IoTs-CCNs and CCNs-BSs (assignments problem) instance in Tables (4-8)−(4-9) with three numbers (Number of IoTs, Number of CCNs, and Number of BSs)—for example, the first instance is (20, 20, 02).

In Figures 4.2−4.3, the general trend is an increase in power consumption as the number of CCNs becomes larger. This trend is almost same for all the tested (assignments problem) instances and algorithms. Ranking the algorithms based on their power consumption performance indicates that power consumption is similar and consistent for all the assignments. In all the IoTs-CCNs and CCNs-BSs instances, the discrete artificial bee colony (DABC) algorithm had the highest performance, followed by the LC-BBO algorithm, the Hybrid IDFWA/LC-BBO algorithm, the GA, the IDFWA, the DFWA, and the FFD algorithm in terms of average power consumption. The FFD algorithm and the GA performed poorly compared to the DABC algorithm, the LC-BBO algorithm, the Hybrid IDFWA/LC-BBO algorithm, the IDFWA and the DFWA. Except for the DABC algorithm, LC-BBO algorithm has the highest performance when compared against the Hybrid IDFWA/LC-BBO algorithm, the IDFWA, the DFWA, and the GA, while the FFD algorithm has the lowest performance. The FFD algorithm, the DFWA, and the IDFWA perform poorly compared to the DABC algorithm, the Hybrid IDFWA/LC-BBO algorithm, the LC-BBO algorithm, and the GA, especially when the number of IoTs becomes large (i.e., ≥ 100) as in Figure 4.3 (a-b). This clearly shows how the DABC and the Hybrid IDFWA/LC-BBO algorithms improve the performance in terms of power consumption, especially for large IoT networks (i.e. ≥ 100), whereas the other algorithms have relatively poor performance. This also shows that the experimental algorithms are scalable and applicable to real-world large IoT networks.

We depict the percentage of power consumption saved by all the other algorithms with respect to the FFD algorithm in Figures 4.4−4.5. As a highest performer, the DABC

algorithm saves power ranging from 17% as in (20, 05, 02) to 56% as in (100, 25, 10). For a smaller number of IoTs (i.e., ≤ 50), the LC-BBO algorithm saves power ranging from 15% as in (50, 50, 05) to 54% as in (20, 06, 02). For a larger number of IoTs (i.e., ≥ 100), the Hybrid IDFWA/LC-BBO algorithm saves power ranging from 6% as in (200, 66, 20) to 31% as in (100, 33, 10). The DFWA and the IDFWA show comparable performance in power saving, and the GA outperforms both the DFWA and the IDFWA.

Figures 4.6 and 4.7 indicate that CPU time increases when the number of IoTs increases for all the algorithms, but the CPU time increase is much more sensitive to IoT increase for the Hybrid IDFWA/LC-BBO algorithm. For ≤ 100 IoT assignments, the GA performs better than the other algorithms in terms of consuming CPU time. However, for ≥ 100 IoT assignments, the DABC algorithm is the fastest, followed by the LC-BBO algorithm, the DFWA, and the IDFWA.

The standard deviations (of cost) of all algorithms with respect to multiple IoT assignments are plotted in Figures 4.8 and 4.9. The DABC algorithm has the lowest standard deviation and the GA has the highest standard deviation, as noted in Tables (4-8) and (4-9). In the next subsection we present these statistics in a different way to obtain further insight into IoT assignments.

4.6.3. Performance analysis

We compare the performance of our proposed Hybrid IDFWA/LC-BBO algorithm with the performance of other algorithms. Table (4-10) provides statistical analysis of our experimental results, which came from running different algorithms. For each (assignments problem) instance, we ran 100 simulations for the Hybrid IDFWA/LC-BBO algorithm, the IDFWA, the DFWA, the LC-BBO algorithm, the DABC algorithm, and the GA. We collected cost function values of the solutions obtained from each simulation and compared the Hybrid IDFWA/LC-BBO algorithm against each of the five other algorithms (IDFWA, DFWA, LC-BBO, DABC, and GA) individually.

For each (assignments problem) instance and each algorithm comparison, the null hypothesis H_0 states that both algorithms produce the same average cost. Also, we performed the t-test of an alternative hypothesis H_1 which states that the Hybrid IDFWA/LC-BBO algorithm produces lower average cost. Table (4-10) shows the p-values of the t-test for each assignment and each comparison. The p-values can be compared against the generally acceptable level of significance $\alpha = 0.05$ to decide whether hypothesis H_1 is accepted. If the average power consumed by the Hybrid IDFWA/LC-BBO algorithm is lower than any compared algorithm and $p \leq \alpha$, then we conclude that there is a statistically significant difference between the Hybrid IDFWA/LC-BBO algorithm and the other experimental algorithms. Otherwise, we conclude that the observed difference is not statistically significant.

The average power consumed by the DABC algorithm is significantly lower than the average power consumed by the Hybrid IDFWA/LC-BBO algorithm, and the average power consumed by the LC-BBO algorithm is also significantly lower than the average power consumed by the Hybrid IDFWA/LC-BBO algorithm for most of the assignments, except (50, 16, 05), (100, 25, 10), (100, 20, 10) and (200, 50, 20) (Table 4-10).

A significant difference was observed in the average power consumed by the Hybrid IDFWA/LC-BBO algorithm and the IDFWA, except for assignment (200, 50, 20). Similarly, the average power consumed by the Hybrid IDFWA/LC-BBO algorithm is significantly lower than the average power consumed by the GA for most of the assignments, except (50, 10, 05), (100, 33, 10), (100, 25, 10) and (100, 20, 10). Table (4-10) shows that for all 20 assignments the average power consumed by the Hybrid IDFWA/LC-BBO algorithm was significantly lower than the average power consumed by the DFWA. In Figures 4.10−4.13, we use boxplots to graphically show statistical results.

Table 4.10 T-test for the IoTs assignment in IoTN

# of IoTs	# of CCNs	# of BSs	Max. # of function evaluations	Algorithms				
				p-value for Hybrid IDFWA/BO vs. IDFWA	p-value for Hybrid IDFWA/BO vs. Low-complexity	p-value for Hybrid IDFWA/BO vs. DFWA	p-value for Hybrid IDFWA/BO vs. Discrete ABC	p-value for Hybrid IDFWA/BO vs. GA
20	20	02	8000	0.0001	0.0001	0.0001	0.0001	0.0001
20	10			0.0001	0.0001	0.0001	0.0001	0.0001
20	6			0.0001	0.0001	0.0001	0.0001	0.0001
20	5			0.0001	0.0001	0.0001	0.0001	0.0001
20	4			0.0001	0.0001	0.0001	0.0001	0.0001
50	50	05	12000	0.0001	0.0001	0.0001	0.0001	0.0001
50	25			0.0001	0.0001	0.0001	0.0001	0.0001
50	16			0.0001	0.6260	0.0001	0.0001	0.0001
50	12			0.0001	0.0001	0.0001	0.0001	0.0001
50	10			0.0001	0.0001	0.0001	0.0001	0.3610
100	100	10	18000	0.0001	0.0020	0.0001	0.0001	0.0001
100	50			0.0001	0.0001	0.0001	0.0001	0.0001
100	33			0.0001	0.0030	0.0001	0.0001	0.4000
100	25			0.0001	0.0260	0.0001	0.0001	0.4410
100	20			0.0070	0.1570	0.0001	0.0001	0.6630
200	200	20	20000	0.0001	0.0001	0.0001	0.0001	0.0001
200	100			0.0001	0.0001	0.0001	0.0001	0.0001
200	66			0.0001	0.0001	0.0001	0.0001	0.0330
200	50			0.4610	0.2570	0.0490	0.0000	0.0000
200	40			0.0130	0.0001	0.0001	0.0001	0.0001

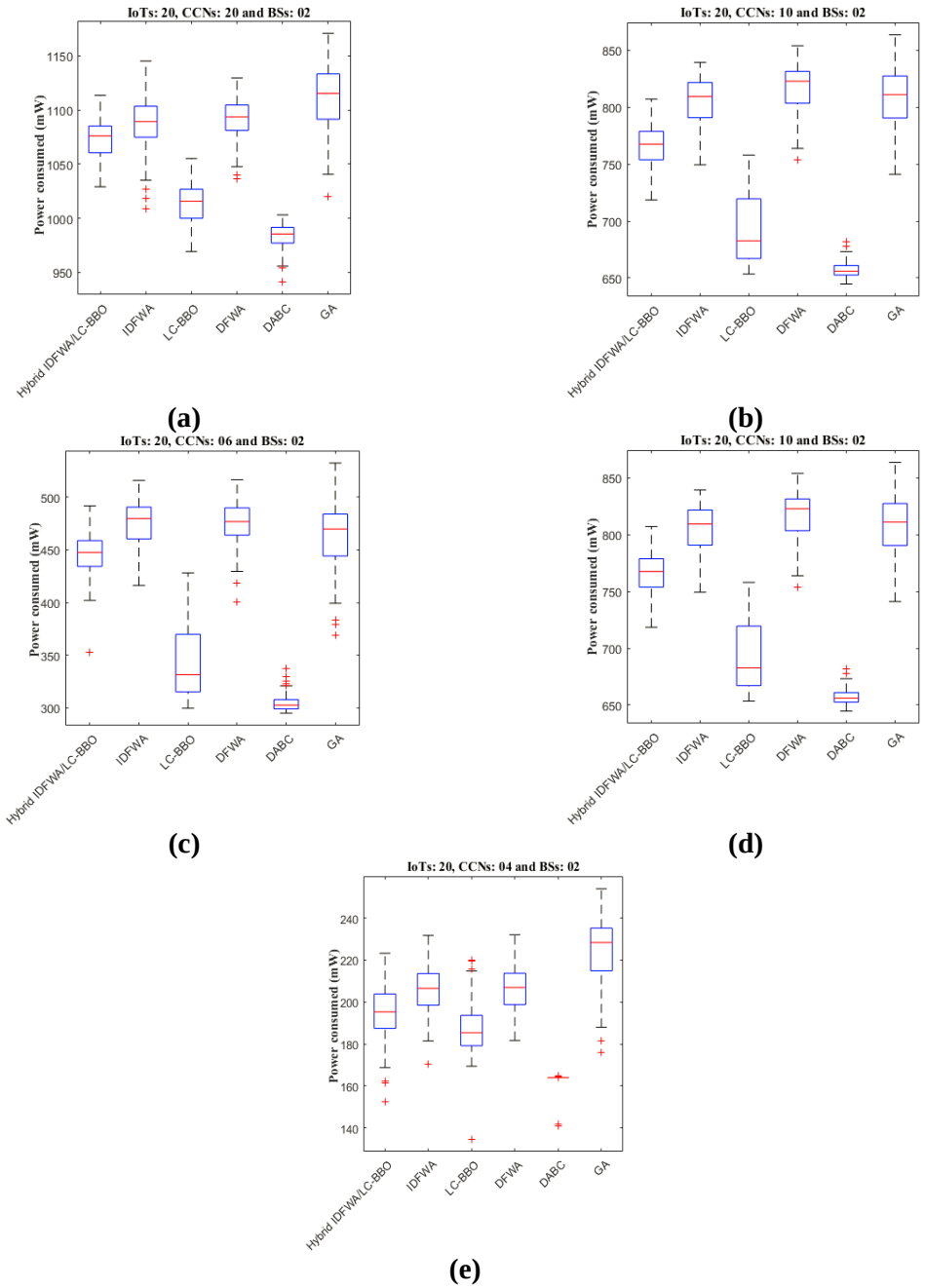


Figure 4.10 Power consumption of 20 IoTs assignment to 20, 10, 06, 05 and 04 CCNs, respectively, using different algorithms.

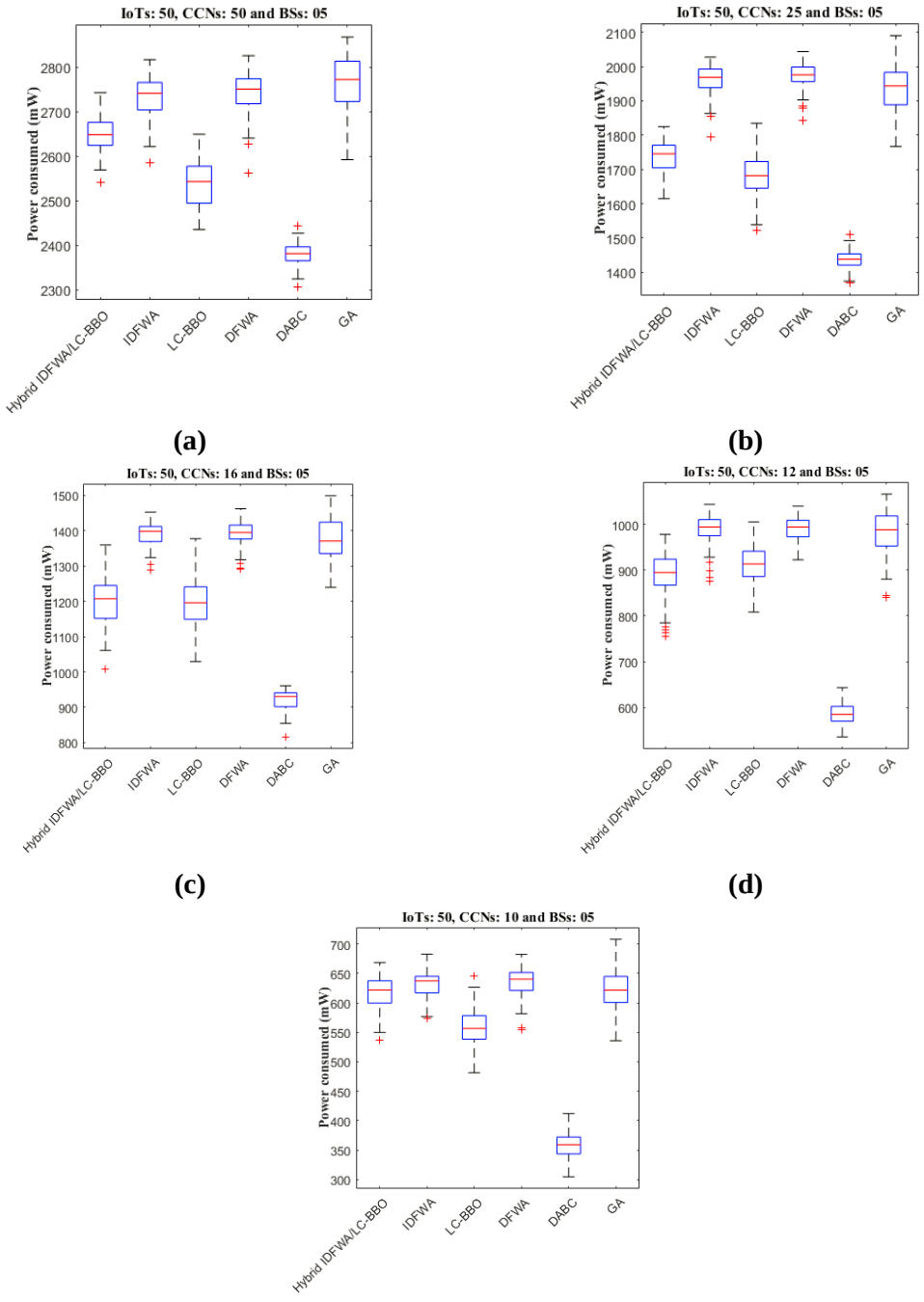


Figure 4.11 Power consumption of 50 IoTs assignment to 50, 25, 16, 12 and 10 CCNs, respectively, using different algorithms.

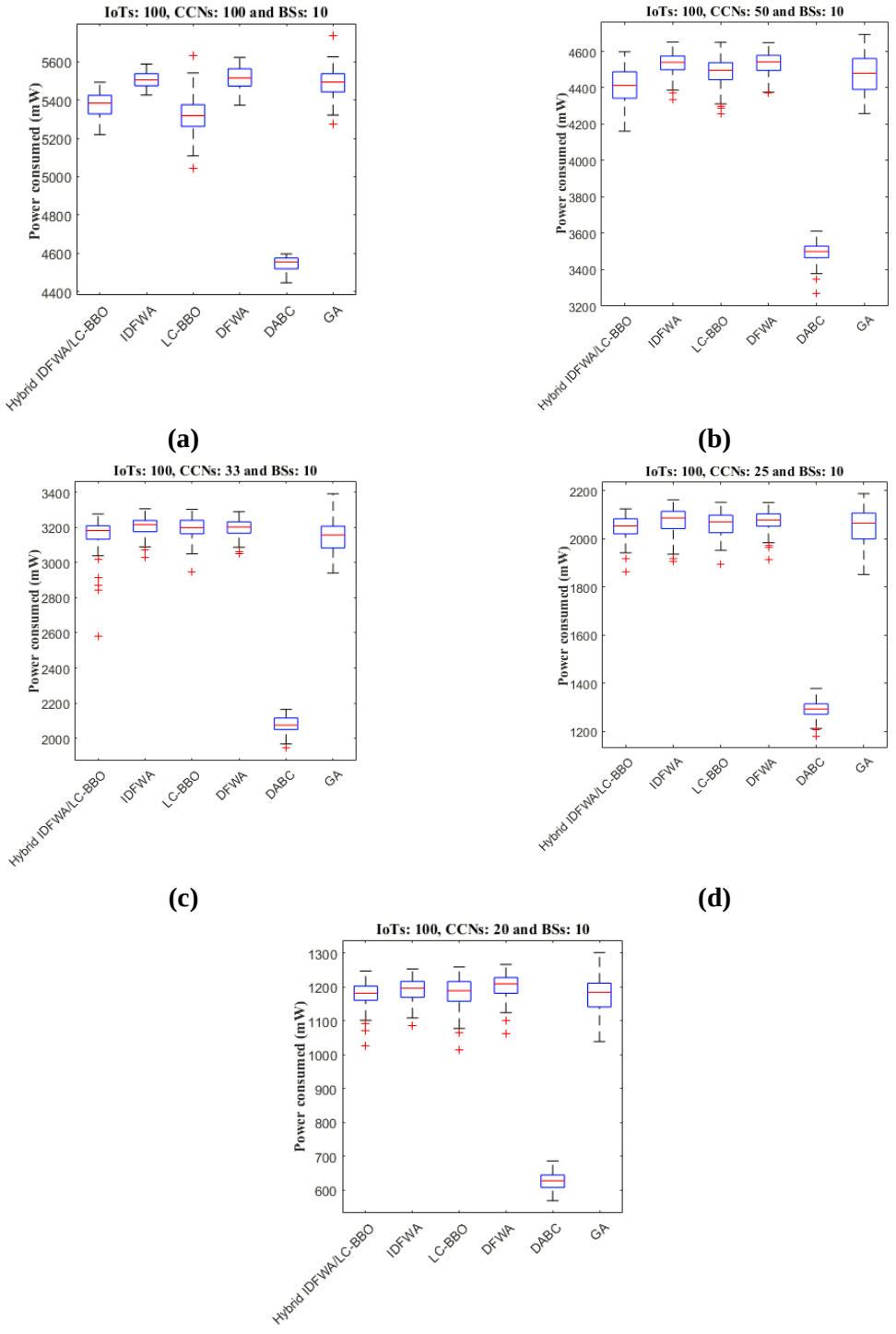


Figure 4.12 Power consumption of 100 IoTs assignment to 100, 50, 33, 25 and 20 CCNs, respectively, using different algorithms.

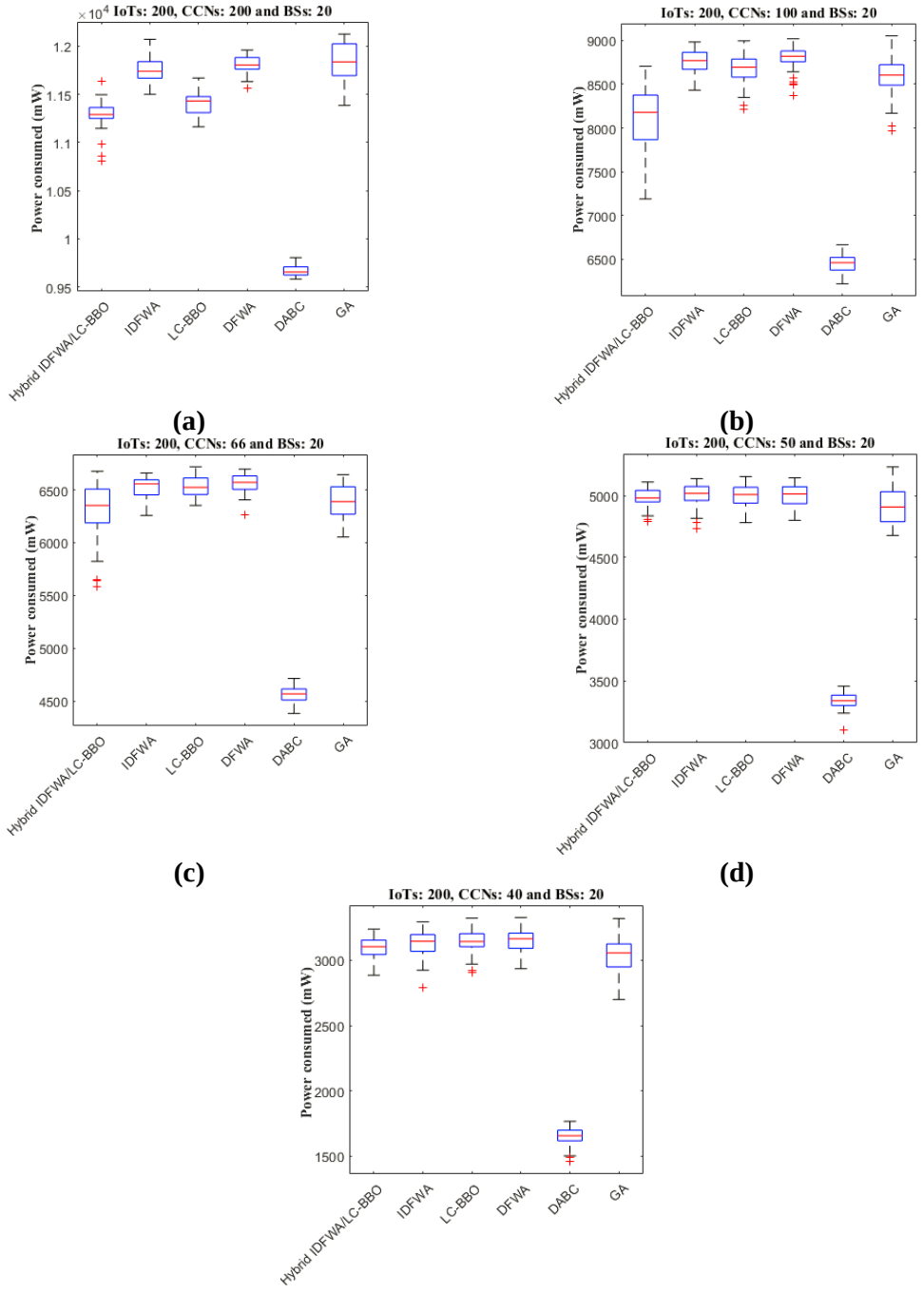


Figure 4.13 Power consumption of 200 IoTs assignment to 200, 100, 66, 50 and 40 CCNs, respectively, using different algorithms.

4.7. Conclusion

We proposed a mathematical framework for optimizing the IoTs-CCNs and CCNs-BSs assignments in a cluster-assisted IoT network. The cluster-assisted IoT network contains three types of nodes: IoTs, CCNs, and BSs. IoTs may or may not be battery powered, but CCNs are battery powered nodes and thus have limited power. Limited power and computing capable CCN is an important node for life span of the IoT network. Therefore, objective of the proposed optimization problem is to minimize weighted sum of both data transmission and computational power in the IoT network. We propose the DFWA, the problem specific information-based DFWA (IDFWA), and the Hybrid IDFWA/LC-BBO algorithms to minimize the power consumption in the IoT network. We experimentally compare the performance of the proposed algorithms against the low-complexity BBO (LC-BBO) algorithm, the DABC algorithm, the GA, and the FFD algorithm. Our experimental results indicate that the DABC algorithm and the LC-BBO algorithm save up to 56 percent and 53 percent power, respectively. The statistical analysis showed that the average power consumed by the Hybrid IDFWA/LC-BBO algorithm is significantly lower than the average power consumed by the IDFWA, the DFWA, and the GA in most of the (assignments problem) instances.

We used four metrics to record the results of experiments: ‘average power consumed,’ ‘standard deviation (Std.),’ ‘percentage of power saved,’ and ‘average CPU time (sec.),’. Our experimental results demonstrate that ‘average power consumed,’ ‘percentage of power saved,’ and ‘average CPU time (sec.),’ can be used to select an appropriate algorithm for a particular scenario. For example, the DABC algorithm would be selected when a very fast algorithm with low power consumption for IoT network is required. The GA is also a relatively fast algorithm that consumed reasonably low power in resource assignment for the IoT network.

Chapter 5. Broadband Wireless Network Plan

5.1. Introduction

High-speed Internet demand is driving 4th generation (4G) broadband wireless network technologies such as WiMAX (Worldwide Interoperability for Microwave Access) and LTE (Long Term Evolution). WiMAX (i.e., IEEE 802.16 standard) was introduced to replace wired technologies and provide wireless, high-speed Internet services in metropolitan areas (Figure 5.1). The Institute of Electrical and Electronics Engineers (IEEE) 802.16 standard was developed to enable communication over a conventional point to multi-point (PMP) WiMAX network in which a subscriber station (SS) can communicate with a base station (BS) directly or indirectly via a relay station (RS). Later, the IEEE 802.16 standard was extended to various versions by introducing an RS node with different node configurations [16], [17] for the WiMAX network. One such variant is the IEEE 802.16*j* standard that can provide coverage and capacity improvements by introducing an RS [17] in the WiMAX network. The WiMAX network-based on the IEEE 802.16*j* standard is also known as a mobile multi-hop relay (MMR) network. The IEEE 802.16*j* standard has transparent and nontransparent relay modes of operation in the WiMAX MMR. In the transparent mode, an SS can communicate directly with a BS or indirectly via an RS. The transparent mode of operation is used mainly to achieve capacity enhancements (e.g., in densely populated urban centers) within the metropolitan area [16]. The nontransparent mode can provide coverage extension for remote areas (e.g., villages, and scattered populations) in which an SS can communicate directly with a BS or via multiple RS [16], [17], [32], [33].

We propose a single-hop broadband wireless network (BWN) (i.e., SS-RS-BS) [32], [33] with links among RS, BS, SS. The BWN ensures that subscribers have sufficiently high link rates in their network and that the total load cannot exceed the maximum load on the network nodes (RS and BS). BWN data traffic communicates in uplink and downlink directions. However, we consider the downlink data traffic only [17] in the proposed BWN. A BWN can be designed from scratch or can be extended from an

existing network [16], our simultaneous BS and RS single-hop BWN was designed from scratch. The objective is to install BS and RS at given candidate sites, to meet the traffic demand of subscribers at minimum network cost [32], [33].

5.2. Related work

An overview of earlier work on the 4G broadband wireless network (BWN) is presented. We describe two types of BWN planning—coverage and capacity.

5.2.1. BWN coverage

In [17], a WiMAX network was presented that extends multi-hop RS coverage in rural areas. The goal of the network model was to find RS locations to serve SS. In [17], several real-life scenarios, such as obstacles to signal transmission like mountains and lakes, were considered. In [108], a 4G/5G heterogeneous network (HetNet) was proposed for small cells (SCs) with additional features of fault-tolerance. SCs were intended to extend network coverage (i.e., multi-hop) and to increase spectrum efficiency. A novel expanded approach was adopted to avoid nonlinearity in the mathematical model of the network.

In [109], a multi-objective hierarchical optimization model was proposed to optimize radio resource management (RRM) and mobile multi-hop relay (MMR) networks. A Markov decision process (MDP) was adopted to obtain the short-term optimal action of an RS. Using the optimal action of each RS, the network planning problem was solved for an RS group by optimizing RS deployment and BS selection. In [110], issues of cost-effective coverage extension in WiMAX multi-hop systems were investigated and several topologies were presented with the resulting cost-effective network coverage; two cost-effective coverage methods with sector-based cellular approaches were wide-beam tri-sector cell (WBTC) and narrow-beam tri-sector cell (NBTC). A three-phase deployment scenario with user traffic density was presented. In [111] omnidirectional and directional antennae were used to model MMR network topologies for network coverage. The authors obtained an optimal network configuration using analytical methods.

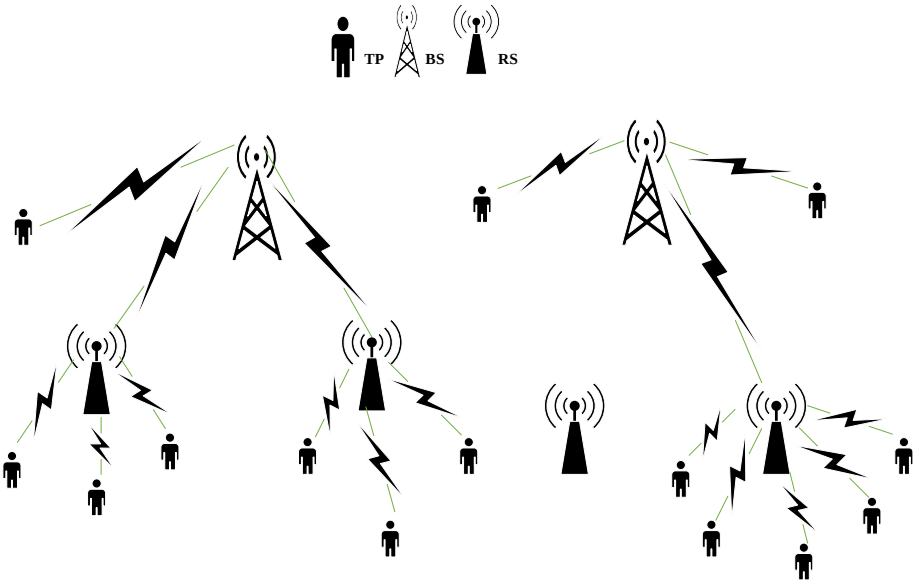


Figure 5.1 Overview of a broadband wireless network.

Multilevel cooperative relaying (CR) has been accepted as an effective design paradigm for achieving throughput (or capacity) enhancement in modern BWNs. In [112], a CR based multi-hop wireless network optimization framework was presented. A two-phase heuristic algorithm was proposed to make the solution of the optimization problem computationally tractable. Similarly, an MMR model with CR was presented for coverage and throughput enhancement in the network [113] An advanced CR technology such as Decode-Forward (D-F) or Compress-Forward (C-F) was used for operating the RS. The goal of the optimization problem was to find optimal RS locations and SS resource allocation. A numerical analysis was conducted to demonstrate the performance of the proposed model.

In [114], a clustering algorithm was proposed to select appropriate locations for BS and RS. This approach considered traffic demands and uniform cluster concepts to make an adaptive decision for selecting the deployment sites for BSs and RSs. The solution of the optimization problem showed reasonable results for the proposed algorithm. Similarly, a clustering approach was used to solve a multi-hop network planning problem for IEEE 802.16*j* networks in [99] with three basic steps: (1) divide the nodes (i.e. BSs and RSs) into small distinct clusters, (2) solve the planning problem separately for each cluster, and (3)

goal of the final optimization is to reduce issues arising at cluster boundaries. In [99], an integer programming model was presented to determine optimal BS and multi-hop RS locations. A state space reduction strategy was used to reduce the search space for the proposed mathematical model. Standard branch and bound algorithms were used to solve the optimization problem.

5.2.2. BWN capacity

In [15], an integer programming problem was formulated to determine BS and RS locations that would enhance network capacity at minimal cost. The authors formulated a joint BS/RS problem, then determined BS locations followed by RS locations. A simultaneous BSs and RSs locations planning problem with link flow [32], [33] was formulated and was solved using evolutionary algorithms. In [115], an integer programming problem was formulated to jointly deploy BS and RS to serve SS in such a way that the cost was within the given budget and the system capacity was maximized. A two-stage network deployment algorithm was presented to analyze the complexity and design of the network.

Cooperative relaying (CR) technology can improve capacity in 4G wireless networks. The authors in [116] formulated an optimization framework to maximize the capacity as well as to meet the minimal traffic demand of each SS. The objectives of optimization were to place RS and allocate bandwidth. A mixed-integer nonlinear program was solved with a genetic algorithm (GA). In [117], the same problem was reformulated as a linear integer problem, which was solved by the CPLEX solver. Note that the CPLEX solver is used to solve integer programming problem.

In [118], an RS deployment mechanism was proposed for a given BS and *k* RS in such a way that the SS bandwidth requirement can be satisfied, and the network throughput can be significantly improved. In [119], RS location planning was formulated for capacity gains in the IEEE 802.16*j* network transparent mode. RS locations were determined from a given set of candidate RS sites, and the optimization problem was solved by the interference aware algorithm.

The IEEE 802.16*j* standard defines relay stations (RS) to enhance network throughput. Deploying RS within the serving area of the base station (BS) can increase network throughput but raises hardware costs. The authors presented a deployment algorithm for an IEEE 802.16*j* network. In [120], a three-phase RS deployment algorithm was proposed. The aim of the first phase was to construct several promising zones where an RS can be deployed. RS deployment in each zone would improve the transmission rate from SS to BS. In the second phase, larger zones were constructed by combining several smaller zones, to reduce the number of deployed RS. When the RS were all deployed in promising zones, the results showed that the transmission delay and the hardware cost was reduced with the proposed algorithm.

In Table (5-1), we compare some existing state-of-the-art broadband wireless network (BWN) models for planning, which either plan capacity or coverage. To the best of our knowledge, limited work has been reported in the literature that considers the objective of simultaneously plan BSs and RSs on the given potential candidate sites while minimizing the path-loss as operational cost.

Table 5.1 Comparison of the recent work for BWN planning

Reference	Objective of the broadband Wireless Network Planning	Coverage Planning for the broadband Wireless Network	Capacity Planning for the broadband Wireless Network	Algorithms/Methods	Remarks
[14]	Planning BSs and RSs with the objective of optimizing power.	✓	✗	Standard branch and bound technique is used to solve the problem	A multi-hops Integer Programming problem is formulated.

[16]	Planning BSs and RSs locations.	✗	✓	Numerical analysis	Two tier network planning.
[17]	Planning RSs locations.	✗	✓	Relay placement mechanism to maximize network capacity.	*k* RSs are deployed region that can be fully covered by the BS.
[99]	Planning BSs and RSs with the objective of optimizing power.	✓	✗	Clustering approach is used to solve the problem.	A multi-hops Integer Programming problem is formulated.
[108]	Minimizing total number of small cells in use.	✓	✗	CPLEX is used for the small size of problem.	Mixed-integer linear problem is solved using CPLEX for small problem size.
[109]	Optimizing the radio resource management and placing RSs.	✓	✗	Hierarchical model is used to gauge quantitative impacts of parameters.	RS-based transmission to maximize utility of a RS.
[110]	Planning RSs locations.	✓	✗	Sectorized cellular approaches is used.	Pre-set topologies and finds the RS locations.
[112]	Planning RSs locations.	✓	✗	A heuristic two-phase algorithm is used.	The cooperative transmission paradigm is used in multi-hop relaying for range extension.
[113]	A single RS placement.	✓	✗	Numerical analysis to show the merits.	Cooperative relaying technology is adopted.
[114]	Planning BSs and RSs for network throughput and coverage.	✓	✗	A clustering algorithm is used to solve the problem	A scheme that makes adaptive decision for selecting the deployment of the BS and RS.
[115]	Deploying BSs and RSs with the objective of capacity and fairness	✗	✓	Two-stage heuristic network deployment algorithm is presented.	Maximizes the network data rate and hence maximizes the network capacity.
[116]	Planning RSs locations.	✗	✓	A mixed integer nonlinear program is solved using a heuristic approach based on genetic algorithm.	RS placement solution that uses the cooperative transmission for capacity enhancement.
[117]	Planning RSs locations.	✗	✓	CPLEX is used to solve the problem	RS placement solution that uses the cooperative transmission for capacity enhancement.

[118]	Minimize the hardware, installation and operational cost.	✓	×	CPLEX is used for the small size of problem.	Mixed-integer linear problem is solved using CPLEX for small problem size.
[119]	Planning RSs locations.	×	✓	Interference aware algorithm is used to plan RS locations.	RSs with more power should be used more frequently.
[120]	Planning RSs locations	×	✓	Cost-aware relay deployment (CARD) mechanism	Deploying RSs within the serving area of the BS that increases network throughput
[111]	Planning BSs and multi-hop RSs in two steps	✓	×	CPLEX is used to solve the problem	Through analytical methods, the optimal network configuration is obtained using traffic density for various topologies.

We formulate a single-hop BWN, which is a computationally challenging integer programming problem. An approximate algorithm is a useful alternative to solve the BWN planning problem with reasonable computing resources.

5.3. System Model and Problem Formulation

5.3.1. System model

A single-hop broadband wireless network consists of base stations (BSs), relay stations (RSs), and test points (TPs). Broadband Wireless Network (BWN) planning should decide the placement of BSs and RSs to the candidate sites and decide the possible connections among them and their further connections to TPs. Here, a connection between two nodes indicates that the corresponding nodes are communicating with each other. The objective is to minimize the hardware and operational cost in capacity planning of broadband wireless network. In the proposed BWN, a test point (TP) can be a subscriber or a group of subscribers with certain data traffic demand. A TP can be connected to a BS directly or communicated to a BS via a RS subject to fulfil its traffic demand. TPs may or may not be battery powered. A TP cannot communicate through more than one RS or more than one BS or cannot communicate through both a BS and a RS at a time. A RS is used to relay data from a BS to TP(s) subject to fulfil its traffic demand. Note that in the proposed

network, we only consider data traffic in the downlink direction. A BS is directly connected to the internet and has reasonable computing capability. As compared to a BS, a RS has limited computing capability. One or more RSs can be connected to a BS.

5.3.2. Problem formulation

The objective of the planning is to minimize the overall cost (hardware, operational) of network functioning in the presence of users' traffic demand. Terms used in this chapter are defined in Table 5-2.

Table 5.2 Notations used in chapter 5

Symbol	Definition
B	set of BSs sites.
R	set of RSs sites.
T	set of test points (TP) or users
(b, r)	denotes the link between BS site b and RS site r
(b, t)	denotes the link between BS site b and TP t
(r, t)	denotes the link between RS site r and TP t
c_b^B	denotes the BS cost at the BS site b
c_r^R	denotes the RS cost at the RS site r
$l_{b,r}^{BR}$	denotes the path-loss associated with the link (b, r) BS site b and RS site r
$l_{b,t}^{BT}$	denotes the path-loss associated with the link (b, t) BS site b and TP t
$l_{r,t}^{RT}$	denotes the path-loss associated with the link (r, t) RS site r and TP t
$m_{b,r}^{BR}$	represents the upper bound (e.g., channel capacity) on the possible information flow rate associated with the link (b, r) BS site b and RS site r
$m_{b,t}^{BT}$	represents the upper bound (e.g., channel capacity) on the possible information flow rate associated with the link (b, t) BS site b and TP
$m_{r,t}^{RT}$	represents the upper bound (e.g., channel capacity) on the possible information flow rate associated with the link (r, t) RS site r and TP t

u_t^T	denotes traffic demand of TP *t*
C_1 and C_2	denotes the maximum capacity (in bits per second) for each deployed BS and RS, respectively
W_1 and W_2	weight parameters for two terms of objective function
y_b^B and y_r^R	denotes the binary decision variables that determine whether a BS *b* and a RS *r* are deployed on BS and RS sites, respectively
$x_{b,r}^{BR}$, $x_{b,t}^{BT}$ and $x_{r,t}^{RT}$	binary decision variables that denote whether a connection is established on the corresponding links (*b, r*) BS site *b* and RS site *r*, (*b, t*) BS site *b* and TP *t*, (*r, t*) RS site *r* and TP *t*, respectively
$f_{b,r}^{BR}$, $f_{b,t}^{BT}$ and $f_{r,t}^{RT}$	are functions that represent flow (bit per second) on the corresponding links. For example, $f_{b,r}^{BR}$ is a function that represent flow from BS site *b* and RS site *r*.

5.3.2.1. Cost function

The objective of the broadband wireless network (BWN) planning is to minimize the hardware and operational cost of the network. Total hardware expenses include the costs of deployed base stations (BS) and deployed relay stations (RS) at their corresponding selected locations. TP (test points) or users can communicate to a deployed BS directly or indirectly via a deployed RS, and a deployed RS must communicate to a deployed BS. Communication among BWN nodes— (TP, BS, RS) occurs at lower cost when nodes are in proximity. In communication among distant nodes, more power is consumed, the quality of the communication may be degraded, and the wireless links might be impaired due to path-loss. Therefore, we treat path-loss as an operational cost for each communicating link (i.e., TP-BS, TP-RS, RS-BS). The hardware cost (HC) of the deployed BSs and RSs is denoted by ξ (i.e., lower case Xi) and expressed as follows:

$$\xi = \sum_{b \in B} c_b^B . y_b^B + \sum_{r \in R} c_r^R . y_r^R,$$

The decision variables $x_{b,r}^{BR}$, $x_{b,t}^{BT}$ and $x_{r,t}^{RT}$ (*see Table 5-2*) are not only used to calculate the operational cost (OC) of the network but are also used to implement various network constraints. The OC as path-loss among the TPs, BSs and RSs communication links is denoted by Ξ (i.e., upper case Xi) and is expressed as follows:

$$\Xi = \sum_{b \in B} \sum_{t \in T} l_{b,t}^{BT} . x_{b,t}^{BT} + \sum_{r \in R} \sum_{t \in T} l_{r,t}^{RT} . x_{r,t}^{RT} + \sum_{b \in B} \sum_{r \in R} l_{b,r}^{BR} . x_{b,r}^{BR}.$$

The cost function and constraints for the BWN are:

$$\min_{\substack{y_b^B, y_r^R \in \{0,1\}, \forall b \in B, r \in R \\ x_{b,r}^{BR}, x_{b,t}^{BT}, x_{r,t}^{RT} \in \{0,1\}, \forall b \in B, r \in R, t \in T}} W_1 \times \xi + W_2 \times \Xi, \tag{5.1}$$

In the cost function, equation (5.1), the first term represents the hardware and installation costs of BS and RS, respectively. As communication between nearby nodes (TP, BS, and RS) is more favorable with respect to cost and clarity than communication between distant nodes, we introduce a second term in equation (5.1) to incorporate the notion of nearby communication. The second term in equation (5.1) represents the path-loss of each communication link between nodes. W_1 *and* W_2 are weight parameters for the first and second terms, respectively, in the cost function equation (5.1). The constraints for the objective function (5.1) are defined in subsequent subsections.

5.3.2.2. Topology constraints

The goal of the Broadband Wireless Network (BWN) planning is to decide the placement of BSs and RSs to the candidate sites and decide the possible connections among them and their further connections to TPs. The objective is to minimize the hardware and operational cost in capacity planning of broadband wireless network. The binary decision variables y_b^B and y_r^R are used to determine whether a BS *b* and a RS *r* can be deployed on candidate BS and RS sites, respectively. For connections among TPs, RSs and BSs, the $x_{b,r}^{BR}$, $x_{b,t}^{BT}$ and $x_{r,t}^{RT}$ binary decision variables are used to determine whether a connection can be established on the corresponding links BSs-RSs, BSs-TPs and RSs-TPs, respectively. The topology constraints are as follows:

$\sum_{b \in B} x_{b,r}^{BR} \leq y_r^R$, $\forall r \in R$, where $x_{b,r}^{BR}$ *and* $y_r^R \in \{0,1\}$, (5.2)

$x_{b,t}^{BT} \leq y_b^B$, $\forall b \in B, t \in T$, where $x_{b,t}^{BT}$ and $y_b^B \in \{0,1\}$, (5.3)

$x_{b,r}^{BR} \leq y_b^B$, $\forall b \in B, r \in R$, where $x_{b,r}^{BR}$ and $y_r^R \in \{0,1\}$, (5.4)

$x_{r,t}^{RT} \leq y_r^R$, $\forall r \in R, t \in T$, where $x_{r,t}^{RT}$ and $y_r^R \in \{0,1\}$, (5.5)

$\sum_{\mathrm{b}\in B} x_{b,t}^{BT} + \sum_{r\in R} x_{r,t}^{RT} = 1,\ \forall t \in T$, where $x_{b,t}^{BT}\ and\ x_{r,t}^{RT} \in \{0,1\}$. (5.6)

Constraint (5.2) ensures that each RS, if deployed (e.g., y_r^R = 1), is connected to one BS only. Constraints (5.3) – (5.4) define that every TP and RS can be connected only to a deployed BS (e.g., y_b^B = 1). Constraint (5.5) confirms that every TP is connected to a deployed RS (e.g., y_r^R = 1). Constraint (5.6) ensures that each TP is connected to either one BS or one RS.

5.3.2.3. Flow constraints

Flow is the amount of data (bits per seconds) that can be transmitted on a wireless link. Here, we define flow in terms of nonnegative real valued function of a binary decision variable. For example, in the BWN planning, the functions $f_{b,r}^{BR}\left(x_{b,r}^{BR}, x_{r,t}^{RT}\right)$, $f_{b,t}^{BT}\left(x_{b,t}^{BT}\right)$, and $f_{r,t}^{RT}\left(x_{r,t}^{RT}\right)$ represent flow on the corresponding links $x_{b,r}^{BR}$, $x_{b,t}^{BT}$ and $x_{r,t}^{RT}$ respectively. For clarity, see Table 5-2 for related notations. Note that every feasible link must adhere to the corresponding maximum capacity $m_{b,r}^{BR}$, $m_{b,t}^{BT}$ and $m_{r,t}^{RT}$.

The flow $f_{b,t}^{BT}$ is a function of binary decision variables $x_{b,t}^{BT}$ and is defined as follows:

$$f_{b,t}^{BT}\left(x_{b,t}^{BT}\right) = \begin{cases} 0, & if\ x_{b,t}^{BT} = 0 \\ u_t^T, & if\ x_{b,t}^{BT} = 1 \end{cases}.$$

For each existing BS-TP connection (i.e., $x_{b,t}^{BT}$ =1), the flow value $f_{b,t}^{BT}$must be within the maximum capacity $m_{b,t}^{BT}$ as follows:

$$f_{b,t}^{BT}\left(x_{b,t}^{BT}\right) \le m_{b,t}^{BT}, \forall b \in B, t \in T. \quad (5.7)$$

The flow $f_{r,t}^{RT}$ is a function of binary decision variable $x_{r,t}^{RT}$ and is defined as follows:

$$f_{r,t}^{RT}\left(x_{r,t}^{RT}\right) = \begin{cases} 0, & if\ x_{r,t}^{RT} = 0 \\ u_t^T, & if\ x_{r,t}^{RT} = 1 \end{cases}.$$

For each existing RS-TP connection (i.e., $x_{r,t}^{RT}$ =1), the flow value $f_{r,t}^{RT}$must be within the maximum capacity $m_{r,t}^{RT}$ as follows:

$$f_{r,t}^{RT}\left(x_{r,t}^{RT}\right) \leq m_{r,t}^{RT}, \forall r \in R, t \in T. \quad (5.8)$$

The flow $f_{b,r}^{BR}$ is a function of binary decision variables $x_{b,r}^{BR}$, $x_{r,t}^{RT}$ and is defined as follows:

$$f_{b,r}^{BR}\left(x_{b,r}^{BR}, x_{r,t}^{RT}\right) = \begin{cases} \sum_{t\in T} u_t^T . x_{r,t}^{RT}, & if \ \ x_{b,r}^{BR} = 1 \\ 0, & if \ \ x_{b,r}^{BR} = 0 \end{cases}.$$

For each existing BS-RS connection (i.e., $x_{b,r}^{BR}$ =1), the flow value $f_{b,r}^{BR}$ must be within the maximum capacity $m_{b,r}^{BR}$ as follows:

$$f_{b,r}^{BR}\left(x_{b,r}^{BR}, x_{r,t}^{RT}\right) \leq m_{b,r}^{BR}, \forall b \in B, r \in R. \quad (5.9)$$

Constraints (5.7) – (5.9) ensure that the flow value on the links (*b*, *r*), (*b*, *t*), (*r*, *t*) are within maximum capacity.

5.3.2.4. Load constraints

$$\sum_{r\in R} f_{b,r}^{BR}(x_{b,r}^{BR}, x_{r,t}^{RT}) + \sum_{t\in T} f_{b,t}^{BT}(x_{b,t}^{BT}) \leq C_1, \ \forall b \in B, \quad (5.10)$$

$$\sum_{t\in T} f_{r,t}^{RT}(x_{r,t}^{RT}) \leq C_2, \ \forall r \in R, \quad (5.11)$$

$$\sum_{b\in B} f_{b,t}^{BT}(x_{b,t}^{BT}) + \sum_{r\in R} f_{r,t}^{RT}(x_{r,t}^{RT}) = u_t^T, \ \forall t \in T. \quad (5.12)$$

Constraints (5.10) and (5.11) confirm that the load on each deployed BS and RS does not exceed the maximum load. Finally, (5.12) guarantees that every TP has enough flow through either a BS or an RS.

5.4. Problem Reformulation

5.4.1. Redefining the decision variables

In the planning problem R, B, T denote sets of RS sites, BS sites, and TPs, respectively. The proposed BWN planning problem has an integer domain, and we define a candidate solution as a vector of nonnegative integers as follows:

$$X = (\mathfrak{I}_1, \mathfrak{I}_2, \dots, \mathfrak{I}_{|T|}, \mathfrak{R}_1, \mathfrak{R}_2, \dots, \mathfrak{R}_{|R|}),$$

where |T| and |R| are the cardinalities of sets T and R. (5.13)

Note that in X, $\mathfrak{I}_t$ represents the t^{th} TP, which must be connected to some BS site b or RS site r , and $\mathfrak{R}_r$ represents the r^{th} RS, which can be connected to some BS site b. $\mathfrak{R}_r$ is zero when the r^{th} RS is not deployed. In X, each TP can be connected to any one BS or RS. Therefore, we represent BS followed by RS in consecutive orders of positive integers in the candidate solution X. Suppose, we have $|B|$ BSs and $|R|$ RSs sites in a candidate solution$X$. In the candidate solution X, BSs sites are represented by 1,2, 3…, |B| and RS sites are represented by 1+|B|, 2+|B|, 3+|B|…, $|R|$+|B|.

In X, integer variable $\mathfrak{I}_t, t = 1,2, \dots, |T|$ takes a value in set $\{1, 2, \dots, |B|, |B| + 1, \dots, |B| + |R|\}$, where element 1, 2… |B| represents the BS sites, and |B| +1… |B|+|R| represents the RS sites. Integer variable $\mathfrak{I}_t$ indicates TP t is connected to a BS or a RS site. In addition, $\mathfrak{I}_t$ also indicates that the t^{th} TP is connected to which BS or RS site. $\mathfrak{I}_t = i$ if $i \leq |B|$ indicates that TP t is connected to a BS site i and a BS is installed at the BS site i. $\mathfrak{I}_t = i$, if $i > |B|$, indicates that TP t is connected to RS site $i - |B|$ and the RS is deployed at site $i - |B|$. Integer variable $\mathfrak{R}_r$ takes a value in {0, 1, 2 … |B|}. Here, $\mathfrak{R}_r = 0$ indicates that a relay is not installed (deployed) at relay site r. $\mathfrak{R}_r = 0$ also indicates the RS is not connected to any TP or any BS; hence RS is not deployed. $\mathfrak{R}_r = b$, if $1 \leq b \leq |B|$, indicates that a relay is installed (deployed) at relay site r and that the relay station site is connected to a base station site b. The same also indicates that BS site b should have a base station installed (or deployed). Both $\mathfrak{I}_t$ and $\mathfrak{R}_r$ variables decide whether there is a BS in BS site b.

For example, if we have three BS sites i.e., B= {1,2,3}, three RSs sites i.e., R= {1,2,3}, and four TPs i.e., T= {1,2,3,4}, the candidate solution is $X = (1,4,2,5,1,2,0)$. In X, we represent three BS sites 1, 2, and 3 and three RS sites as 1+|B| (e.g., 4), 2+ |B| (e.g., 5), and 3+ |B| (e.g., 6); where in this example |B| =3 is the cardinality of set B. In X, the first four components represent TPs and the last three components represent RS sites. Clearly, $\mathfrak{I}_1$ is connected to BS site 1, $\mathfrak{I}_2$ is connected to RS site 1 (4=3+|B|), $\mathfrak{I}_3$ is connected to BS site 2, $\mathfrak{I}_4$ is connected to RS site 2 (5=2+|B|), and $\mathfrak{R}_1$, $\mathfrak{R}_2$ sites are connected to BS site 1 and BS site 2, respectively. Further, from X note that no BS is

deployed at BS site 3 because no component of *X* is associated with 3. Similarly, no RS is deployed at RS site $\Re_3$ because neither a TP *t* is connected to RS site $\Re_3$ (3+ |B| = 6) nor $\Re_3$ is connected to any BS site. The same also indicates that BS site *b* should have a base station installed (or deployed). Both $\Im_t$ and $\Re_r$ variables decide whether there is a BS in BS site *b*.

5.4.2. Reformulating broadband network planning

In reformulated network planning problem, we use a decision vector of nonnegative integers $X = (\Im_1, \Im_2, \dots, \Im_{|T|}, \Re_1, \Re_2, \dots, \Re_{|R|})$, where |T| and |R| are the cardinalities of sets T and R. Here, each *X* is a candidate configuration of the BWN, which was presented in the section 5.3.3. The parameters of the BWN planning problem are the same before and after encoding, except a small adjustment for the implementation and experimentation. Now, we reformulate the BWN planning problem using variable *X*; the indices are used as: $t = 1,2, \dots, |T|$, $r = 1,2, \dots, |R|$ and $b = 1,2, \dots, |B|$.

5.4.2.1. Cost function

We reformulate the plan for the broadband wireless network (BWN) using the encoded vector *X*. The cost function of the BWN plan is to minimize the weighted sum of hardware and operational cost of the network. Reformulations of the cost function and constraints for the BWN are as follows:

Let a function $\mathscr{b}_b^B$ represents whether a BS is installed in the BS site *b* in vector *X*:

$$\mathscr{b}_b^B = \begin{cases} 1, & \sum_{\{i:1 \le i \le |T|+|R| \wedge X_i = b\}} X_i \ge 1 \\ 0, & otherwise \end{cases}.$$

Let a function S_r^R represents whether a RS is installed in the RS site *r* in vector *X*:

$$S_r^R = \begin{cases} 1, & \sum_{\{t:1 \le t \le |T| \wedge X_t = r+|B|\}} X_t \ge 1 + |B| \\ 0, & otherwise \end{cases}.$$

The hardware cost (HC) of the deployed BSs and RSs is as follows:

$\xi' = \sum_{b \in B} \mathscr{b}_b^B . c_b^B + \sum_{r \in R} S_r^R . c_r^R .$

The operational cost (OC) as path-loss among the TPs, BSs and RSs communication links is as follows:

$$\Xi' = \sum_{b\in B}\sum_{\{t:1\le t\le |T| \wedge X_t=b\}} l_{b,t}^{BT} + \sum_{r\in R}\sum_{\{t:1\le t\le |T| \wedge X_t=r+|B|\}} l_{r,t}^{RT} + \sum_{b\in B}\sum_{\{r:|T|+1\le r\le |R|+|T| \wedge X_r=b\}} l_{b,r}^{BR}.$$

$$\min_{X} W_1 \times \xi' + W_2 \times \Xi', \qquad (5.14)$$

In the cost function, equation (5.14), the first term represents the hardware and installation costs of BS and RS, respectively. The second term in equation (5.14) represents the path-loss of each communication link between nodes. W_1 *and* W_2 are weight parameters for the first and second terms, respectively.

5.4.2.2. Topology constraints

To reduce the constraints checks, we defined a vector of nonnegative integers $X = (\mathfrak{I}_1, \mathfrak{I}_2, \dots, \mathfrak{I}_{|T|}, \mathfrak{R}_1, \mathfrak{R}_2, \dots, \mathfrak{R}_{|R|})$, where |T| and |R| are the cardinalities of sets *T* and *R*. Constraints (5.2) – (5.6) are implicitly enforced in the vector X.

5.4.2.3. Flow constraints

Flow is the amount of data (bits per seconds) that can be transmitted on a link. Here, we define flow in terms of nonnegative real valued function in terms of decision variable X. For example, in BWN planning, the functions $f_{b,r}^{BR}(X)$, $f_{b,t}^{BT}(X)$, and $f_{r,t}^{RT}(X)$ represent flow on the corresponding links in X. See *section 5.4.1* for related notations and definition of *X*. Note that every feasible link in the network must adhere to the corresponding maximum capacity $m_{b,r}^{BR}, m_{b,t}^{BT}$ and $m_{r,t}^{RT}$. The flow $f_{b,t}^{BT}$ is a function of decision variable *X* and is defined as follows:

$$f_{b,t}^{BT}(X) = \begin{cases} u_t^T, & if\ X_t = b \\ 0, & otherwise \end{cases}, \forall b = 1,2,\dots,|B| \text{ and } \forall t = 1,2,\dots,|T|.$$

For each connection between a BS and a TP, the flow value $f_{b,t}^{BT}$ must be within the maximum capacity $m_{b,t}^{BT}$ as follows:

$$f_{b,t}^{BT}(X) \le m_{b,t}^{BT}, \forall b = 1,2,\dots,|B| \text{ and } \forall t = 1,2,\dots,|T| \quad . \qquad (5.15)$$

The flow $f_{r,t}^{RT}$ is a function of decision variable *X* and is defined as follows:

$$f_{r,t}^{RT}(X) = \begin{cases} u_t^T, & if\ X_t = r + |B| \\ 0, & otherwise \end{cases}, \forall r = 1,2,\dots,|R| \text{ and } \forall t = 1,2,\dots,|T|.$$

For each connection between a RS and a TP, the flow value $f_{r,t}^{RT}$ must be within the maximum capacity $m_{r,t}^{RT}$ as follows:

$$f_{r,t}^{RT}(X) \leq m_{r,t}^{RT}, \forall r = 1,2,\dots,|R| \text{ and } \forall t = 1,2,\dots,|T|. \quad (5.16)$$

The flow $f_{b,r}^{BR}$ is a function of decision variable X and is defined as follows:

$$f_{b,r}^{BR}(X) = \begin{cases} \sum_{t=1}^{|T|} f_{r,t}^{RT}(X), & if\ X_j = b\ where\ j = r + |T| \\ 0, & otherwise \end{cases},$$

$$\forall b = 1,2,\dots,|B| \text{and } \forall r = 1,2,\dots,|R|.$$

For each existing connection between a BS and an RS, the flow value $f_{b,r}^{BR}$ must be within the maximum capacity $m_{b,r}^{BR}$ as follows:

$$f_{b,r}^{BR}(X) \leq m_{b,r}^{BR}, \forall b = 1,2,\dots,|B| \text{ and } \forall r = 1,2,\dots,|R|. \quad (5.17)$$

5.4.2.4. Load constraints

$$\sum_{t=1}^{|T|} f_{b,t}^{BT}(X) + \sum_{r=1}^{|R|} f_{b,r}^{BR}(X) \leq C_1, \forall b = 1,2,\dots,|B|, \quad (5.18)$$

$$\sum_{t=1}^{|T|} f_{r,t}^{RT}(X) \leq C_2, \forall r = 1,2,\dots,|R|, \quad (5.19)$$

$$\sum_{b=1}^{|B|} f_{b,t}^{BT}(X) + \sum_{r=1}^{|R|} f_{r,t}^{RT}(X) = u_t^T, \forall t = 1,2,\dots,|T|. \quad (5.20)$$

Constraints (5.18) and (5.19) ensure that the load on each BS and RS does not exceed the maximum load and (5.20) guarantees that every TP has enough flow, either through a BS or an RS.

BWN planning is a combinatorial integer space optimization problem and author does not know any polynomial time algorithm to solve such problems. Exhaustive search requires high computing cost to find the optimal solution for these problems. A practical approach is to solve such problems using approximate algorithms, e.g., evolutionary algorithms, in reasonable computing resources. Without any guarantee of optimal solution, EAs can provide a high-quality solution using moderate computing resources. Therefore, we propose a relatively new swarm intelligence-based EA i.e., discrete fireworks algorithm (DFWA) and its variants for the BWN planning.

5.5. Discrete fireworks algorithm

The local search method adopted in the EFWA cannot be used for discrete space optimization [23]. In chapter 3, and chapter 4, we modify EFWA operators to convert their real value into their integer value to operate on integer space optimization. In another version of DFWA [30], the local search (LS) method of the EFWA is replaced with a neighborhood mapping-based LS method such as '*insert,*' '*interchange,*' and '*swap*'. In the DFWA, '*insert,*' '*interchange,*' and '*swap*' LS methods exchange/replace one or more components of a firework as a perturbation to generate sparks. Like the EFWA, the DFWA also has four basics operations: an explosion operator, a mutation operator, a repair mechanism, and a selection operation.

5.5.1. Explosion operator

Initially, the DFWA randomly generates a population of *N* fireworks, and each of the fireworks is evaluated using the cost function in (5.13). The cost and parameters are used to determine the criteria of the explosion operator. The explosion operator uses the local search (LS) method with two parameters, the explosion strength and the explosion radius [23], [31], to generate sparks. The DFWA explosion operator determines the number of sparks and the explosion radius in proportion to the cost value of fireworks.

5.5.1.1. Explosion strength

In the discrete fireworks algorithm (DFWA), the explosion strength determines the number of sparks that are generated by the explosion of a firework. The cost value of a firework and user-defined control parameters are used to determine the number of sparks that are generated by a firework. Like the EFWA, the DFWA computes the number of sparks, s_i, for the i^{th} firework, for each of the $i = 1,2,\dots,N$ fireworks:

$$s_i = \text{round}\left(M_e \times \frac{\left(\text{Y}_{max} - f(X^i)\right)+\varepsilon}{\sum_{i=1}^{N}\left(\text{Y}_{max} - f(X^i)\right)+\varepsilon}\right), \text{where } i = 1,2,\dots,N \tag{5.21}$$

where s_i refers to the number of sparks generated from the i^{th} firework, and Y_{max} is the maximum cost value among the N fireworks in the current algorithm iteration. Note that $f(X^i)$ represents the cost of the i^{th} firework, where each of the $i = 1,2,\dots,N$. M_e is a constant to control the total number of sparks generated from the i^{th} firework ε is a small constant used to avoid the division by zero in (5.21).

5.5.1.2. Local search method

In the DFWA [23], an LS is used to perturb one or more components of the i^{th} firework to generate number of, s_i, sparks for each of the $i = 1,2,\dots,N$. The LS method introduced in the DFWA is different from the LS method in the EFWA [23], which is designed for discrete (integer) space optimization problems [30]. In the EFWA, one or more components of a firework is probabilistically selected with user-determined probability and the selected component(s) are perturbed by adding offset displacements. In the DFWA, multiple components of a firework are perturbed by using explosion radius. Note that the explosion radius is defined in proportion to the cost value of fireworks. The perturbation in the DFWA using LS method is made by exchanging/replacing multiple components of a firework. We use various neighborhood-based LS methods to generate sparks in the DFWA [30] [121]. For the neighborhood definition, we consider $\pi = (\pi_1, \pi_2, \dots, \pi_d)$ as a candidate solution, where d is the dimension of a candidate solution π.

Neighborhood search: Neighborhood search is a widely used methodology to solve combinatorial optimization problems. In neighborhood search, a solution $\pi \in \Pi$ is a vector, where Π is a set of all feasible solutions. The $c(\pi)$ is the cost of the solution π—typically called the objective function. Each solution $\pi \in \Pi$ has an associated set of *neighbors*, $\mathcal{N}(\pi) \subset \Pi$, called the neighborhood of π. Each solution $\pi' \in \mathcal{N}(\pi)$ can be reached directly from π by an operation, called a *move*, and π is said to *move* to π' when such an operation is imposed [121].

The neighborhood search method defines *neighbors* with respect to a single solution π. For a binary vector π, a simple neighborhood might be set of all single-bit changes to π. For example, if $\pi = (1,1,1,0)$, then

$$\mathcal{N}_1(\pi) = \{(0,1,1,0), (0,0,1,0), (0,1,0,0), (0,1,1,1)\}.$$

Note that π_i (where $1 \leq i \leq d$) indicates the i^{th} component in d-dimensional vector π. The local search operators used in this chapter are defined below.

i. Swap Operator

The swap operator is used to swap two adjacent components in a vector π. If the swap operator is imposed on the i^{th} component in π, we will get π', which can be denoted as follows:

$$\pi' = (\pi_1, \ldots \pi_{i-1}, \pi_{i+1}, \pi_i, \pi_{i+2}, \ldots \pi_d).$$

In other words, the swap operator swaps the component π_i with adjacent component π_{i+1} in π.

ii. Interchange Operator

The interchange operator chooses two components π_i and π_j randomly in π, and their positions are exchanged. If $i > j$, we will get π' as follows:

$$\pi' = (\pi_1, \ldots \pi_{j-1}, \pi_i, \pi_{j+1} \ldots \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots \pi_d).$$

iii. Insert Operator

The insert operator selects two components π_i and π_j randomly, not necessarily the adjacent ones, and put the first component behind the second one. We will get π' by imposing this operator on π as follows:

$$\pi' = (\pi_1, \ldots \pi_{i-1}, \pi_{i+1} \ldots \pi_j, \pi_i, \pi_{j+1}, \ldots \pi_d).$$

If $i > j$, we will get π' by imposing this operator on π as follows:

$$\pi' = (\pi_1, \dots \pi_j, \pi_i, \pi_{j+1} \dots \pi_{i-1}, \pi_{i+1}, \dots \pi_d).$$

For the broadband wireless network planning, we defined a vector of nonnegative integers $X = (\mathfrak{I}_1, \mathfrak{I}_2, \dots, \mathfrak{I}_{|T|}, \mathfrak{R}_1, \mathfrak{R}_2, \dots, \mathfrak{R}_{|R|})$, where |T| and |R| are the cardinalities of sets *T* and *R*. See section 5.4.1 for the definition of variable *X*. For the sake of convenience in the neighborhood definition, we consider $\pi = (\pi_1, \pi_2, \dots, \pi_{|T|}, \pi_{|T|+1}, \dots, \pi_d)$ as a candidate solution, where $d = |T| + |R|$ is the dimension of a candidate solution. However during implementation, we apply neighborhood-based LS methods separately on $1, 2, 3, \dots |T|$ components (i.e., $\mathfrak{I}_1, \mathfrak{I}_2, \dots, \mathfrak{I}_{|T|}$) and $|T| + 1,\ |T| + 2,\ |T| + 3, \dots |T| + |R|$ components (i.e., $\mathfrak{R}_1, \mathfrak{R}_2, \dots, \mathfrak{R}_{|R|}$) of variable *X*. Pseudo code of the Algorithm 5.1 is run once to generate a spark.

Algorithm 5.1: Generating explosion sparks in the DFWA with local search

Inputs:

- *X*: a vector of *m* component. Note that *X* is a firework (a candidate solution).

Algorithm parameters:

- $\mathbb{S}$: Explosion radius *(see 5.5.1.3)*
- Op: Local search operator *// 'inter-change', 'insert', or 'swap'*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

for *q = 1 to* $\mathbb{S}$ *// explosion radius* $\mathbb{S}$ *(see section 5.5.1.3)*
// $\mathbb{S}$ number of times a LS operator is applied to candidate solution X.
Choose ϖ, one of the local search Op ('*inter-change*', '*insert*', '*swap*'),
Apply ϖ (i.e., lower case Pi Variant) on X.
// inter-change', 'insert', or 'swap' operators are defined (see 5.5.1.2).
end

5.5.1.3. Explosion radius

The explosion radius is used to determine the number of times a local search (LS) operator is applied to perturb multiple components of the i^{th} firework X^i. The cost value of the i^{th} firework, for each of $i = 1, 2, \dots, N$, and parameters are used to determine the number of times an LS operator is applied on that firework [30]. In the DFWA [31], a firework with a lower cost value should generate sparks with smaller radius around that firework, and a firework with a higher cost function value should generate sparks with larger radius

around that firework. The rationale behind generating sparks with smaller radius is to exploit the low cost of the good firework and conduct a thorough search to find a better solution around the good firework. Sparks generated from the bad firework with larger radius are used to explore the search space and prevent the algorithm from being trapped in a local minimum. The DFWA computes the explosion radius, $\mathbb{S}_i$, for the i^{th} firework, as follows:

$$\mathbb{S}_i = round\left(\hat{a} \times \frac{f(X^i) - Y_{min} + \varepsilon}{\sum_{i=1}^{N}(f(X^i) - Y_{min}) + \varepsilon}\right), \text{ where } i = 1,2,\dots,N. \quad (5.22)$$

where $\mathbb{S}_i$ is the explosion radius associated with the i^{th} firework, Y_{min} is the minimum cost value among the N fireworks in the current algorithm iteration, $f(X^i)$ represents the cost value of the i^{th} firework, â is a constant used to control the maximum number of times an LS operator is applied on X^i, and ε is a small constant used to avoid division by zero in (5.22).

5.5.2. Mutation operator

In the DFWA, a modified mutation operator that uses the random integer function *'randi'* for the mutation is adopted. A set $\mathcal{Z}$ of fireworks to be mutated from N fireworks to set up sparks with the mutation operator, where $|\mathcal{Z}| < N$ and $|\mathcal{Z}|$ is the cardinality of set $\mathcal{Z}$. One or more components of a mutation firework $X^i \in \mathcal{Z}$ are probabilistically selected with the user-determined probability '*mutateProb*' and replaced with the new component. Like the EFWA, the DFWA generates one spark for each mutation spark $X^i \in \mathcal{Z}$ using the mutation operator as follows:

$$\widetilde{X_q^i} = randi\left(X_q^{min}, X_q^{max}\right). \quad (5.23)$$

where $\widetilde{X_q^i}$ is the component of a newly generated spark to replace X_q^i in the current algorithm generation and $X_q^{min}$and X_q^{max} are lower and upper bounds of the search space in dimension *q*. Pseudo code of the Algorithm 5.2 is run once to generate a mutation spark. Pseudo code of the Algorithm 5.2 is run once to generate a spark.

Algorithm 5.2: Generating Mutation sparks in the DFWA with local search

Inputs:

- *X*: a vector of *m* component. Note that *X* is a firework (a candidate solution).

Algorithm parameters:

- *mutateProb:* spark probability [0,1] *// user determined mutation probability.*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. ***for*** *q* = 1 to *m* *// m is number of components in X*
2. ***if*** *rand < mutateProb*
3. $\widetilde{X_q^i} = randi\left(X_q^{min}, X_q^{max}\right)$ *// perturbing the q^{th} component (see 5.5.2)*
   // note that randi() returns integer between X_q^{min} and X_q^{max}
4. ***end if***
5. ***end for***

5.5.3. Repair mechanism

Randomly generated fireworks, sparks, and mutation sparks (i.e., candidate solutions) may fall in the infeasible space after executing DFWA operations. Sparks in the infeasible space are considered useless for further algorithm operation. Therefore, infeasible sparks need to be returned to the feasible space. A candidate solution, as defined in (5.13) of the broadband wireless network (BWN) plan is infeasible if it falls outside the feasible space or violates constraints. We proposed a repair algorithm to check feasibility or repair the infeasible candidate solutions. The repair algorithm is described below.

5.5.3.1. Repair algorithm

The implementation details and pseudo code of the repair algorithm for the broadband wireless network (BWN) planning is discussed in appendix of this chapter. In this section, we concisely present the repair algorithm with pseudo code in Table 5-3. A candidate solution, *X* in (5.13), may violate one or more constraints of the BWN planning,

and therefore become infeasible. Evolutionary decisions of the experimented algorithms use feasible candidate solutions during their evolutionary operations. The proposed repair algorithm checks the feasibility of candidate solutions and repairs the infeasible ones.

The system parameters, as defined in the *section 5.2*, and a candidate solution *X* are input to the repair algorithm. The repair algorithm for the BWN planning comprises of two levels of feasibility checks: (1) feasibility of wireless links among communicating nodes (BSs, RSs, and TPs), and (2) feasible load on BSs and RSs. A communication link between any two nodes (i.e., BS, RS, TP) is feasible until flows among communicating nodes is less or equal to the maximum link (or channel) capacity. Note that the upper bound of link capacity (e.g., channel capacity) is defined in the Table 5-2. Each BS and RS has maximum load capacity, which is defined in the Table 5-2. The repair algorithm makes sure that the load on BSs and RSs should not be greater than the maximum load capacity.

The repair algorithm computes link flow between any two nodes (BSs, RSs, TPs) and then checks feasibility of their corresponding links. A communication link between any two nodes is infeasible if flow among communicating nodes (i.e., BS, RS, TP) is greater than the maximum link capacity. In case any of the communicating links among nodes are not feasible, the repair algorithm disconnects infeasible links and try to establish feasible links among corresponding nodes in steps 2−8 of Table (5-3). Similarly, the repair algorithm checks the load constraints on BSs and RSs for the second level of feasibility. The repair algorithm computes the load on BSs and RSs and checks whether the loads on the deployed BSs and RSs is greater than the maximum loads capacity. In case the load on BSs is greater than the load capacity, the repair algorithm disconnects TPs/RSs from the overloaded BSs and reconnect TPs/RSs to underloaded BSs. Similarly, if load on RSs is greater than the load capacity, the repair algorithm disconnects TPs from the overloaded RSs and reconnect TPs to underloaded BSs/RSs. A candidate soliton *X* is considered infeasible, if one or more BSs/RSs are overloaded. In contrast, BSs/RSs are considered underloaded in *X*, if current load does not exceed the maximum load capacity.

In the first level of feasibility check, the repair algorithm makes sure that candidate solution *X* should have feasible wireless links among communicating nodes such as BSs-

RSs, BSs-TPs and RSs-TPs. Then, in the second level of feasibility check, the repair algorithm checks the load feasibility on BSs and RSs. In other words, the repair algorithm checks the load constraints (on BSs and RSs) only, if the repair algorithm successfully checks the feasibility of wireless link among communicating nodes. If candidate solution *X* is irreparable during first level of feasibility check, it randomly generates a new candidate solution, *X*, and the same is repaired by executing the steps 2−8 i.e., wireless link feasibility. However, if a candidate solution *X* is irreparable due to infeasible load (till step 19), then the repair algorithm (using the steps 21−23 in the Table 5-3) randomly generates a new candidate solution, *X*, and the same is repaired by executing both the link feasibility and load feasibility checks (in steps 2−19). During two level of feasibility checks, candidate solution *X* is updated on steps 7 and 17 in the Table 5-3.

The proposed repair algorithm does not guarantee that each of the repairable (or infeasible) candidate solution will become feasible after executing the steps 2−8 and steps 14−19 in the Table 5-5. The reason is that the proposed repair algorithm is not checking each communicating wireless link and load on each of the deployed BSs/RSs exhaustively. In other words, the repair algorithm only checks for the first available feasible wireless link to replace the corresponding infeasible wireless links. Similarly, the repair algorithm only checks for the first available feasible nodes (BSs, or RSs) to replace the infeasible (or overloaded) nodes. Finally, the repair algorithm returns the feasible candidate solution at step 24 in the Table 5-3.

Table 5.3 Repair algorithm for infeasible solutions

A. Inputs	**Steps:** 1. (a) System parameters such as BSs and RSs hardware costs and maximum loads, TPs data traffic demand, path loss in various wireless links, etc. (b) Candidate solution *X*.
B. Execution	**Steps:** 2. ***for*** (check links feasibility in *X*) 3. Compute link flows among TP-RS, TP-BS, BS-RS links. *// See section 5.4.1 and 5.4.2 for link flow among nodes* 4. Disconnect an infeasible BS-RS link try to establish a feasible link between a BS-RS. 5. Disconnect an infeasible BS-TP link and try to establish a feasible link between a BS-TP/RS-TP.

	6. Disconnect an infeasible RS-TP link and try to establish a feasible links between a BS-TP/RS-TP. 7. Update the candidate solution *X*. 8. ***end for*** *// end links validation* 9. ***while*** (links in *X* is still infeasible) 10 Randomly generate a candidate solution *X*. 11. Repeat steps 2 to 8. 12. ***end while*** *// Solution X with feasible links* 13. Calculate load on each BS and RS in *X*. 14. ***for*** (load on each BS/RS in *X*) 15. ***if*** **(**load on a BS/RS is infeasible) 16. Disconnect TPs from an overloaded BS/RSs and reconnect to an underloaded BS/RS subject to maximum link capacity and maximum loads on BSs and RSs. 17. Update the candidate solution *X*. 18. ***end if*** 19. ***end for*** *// X with feasible links, BS loads, and RS loads* 20. ***while*** **(**load on BSs and RSs is still infeasible) 21. Generate a candidate solution *X* randomly. 22. Repeat lines from 2 to19. 23. ***end while*** *// X with feasible links, BS loads, and RS loads*
C. Output	24. ***return*** the feasible candidate *X*.

5.5.4. Selection operation

Each generation of the DFWA produces number of candidate solutions greater than the population of *N* fireworks. Therefore, after applying all the DFWA operators, a new population of *N* fireworks needs to be selected from the current candidate solutions. The DFWA adopts the same elitism-random selection strategy as that adopted in the enhanced fireworks algorithm (EFWA) [23], [30]. In DFWA, first, the solution with the minimum cost value is selected, then (*N-1*) candidate solutions are randomly selected from the remaining candidate solutions for the next algorithm generation.

5.5.5. DFWA operation

The pseudo code for the DFWA is shown in Table 5-4. Initially, the population of *N* fireworks is randomly generated, and algorithm parameters are initialized. After computing the cost of the *N* fireworks using (5.14) – (5.20), the number of sparks, s_i, and the explosion radius, $\mathbb{S}_i$, are computed using (5.21) and (5.22) for each of $i = 1,2,\dots,N$.

The DFWA uses any one of the local-search (LS) methods from section 5.5.1.2—*swap, insert,* and *interchange*—to perturb multiple components of a firework. This perturbation process exploits the existing small region (around a firework) and conducts a thorough search in a small region to generate sparks. Sparks generated from *N* fireworks are evaluated using the cost function (5.14).

Now, the DFWA selects a set $\mathcal{Z}$ of fireworks to be mutated from the *N* fireworks to execute the exploration process. For each firework $X^i \in \mathcal{Z}$, the mutation operator (5.23) is used to generate mutation sparks with a user-determined '*mutateProb*' probability. After executing the exploration process on the $|\mathcal{Z}|$ fireworks, the mutation sparks are evaluated using the cost function (5.14).

After performing exploitation and exploration for one algorithm generation, the DFWA selects a new population of *N* fireworks. First, the solution with the minimum cost value is selected, then (*N*-1) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation [23], [31].

Table 5.4 Discrete FWA (with local search) pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$ 2. Initialize the *sparkProb* and *mutateProb*. 3. Declare *S* as an empty set of sparks.
B. Execution	4. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 5. ***while*** (stopping criteria not satisfied) 6. ***for*** $i = 1,2, \dots, N$ 7. Calculate the number of sparks s_i and the explosion radius $\mathbb{S}_i$ for the i^{th} Firework X^i using (5.21) and (5.22) respectively. 8. ***for*** j = 1 to s_i 9. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 5.1.** 10. Add generated sparks in the set *S*. 11. ***end for*** 12. ***end for*** 13. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 5.5.2*) from a population of *N* fireworks. 14. ***for*** each firework X in $\mathcal{Z}$ 15. Generate mutation spark $\check{X}$ using **Algorithm 5.2**.

	16. Add generated spark in the set *S*. 17. ***end for*** 18. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 19. Select the best solution, and (*N*–1) solutions to make a new population of the *N* fireworks. 20. ***end while***
C. Output	21. ***return*** the best solution found so far.

5.6. Proposed DFWA with an ensemble of LS methods

We propose a discrete FWA (DFWA) with an ensemble of local search (LS) methods i.e., '*insert,*' '*interchange,*' and '*swap*'. Typically, in one DFWA generation with a user-defined stopping criterion, a single LS method is used to perturb multiple components of a firework to generate sparks. A LS method is used as a perturbation to replace one or more components of a firework in the DFWA. An ensemble of LS methods can also be used [30], [31] to perturb one or more components of a firework. Multiple LS method can be used with different variations from an ensemble of LS methods in each DFWA generation. The ensemble in the proposed DFWA consists of the combinations of '*insert,*' '*interchange,*' and '*swap*' LS methods.

In the next subsections, two ways to combine LS methods in a DFWA are presented. These algorithms are: (1) the DFWA with fixed-rate ensemble of LS methods (DFWA-with-FR-3-LS), and (2) the DFWA with dynamic ensemble of LS methods such as DFWA-with-Dy-3-LS.

5.6.1. DFWA with ensemble of fixed-rate (FR) local search methods

We used the LS methods (i.e., '*insert*', '*interchange*', and '*swap*') individually in the DFWA for the BWN planning. In this experiment, several performance metrics were recorded for the experimented LS methods such as cost of the objective function value, CPU time, and standard deviation. Using any one of the recorded metrics, experimented LS methods can be ranked based on their individual performance. In the above experiment,

'*insert*', '*swap*' and '*interchange*' LS methods in the DFWA are ranked as the first, second and third performers in terms of their average cost for the BWN planning. In the DFWA-with-FR-3-LS, this predetermined ranking information is used to build an ensemble of LS methods to be used in the DFWA. For example, a better performing LS method has a higher probability of being selected than a relatively poorly performing LS method. After testing the performance of individual LS methods in previous experiments, the better performing LS method is assigned a higher user-determined probability ∂_1 as compared to the relatively poorly performing LS methods are assigned with lower user-determined probabilities ∂_2, and ∂_3 respectively.

Like the DFWA, the DFWA-with-FR-3-LS has four basics operations: an explosion operator, a mutation operator, a repair mechanism, and a selection operation. Except for the LS methods in the explosion operator, the operators in the DFWA-with-FR-3-LS are the DFWA. In this section, we discuss the explosion operator using the ensemble of LS methods in the DFWA-with-FR-3-LS.

5.6.1.1. Explosion operator

In the DFWA-with-FR-3-LS, cost value and control parameters are used to determine the criteria of the explosion operator. The explosion operator uses ensemble of LS methods with two parameters: explosion strength and explosion radius. In the DFWA-with-FR-3-LS, the explosion operator determines the number of sparks and the radius of those sparks in proportion to the cost value of fireworks.

A. Explosion strength

In the DFWA-with-FR-3-LS, the explosion strength determines the number of sparks that are generated by the explosion of a firework. The cost of a firework and user-defined parameters determine the number of sparks that are generated by a firework. The DFWA-with-FR-3-LS computes the number of sparks, s_i, for the i^{th} firework in (5.21), where $i = 1,2, \dots, N$, as in the DFWA (*See section 5.5.1–A*).

B. Selecting an LS method with user-determined probability from an ensemble of LS methods

In the DFWA-with-FR-3-LS, three local search methods '*insert*', '*interchange*', and '*swap*' are used simultaneously to generate sparks in one algorithm generation. The performance variation in the DFWA is observed using different LS methods individually [30]. In the absence of any scientific methodology to select a better performing LS method for the DFWA, we propose a simultaneous use of multiple LS methods, which is also known as ensemble of LS methods. In one generation of the DFWA-with-FR-3-LS, a selection criterion is devised to use a LS method from the ensemble (i.e., '*insert*', '*interchange*', and '*swap*'). The ensemble of three LS methods is used for fireworks explosion in the DFWA-with-FR-3-LS to generate sparks.

Three local search (LS) methods are labelled *Op1, Op2, Op3* and are assigned user-determined probabilities ∂_1, ∂_2, and ∂_3 (i.e., in (0,1)), respectively, according to their individual performances in the DFWA for the BWN planning. Note that sum of the user-determined probabilities assigned to the ∂_1, ∂_2, and ∂_3 is 1. The rationale of assigning different values to various LS methods (i.e., *Op1, Op2,* and *Op3*) in the ensemble is to use LS methods according to their predetermined performances (from previous experiment) in the DFWA. Let us assume that the LS method *Op1* associated with user-determined probability ∂_1 is the best performer and it has a higher chance of being selected as compared to other LS operators (*Op2* and *Op3*). Similarly, LS method *Op2* associated with user-determined probability ∂_2 has a higher chance of being selected in the DFWA as compared to the LS method *Op3*. In contrast, the LS method *Op3* associated with a user-determined probability ∂_3 has a lower chance of being selected as compared to LS methods *Op1* and *Op2* in the DFWA. This way, we use three LS methods simultaneously to generate sparks in one algorithm generation of the DFWA-with-FR-3-LS.

The DFWA-with-FR-3-LS probabilistically selects an LS method from the ensemble of LS methods with user-determined probabilities ∂_1, ∂_2, and ∂_3 associated with LS methods *Op1, Op2,* and *Op3* respectively. The selected LS method then used to perturb $\mathbb{S}_i$ times (changing/exchanging components of a firework) to generate a spark. In other

words, the selected LS method is imposed on the i^{th} firework $\mathbb{S}_i$ times to generate a spark, where $\mathbb{S}_i$ is an integer that represents the explosion radius. This way, the number of sparks, s_i, are generated using probabilistically selected LS methods for each of the i^{th} firework, where $i = 1,2, \dots, N$. Pseudo code of the Algorithm 5.3 is run once to generate an explosion spark in the DFWA-with-FR-3-LS.

Algorithm 5.3: Generating explosion sparks in the DFWA-with-FR-3-LS

Inputs:

- X: a vector of *m* component. Note that X is a firework (a candidate solution).

Algorithm parameters:

- ∂_1, ∂_2, and ∂_3: probabilities to select local search operators (Op) (0,1) according to predetermined performance of these operators *(see 5.6.1.1-B).*
- Op: Local search operators // *Op1 = 'inter-change', Op2 = 'insert', or Op3 = 'swap'*
- $\mathbb{S}$: Explosion radius *(see 5.5.1.3).*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. *randNum = rand () // Generate random number in (0,1)*
2. ***for*** *q* = 1: $\mathbb{S}$ *// number of times an Op applied on a firework (see 5.6.1.1 -C)*
3. ***if*** *randNum* <= ∂_1 *// Say* ∂_1 *= 0.6* ---here ∂ partial differential.
4. Choose ϖ local search operator *Op1*,
5. Apply ϖ (lower case Pi Variant) on X.
6. ***end***
7. ***if*** *randNum* > ∂_2 **&** *sparkProb* <= ∂_3 *// Say* ∂_2 *> 0.6 &* ∂_3 *< 0.9*
8. Choose ϖ local search operator *Op2,*
9. Apply ϖ on X.
10. ***end***
11. ***if*** *randNum* > ∂_3
12. Choose ϖ local search operator *Op3*
13. Apply ϖ (lower case Pi Variant) on X.
14. ***end***

// for local search 'interchange', 'insert', or 'swap' operators (see 5.5.1.2).

15.***end for***

C. Explosion radius

The explosion radius is an integer value used to determine the number of times a local search (LS) operator is applied to perturb one or more components of the i^{th} firework. The DFWA-with-FR-3-LS computes the explosion radius, $\mathbb{S}_i$, for the i^{th} firework, in (5.22).

5.6.1.2. DFWA-with-FR-3-LS operation

The pseudo code for the DFWA with an ensemble of three fixed-rate LS methods (DFWA-with-FR-3-LS) is shown in Table 5-5. Initially, a population of *N* fireworks is randomly generated, and algorithm parameters are initialized. After computing the cost value of the *N* fireworks using (5.14) – (5.20), the number of sparks, s_i, and the explosion radius, $\mathbb{S}_i$, are computed using (5.21) and (5.22), respectively, for each of the $i = 1,2, \dots, N$. The DFWA-with-FR-3-LS uses an ensemble of local search (LS) methods (*swap*, *insert*, *interchange*) to perturb one or more components of a firework using explosion radius (*section 5.6.1.1-B*). This perturbation process exploits the small region around a firework and a thorough search is conducted in that small region to generate sparks. All the sparks generated from the *N* fireworks are evaluated by using the cost function (5.14).

Now, the DFWA-with-FR-3-LS selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of *N* fireworks to execute the exploration process. For each firework $X^i \in \mathcal{Z}$, the mutation operator (5.23) is used to generate mutation sparks with user-determined '*mutateProb*' probability. After executing the exploration process on the $|\mathcal{Z}|$ fireworks, the mutation sparks are evaluated by using the cost function (5.14).

After performing exploitation and exploration for one DFWA-with-FR-3-LS generation, a new population of *N* fireworks is selected. First the solution with the minimum cost value is selected, then (*N*-1) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation [23], [31].

Table 5.5 DFWA-with-FR-3-LS pseudo code

A. Initialization	1. Randomly generate a population of the *N* fireworks, $X^i, i = 1,2, \dots, N$ 2. Initialize the *mutateProb*. 3. Local search operators *Op1, Op2,* and *Op3*.

	4. ∂_1, ∂_2, and ∂_3 user-determined probability to select local search operators (Op) [0,1], according to predetermined performance of these operators *(see 5.6.1.1).* 5. Declare *S* as an empty set of sparks.
B. Execution	6. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 7. ***while*** (stopping criteria not satisfied) 8. ***for*** $i = 1,2, \dots, N$ 9. Calculate the number of sparks s_i and the explosion radius $\mathbb{S}_i$ for the i^{th} Firework X^i using (5.21) and (5.22) respectively. 10. ***for*** j = 1 to s_i 11. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 5.3.** 12. Add generated sparks in the set *S*. 13. ***end for*** 14. ***end for*** 15. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 5.5.2*) from a population of *N* fireworks. 16. ***for*** each firework X in $\mathcal{Z}$ 17. Generate mutation spark $\check{X}$ using **Algorithm 5.2**. 18. Add generated spark in the set *S*. 19. ***end for*** 20. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 21. Select the best solution, and (*N*-1) solutions to make a new population of the *N* fireworks. 22. ***end while***
C. Output	23. ***return*** the best solution found so far.

5.6.2. DFWA with an ensemble of dynamic local search methods

In the DFWA-with-FR-3-LS, we used predetermined ranking information to assign user-determined probabilities to the LS methods in the ensemble. This way of building ensemble can be a tedious job either due to ensemble contains a large number of LS methods or due to limited availability of predetermined ranking information about LS methods. To avoid these difficulties, we propose the DFWA with an ensemble of dynamic 3 LS methods (DFWA-with-Dy-3-LS). In the next subsections, first we discuss the potential disadvantages of the fixed-rate LS methods (i.e., DFWA-with-FR-3-LS), and then we discuss the proposed dynamic use of LS methods in the DFWA.

5.6.2.1. Disadvantage of fixed-rate LS methods

In the *section 5.6.1*, ensemble of three LS methods is used for firework explosion to generate sparks in the DFWA-with-FR-3-LS. In the DFWA-with-FR-3-LS, selecting a LS method probabilistically from the ensemble may choose poor performing LS methods more frequently than the better performing LS methods. This probabilistic selection of LS methods from ensemble is one of the disadvantages that may affect the overall performance of the DFWA-with-FR-3-LS.

Assigning fixed probabilities to LS methods is relatively easier if (1) ensemble of LS methods consists of relatively small number (two or three) of LS methods (2) predetermined information about the LS methods is useful to clearly rank the LS methods in the ensemble. In contrast, assigning fixed probabilities to LS methods can be a tedious job, if (1) ensemble of LS methods consists of large number (say10 or more) of LS methods, (2) predetermined information about the LS methods may be unable to clearly rank the LS methods in the ensemble. Without scientific information, assigning the LS methods in the 2nd scenario may affect the overall performance of the DFWA-with-FR-3-LS. Therefore, we propose DFWA-with-Dy-3-LS to avoid such type of potential disadvantages.

5.6.2.2. Dynamic LS methods

Selecting an LS method for the DFWA (or any algorithm) is a tedious job, especially when uncertainty in the performance of the LS operators is observed as in [30]. In the absence of experimental evidence and in the presence of variations in the performance of LS methods, a dynamic ensemble of local search methods can avoid a fixed-rate selection of LS methods that might be unsatisfactory. We present the DFWA-with-Dy-3-LS with an ensemble of three dynamic LS methods (i.e., '*insert,*' '*interchange,*' and '*swap*'). In contrast to the DFWA-with-FR-3-LS, the DFWA-with-Dy-3-LS in each generation can toggle LS methods from ensemble. Initially, LS method is randomly assigned to each of the i^{th} Firework, for each $i = 1,2, \dots, N$. From the 2nd generation onward in the DFWA-with-Dy-3-LS, if a LS method cannot produce better spark(s), than LS method for the current firework is replaced randomly with one of the remaining LS

methods from ensemble. This way, LS methods in ensemble can be dynamically used with search progress during the DFWA-with-Dy-3-LS operation.

Like the DFWA, the DFWA-with-Dy-3-LS has four basics operations: an explosion operation, a mutation operation, a repair mechanism and a selection operation. Except for the LS method, the operators in the DFWA-with-Dy-3-LS are like the operators in the DFWA.

5.6.2.3. Explosion operator

In the DFWA-with-Dy-3-LS, the cost value and the parameters determine the criteria of the explosion operator. The explosion operator uses LS methods with two parameters: explosion strength and explosion radius. The DFWA-with-Dy-3-LS explosion operator determines the number of sparks and the radius of those sparks in proportion to the cost value of fireworks.

A. **Explosion strength**

In the DFWA-with-Dy-3-LS, the explosion strength determines the number of sparks that are generated by the explosion of a firework. The cost of a firework and user-defined control parameters determine the number of sparks that are generated by a firework. The DFWA-with-Dy-3-LS computes the number of sparks, s_i, for the i^{th} firework, where $i = 1,2, \dots, N$, as in (5.20).

B. **Dynamically selecting an LS method from an ensemble of LS methods**

In the DFWA-with-Dy-3-LS, we propose a new algorithm to dynamically select an LS method from an ensemble of LS methods. We denote three LS methods as a set of integers $\wp$ (i.e., calligraphic lower case p). For example, the set $\wp = \{\wp_1, \wp_2, \wp_3\}$ is an ensemble of LS methods in which each element $\wp_i$ represents a LS method such as $\wp_1 = 1$ (LS operator 1), $\wp_2 = 2$ (LS operator 2), $\wp_3 = 3$ (LS operator 3).

Algorithm 5.4: Generating explosion sparks in the DFWA-with-Dy-3-LS

Inputs:

- X: a vector of m components. Note that X is a firework (a candidate solution).

Algorithm parameters:

- $\wp = \{\wp_1, \wp_2, \wp_3\}$ *// $\wp_1 = 1$ (LS operator 1), $\wp_2 = 2$ (LS operator 2), $\wp_3 = 3$ (LS operator 3) (see 5.6.2.3-B).*
- ϱ: a vector of *N* components

// Each component of ϱ represents a local search operator (Op) associated with //corresponding firework in the population of N fireworks (see 5.6.2.3-B).

- $\mathbb{S}$: Explosion radius *(see 5.6.2.3-C).*

Output:

- $\check{X}$, a *spark, a vector of m components*

Steps:

1. ***for*** *q* = 1: $\mathbb{S}$ *// number of times an Op applied on a firework (see 5.6.2.3-C)*
2. Choose a ϖ local search operator from the vector ϱ (i.e. lower case Rho variant),
3. Apply ϖ (i.e., lower case pi variant) on X.
   // Note that each component of vector ϱ represents a local search operator.
   // for perturbation, any one of the associated 'interchanges', 'insert', or 'swap'
   // local search operators in ϱ can be used (see 5.6.2.3-B).
4. ***end for***

Initially, the DFWA-with-Dy-3-LS generates randomly an integer vector ϱ with *N* components. Each of the i^{th} component in integer vector ϱ is considered an associated LS method to the i^{th} Firework, for each $i = 1,2, \dots, N$. In the first generation of the DFWA-with-Dy-3-LS, each firework generates, s_i, number of sparks using assigned LS method (i.e., from ϱ), where $i = 1,2, \dots, N$, and the generated sparks are evaluated by using the cost function (5.14). From the 2nd generation of the DFWA-with-Dy-3-LS, if there is no improvement observed in the cost value of the sparks generated from the i^{th} firework for each of $i = 1,2, \dots, N$, then currently assigned LS method is replaced for the i^{th} firework (i.e., i^{th} component of the integer vector ϱ) to be used in the next algorithm generation. The replaced LS method is randomly selected from the remaining LS methods in the ensemble for the i^{th} firework, where $i = 1,2, \dots, N$.

For example, the population of fireworks is *N* =10, set of LS methods is $\wp = \{\wp_1, \wp_2, \wp_3\}$, where each element of the set $\wp$ is an integer such as $\wp_1 = 1$, $\wp_2 = 2$, $\wp_3 = 3$. In the DFWA-with-Dy-3-LS, each of the *N* components of vector ϱ is randomly assigned an integer from the set $\wp$ such as ϱ = (3, 3, 1, 2, 2, 3, 3, 1, 1, 1). Each of the i^{th} component of the ϱ represents the LS method associated with the i^{th} firework. For example, in the vector ϱ of *N* components, the first firework is associated with the 3rd LS method while the last firework is associated with the 1st LS method. In the 1st generation, each of the i^{th}

firework is generating sparks using associated LS method in the i^{th} component of ϱ. If no improvement is observed in the sparks generated from the first firework using 3rd LS method, then 3rd LS method is replaced with randomly selected LS method from the ensemble and the updated vector may be expressed as: ϱ = (2, 3, 1, 2, 2, 3, 3, 1, 1, 1). Now, in the 2nd generation, the first firework is using 2nd LS method to generate explosion sparks. Pseudo code of the Algorithm 5.4 is run once to generate an explosion spark.

C. **Explosion radius**

The explosion radius is an integer value used to determine the number of times a local search (LS) operator is applied to perturb one or more components of the i^{th} firework. The cost of the i^{th} firework, for each of $i = 1,2, \dots, N$ fireworks, and parameters are used to determine the number of times an LS method is applied on that firework [30]. The DFWA-with-Dy-3-LS computes the explosion radius, $\mathbb{S}_i$, for the i^{th} firework, as in (5.22).

5.6.2.4. DFWA-with-Dy-3-LS operation

The pseudo code of the DFWA-with-Dy-3-LS is shown in Table 5-6. Initially, the population of *N* fireworks is randomly generated, and algorithm parameters are initialized. After computing the cost of the *N* fireworks using (5.14) – (5.20), the sparks, s_i, and the explosion radius, $\mathbb{S}_i$, are computed using (5.21) and (5.22) for each of the $i = 1,2, \dots, N$ fireworks. The DFWA-with-Dy-3-LS uses the LS methods *swap*, *insert*, and *interchange* dynamically to perturb one or more components of a firework by imposing local search operator $\mathbb{S}_i$ times. This perturbation exploits the existing small region around a firework and a thorough search is conducted in a small region to generate sparks. All the sparks generated from the *N* fireworks are evaluated using the cost function (5.14).

Now, the DFWA-with-Dy-3-LS selects a set $\mathcal{Z}$ of fireworks to be mutated from the population of *N* fireworks to execute the exploration process. For each firework $X^i \in \mathcal{Z}$, the mutation operator (5.23) is used to generate mutation sparks with user-determined '*mutateProb*' probability. After executing the exploration process on the $|\mathcal{Z}|$ fireworks, the mutation sparks are evaluated by using the cost function (5.14).

After performing exploitation and exploration for one algorithm generation, the DFWA selects the new population from *N* fireworks. In the DFWA-with-Dy-3-LS, first the solution with the minimum cost value is selected, then, (*N*-1) fireworks are selected randomly from the remaining candidate solutions for the next algorithm generation [23], [31].

Table 5.6 DFWA-with-Dy-3-LS pseudo code

A. Initialization	1. Randomly generate a population of *N* fireworks and initialize the *mutateProb*. 2. $\wp = \{1, 2, 3\}$ *// each element of set* $\wp$ *represents a LS operator* *// (see 5.6.2.3-B).* 3. ϱ : Randomly generate a vector of *N* components in the set $\wp$. *// Each component of* ϱ *represent in* the set $\wp$ (*see 5.6.2.3-B).* 4. Declare *S* be the empty set of sparks.
B. Execution	5. Check the feasibility of the *N* fireworks or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 6. ***while*** (stopping criteria not satisfied) 7. ***for*** $i = 1,2, \dots, N$ 8. Calculate the number of sparks s_i and the explosion radius $\mathbb{S}_i$ for the i^{th} Firework X^i using (5.21) and (5.22) respectively. 9. ***for*** j = 1 to s_i 10. Generate j^{th} explosion spark $\widetilde{X^j}$ using **Algorithm 5.4** 11. Add generated sparks in *S*. 12. ***end for*** 13. Check the feasibility of the sparks in *S* using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 14. i^{th} component of the vector ϱ (associated with the i^{th} Firework) is replaced with randomly selected remaining local search methods, if generated sparks are not better than the corresponding fireworks. 15. ***end for*** 16. Randomly select a set $\mathcal{Z}$ of fireworks to be mutated (*see 5.5.2*) from a population of *N* fireworks. 17. ***for*** each firework X in $\mathcal{Z}$ 18. Generate mutation spark $\check{X}$ using **Algorithm 5.2**. 19. Check the feasibility of all the sparks in *S* or repair the infeasible ones using the repair algorithm in Table (5-3) and evaluate using the cost function in (5.14). 20. Add generated spark in *S*. 21. ***end for***

	23. Select the best solution, and (*N*-1) solutions from *S* to make a new population of the *N* fireworks for next algorithm generation. 23. ***end while***
C. Output	24. ***return*** the best solution found so far.

5.7. Results and Discussion

5.7.1. Simulation setup

We defined parameters for the broadband wireless network (BWN) with single-hop planning problem as formulated in *section 5.3*. The experiment was conducted with eight different problem instances. Problem specific parameters such as number of BSs, RSs, and TPs are shown in Table 5-7, and the algorithm parameters are shown in Table 5-8.

Table 5.7 Algorithm specific parameters

Algorithms	Algorithm parameters	Common parameters
DFWA and its variants	Mutation Prob. = 0.01, $\mathbb{S}_i$, times LS Methods are applied on each firework, # of fireworks = 10, # of Mutation fireworks = 5 Maximum # of Sparks = 40, Minimum # of sparks = 2	Population size: 30
GA	Mutation Prob. = 0.01, Probability of crossover = 0.9, Probability of selection = 0.5.	# of Fireworks:10 # of mutation Fireworks: 5
Low-complexity BBO	λ is defined is in chapter 2. Emigrating method is in chapter 2. Probability of mutation = 0.01	
Discrete ABC	Limit trial t = 1.2×Population size	

The TP (users) demand is a real vector which is randomly generated in the interval [0.01 3.0]. The cost for each installed BS and RS is set as: BS = 25 and RS = 5. The real matrices representing the path loss for each link $l_{b,r}^{BR}$, $l_{b,t}^{BT}$ and $l_{r,t}^{RT}$ were randomly generated using the Matlab '*rand*' function. The maximum link rate $m_{b,r}^{BR}$, $m_{b,t}^{BT}$ and $m_{r,t}^{RT}$ for each existing link (i.e., $x_{b,r}^{BR}$ =1, $x_{b,t}^{BT}$=1, $x_{r,t}^{RT}$=1) is defined in Tables 5-9 and 5-10.

Table 5.8 Parameters

Problem instance #	Parameters			
	# of TPs	# of BSs sites	# of RSs sites	# of function evaluations
1	100	10	20	1500
2	200	20	40	5000
3	300	50	24	8000
4	400	70	34	10000
5	500	80	40	12000
6	600	92	46	15000
7	700	100	50	18000
8	800	112	54	20000

Table 5.9 BS to RS link rate

Path loss ($l_{b,r}^{BR}$)	Link Rate ($m_{b,r}^{BR}$ *mbps*)
<= 0.2	*20*
<= 0.4	*18*
<= 0.6	*16*
<= 0.8	*14*
<=0.9	*12*
else	*10*

Table 5.10 BS/RS to TP link rate

Path loss ($l_{b,t}^{BT}$& $l_{r,t}^{RT}$)	Link Rate ($m_{b,t}^{BT}$ & $m_{r,t}^{RT}$ *mbps*)
<= 0.2	*4.0*
<= 0.4	*3.5*
<= 0.6	*3.0*
<= 0.8	*2.0*
<=0.9	*1.0*
else	*0.5*

5.7.2. Performance

We used three performance metrics: Average cost, Average CPU time (seconds) and standard deviation to record the performance of experimented algorithms. The results presented in this chapter are the average of 100 independent trails of each problem instance. In Table 5-11, we recorded the performance metrics of the discrete fireworks algorithm (DFWA) with three different local search (LS) methods and the same performance metrics for the low-complexity biogeography-based optimization (LC-BBO) algorithm, the discrete artificial bee colony (DABC) algorithm and the genetic algorithm (GA) are presented in the Table 5-12. Similarly, the performance metrics for the DFWA with a dynamic ensemble of three LS methods (DFWA-with-Dy-3-LS), the DFWA with a dynamic ensemble of two local search methods (DFWA-with-Dy-2-LS), and the DFWA with a fixed-rate ensemble of three LS methods (DFWA-with-FR-3-LS) are recorded in the Table 5-13. The number of objective function evaluations is the stopping criteria for the experimented algorithms as mentioned in the 5th column of the Table 5.8.

Table 5.11 DFWA using various LS operators

Problem instance #	**Algorithms**					
	DFWA-*Insert*		**DFWA-*Swap***		**DFWA-*Interchange***	
	Average Cost (Std.)	Avg. Matlab CPU time (sec.)	Average Cost (Std.)	Avg. Matlab CPU time (sec.)	Average Cost (Std.)	Avg. Matlab CPU time (sec.)
1	99.38(6.6)	6.97	143.42(10.5)	4.27	177.55(8.2)	4.36
2	186.02(11.8)	35.06	284.13(17.4)	18.43	337.58(15.3)	18.55
3	245.21(9.4)	63.94	377.02(21.5)	32.96	427.01(15.2)	32.84
4	335.40(23.7)	120.56	531.42(25.4)	68.10	592.24(17.5)	70.65
5	409.93(19.9)	168.04	649.97(24.4)	94.50	708.14(20.7)	99.32
6	516.86(19.5)	244.27	745.96(28.2)	135.13	808.84(20.8)	143.18
7	574.51(28.3)	308.17	840.98(32.2)	168.65	900.57(22.4)	181.93
8	667.10(25.6)	415.05	928.25(24.6)	212.90	980.96(20.4)	234.47

We plotted the performance of the DFWA-with-Dy-3-LS for three metrics: Average cost, Average CPU time (seconds) and standard deviation against a group of experimented algorithms. First, we plotted the performance of DFWA-with-Dy-3-LS against the DFWA-*insert*, DFWA-*swap*, and the DFWA-*interchange* for Average cost, Average CPU time (seconds) and standard deviation (of cost) in Figures 5.2, 5.5 and 5.8 respectively. Secondly, we plotted the performance of DFWA-with-Dy-3-LS against the

Table 5.12 Results for Discrete ABC, BBO, and GA

Problem instance #	**Algorithms**					
	Low-complexity BBO		**Discrete ABC**		**GA**	
	Average Cost (Std.)	Average Matlab CPU time (sec.)	Average Cost (Std.)	Average Matlab CPU time (sec.)	Average Cost (Std.)	Avg. Matlab CPU time (sec.)
1	156.39(3.9)	4.56	165.09(6.5)	3.99	166.65(6.5)	3.23
2	298.61(11.0)	26.31	345.88(6.5)	28.48	350.73(5.0)	23.96
3	362.21(12.4)	52.77	442.78(6.0	56.88	447.32(5.7)	49.03
4	508.53(15.6)	75.25	627.73(8.4)	101.50	633.27(7.1)	90.67
5	594.91(21.5)	107.41	752.04(8.0)	146.23	756.85(5.9)	132.67
6	671.08(21.6)	160.52	868.07(8.3)	243.10	873.15(7.2)	221.72
7	741.90(21.1)	196.89	963.11(7.2)	202.08	968.25(7.2)	179.59
8	807.85(22.3)	255.15	1050.40(7.5)	282.55	1056.94(6.3)	251.58

LC-BBO, DFWA and GA for Average cost, Average CPU time (seconds) and standard deviation in Figures 5.3, 5.6 and 5.9 respectively. Lastly, we plotted the performance of DFWA-with-Dy-3-LS against the DFWA-with-Dy-2-LS and DFWA-with-FR-3-LS for Average cost, Average CPU time (seconds) and standard deviation in Figures 5.4, 5.7 and 5.10 respectively.

In the first experiment, we used three local search (LS) methods ('*insert*,' '*interchange*,' and '*swap*') one-by-one in the discrete fireworks algorithm (DFWA) and

Table 5.13 DFWA using various LS operators

Problem instance #	**Algorithms**					
	DFWA-with-Dy-3-LS		**DFWA-with-Dy-2-LS**		**DFWA-with-FR-3-LS**	
	Average Cost (Std.)	Avg. Matlab CPU time (sec.)	Average Cost (Std.)	Avg. Matlab CPU time (sec.)	Average Cost (Std.)	Avg. Matlab CPU time (sec.)
1	104.42(8.6)	5.89	103.96(8.6)	5.58	110.30(9.4)	5.20
2	195.24(14.2)	32.32	194.17(13.8)	35.56	209.45(15.0)	30.52
3	256.65(15.3)	68.09	255.34(13.6)	67.98	276.19(17.5	60.16
4	356.32(23.7)	103.00	361.29(22.0)	105.73	384.06(21.7)	90.95
5	442.64(26.2)	141.81	447.28(25.9)	150.76	474.66(26.1)	124.84
6	552.45(26.3)	206.93	547.98(24.4)	216.24	579.13(21.9)	180.97
7	612.39(29.3)	264.08	604.42(27.0)	272.93	638.66(30.5)	220.29
8	703.95(33.0)	334.01	702.48(28.8)	343.24	736.03(29.4)	281.09

compared the results against the DFWA-with-Dy-3-LS. The DFWA-*insert* outperformed the DFWA-*swap*, the DFWA-*interchange*, and the DFWA-with-Dy-3-LS in terms of average cost as shown in Figure 5.2 and Tables 5-11 and 5-13. However, the DFWA-with-Dy-3-LS performs better than the DFWA-*swap* and the DFWA-*interchange* for the average cost. The DFWA-*insert* consumed a higher average Matlab CPU time than DFWA-*swap*, DFWA-*interchange*, and the DFWA-with-Dy-3-LS as shown in Figure 5.5. The standard deviation (of cost) for the DFWA-*insert* is the smaller than the standard deviation of the DFWA-*swap*, the DFWA-*interchange*, and the DFWA-with-Dy-3-LS for the instances with TPs 100 to 300 and TPs 500 to 600 as shown in Figure 5.8. For the remaining instances with TPs 400, 700 and 800, the standard deviation of cost of the DFWA-*interchange* is smaller than the DFWA-with-Dy-3-LS, DFWA-*swap*, and DFWA-*insert*.

In the second experiment, we compared the performance of the DFWA-with-Dy-3-LS against the LC-BBO algorithm, the discrete ABC (DABC) algorithm, and the genetic algorithm (GA). The DFWA-with-Dy-3-LS algorithm outperformed the LC-BBO, the discrete ABC algorithm, and the GA in terms of average cost as shown in Figure 5.3 and Tables 5-12 and 5-13. The performance of the DABC and the GA is comparable but LC-BBO outperforms both DABC and GA for the average cost. The standard deviation (Std.) for the GA is the smaller than the standard deviation the LC-BBO, the discrete ABC, and the DFWA-with-Dy-3-LS except for the instance with TPs 100. On other hand, standard deviation for the DABC is higher than the LC-BBO, DFWA-with-Dy-3-LS, and GA as shown in Figure 5.9 and Tables 5-12 and 5-13. The DFWA-with-Dy-3-LS consumed a higher average Matlab CPU time than low-complexity BBO, discrete ABC and GA except for the instances with TPs 500 and 600 as shown Figure 5.6 and Tables 5-12 and 5-13.

In the third experiment, we compared the performance of the DFWA-with-Dy-3-LS against the DFWA-with-Dy-2-LS, and DFWA-with-FR-3-LS. The performance of the DFWA-with-Dy-3-LS and DFWA-with-Dy-2-LS is comparable for the average cost value. However, DFWA-with-Dy-3-LS outperformed the DFWA-with-FR-3-LS in terms of average cost as shown in the Figure 5.4 and Tables 5-13. As far as standard deviation of cost is concerned, standard deviation of the DFWA-with-Dy-3-LS is higher than the standard deviation of the DFWA-with-FR-3-LS except for the instance with TPs 300 as shown in the Figure 5.10 and Table 5-13. On the other hand, DFWA-with-Dy-2-LS consumed a higher average Matlab CPU time as compared to the DFWA-with-FR-3-LS and DFWA-with-Dy-3-LS as shown in the Figure 5.7 and Table 5-13.

Overall, the DFWA-*insert* is the best and DFWA-with-Dy-3-LS algorithm is 2nd best performing algorithm as compared to rest of the all algorithms in terms of average cost as shown in the Figures 5.2−5.4. Highlights of the experiments are:

1- Performance differences in DFWA-*insert*, DFWA-*swap*, and DFWA-*interchange* suggest that it may be inefficient to randomly select and or give priority to one local search over another local search method in the DFWA.

2- The DFWA-with-FR-3-LS algorithm did not perform better than the dynamic use of LS methods such as DFWA-with-Dy-3-LS and DFWA-with-Dy-2-LS algorithms.

3- In the absence of experimental and scientific data for LS methods, an ensemble of dynamic LS methods is a good choice for the BWN planning.

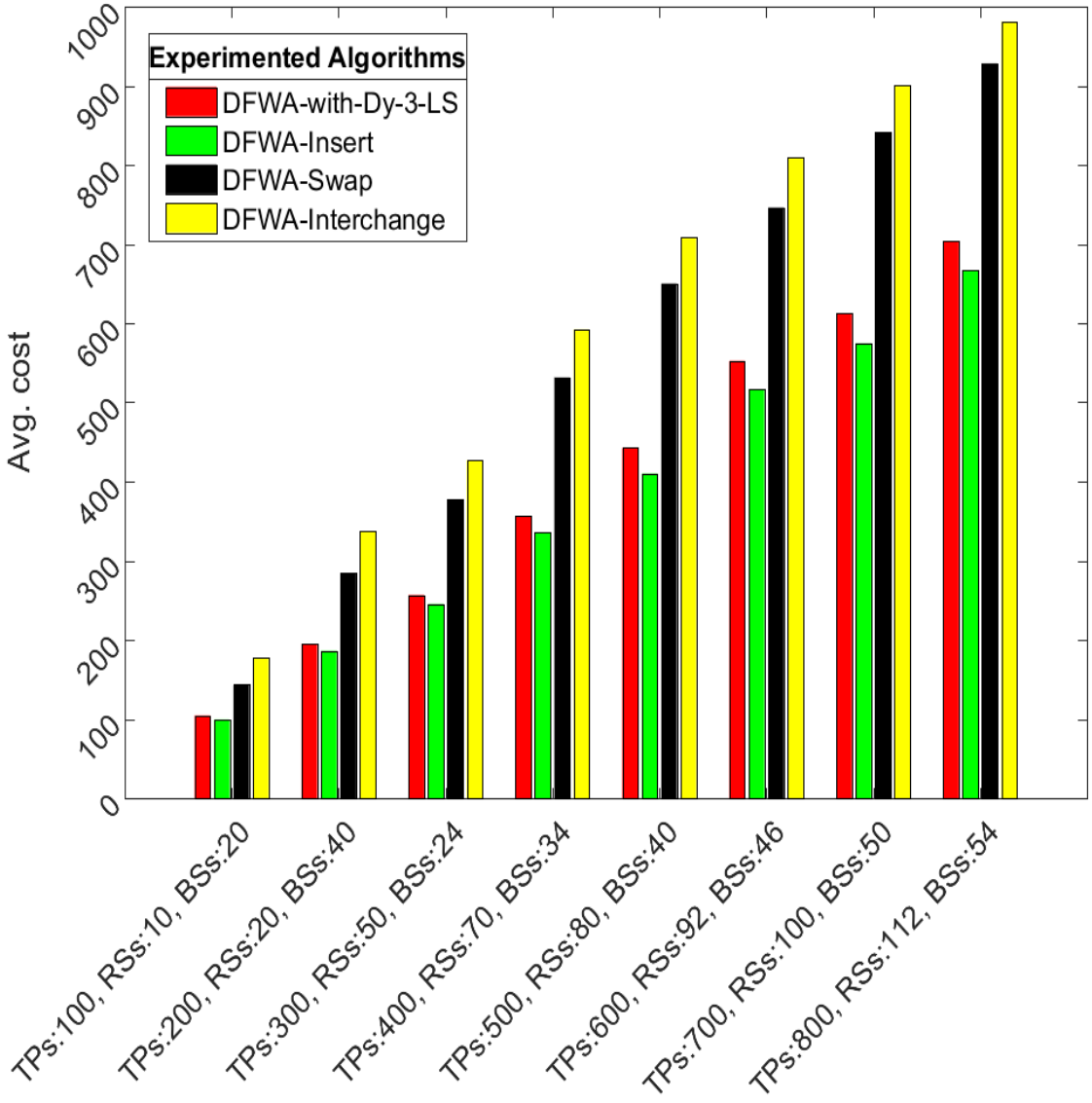


Figure 5.2 Avg. cost of DFWA-with-Dy-3-LS vs. DFWA with three individual LS methods

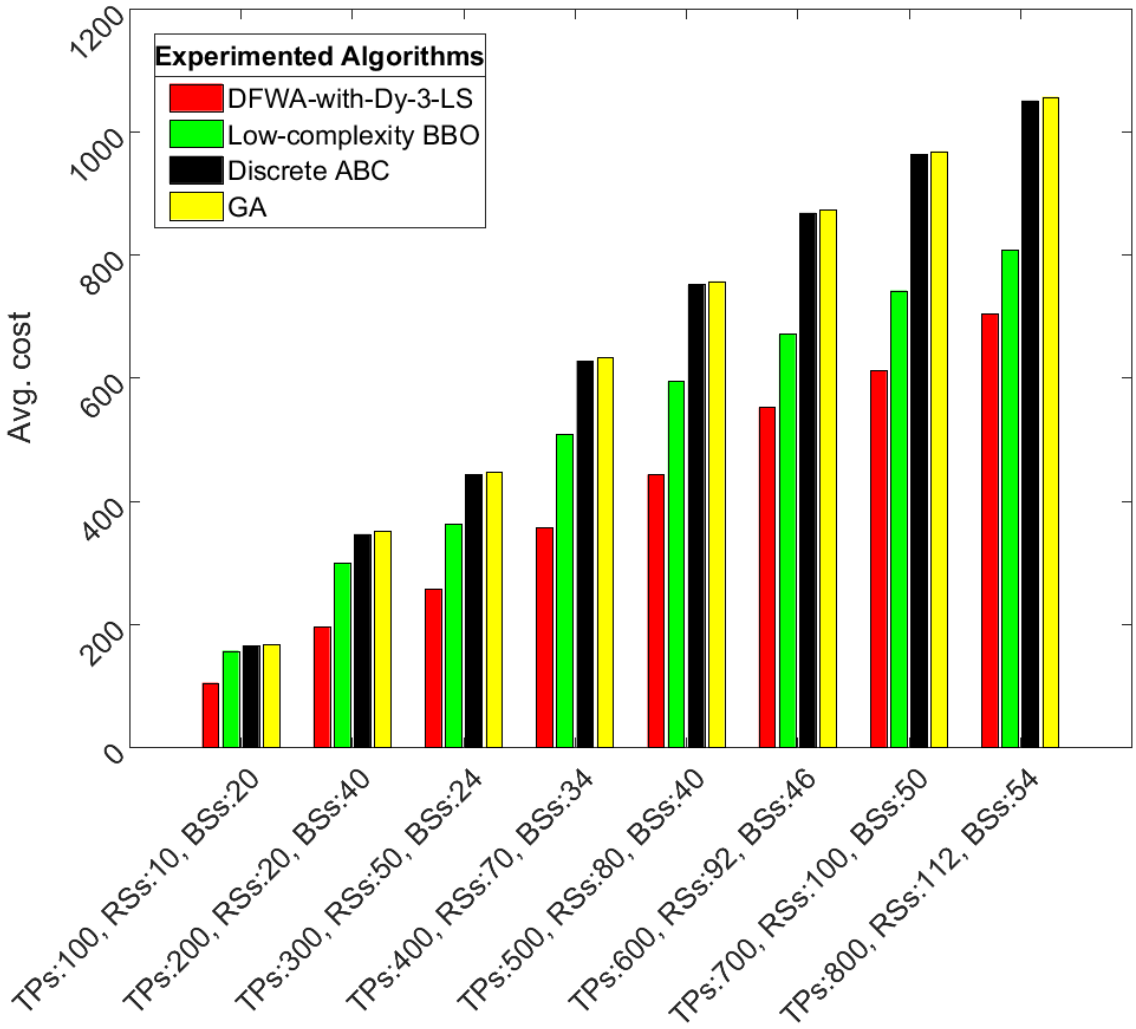


Figure 5.3 Avg. cost of DFWA-with-Dy-3-LS vs. LC-BBO, DABC, and GA.

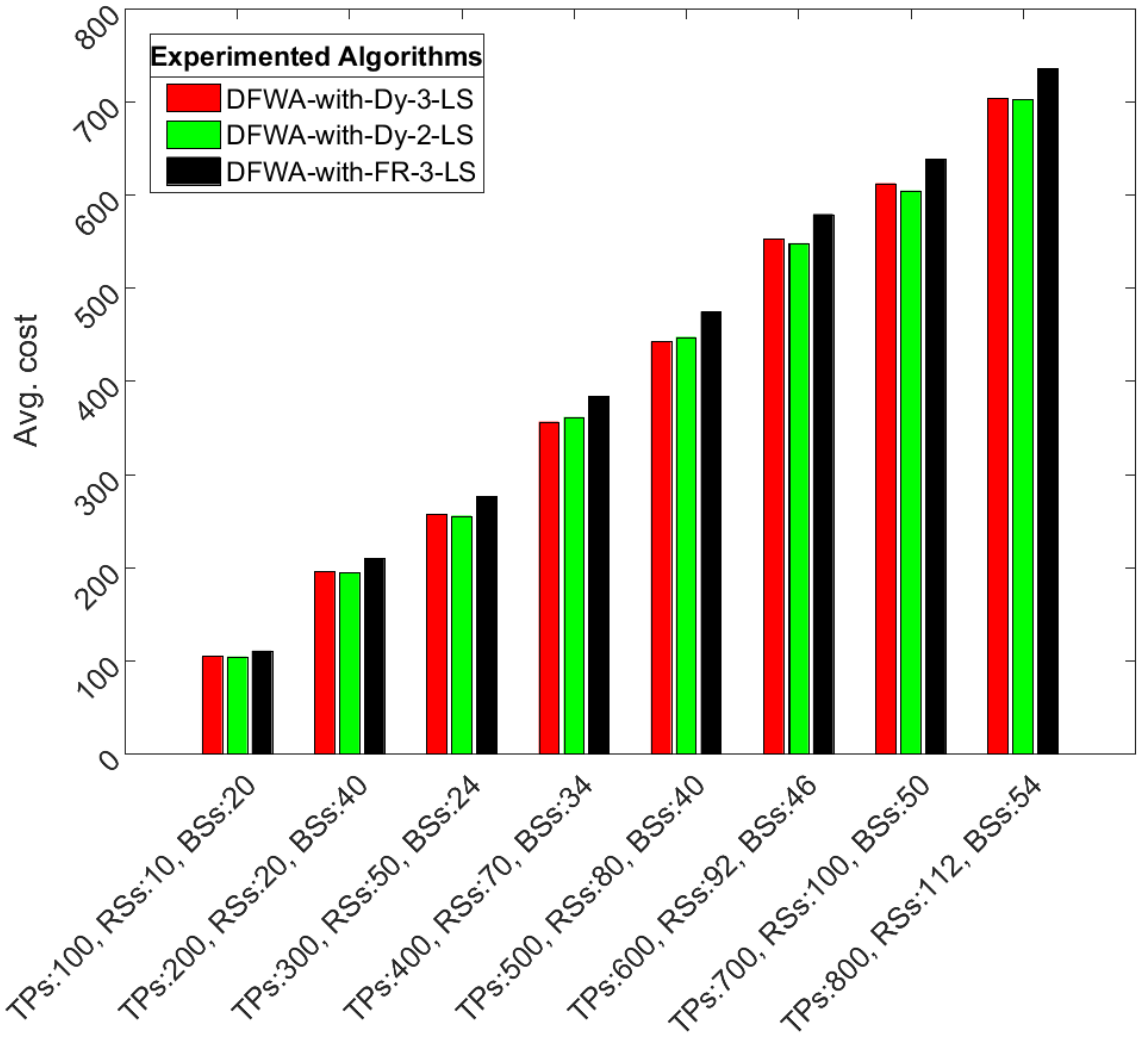


Figure 5.4 Avg. cost of DFWA-with-Dy-3-LS vs. DFWA-with-Dy-2-LS, DFWA-with-FR-3-LS.

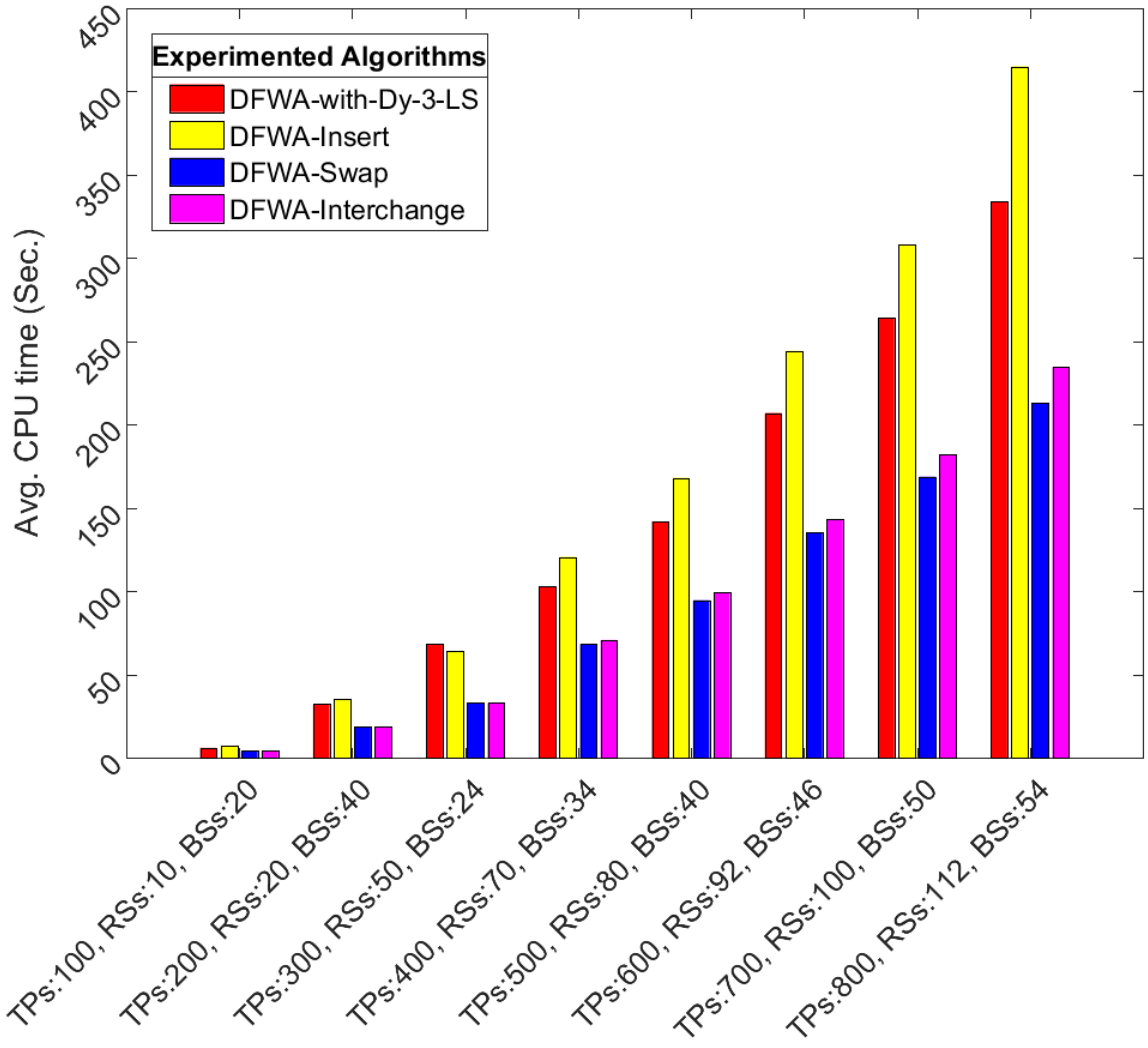


Figure 5.5 Avg. CPU time of DFWA-with-Dy-3-LS vs. DFWA with three individual LS methods.

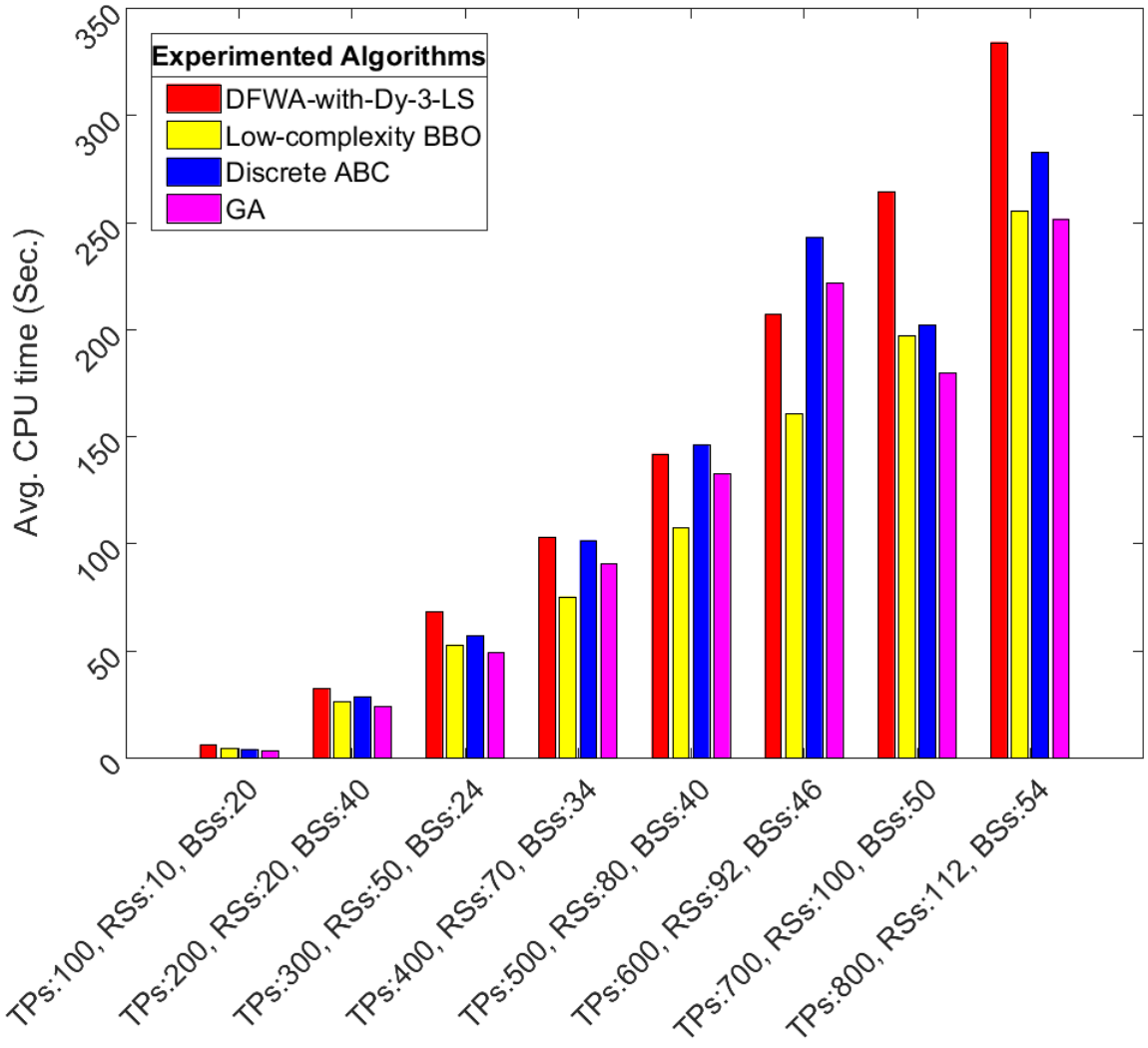


Figure 5.6 Avg. CPU time of DFWA-with-Dy-3-LS vs. LC-BBO, DABC, and GA.

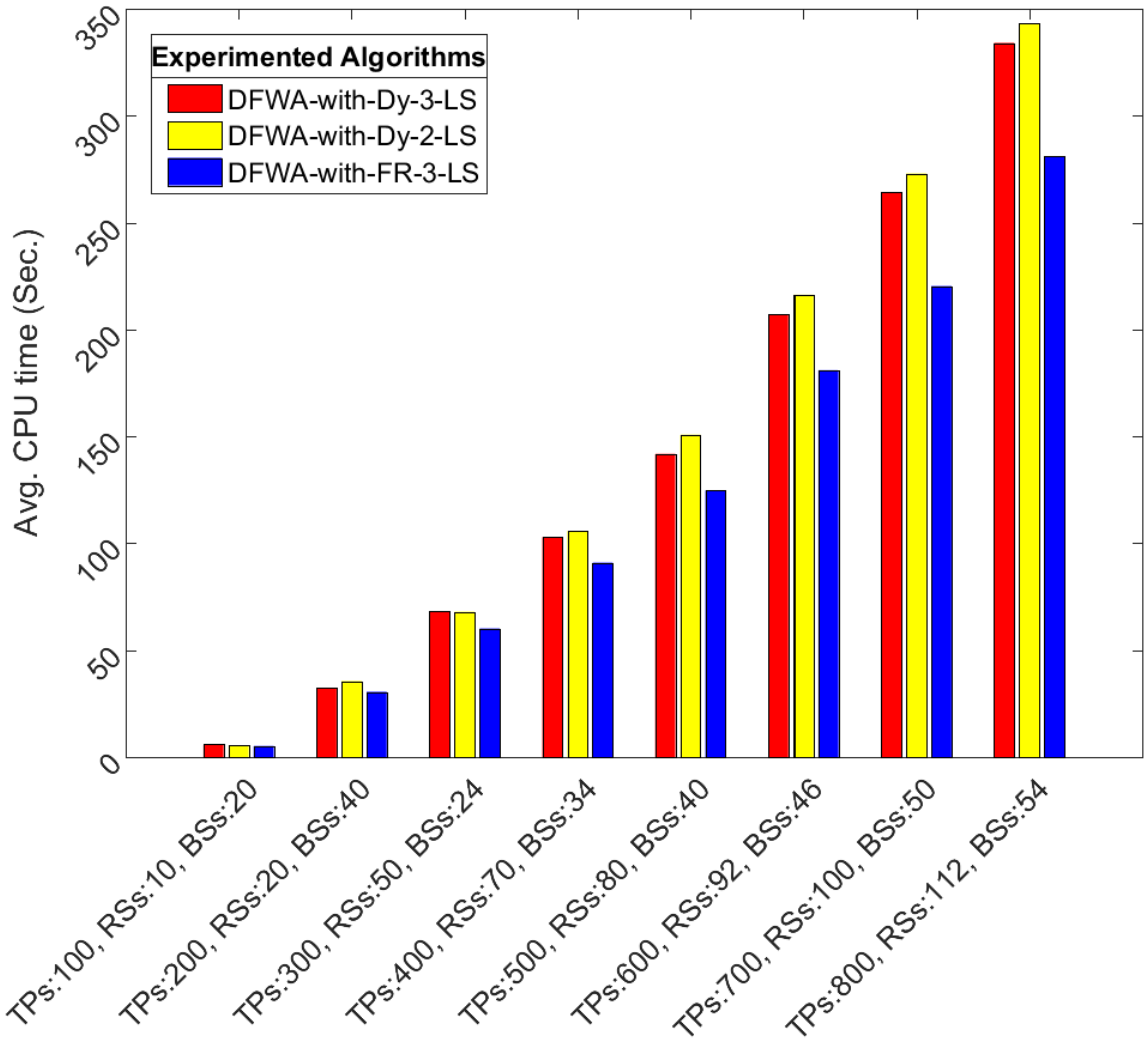


Figure 5.7 Avg. CPU time of DFWA-with-Dy-3-LS vs. DFWA-with-Dy-2-LS, DFWA-with-FR-3-LS

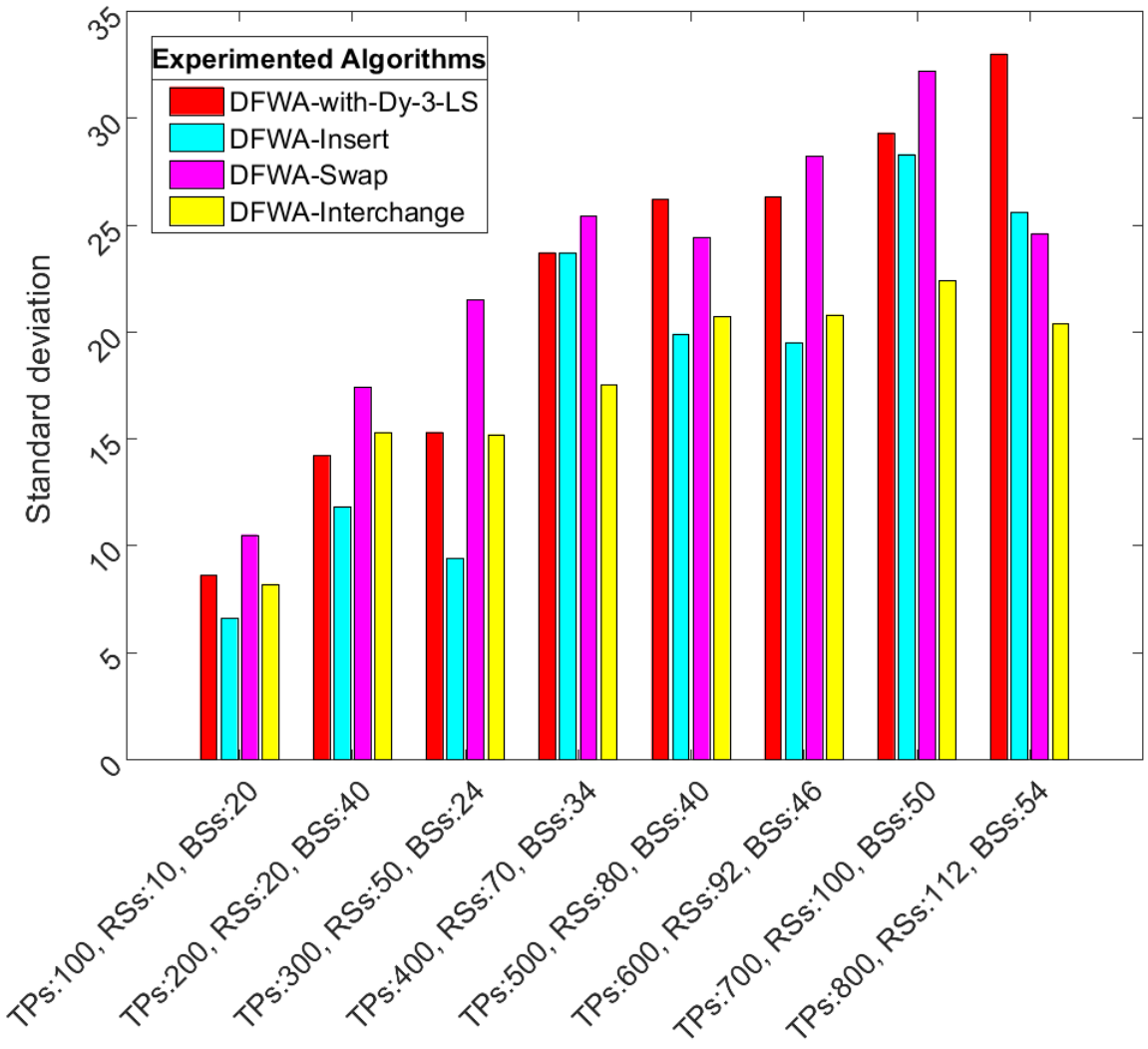


Figure 5.8 Standard deviation of DFWA-with-Dy-3-LS vs. DFWA with three individual LS methods.

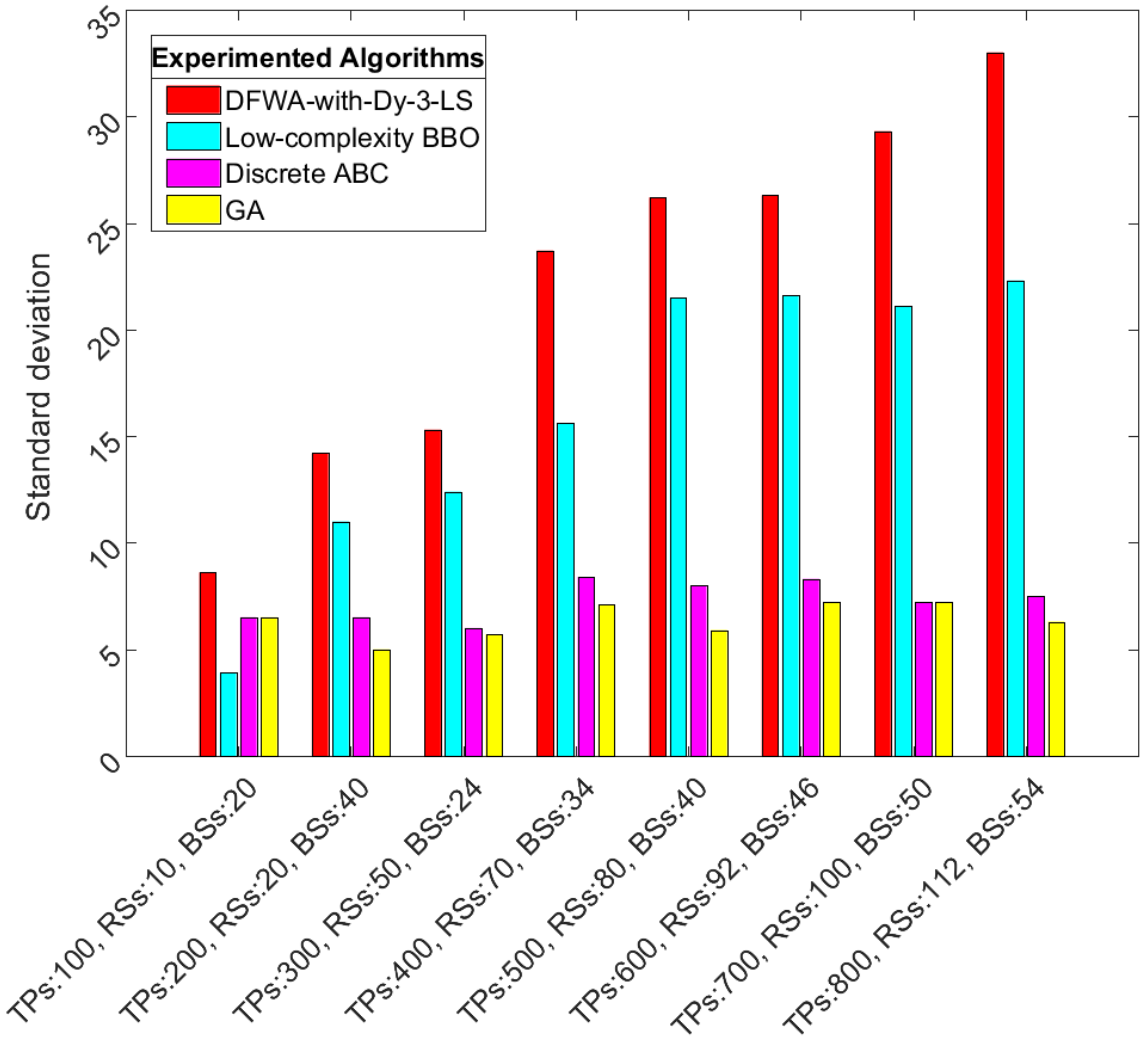


Figure 5.9 Standard deviation of DFWA-with-Dy-3-LS vs. LC-BBO, DABC, and GA.

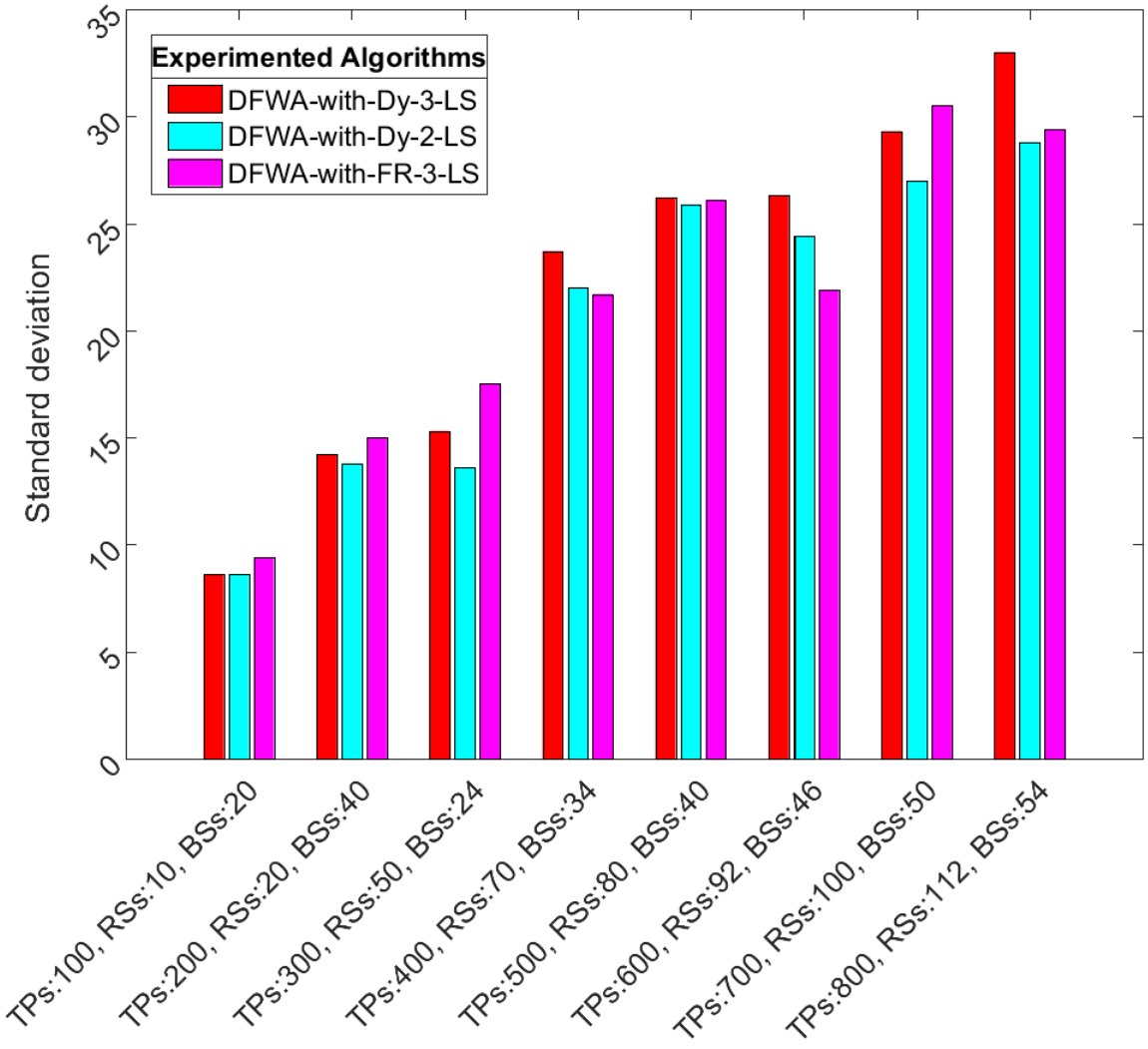


Figure 5.10 Standard deviation of DFWA-with-Dy-3-LS vs. DFWA-with-Dy-2-LS, DFWA-with-FR-3-LS.

5.7.3. Performance significance of the DFWA-with-Dy-3-LS

A T-test showed a significant difference between the performance of the DFWA-with-Dy-3-LS algorithm and the performances of DFWA-*insert*, DFWA-*swap*, DFWA-*interchange*, DFWA-with-FR-3-LS, DFWA-with-Dy-2-LS, LC-BBO, discrete ABC, and genetic algorithms. The null hypothesis H_0 states that both algorithms produce the same average cost. We performed the T-test of an alternative hypothesis H_1 which states that the DFWA-with-Dy-3-LS algorithm produces lower average cost. Table (5-14) shows the p-values of the T-test for each problem instance against each compared algorithm. The p-values can be compared against the generally acceptable level of significance $\alpha = 0.05$ to decide whether hypothesis H_1 is accepted. If the average cost by the DFWA-with-Dy-3-LS algorithm is lower than any compared algorithm and $p \leq \alpha$, then we conclude that there is a statistically significant difference between the DFWA-with-Dy-3-LS algorithm and the other experimental algorithms. Otherwise, we conclude that the observed difference is not statistically significant.

The DFWA-*insert* algorithm showed a lower average cost when compared to the other experimental algorithms, and the p-value was also lower than 0.05. Therefore, the performance of the DFWA-*insert* algorithm was significantly better than the performance of the DFWA-with-Dy-3-LS algorithm. Because of a lower average cost and a p-value lower than 0.05, the performance of the DFWA-with-Dy-3-LS algorithm was significantly better than the performance of DFWA-*swap*, DFWA-*interchange*, DFWA-with-FR-3-LS,

LC-BBO, discrete ABC, and genetic algorithms. No significant difference in performance was observed between DFWA-with-Dy-3-LS and DFWA-with-Dy-2-LS algorithms.

Table 5.14 T-test for a single-hop network planning problem

Problem instance #	Algorithms							
	p-value for DFWA-with-Dy-3-LS vs DFWA-*Insert*	**p-value for DFWA-with-Dy-3-LS vs DFWA-*Swap***	**p-value for DFWA-with-Dy-3-LS vs DFWA-*Interchange***	**p-value for DFWA-with-Dy-3-LS vs Low-complexity BBO**	**p-value for DFWA-with-Dy-3-LS vs DFWA-with-FR-3-LS**	**p-value for DFWA-with-Dy-3-LS vs DFWA-with-Dy-2-LS**	**p-value for DFWA-with-Dy-3-LS vs Discrete ABC**	**p-value for DFWA-with-Dy-3-LS vs GA**
1	0.0001	0.0001	0.0001	0.0001	0.0001	**0.7080**	0.0001	0.0001
2	0.0001	0.0001	0.0001	0.0001	0.0001	**0.5900**	0.0001	0.0001
3	0.0001	0.0001	0.0001	0.0001	0.0001	**0.5230**	0.0001	0.0001
4	0.0001	0.0001	0.0001	0.0001	0.0001	**0.1260**	0.0001	0.0001
5	0.0001	0.0001	0.0001	0.0001	0.0001	**0.2090**	0.0001	0.0001
6	0.0001	0.0001	0.0001	0.0001	0.0001	**0.2150**	0.0001	0.0001
7	0.0001	0.0001	0.0001	0.0001	0.0001	**0.0470**	0.0001	0.0001
8	0.0001	0.0001	0.0001	0.0001	0.0001	**0.7390**	0.0001	0.0001

5.7.4. Performance analysis

The BWN planning problem instances are presented in Table 5-7 and can be named as: (100 10 20), (200 20 40), (300 24 50), (400 34 70), (500 40 80), (600 46 92), (700 50

100) and (800 54 112). Box-plots show the comparative performance of the algorithms tested in the BWN planning problem in Figures 5.11 to 5.18.

Although variability is observed among the algorithms in Figures 5.11 to 5.18, there is consistency among some of the algorithms. For example, the DABC algorithm and the GA have low variability in all eight experiments. In addition, less variability is also observed in the (100 10 20) and (300 24 50) problem instances for the DFWA-*insert*, and in the (100 10 20) test of the LC-BBO algorithm.

In the first experiment, the better performance of the DFWA-*insert* is observed in terms of average cost. The proposed DFWA-with-Dy-3-LS achieves better average cost value as compared to the DFWA-*swap*, DFWA-*interchange*, DFWA-with-FR-3-LS, LC-BBO, discrete ABC and genetic algorithms in most of the experiments.

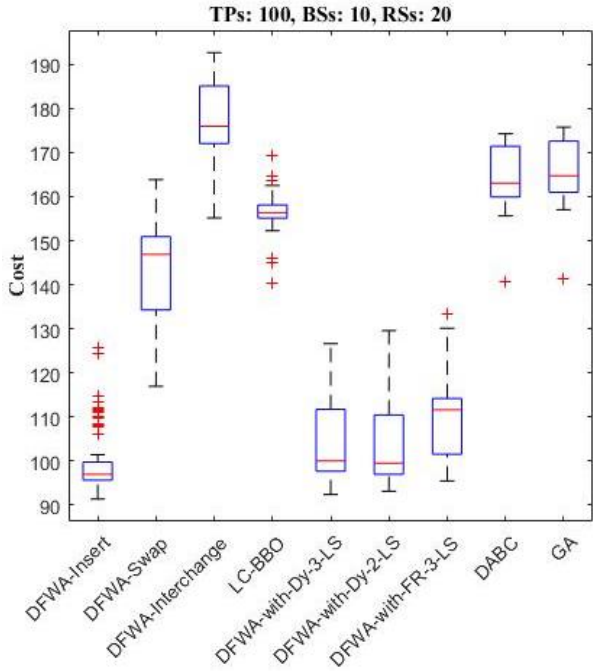


Figure 5.11 Comparing the experimental algorithms problem 1.

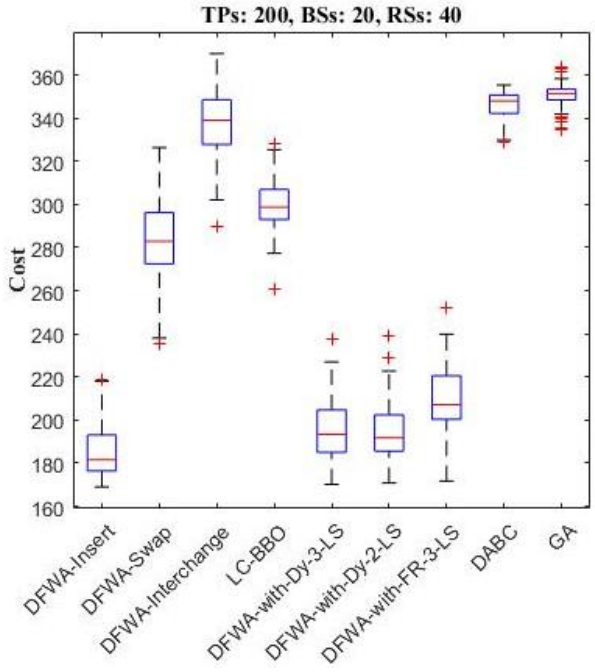


Figure 5.12 Comparing the experimental algorithms problem 2.

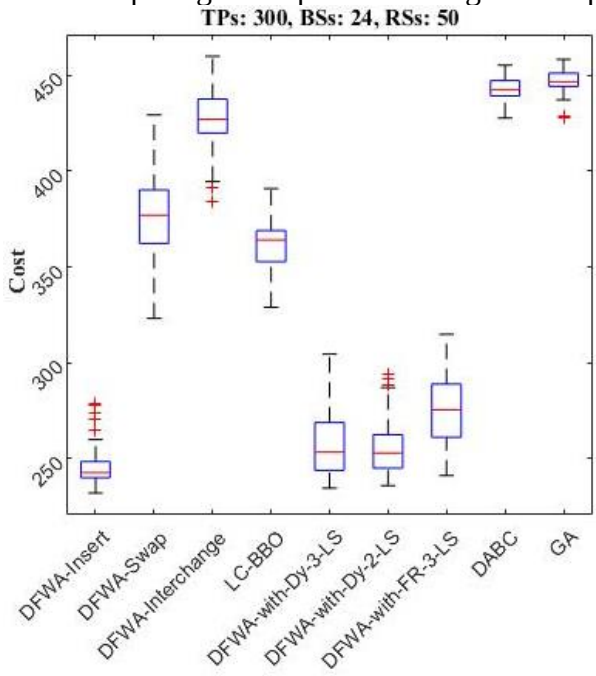


Figure 5.13 Comparing the experimental algorithms problem 3.

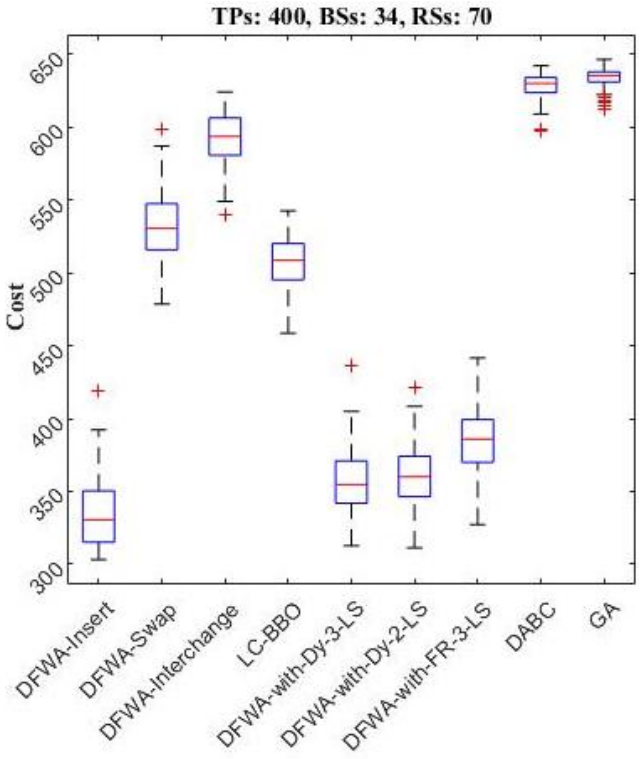


Figure 5.14 Comparing the experimental algorithms problem 4.

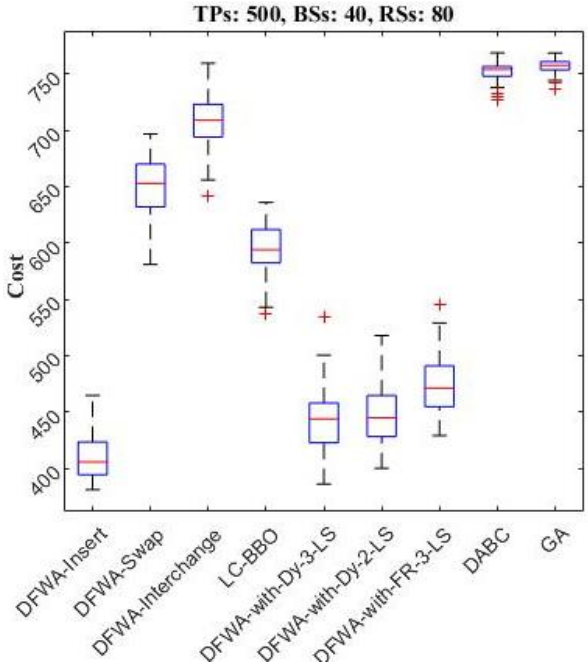


Figure 5.15 Comparing the experimental algorithms problem 5.

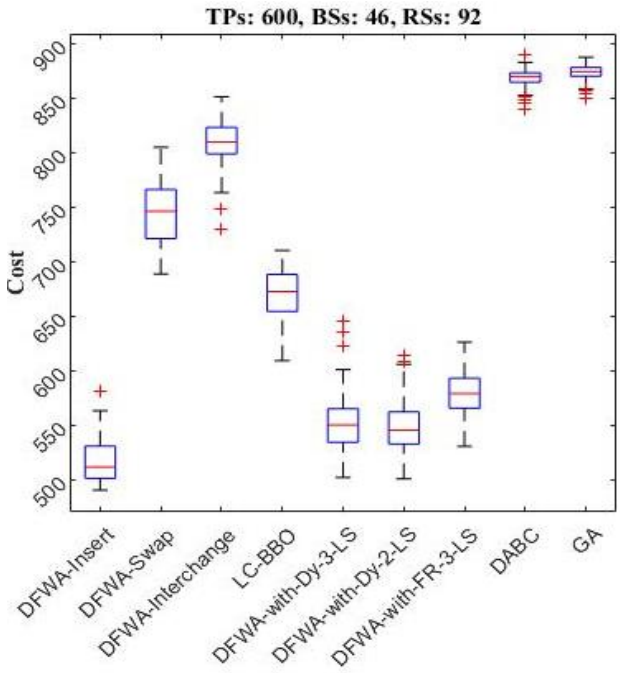


Figure 5.16 Comparing the experimental algorithms problem 6.

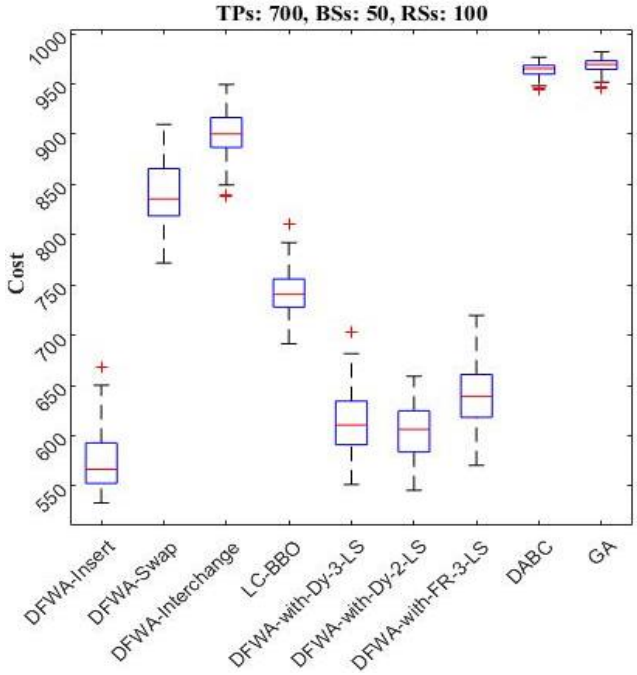


Figure 5.17 Comparing the experimental algorithms problem 7.

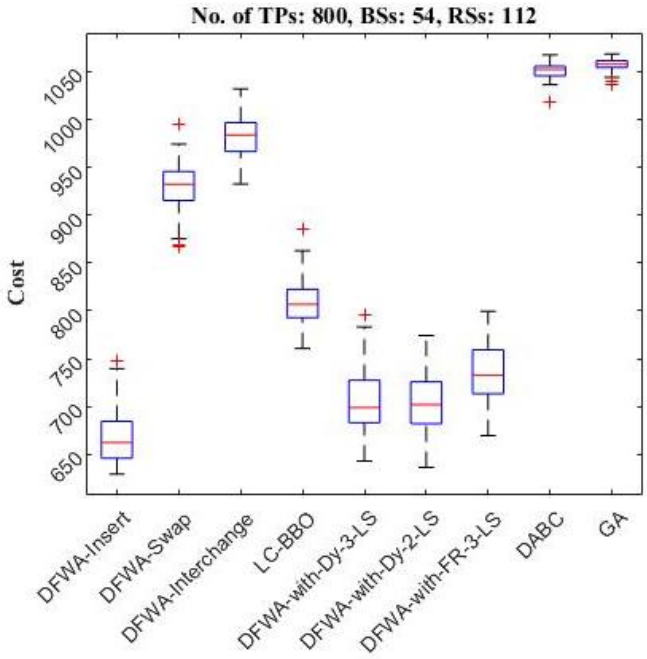


Figure 5.18 Comparing the experimental algorithms problem 8.

Using boxplots (Figures 5.11 to 5.18) we can see the outliers (the red "+") in the data set. For example, the DFWA-with-Dy-3-LS, DFWA-with-Dy-2-LS, DFWA-with-FR-3-LS, discrete ABC, and genetic algorithm data are symmetric in the (100 10 20) as shown in Figure 5.6. However, left and right skewness is observed for the DFWA-*swap* and DFWA-*interchange* algorithms, respectively, in Figure 5.11. Note that skewness indicates the direction and a relative magnitude of how far a distribution deviates from normal.

5.8. Conclusion

In this chapter, we discussed an integer programming formulation of a broadband wireless network planning problem with a single-hop. The network planning problem consists of three nodes: a base station (BS), a relay station (RS), and test points (TP or users). A TP can communicate with a BS directly or via an RS. The objective of this optimization was to minimize the overall operating cost of the network. Finding an optimal solution using exhaustive search was impractical due to the high computing demand.

We used Discrete Fireworks Algorithm (DFWA) with '*insert*', '*interchange*', and '*swap*' local search (LS) methods and observed difference in the performance of these LS methods for the BWN planning problem. Therefore, we proposed various combinations of LS methods for the DFWA. These algorithms included a DFWA with a fixed-rate of LS (DFWA-with-FR-3-LS), a DFWA with 3 dynamic LS methods (DFWA-with-Dy-3-LS), and a DFWA with 2 dynamic LS methods (DFWA-with-Dy-2-LS).We compared the performance of DFWA-with-Dy-3-LS against all the experimented algorithms such as DFWA-*insert*, DFWA-*swap*, DFWA-*interchange*, low-complexity BBO, discrete ABC, genetic algorithm (GA), DFWA-with-FR-3-LS and DFWA-with-Dy-2-LS. The DFWA-*insert* and DFWA-with-Dy-3-LS algorithms are the 1st and 2nd top performers in terms of average cost of the network and was significantly better than the other algorithms according to T-test results. Simulation results demonstrated the merits and demerits of individual LS methods versus an ensemble of LS methods for the DFWA.

Our experimental results highlight some key findings. First, performance difference is observed in the '*insert*,' '*interchange,*' and '*swap*' LS methods of the DFWA for the BWN planning. Second, we observed sensitivity of selecting an LS method randomly for the DFWA. Third, an ensemble of LS methods can work better than a randomly selected LS method. Fourth, the DFWA-with-Dy-3-LS produced better results than the DFWA-with-FR-3-LS, DFWA-*swap*, DFWA-*interchange*, low-complexity BBO, discrete ABC, and GA. Finally, statistical analysis showed that the DFWA-with-Dy-3-LS performed significantly better than DFWA-*swap*, DFWA-*interchange*, low-complexity BBO, discrete ABC and GA.

Chapter 6. Summary, Future Work, and Conclusion

6.1. Thesis summary

Three problems are addressed in this thesis: (i) a virtual machine (VM) placement was reformulated to minimize the power consumption in a datacenter, (ii) a resource assignment problem was formulated for Internet of things network (IoTN) with the objective of minimizing operating power, and (iii) a single-hop broadband wireless network planning problem was formulated with the objective of minimizing the weighted sum of operating and infrastructure costs. All three problems are combinatorial in nature, and these problems are solved using approximate algorithms (e.g., evolutionary algorithms), which often return good-quality solutions without excessive computing resources. In addition to the problems formulated, some enhancements to the swarm intelligence-based evolutionary algorithm (EA) (i.e., Discrete Fireworks Algorithm (DFWA) and its variants) are proposed and compared against the low-complexity biogeography-based optimization (LC-BBO) algorithm, the discrete artificial bee colony (DABC) algorithm, and genetic algorithm (GA). The subsequent sections include contributions in this thesis and suggest ideas for extending this work.

6.2. Virtual machine placement

In chapter 3, virtual machine (VM) placement with the objective of minimizing the power consumption in a datacenter is considered. The VM placement problem formulation for binary space and the power formulas are taken from [8]. The binary space VM placement problem is reformulated as an integer space VM placement to reduce the constraint checks.

6.3. Optimizing power in IoT network

Real-time feedback to delay sensitive Internet of things (IoT) applications using datacenters brought about a new concept of fog computing that acts as a bridge between IoT nodes and classic cloud computing. The idea behind fog computing is to bring the cloud closer to IoT nodes mainly to mitigate the latency. The fog cloud node can be a server or a set of servers with large computing and storing capacities that receive, process, and analyze data collected from IoT nodes. However, conventional model of fog computing may not be feasible in some mission-specific conditions or in the remote areas where power is a scarce resource. In this work a special case of fog computing model is introduced. In this model, a battery powered node with computing capabilities is included in the IoT network for real-time feedback. The proposed IoT network comprises of three nodes: IoT, core cluster node (CCN), and base station (BS). This cluster-assisted IoT network has a battery powered CCN that contains computing resources such as a CPU and memory. A CCN acts as a cluster head (CH) and its power is critical for the life span of the IoT network. CCN's power can be better utilized by efficient resources (i.e., memory and CPU) assignment in the IoT network. Optimizing power by assigning efficient resources in the IoT network is a challenging task. The objective of the proposed optimization problem is to minimize the weighted sum of data transmission power between IoTs and CCNs, between CCNs and BSs, and computational power at CCNs. The proposed resource assignment in IoT network described in chapter 4 may be extended in the future to the following areas:

(i) Planning a fog node location in the IoT network:

Some of the emerging challenges of the last decade, mobile computing, control, network management functions and data storage are shifted to centralized data centers. However, traditional cloud computing is facing serious challenges in meeting many new requirements in the Internet of Things (IoT). Fog computing is an architecture that distributes computation, communication, control and storage closer to IoT. The relevance of fog model is rooted in both the inadequacy of the traditional cloud and the emergence of new opportunities for the IoT [98]. The fog cloud node can be a server or a set of servers

with large computing and storing capacities. Before a fog node can be installed in an IoT network, complete knowledge about computing needs, the geography over which the fog node will be installed, and information about inadequacy of traditional cloud is helpful. In the future, current work can be extended to plan fog node locations by using metrics such as delay, geographic conditions or variable power resources.

(ii) Explore a new power model for the proposed IoT network:

In the future, different power formulae can be employed in place of current power formulae to calculate CCN's computational power and its effect on the IoT network.

6.4. Planning the single-hop broadband wireless network

A broadband wireless network (BWN) consists of three nodes: subscribers (i.e., test points), base stations, and relay stations. In chapter 5, two equivalent formulations are proposed for a single-hop BWN to enhance its capacity in populated urban centers. The first formulation is a binary space optimization, and the second is an integer space optimization problem. Reducing the number of variables and constraint checks is the main advantage of converting binary space into integer space optimization problem. The objective of BWN planning is to simultaneously minimize infrastructure (base stations and relay stations) and the operating cost (path-loss) of the BWN. The BWN plan described in chapter 5 may be extended in the future to the following areas:

(i) A multi-hop broadband wireless network:
The current work can be extended to multi-hop broadband wireless network by allowing more than one RSs between communication of TPs and BSs. This work can be used to extend the coverage of the network in remote areas for sparse and scattered population.

(ii) Planning a 4G/5G heterogeneous wireless network:
This work can be further investigated for planning 5G radio access technology (RAT) [122].

6.5. Discrete fireworks algorithm

In chapters 3–5, discrete space optimization problems such as virtual machine (VM) placement, optimizing power in emerging IoT applications, and broadband wireless network (BWN) planning are considered. VM placement, IoT applications and BWN planning are integer space optimization problems. In chapters 3–5, candidate solutions for VM placement, IoT network and a BWN are formulated as vectors of nonnegative integers. We proposed two different types of discrete fireworks algorithms to solve the above optimization problems.

In chapter 3 and 4, the proposed discrete fireworks algorithm (DFWA) is modification of the enhanced fireworks algorithm (EFWA) to solve integer space VM placement and resource assignment in IoT network. To discretize the EFWA, the '*round*' and '*ceil*' functions are used to convert real values into integer values for the explosion amplitude and the offset displacement, respectively, for an integer space VM placement and resources assignment in IoT network. In the DFWA, an offset displacement is added to one or more selected components of a firework to generate sparks.

In chapter 5, instead of converting the original local search (LS) method of the EFWA to solve the integer space problem in chapter 3 and chapter 4, the *insert*, *swap*, and *interchange* as LS methods are employed in the DFWA to plan the integer space BWN. In the DFWA, '*insert,*' '*interchange,*' and '*swap*' LS methods are used to exchange/replace one or more components of a firework as a criterion of perturbation to generate sparks.

6.5.1. Enhancing the discrete fireworks algorithm

In chapter 3 and 4, the problem specific information-based DFWA (IDFWA) is introduced to incorporate some domain knowledge for the VM placement and resources assignment in IoT network in the DFWA.

In chapter 5, first, the *insert*, *interchange*, and *swap* LS methods were ranked based on their individual performance in the BWN plan using the DFWA. This predetermined

information was used to build an ensemble of LS methods for the DFWA. A better LS method has a greater probability of being selected than a relatively poor LS method. Information about the performance of individual LS methods in the DFWA allowed the better performing LS methods to be assigned a more user-determined probability. Because a constant user-determined probability is assigned to each LS method at the start of an experiment, the DFWA that incorporated an ensemble of fixed-rate LS methods was called a DFWA-with-FR-3-LS.

Second, a DFWA with an ensemble of three dynamic LS (i.e., *insert*, *interchange*, and *swap*) methods—a DFWA-with-Dy-3-LS algorithm—was proposed to avoid manually assigning a user-determined probability to the LS methods. In the DFWA-with-Dy-3-LS algorithm, a new criterion to dynamically select an LS method was adopted from an ensemble of LS methods. The three LS methods were a set of integers $\wp$. For example, the set $\wp = \{\wp_1, \wp_2, \wp_3\}$ is an ensemble of LS methods in which each element $\wp_i$ represents a LS method such as $\wp_1 = 1$ (LS operator 1), $\wp_2 = 2$ (LS operator 2), $\wp_3 = 3$ (LS operator 3), respectively. Initially, the DFWA-with-Dy-3-LS algorithm randomly assigned an LS method from the set $\wp$ to each of a population of N fireworks. After the 1st iteration of the DFWA-with-Dy-3-LS algorithm, if no improvement was observed in the cost value of the sparks generated from the i^{th} firework for each of the $i = 1,2, \ldots, N$ fireworks, then the currently assigned LS method was replaced with a random selection of one of the two remaining LS methods for the i^{th} firework in the next algorithm generation.

Third, a DFWA with an ensemble of two dynamic LS (*insert* and *swap*) methods was proposed and was abbreviated as a DFWA-with-Dy-2-LS algorithm. In the DFWA-with-Dy-2-LS algorithm, the two better performing LS methods were employed as an ensemble from the three *insert*, *interchange*, and *swap* LS methods in the first experiment.

The performance of future DFWA algorithms could be improved as follows:

(i) In the current work, we used three LS methods (*insert*, *interchange*, and *swap*) to build an ensemble of LS methods. In the future, we will incorporate more than three LS methods to expand the local search ensemble.

(ii) In the current work, we proposed fixed-rate ensemble of three LS methods (DFWA-with-FR-3-LS) and dynamic ensemble of three LS methods (DFWA-with-Dy-3-LS). In the future, a fuzzy rule-based system might be adapted to an ensemble of LS methods [123] to expand the search for solutions to problems like VM placement, resources assignment in IoT network, and BWN planning.

6.6. Hybrid IDFWA/LC-BBO algorithm

A hybrid of the IDFWA and the LC-BBO algorithm was proposed to solve VM placement and resource assignment in IoT network in chapter 3 and chapter 4 respectively. In each generation of the hybrid IDFWA/LC-BBO algorithm, either the migration procedure of the LC-BBO algorithm or the explosion procedure of the IDFWA is probabilistically selected to generate spark(s) for each of the N fireworks. The hybrid IDFWA/LC-BBO algorithm outperformed the DFWA, the IDFWA, the LC-BBO algorithm, the DABC algorithm and the GA in terms of average power consumed for VM placement.

In the future, the IDFWA and the LC-BBO algorithm can be hybridized using a fuzzy rule-based system to control the operators, the fuzzy rule-based system could be used to decide whether to select a migration procedure of the LC-BBO algorithm or an explosion procedure of the IDFWA [123] to generate the sparks.

6.7. Repair algorithms

In this thesis, a candidate solution is mathematically represented by a vector of integers for each optimization problem. An operator in each experimented algorithm perturbs multiple components of a candidate solution. This evolution during algorithm operation may violate one or more constraints of the optimization problem, so a candidate solution may become infeasible during the algorithm operation. We propose three repair algorithms to check feasibility or repair infeasible candidate solutions during

implementation of VM placement, resource assignment in IoT network and BWN planning problems.

6.8. Conclusion

This thesis contains two types of contributions: formulations of optimization problems and algorithms to solve these optimization problems. The contribution for formulating optimization problems includes: (i) two equivalent formulations for resource assignment in IoT network, (ii) two equivalent formulations for BWN planning and (iii) a reformulation for an existing VM placement problem. The purpose of the second formulations of the first two problems and reformulation of the existing problem was to reduce the constraint checks during the implementation of these problems. The algorithms proposed to solve these optimization problems include the DFWA, IDFWA and hybrid IDFWA/LC-BBO algorithms to solve resource assignment in IoT network and VM placement. In addition, the DFWA with three different combination of LS methods (i.e., DFWA-with-Dy-3-LS, DFWA-with-Dy-2-LS and DFWA-with-FR-3-LS) were proposed to solve BWN planning. After conducting T-tests, the conclusion was that the Hybrid IDFWA/LC-BBO algorithm significantly outperforms the DFWA, the IDFWA, the LC-BBO algorithm, the DABC algorithm and the GA in terms of cost-effective power consumed in VM placement. However, the DABC algorithm outperformed the Hybrid IDFWA/LC-BBO algorithm, DFWA, the IDFWA, the LC-BBO algorithm in terms of average power consumed in IoT network. The performance of the DFWA-with-Dy-3-LS algorithm is better than the performances of the DFWA-*Swap*, the DFWA-*Interchange*, the DFWA-with-FR-3-LS, the LC-BBO algorithm, the discrete ABC algorithm, and the GA in terms of lower average infrastructure and operating costs.

References

[1] M. W. Toffel and A. Horvath, “Environmental Implications of Wireless Technologies: News Delivery and Business Meetings,” *Environ. Sci. Technol.*, vol. 38, no. 11, pp. 2961–2970, Jun. 2004.

[2] S. Joseph, V. Namboodiri, and V. C. Dev, “Toward environmentally sustainable mobile computing through an economic framework,” *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2. pp. 212–224, 2014.

[3] WK. Kuo and CT. Hsu, “Study on Energy Conservation for Cellular Systems: A Global Optimization Approach,” *IEEE Syst. J.*, vol. 12, no. 1, pp. 627–638, 2018.

[4] N. Mhaisen, O. Abazeed, Y. Al Hariri, A. Alsalemi, and O. Halabi, “Self-Powered IoT-Enabled Water Monitoring System,” in *2018 International Conference on Computer and Applications, ICCA 2018*, 2018, pp. 41–45.

[5] E. Gelenbe and Y. Caseau, “The impact of information technology on energy consumption and carbon emissions,” *Ubiquity*, vol. 2015, no. June, pp. 1–15, 2015.

[6] A. Fehske, J. Malmodin, G. Biczok, and G. Fettweis, “The Global Carbon Footprint of Mobile Communications - The Ecological and Economic Perspective,” *IEEE Communications Magazine*, vol. 49, no. 8, 2011.

[7] M. F. Bari *et al.*, “Data center network virtualization: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[8] Y. Wu, M. Tang, and W. Fraser, “A simulated annealing algorithm for energy efficient virtual machine placement,” in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2012, pp. 1245–1250.

[9] H. R. Arkian, A. Diyanat, and A. Pourkhalili, “MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications,” *J. Netw. Comput. Appl.*, vol. 82, pp. 152–165, 2017.

[10] A. Brogi and S. Forti, “QoS-aware deployment of IoT applications through the fog,” *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1–8, 2017.

[11] W. Ejaz and M. Ibnkahla, “Multiband Spectrum Sensing and Resource Allocation for IoT in Cognitive 5G Networks,” *IEEE Internet Things J.*, vol. 5, no. 1, pp. 150–163, 2018.

[12] H. Atlam, R. Walters, and G. Wills, “Fog Computing and the Internet of Things: A Review,” *Big Data Cogn. Comput.*, vol. 2, no. 2, p. 10, 2018.

[13] E. Gateway, I. Fog, M. Suárez-albela, L. Castedo, T. M. Fernández-caramés, and P. Fraga-lamas, “A Practical Evaluation of a High-Security Computing Applications,” *Sensors*, vol. 17, no. 1979, pp. 1–39, 2017.

[14] Y. Yu, S. Murphy, and L. Murphy, “Planning base station and relay station locations in IEEE 802.16j multi-hop relay networks,” in *2008 5th IEEE Consumer Communications and Networking Conference, CCNC 2008*, 2008, pp. 922–926.

[15] T. Hu, Y. P. Chen, and W. Banzhaf, “WiMAX Network Planning Using Adaptive-Population-Size Genetic Algorithm,” in *Springer*, 2010, pp. 31–40.

[16] Y. Yu, S. Murphy, and L. Murphy, “Planning base station and relay station locations for IEEE 802.16j network with capacity constraints,” in *2010 7th IEEE Consumer Communications and Networking Conference, CCNC 2010*, 2010.

[17] Z. Abichar, A. E. Kamal, and J. M. Chang, “Planning of relay station locations in IEEE 802.16 (WiMAX) networks,” in *IEEE Wireless Communications and Networking Conference, WCNC*, 2010.

[18] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*, vol. 9780521195. 2011.

[19] X. Chen, Y. S. Ong, M. H. Lim, and K. C. Tan, “A multi-facet survey on memetic computation,” *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5. pp. 591–607, 2011.

[20] S. Dan, *Evolutionary Optimization Algorithm : Biologically Inspired and Population-Based Approaches to Computer Intelligence*, 1st Ed. Wiley, 2013.

[21] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms (Decision Engineering)*. 2010.

[22] Y. Tan, *Fireworks Algorithm: A Novel Swarm Intelligence Method*, 1st Ed. Springer, 2015.

[23] S. Zheng, A. Janecek, and Y. Tan, “Enhanced fireworks algorithm,” in *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, 2013, pp. 2069–2077.

[24] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, “A comprehensive survey: Artificial bee colony (ABC) algorithm and applications,” *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, Jun. 2014.

[25] M. H. Kashan, N. Nahavandi, and A. H. Kashan, “DisABC: A new artificial bee colony algorithm for binary optimization,” *Appl. Soft Comput. J.*, vol. 12, no. 1, pp. 342–352, 2012.

[26] S. Ashrafinia, U. Pareek, M. Naeem, and D. Lee, “Source and relay power selection using biogeography-based optimization for cognitive radio systems,” in *IEEE Vehicular Technology Conference*, 2011.

[27] H. M. Ali and D. C. Lee, “A biogeography-based optimization algorithm for energy efficient virtual machine placement,” in *IEEE SSCI 2014 - 2014 IEEE Symposium Series on Computational Intelligence - SIS 2014: 2014 IEEE Symposium on Swarm Intelligence, Proceedings*, 2015, pp. 231–236.

[28] H. M. Ali and D. C. Lee, “Optimizing the energy efficient VM placement by IEFWA and hybrid IEFWA/BBO algorithms,” in *Proceedings of the 2016 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2016 - Part of SummerSim 2016 Multiconference*, 2016.

[29] H. M. Ali, W. Ejaz, D. C. Lee, and I. M. Khater, “Optimising the power using firework-based evolutionary algorithms for emerging IoT applications,” *IET Networks*, vol. 8, no. 1, pp. 15–31, Jan. 2019.

[30] Z. Liu, Z. Feng, and L. Ke, “Fireworks algorithm for the multi-satellite control resource scheduling problem,” in *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, 2015, pp. 1280–1286.

[31] G. Iacca, F. Neri, F. Caraffini, and P. N. Suganthan, “A differential evolution framework with ensemble of parameters and strategies and pool of local search algorithms,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8602, pp. 615–626.

[32] H. M. Ali, S. Ashrafinia, J. Liu, and D. Lee, “Broadband wireless network planning using evolutionary algorithms,” in *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, 2013, pp. 1045–1052.

[33] H. M. Ali, J. S. Oberoi, J. Liu, and D. Lee, “Base station and relay station broadband network planning using immune quantum evolutionary algorithm,” in *IEEE Vehicular Technology Conference*, 2013.

[34] H. M. Ali, D. Mitchell, and D. C. Lee, “MAX-SAT problem using evolutionary algorithms,” in *2014 IEEE Symposium on Swarm Intelligence*, 2014, pp. 1–8.

[35] H. M. Ali and D. Lee, “Solving the MAX-SAT problem by binary enhanced fireworks algorithm,” in *Sixth IEEE International Conference on Innovative Computing Technology (INTECH)*, 2016.

[36] H. M. Ali, S. Ashrafinia, J. Liu, and D. C. Lee, “Wireless mesh network planning using quantum inspired evolutionary algorithm,” in *IEEE Vehicular Technology Conference*, 2011.

[37] M. Naeem, H. M. Ali, and D. C. Lee, “Quantum Inspired Evolutionary Algorithm for Optimizing Sensor Selection,” in *IASTED Technology Conferences / 696:MS / 697:CA / 698: WC / 699: EME / 700: SOE*, 2010.

[38] T. D. Seeley, *The wisdom of the hive: the social physiology of honey bee*, vol. 40. 1995.

[39] V. Tereshko, *Reaction-diffusion model of a honeybee colony’s foraging behaviour*, vol. 1917. 2000.

[40] D. Karaboga and B. Basturk, “A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm,” *J. Glob. Optim.*, vol. 39, no. 3, pp. 459–471, Oct. 2007.

[41] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd Editio. Springer, 2015.

[42] C. Mihăilă, “Evolutionary Computation in Scheduling,” Babeş-Bolyai University, 2011.

[43] S. Ashrafinia, U. Pareek, M. Naeem, and D. Lee, “Biogeography-based optimization for joint relay assignment and power allocation in cognitive radio systems,” in *IEEE SSCI 2011 - Symposium Series on Computational Intelligence - SIS 2011: 2011 IEEE Symposium on Swarm Intelligence*, 2011, pp. 237–244.

[44] D. Simon, “Biogeography-based optimization,” *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, 2008.

[45] S. Ashrafinia, “Novel ABC-and BBO-based evolutionary algorithms and their illustrations to wireless communications,” Simon Fraser University, 2012.

[46] Y. Tan and Y. Zhu, “Fireworks algorithm for optimization,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6145 LNCS, no. PART 1, pp. 355–364.

[47] M. Dayarathna, Y. Wen, R. F.-I. C. Surveys, and U. 2016, “Data Center Energy Consumption Modeling: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[48] W. Xia, P. Zhao, Y. Wen, and H. Xie, “A Survey on Data Center Networking (DCN): Infrastructure and Operations,” *IEEE Communications Surveys and Tutorials*, vol. 19, no. 1. pp. 640–656, 2017.

[49] “Datacenter.” [Online]. Available: https://en.wikipedia.org/wiki/Data_center. [Accessed: 21-May-2018].

[50] X. Zhang, Y. Zhao, S. Guo, and Y. Li, “Performance-aware Energy-efficient Virtual Machine Placement in Cloud data center,” in *IEEE International Conference on Communications*, 2017.

[51] D. A. Alboaneen, H. Tianfield, and Y. Zhang, “Metaheuristic approaches to virtual machine placement in cloud computing: A review,” in *Proceedings - 15th International Symposium on Parallel and Distributed Computing, ISPDC 2016*, 2017, pp. 214–221.

[52] A. Al-Dulaimy, A. Zekri, W. Itani, and R. Zantout, “Paving the way for energy efficient cloud data centers: A type-aware virtual machine placement strategy,” in *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*, 2017, pp. 5–8.

[53] X. Pan, L. Wu, D. Wu, and Y. Sheng, “Ant colony optimization of virtual machine placement for data latency minimization in cloud systems,” in *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing, ICCWAMTIP 2015*, 2016, pp. 49–54.

[54] N. Su, A. Shi, C. Chen, E. Chen, and Y. Wang, “Research on virtual machine placement in the cloud based on improved simulated annealing algorithm,” in *World Automation Congress Proceedings*, 2016, vol. 2016–Octob.

[55] L. Li and K. Liu, “Guarantee-Aware Cost Effective Virtual Machine Placement Algorithm for the Cloud,” in *Proceedings - 2017 IEEE 19th Intl Conference on High Performance Computing and Communications, HPCC 2017, 2017 IEEE 15th Intl Conference on Smart City, SmartCity 2017 and 2017 IEEE 3rd Intl Conference on Data Science and Systems, DSS 2017*, 2018, vol. 2018–Janua, pp. 506–513.

[56] X. Ye, Y. Yin, and L. Lan, “Energy-efficient many-objective virtual machine placement optimization in a cloud computing environment,” *IEEE Access*, vol. 5, pp. 16006–16020, 2017.

[57] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, “Power-Aware and Performance-Guaranteed Virtual Machine Placement in the Cloud,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1385–1400, 2018.

[58] G. Portaluri, D. Adami, A. Gabbrielli, S. Giordano, and M. Pagano, “Power Consumption-Aware Virtual Machine Placement in Cloud Data Center,” *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 4, pp. 541–550, 2017.

[59] F. Alharbi, Y. C. Tain, M. Tang, and T. K. Sarker, “Profile-based static virtual machine placement for energy-efficient data center,” in *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 2017, pp. 1045–1052.

[60] M. K. Gupta and T. Amgoth, “Resource-aware algorithm for virtual machine placement in cloud environment,” in *2016 9th International Conference on Contemporary Computing, IC3 2016*, 2017.

[61] T. H. Duong-Ba, T. Nguyen, B. Bose, and T. T. Tran, “A Dynamic Virtual Machine Placement and Migration Scheme For Data Centers,” *IEEE Transactions on Services Computing*, 2018.

[62] X. K. Li, C. H. Gu, Z. P. Yang, and Y. H. Chang, “Virtual machine placement strategy based on discrete firefly algorithm in cloud environments,” in *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing, ICCWAMTIP 2015*, 2016, pp. 61–66.

[63] R. A. C. Da Silva and N. L. S. Da Fonseca, “Algorithm for the placement of groups of virtual machines in data centers,” in *IEEE International Conference on Communications*, 2015, vol. 2015–Septe, pp. 6080–6085.

[64] P. Wattanasomboon and Y. Somchit, “Virtual machine placement method for energy saving in cloud computing,” in *Proceedings - 2015 7th International Conference on Information Technology and Electrical Engineering: Envisioning the Trend of Computer, Information and Engineering, ICITEE 2015*, 2015, pp. 275–280.

[65] X. Zheng and Y. Cai, “Energy-efficient statistical live virtual machine placement for big data information systems in cloud computing environments,” in *Proceedings - 2015 IEEE International Conference on Smart City, SmartCity 2015, Held Jointly with 8th IEEE International Conference on Social Computing and Networking, SocialCom 2015, 5th IEEE International Conference on Sustainable Computing and Communic*, 2015, pp. 1053–1058.

[66] C. S. Verma, V. Dinesh Reddy, G. R. Gangadharan, and A. Negi, “Energy efficient virtual machine placement in cloud data centers using modified intelligent water drop algorithm,” in *Proceedings - 13th International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2017*, 2018, vol. 2018–Janua, pp. 13–20.

[67] M. Gaggero and L. Caviglione, “Model Predictive Control for Energy-Efficient, Quality-Aware, and Secure Virtual Machine Placement,” *IEEE Transactions on Automation Science and Engineering*, 2018.

[68] C. Sonklin, M. Tang, and Y. C. Tian, “A decrease-and-conquer genetic algorithm for energy efficient virtual machine placement in data centers,” in *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, 2017, pp. 135–140.

[69] C. Gao, H. Wang, L. Zhai, Y. Gao, and S. Yi, “An energy-aware ant colony algorithm for network-aware virtual machine placement in cloud computing,” in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2017, pp. 669–676.

[70] D. A. Alboaneen, H. Tianfield, and Y. Zhang, “Glowworm Swarm Optimisation Algorithm for Virtual Machine Placement in Cloud Computing,” in *Proceedings - 13th IEEE International Conference on Ubiquitous Intelligence and Computing, 13th IEEE International Conference on Advanced and Trusted Computing, 16th IEEE International Conference on Scalable Computing and Communications, IEEE Internationa*, 2017, pp. 808–814.

[71] X. F. Liu, Z. H. Zhan, J. D. Deng, Y. Li, T. Gu, and J. Zhang, “An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing,” *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 113–128, 2018.

[72] L. Hong and G. Yufei, “GACA-VMP: Virtual machine placement scheduling in cloud computing based on genetic ant colony algorithm approach,” in *Proceedings - 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing, 2015 IEEE 12th International Conference on Advanced and Trusted Computing, 2015 IEEE 15th International Conference on Scalable Computing and Communications, 20*, 2016, pp. 1008–1015.

[73] D. Liu, X. Sui, and L. Li, “An energy-efficient virtual machine placement algorithm in cloud data center,” *2016 12th Int. Conf. Nat. Comput. Fuzzy Syst. Knowl. Discov. ICNC-FSKD 2016*, no. C, pp. 719–723, 2016.

[74] B. Zhang, M. X. Zhang, and Y. J. Zheng, “A hybrid biogeography-based optimization and fireworks algorithm,” in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, 2014, pp. 3200–3206.

[75] A. John, A. Rajput, and K. V. Babu, “Dynamic cluster head selection in wireless sensor network for Internet of Things applications,” in *2017 International Conference on Innovations in Electrical, Electronics, Instrumentation and Media Technology (ICEEIMT)*, 2017, pp. 45–48.

[76] B. Hammi, R. Khatoun, S. Zeadally, A. Fayad, and L. Khoukhi, “IoT technologies for smart cities,” *IET Networks*, vol. 7, no. 1, pp. 1–13, Jan. 2018.

[77] G. Song, W. Li, B. Wang, and S. C. M. Ho, “A review of rock bolt monitoring using smart sensors,” *Sensors (Switzerland)*, vol. 17, no. 4. 2017.

[78] M. K. Mosleh, Q. K. Hassan, and E. H. Chowdhury, “Application of remote sensors in mapping rice area and forecasting its production: A review,” *Sensors (Switzerland)*, vol. 15, no. 1. pp. 769–791, 2015.

[79] M. Marcelli, V. Piermattei, A. Madonia, and U. Mainardi, “Design and application of new low-cost instruments for marine environmental research,” *Sensors (Switzerland)*, vol. 14, no. 12, pp. 23348–23364, 2014.

[80] M. Wu, L. Tan, and N. Xiong, “A Structure Fidelity Approach for Big Data Collection in Wireless Sensor Networks,” *Sensors*, vol. 15, no. 1, pp. 248–273, 2015.

[81] T. Kim, J. Park, S. Heo, K. Sung, and J. Park, “Characterizing Dynamic Walking Patterns and Detecting Falls with Wearable Sensors Using Gaussian Process Methods,” *Sensors*, vol. 17, no. 6, p. 1172, 2017.

[82] S. K. Gharghan, R. Nordin, and M. Ismail, “Energy-efficient ZigBee-based wireless sensor network for track bicycle performance monitoring,” *Sensors (Switzerland)*, vol. 14, no. 8, pp. 15573–15592, 2014.

[83] J. V. Capella, A. Perles, A. Bonastre, and J. J. Serrano, “Historical building monitoring using an energy-efficient scalable wireless sensor network architecture,” *Sensors*, vol. 11, no. 11, pp. 10074–10093, 2011.

[84] C. S. Chen and D. S. Lee, “Energy saving effects of wireless sensor networks: A case study of convenience stores in Taiwan,” *Sensors*, vol. 11, no. 2, pp. 2013–2034, 2011.

[85] P. Tarrío, A. M. Bernardos, and J. R. Casar, “An energy-efficient strategy for accurate distance estimation in wireless sensor networks,” *Sensors (Switzerland)*, vol. 12, no. 11, pp. 15438–15466, 2012.

[86] K. P. Musaazi, T. Bulega, and S. M. Lubega, “Energy efficient data caching in wireless sensor networks: A case of precision agriculture,” in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 2015, vol. 147, pp. 154–163.

[87] E. Gateway, I. Fog, M. Suárez-albela, L. Castedo, T. M. Fernández-caramés, and P. Fraga-lamas, “A Practical Evaluation of a High-Security Computing Applications,” *Sensors*, vol. 17, no. 1979, pp. 1–39, 2017.

[88] H. Yu, Y. Zhang, S. Guo, Y. Yang, and L. Ji, “Energy efficiency maximization for WSNs with simultaneous wireless information and power transfer,” *Sensors (Switzerland)*, vol. 17, no. 8, 2017.

[89] Y. Chang, H. Tang, Y. Cheng, Q. Zhao, B. Li, and X. Yuan, “Dynamic hierarchical Energy-Efficient method based on combinatorial optimization for wireless sensor networks,” *Sensors (Switzerland)*, vol. 17, no. 7, 2017.

[90] H. Ouchitachen, A. Hair, and N. Idrissi, “Minimizing energy consumption in mission-specific mobile sensor networks by placing sensors and base station in the best locations: Genetic algorithms approach,” in *International Conference on Wireless Networks and Mobile Communications, WINCOM 2015*, 2016.

[91] Shafali, S. Sharma, N. S. Randhawa, and D. Sharma, “An algorithm to minimize energy consumption using nature-inspired technique in wireless sensor network,” in *IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2015*, 2015.

[92] Z. T. Alisa and H. A. Nassrullah, “Minimizing energy consumption in wireless sensor networks using modified genetic algorithm and an energy balance filter,” in *Al-Sadiq International Conference on Multidisciplinary in IT and Communication Techniques Science and Applications, AIC-MITCSA 2016*, 2016, pp. 262–267.

[93] A. Al-Baz and A. El-Sayed, “A new algorithm for cluster head selection in LEACH protocol for wireless sensor networks,” *Int. J. Commun. Syst.*, vol. 31, no. 1, p. e3407, Jan. 2018.

[94] K. Sundaran, V. Ganapathy, and P. Sudhakara, “Fuzzy logic based Unequal Clustering in wireless sensor network for minimizing Energy consumption,” in *Proceedings of the 2017 2nd International Conference on Computing and Communications Technologies, ICCCT 2017*, 2017, pp. 304–309.

[95] M. A. Razzaque and S. Dobson, “Energy-efficient sensing in wireless sensor networks using compressed sensing,” *Sensors (Switzerland)*, vol. 14, no. 2. pp. 2822–2859, 2014.

[96] Ó. García, J. Prieto, R. S. Alonso, and J. M. Corchado, “A framework to improve energy efficient behaviour at home through activity and context monitoring,” *Sensors (Switzerland)*, vol. 17, no. 8, 2017.

[97] H. Jawad *et al.*, “Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review,” *Sensors*, vol. 17, no. 8, p. 1781, 2017.

[98] M. Chiang and T. Zhang, “Fog and IoT: An Overview of Research Opportunities,” *IEEE Internet of Things Journal*, vol. 3, no. 6. pp. 854–864, 2016.

[99] Y. Yu, S. Murphy, and L. Murphy, “A clustering approach to planning base station and relay station locations in IEEE 802.16j multi-hop relay networks,” in *IEEE International Conference on Communications*, 2008, pp. 2586–2591.

[100] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: Utility-Aware Resource Allocation for Edge Computing,” *Proc. - 2017 IEEE 1st Int. Conf. Edge Comput. EDGE 2017*, pp. 47–54, 2017.

[101] X. Xu *et al.*, “Dynamic Resource Allocation for Load Balancing in Fog Environment,” *Wirel. Commun. Mob. Comput.*, vol. 2018, 2018.

[102] K. Wang, W. Zhou, and S. Mao, “On Joint BBU/RRH Resource Allocation in Heterogeneous Cloud-RANs,” *IEEE Internet Things J.*, vol. 4, no. 3, pp. 749–759, 2017.

[103] Y. T. Chen, M. F. Horng, C. C. Lo, S. C. Chu, J. S. Pan, and B. Y. Liao, “A transmission power optimization with a minimum node degree for energy-efficient wireless sensor networks with full-reachability,” *Sensors (Switzerland)*, vol. 13, no. 3, pp. 3951–3974, 2013.

[104] K. Liu, S. Wu, B. Huang, F. Liu, and Z. Xu, “A power-optimized cooperative MAC protocol for lifetime extension in wireless sensor networks,” *Sensors (Switzerland)*, vol. 16, no. 10, 2016.

[105] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, and R. Spalazzese, “Model-Driven Engineering for Mission-Critical IoT Systems,” *IEEE Softw.*, vol. 34, no. 1, pp. 46–53, 2017.

[106] J. Li, X. Hou, D. Su, and J. D. D. Munyemana, “Fuzzy power-optimised clustering routing algorithm for wireless sensor networks,” *IET Wirel. Sens. Syst.*, vol. 7, no. 5, pp. 130–137, Oct. 2017.

[107] S. Hu and J. Han, “Power control strategy for clustering wireless sensor networks based on multi-packet reception,” *IET Wirel. Sens. Syst.*, vol. 4, no. 3, pp. 122–129, Sep. 2014.

[108] T. Omar, Z. Abichar, A. E. Kamal, J. M. Chang, and M. A. Alnuem, “Fault-Tolerant Small Cells Locations Planning in 4G/5G Heterogeneous Wireless Networks,” *IEEE Trans. Veh. Technol.*, vol. 66, no. 6, pp. 5269–5283, 2017.

[109] D. Niyato, E. Hossain, D. I. Kim, and Z. Han, “Relay-centric radio resource management and network planning in IEEE 802.16j mobile multihop relay networks,” *IEEE Trans. Wirel. Commun.*, vol. 8, no. 12, pp. 6115–6125, 2009.

[110] S. J. Kim, S. Y. Kim, B. B. Lee, S. W. Ryu, H. W. Lee, and C. H. Cho, “Multi-hop relay based coverage extension in the IEEE802.16J based mobile WiMAX systems,” in *Proceedings - 4th International Conference on Networked Computing and Advanced Information Management, NCM 2008*, 2008, vol. 1, pp. 516–522.

[111] S. Kim, B. Lee, S. Ryu, H. Lee, and C. Cho, “Cost Effective Coverage Extension in IEEE802 . 16j Based Mobile WiMAX Systems,” *Qual. Serv. Resour. Alloc. WiMAX*, 2008.

[112] B. Lin and P. H. Ho, “Dimensioning and location planning of broadband wireless networks under multi-level cooperative relaying,” *IEEE Trans. Wirel. Commun.*, vol. 8, no. 11, pp. 5682–5691, 2009.

[113] B. Lin, P. Ho, L. Xie, and X. Shen, “Optimal relay station placement in IEEE 802.16j networks,” *Proc. Int. Conf. Wirel. Commun. Mob. Comput.*, pp. 25–30, 2007.

[114] J. Y. Chang and Y. S. Lin, “A clustering deployment scheme for base stations and relay stations in multi-hop relay networks,” *Comput. Electr. Eng.*, vol. 40, no. 2, pp. 407–420, 2014.

[115] I. Networks, H. Lu, and W. Liao, “Joint Base Station and Relay Station Placement for,” *Commun. Soc.*, pp. 1–5, 2009.

[116] B. Lin, M. Mehrjoo, P. H. Ho, L. L. Xie, and X. Shen, “Capacity enhancement with relay station placement in wireless cooperative networks,” in *IEEE Wireless Communications and Networking Conference, WCNC*, 2009.

[117] B. Lin, P. H. Ho, L. L. Xie, X. S. Shen, and J. Tapolcai, “Optimal relay station placement in broadband wireless access networks,” *IEEE Trans. Mob. Comput.*, vol. 9, no. 2, pp. 259–269, 2010.

[118] C. Y. Chang, C. T. Chang, M. H. Li, and C. H. Chang, “A novel relay placement mechanism for capacity enhancement in IEEE 802.16j WiMAX networks,” in *IEEE International Conference on Communications*, 2009.

[119] Y. Yu, S. Murphy, L. M.-P. of the 4th A. workshop On, and U. 2009, "Interference aware relay station location planning for IEEE 802.16 J mobile multi-hop relay network," *dl.acm.org*, pp. 201–208, 2009.

[120] C. Y. Chang and M. H. Li, "A placement mechanism for relay stations in 802.16j WiMAX networks," *Wirel. Networks*, vol. 20, no. 2, pp. 227–243, Feb. 2014.

[121] C. R. Reeves, "Genetic algorithms and neighbourhood search," in *Springer*, 1994, pp. 115–130.

[122] H. Atlam, R. Walters, and G. Wills, "Fog Computing and the Internet of Things: A Review," *Big Data Cogn. Comput.*, vol. 2, no. 2, p. 10, 2018.

[123] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of evolutionary fuzzy systems," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 2, pp. 109–119, 1999.

[124] "IBM: CPLEX" [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/preface/whatdoes.html [Accessed: 12-Jun-2019].

Appendix A. Repair algorithm for VM placement

A candidate solution *X*, as defined in (3.11), either generated randomly or evolved by the experimented evolutionary algorithm (EA), may violate one or more constraints of the VM placement problem, and therefore becomes infeasible. In this chapter, each of the randomly generated candidate solution or evolved by the EA is checked for its feasibility and is repaired infeasible one using the proposed repair algorithm. The stepwise detailed pseudo code of the repair algorithm for the VM placement is presented in Table A.

In step 1 of the Table A, the system parameters, as defined in the *section 3.2*, and population of candidate solution to repair is the input to the repair algorithm. For each candidate solution *X*, as defined in (3.11) and parameters in step 1, the repair algorithm computes the current load status (of CPU, Memory, and Bandwidth) of PMs in vectors *VMcpuSumMat*, *VMmemSumMat*, and *VMbwdSumMat* in step 2. Load on a PM is the sum of the VMs' demand of CPU/memory/bandwidth connected to that PM. Note that a PM is considered overloaded, if current load of a PM exceeds the capacity of that PM. In contrast, a PM is underloaded, if current load of a PM does not exceed the capacity of that PM.

Repair algorithm (RA) enters in a main loop to check feasibility of a candidate solution and repair infeasible candidate solution *X* in step 3. For each *X*, RA computes the overloaded PMs in terms of CPU, Memory, Bandwidth and recorded the same in a vector *pm2Unload* in step 4(a) using information in step 2. Similarly, for each *X*, RA indicates the underloaded PMs in terms of CPU, Memory Bandwidth and recorded the same in a vector *pm2Assign* in step 4(b) using information in step 2.

In step 5, the repair algorithm (RA) checks whether the current solution *X* need to be repaired using information in step 2 and step 4. If a candidate solution *X* is feasible, the RA skips steps 6−22 and accumulate the repaired candidate solution *X* in the set S_1 in step 23. The RA runs from steps 3−24, if more candidate solutions need to be checked for feasibility and the RA terminates at step 25, otherwise.

In case a candidate solution *X* is decided infeasible in step 5, the repair algorithm (RA) runs steps 6−22 to repair the infeasible solution. In *X*, the RA enters a loop for each overloaded PM in step 6. The RA checks overloaded information about each PM in the vector *pm2Unload*. An overloaded PM in the *pm2Unload* can be brought back to less or equal to its maximum load, as defined in the *section 3.2*. In the step 7, a loop is started to disconnect VMs one by one from the overloaded PMs. A disconnected VM from overloaded PM (in *pm2Unload* in step 4 (a)) need to be reconnected to an underloaded PM (in *pm2Assign* in step 4 (b)). Here, in step 8, RA checks whether a disconnected VM can be legitimately reconnected to an underloaded PM in the vector *pm2Assign*. In case this reconnection is feasible, three steps are executed: a VM is assigned to a PM in step 9, the step 2 is repeated in step 10, and a candidate solution *X* is updated in step 11. If the current PM is no more overloaded after disconnecting a VM from overloaded PM, then steps 13−15 are executed and the loop from steps 7−16 is broken. On the other hand, if current overloaded PM is still overloaded, then RA steps 13−15 are skipped and the loop in steps 7−16 continues. The RA loop in steps 7−16, iteratively disconnects VMs from the overloaded PMs and reconnects VMs to underloaded PMs. The RA loop in steps 6−18 is run for each overloaded PM in vector *pm2Unload* to check feasibility of the candidate solution *X*.

The proposed repair algorithm (RA) to repair candidate solution *X*, in (3.11), does not guarantee that each of the repairable (or infeasible) solutions will become feasible solution after executing step 4−18. The reason is that the proposed RA is not checking each VM connection to each PM exhaustively. In other words, the RA only checks for the first available feasible connection between a VM to PM to replace an infeasible connection. If a candidate solution is not repairable (or no valid VM to PM connection is available), the proposed RA randomly generates a new candidate solution *X* and checks its feasibility in steps 19−22. In step 23, the repair algorithm (RA) accumulates the repaired candidate solution(s). The RA loop in steps 3−24 is executed for each candidate solution *X* either to check its feasibility or to repair infeasible one. The set S_1 of feasible (or repaired) solutions is returned by the RA in step 25.

Table A. Repair algorithm for infeasible solutions	
A. Inputs	**Steps:** 1. (a) System parameters such as VMs CPU, memory, and bwd demand of VMs, and PMs CPU, memory, and bwd capacity of PMs, etc. (b) Population of candidate solution(s) to repair.
B. Execution	**Steps:** 2. Calculate demands of CPU/Memory/Bandwidth of all VMs to the corresponding PMs in the vectors *VMcpuSumMat*, *VMmemSumMat*, and *VMbwdSumMat*. 3. ***for*** (each candidate solution *X* to repair) 4. (a) In the *X*, compute overloaded PMs, in vector *pm2Unload*, in terms of CPU, Memory and Bandwidth. (b) In the *X*, compute underloaded PMs, in vector *pm2Assign*, in terms of CPU, Memory and Bandwidth. 5. ***if*** (PMs are overloaded in *pm2Unload*) *// No PM overloaded means X is feasible* 6. ***for*** (each PM to unload in *pm2Unload*) 7. ***for*** (each VM disconnects from PM in *pm2Unload*) 8. ***if*** (a VM assigned to a PM in *pm2Assign* is *feasible*) 9. Assign a VM to a PM in *pm2Assign*. 10. Repeat Step 2. 11. Update the candidate solution *X*. 12. ***end if*** *// a VM is reassigned to a PM* 13. ***if*** (PM is not overloaded) *// No overloaded PM in X* 14. ***break*** *// for loop in steps 7−16 is broken* 15. ***end if*** 16. ***end for*** *// disconnect VMs and reassign to PMs* 17. ***end for*** *// for each overloaded PM* 18. ***end if*** // *No overloaded PMs, feasible X.* 19. ***while*** (*X* is not feasible) 20 Randomly generate a candidate solution *X*. 21. Repeat steps 4 to 18. 22. ***end while*** *// A solution is repaired* 23. Accumulate a repaired solution in the set S_1. 24. ***end for*** *// All feasible solution(s)*
C. Output	25. *r**eturn*** feasible solution(s) set S_1.

Appendix B. Repair algorithm for IoT assignment

A candidate solution, X in (4.14), either generated randomly or evolved by any evolutionary algorithm (EA), may violate one or more constraints of the IoT assignment problem, and therefore become infeasible candidate solution. In this chapter, each of the randomly generated candidate solution or evolved by the experimented EA is checked for its feasibility and is repaired the infeasible one using the proposed repair algorithm (RA). The stepwise pseudo code for the repair algorithm is presented in Table B, and we discuss the operational steps of the proposed repair algorithm for the IoTs-CCNs and CCNs-BSs assignment problem.

In step 1, the system parameters, as defined in the *section 4.2*, and population of candidate solution to repair are input to the repair algorithm. The proposed IoT network comprises of two levels of resource assignments: between IoTs and CCNs, and between CCNs and BSs as discussed in the *section 4.2*. In the repair algorithm, candidate solution X splits into two vectors, $\dot{X} = (X_1, X_2, \dots, X_{|\mathcal{H}|})$ and $\ddot{X} = (X_{|\mathcal{H}|+1}, X_{|\mathcal{H}|+2}, \dots, X_{|\mathcal{H}|+|\mathcal{M}|})$ in step 2. The repair algorithm checks feasibility or repairs the infeasible vectors $\dot{X}$ and $\ddot{X}$ separately and concatenates both $\dot{X}$ and $\ddot{X}$ vectors as an X vector to return as a feasible candidate solution in step 32. For each $\dot{X}$ using parameters in step 1, the repair algorithm computes vectors *IoTcpuSumMat* and *IoTmemSumMat* to record the current load status of CCNs in step 3. Note that a CCN is considered overloaded, if current load of a CCN exceeds capacity of a CCN. The load on a CCN is the sum of IoTs' CPU and memory demands connected to that CCN. A candidate soliton X is considered infeasible, if one or more CCNs are overloaded in $\dot{X}$. In contrast, a CCN is considered underloaded in $\dot{X}$, if current load does not exceed the capacity of that CCN.

Repair algorithm enters in a main loop to repair (or to check feasibility of) each candidate solution X in step 4. Using $\dot{X}$ and information in step 3, the repair algorithm computes the overloaded CCNs in terms of CPU, Memory and recorded the same in a vector *ccn2Unload* in step 5(a). Similarly, using $\dot{X}$ and information in step 3, the repair

algorithm computes underloaded CCNs in terms of CPU, Memory and recorded the same in a vector *ccn2Assign* in step 5(b).

In step 6, the repair algorithm checks whether the vector $\dot{X}$ need to be repaired using information in step 3 and step 5. If the $\dot{X}$ is feasible, the repair algorithm skips steps 7−23. However, the repair algorithm runs from steps 7−23, in case $\dot{X}$ vector is infeasible and need to be repaired.

In case a vector $\dot{X}$ is decided infeasible in step 6, the repair algorithm runs steps 7−23 to generate a feasible solution. In $\dot{X}$, the repair algorithm enters a loop for each overloaded CCN in step 7. The RA checks overloaded information about each CCN in the vector *ccn2Unload*. An overloaded CCN in the *ccn2Unload* can be brought back to less or equal to its maximum load, as defined in the *section 4.2*. In the step 8, a loop is used to disconnect IoTs one by one from the overloaded CCNs. A disconnected IoT from an overloaded CCN (in *ccn2Unload* in step 5 (a)) need to be reconnected to an underloaded CCN (in *ccn2Assign* in step 5 (b)). Here, in step 9, the repair algorithm checks whether a disconnected IoT can be feasibly reconnected to an underloaded CCN in the *ccn2Assign*. In case this reconnection is feasible, three steps are executed: an IoT is assigned to a CCN in step 10, the step 3 is executed in step 11, and a vector $\dot{X}$ is updated in step 12. If the current CCN is no more overloaded after disconnecting an IoT from overloaded CCN, then steps 14−16 are executed and the loop from steps 8−17 is broken. On the other hand, if currently overloaded CCN is still overloaded, then the repair algorithm skips the steps 14−16 and the loop in steps 8−17 continues. The repair algorithm loop in steps 8−17, iteratively disconnects IoTs from the overloaded CCNs and reconnects IoTs to underloaded CCNs. The repair algorithm loop in steps 7−19 is run for each overloaded CCN in the *ccn2Unload* to check feasibility of the vector $\dot{X}$.

The proposed repair algorithm does not guarantee that each of the repairable (or infeasible) vector $\dot{X}$ will become feasible after executing steps 5−19. The reason is that the repair algorithm is not checking each IoT connection to each CCN exhaustively. In other words, the repair algorithm only checks for the first available feasible connection between

an IoT to a CCN to replace an infeasible connection. If a candidate solution is not repairable (or no feasible IoT to CCN connection is available), the repair algorithm randomly generates a new vector $\dot{X}$ in step 21 and checks its feasibility.

After checking feasibility or repairing infeasible vector $\dot{X}$, the repair algorithm checks feasibility or repair infeasible vector $\ddot{X}$ in steps 24−31. The repair algorithm enters the loop for each CCN in IoT network at step 24 to check the used/unused status of CCNs in the $\dot{X}$. If a CCN is not serving any IoT in the vector $\dot{X}$, assign a '*0*' value to the corresponding CCN in $\ddot{X}$ (each component in $\ddot{X}$ represents a CCN, *see section 4.3.1*). Note that '*0*' value in $\ddot{X}$ means the corresponding CCN is not in use. On the other hand, if a CCN is serving IoT(s) in $\dot{X}$ and the corresponding CCN is a '*0*' value in $\ddot{X}$, then assign a BS randomly (from $k = 1,2, \dots, |\mathcal{G}|$) to the corresponding component in the vector $\ddot{X}$. Note that any nonzero value in $\ddot{X}$ means the corresponding CCN is in use. For each vector $\dot{X}$, the repair algorithm in steps 24−31 returns the feasible vector $\ddot{X}$.

In step 32, the repair algorithm concatenates $\dot{X}$ and $\ddot{X}$ vector to the vector X and accumulates the repaired candidate solution in the set S_1. For each candidate solution X, the repair algorithm loop, in steps 4−34, is executed for checking feasibility or repairing the infeasible ones. The set S_1 of feasible candidate solution(s) is returned by the repair algorithm in step 35.

Table B. Repair algorithm for infeasible solutions

A. Inputs	**Steps:** 1. (a) System parameters such as IoTs: CPU and memory demand, CCNs: CPU and memory capacity, etc. (b) Population of candidate solution(s) to repair.
B. Execution	**Steps:** 2. Split candidate solution X into two vectors, $\dot{X} = (X_1, X_2, \dots, X_{\lvert\mathcal{H}\rvert})$ and $\ddot{X} = (X_{\lvert\mathcal{H}\rvert+1}, X_{\lvert\mathcal{H}\rvert+2}, \dots, X_{\lvert\mathcal{H}\rvert+\lvert\mathcal{M}\rvert})$ *// see section 4.3.1.* 3. Calculate demands of CPU/Memory of all IoTs to a CCN in the vectors *IoTcpuSumMat* and *IoTmemSumMat*. 4. ***for*** (each candidate solution X to repair) 5. (a) Using $\dot{X}$, compute overloaded CCNs, in vector *ccn2Unload*, in terms of CPU and Memory. (b) Using $\dot{X}$, compute underloaded CCNs, in vector

	ccn2Assign, in terms of CPU and Memory. 6. ***if*** (CCNs are overloaded in *ccn2Unload*) *// No CCN overloaded means connections in $\dot{X}$ is feasible* 7. ***for*** (each CCN to unload in *ccn2Unload*) 8. ***for*** (each IoT disconnects from a CCN in *ccn2Unload*) 9. ***if*** (an IoT assigned to a CCN in *ccn2Assign* is feasible) 10. Assign an IoT to a CCN in *ccn2Assign*. 11. Repeat Step 3. 12. Update the vector $\dot{X}$. 13. ***end if*** *// an IoT is reassigned to a CCN* 14. ***if*** (CCN is not overloaded) *// No overloaded CCN in $\dot{X}$* 15. ***break*** *// for loop in steps 8−17 is broken* 16. ***end if*** 17. ***end for*** *// disconnect IoTs and reassign to CCNs* 18. ***end for*** *// for each overloaded CCN* 19. ***end if*** // *No overloaded CCNs, and $\dot{X}$ is feasible.* 20. ***while*** ($\dot{X}$ is not feasible) 21 Randomly generate a vector $\dot{X}$. 22. Repeat steps 5 to 19. 23. ***end while*** *// $\dot{X}$ is finally repaired* 24. ***for*** (each CCN in IoT network) 25. ***if*** (CCN is not serving any IoT in $\dot{X}$) *// see 4.3.1 for further clarification on X, $\dot{X}$, and $\ddot{X}$.* 26. Assign a '*0*' value to the corresponding CCN in $\ddot{X}$. *// Note that '0' value in $\ddot{X}$ means CCN is not in use* 27. ***end if*** 28. ***if*** (CCN is serving IoT(s) in $\dot{X}$ and has '*0*' value in $\ddot{X}$) 29. Assign a BS randomly to the corresponding CCN in $\ddot{X}$. *// Note that replacing '0' in $\ddot{X}$ means CCN is in use* 30. ***end if*** 31. ***end for*** *// repaired $\ddot{X}$* 32. $X = \dot{X} + \ddot{X}$ *// Concatenate $\dot{X}$* and $\ddot{X}$ 33. Accumulate a repaired solution X in the set S_1. 34. ***end for*** *// All feasible solution(s)*
C. Output	35. ***return*** feasible solution(s) set S_1.

Appendix C. Repair algorithm for BWN planning

A candidate solution, X in (5.13), either generated randomly or evolved by any evolutionary algorithm (EA), may violate one or more constraints of the broadband wireless network (BWN) planning, and therefore become infeasible. In this chapter, each of the randomly generated candidate solution or evolved by the experimented EA is checked for its feasibility and is repaired the infeasible one using the proposed repair algorithm. The stepwise pseudo code for the repair algorithm is presented in Table C, and we also discuss the operational steps of the proposed algorithm.

In step 1 of the Table C, the system parameters, as defined in the *section 5.2*, and population of candidate solution to repair are input to the repair algorithm. The repair algorithm for the broadband wireless network (BWN) planning comprises of mainly two levels of feasibility check: (1) steps 3−15 (check wireless links feasibility among communicating nodes) and (2) steps 20−32 (feasible load on BSs and RSs). A communication link between any two nodes (i.e., BS, RS, and TP) is considered infeasible until flow among communicating nodes is greater than the maximum link (or channel) capacity. Note that the upper bound of link capacity (e.g., channel capacity) is defined in the Table 5.2. Each BS and RS has maximum load capacity, which is defined in the Table 5.2. The repair algorithm makes sure that the load on deployed BSs and RSs must not be greater than the maximum loads capacity. In case the load on BSs/RSs is greater than the maximum load capacity, these BSs/RSs are considered overloaded nodes. A candidate soliton X is considered infeasible, if one or more BSs/RSs are overloaded. In contrast, BSs/RSs are considered underloaded in X, if current load does not exceed the maximum load capacity.

In the first level of feasibility check, the repair algorithm makes sure that candidate solution X should have feasible wireless links among various communicating nodes (i.e., BS, RS, and TP). Then, repair algorithm checks the load feasibility on BSs and RSs in the second level. In other words, the repair algorithm checks the load constraints (on BSs and RSs) only, if repair algorithm successfully validates the wireless link constraints among various communicating nodes in the second level. If candidate solution X is irreparable

until the step 14, then the repair algorithm, using steps 16−19, randomly generates a new candidate solution, *X*, and the same is repaired by executing the steps 3−15. On the same token, if candidate solution *X* is irreparable until the step 32, then the repair algorithm, using the steps 33−36, randomly generates a new candidate solution, *X*, and the same is repaired by executing the steps 3−32 in the Table C.

The repair algorithm checks feasible wireless links among BSs, RSs, TPs in *X* in steps 3−15. In step 4, the repair algorithm computes link flows between a BS to RS (i.e., *f-BS-RS*), a BS to TP (i.e., *f-BS-TP*), a RS to TP (i.e., *f-RS-TP*) respectively. The repair algorithm in the step 5, verifies whether flow between a BS to RS (i.e., *f-BS-RS*) is greater than the maximum link capacity. In case the link is not feasible, the repair algorithm in step 6 disconnects an infeasible BS to RS link and try to establish a feasible link between a RS to other BSs nodes. Similarly, the repair algorithm verifies the feasibility or repairs (in case of infeasible) links between BSs to TPs (steps 8−9) and between RS to TPs (steps 11−12). In step 14 of the repair algorithm, candidate solution *X* is updated with feasible links and the links verification loop among various nodes ends from steps 3−15.

The proposed repair algorithm does not guarantee that each of the repairable (or infeasible) link (among communicating nodes) will become feasible after executing steps 3−15. The reason is that the proposed repair algorithm is not checking each communicating link exhaustively. In other words, the repair algorithm only checks for the first available feasible link among communicating nodes to replace the infeasible link. If a candidate solution is not repairable (or no feasible link among communicating node is available), the proposed repair algorithm randomly generates a new candidate solution *X* and checks its feasibility in steps 16−19.

The repair algorithm in steps 20−32 is checking the load constraints on BSs (in steps 21−26) and RSs (in steps 27−32) for the second level of feasibility check. In step 20, the repair algorithm computes the load on BSs and RSs in the vectors *loadOnBSs* and *loadOnRSs* respectively. For each overloaded BS in steps 21−26, the repair algorithm disconnects TPs from the overloaded BS and reconnect disconnected TPs to an

underloaded BS/RS subject to maximum link capacity and maximum loads on BSs and RSs in step 23. After executing step 23, the repair algorithm updates the candidate solution *X* and updates the vectors *loadOnBSs* and *loadOnRSs* in step 24. Similarly, for each overloaded RSs in steps (27−28), the repair algorithm disconnects TPs from the overloaded RS and reconnect TPs to an underloaded BS/RS subject to maximum link capacity and maximum loads on BSs and RSs in step 29. Then, the repair algorithm again updates the candidate solution *X* and updates the vectors *loadOnBSs* and *loadOnRSs* in step 30.

The repair algorithm may fail to repair (in step 33) load constraints (on BSs and RSs) for a candidate solution *X* after executing the steps 20−32. In this case, the repair algorithm generates a new candidate solution *X* (in step 34) and then repair algorithm runs both levels of feasibility check (from steps 3−32) again in step 35. The feasible candidate solution is accumulated in set S_1 (in step 37). The main loop of repair algorithm (in steps 2−38) runs for each candidate solution *X* of the population. Finally, the repair algorithm returns the set S_1 of feasible candidate solution(s).

Table C. Repair algorithm for infeasible solutions

A. Inputs	**Steps:** 1. (a) System parameters such as BSs and RSs hardware costs and maximum loads, TPs data traffic demand, path loss in various wireless links, etc. (b) Population of candidate solution(s) to repair.
B. Execution	**Steps:** 2. ***for*** (each candidate solution *X* in a population) 3. ***for*** (each link to validate in *X*) 4. Compute link flows among BSs, RSs, TPs links *f-BS-RS*, *f-BS-TP*, *f-RS-TP* respectively. 5. ***if*** (*f-BS-RS* is infeasible) *// greater than link capacity* 6. Disconnect an infeasible BS-RS link try to establish a feasible link between a BS-RS. 7. ***end if*** 8. ***if*** (*f-BS-TP* is infeasible) *// greater than link capacity* 9. Disconnect an infeasible BS-TP link and try to establish a feasible link between a BS-TP/RS-TP. 10. ***end if*** 11. ***if*** (*f-RS-TP* is infeasible) *// greater than link capacity* 12 Disconnect an infeasible RS-TP link and try to

	establish a feasible links between a BS-TP/RS-TP. 13. ***end if*** 14. Update the candidate solution *X*. 15. ***end for*** *// end links validation* 16. ***while*** (link(s) of *X* are not validated) 17 Randomly generate a candidate solution *X*. 18. Repeat steps 3 to 15. 19. ***end while*** *// Solution X with validated links* 20. Calculate load on each BS and RS in vectors *loadOnBSs* and *loadOnRSs* in *X*. 21. ***for*** (load on each BS in *X*) 22. ***if*** **(**load on a BS is infeasible in *loadOnBSs*) 23. Disconnect TPs from an overloaded BS and reconnect to an underloaded BS/RS subject to maximum link capacity and maximum loads on BSs and RSs. 24. Update *X* and update vectors *loadOnBSs* and *loadOnRSs*. 25. ***end if*** 26. ***end for*** *// Solution X with validated links, BS loads* 27. ***for*** (load on each RS in *X*) 28. ***if*** **(**load on a RS is infeasible in *loadOnRSs*) 29. Disconnect TPs from an overloaded RS and reconnect to an underloaded BS/RS subject to maximum link capacity and maximum loads on BSs and RS. 30. Update *X* and update vectors *loadOnBSs* and *loadOnRSs*. 31. ***end if*** 32. ***end for*** *// possibly, X with validated links, BS loads, and RS loads* 33. ***while*** **(**load on BSs and RSs is still infeasible) 34. Generate a candidate solution *X* randomly. 35. Execute lines from 3-32. 36. ***end while*** *// X with validated links, BS loads, and RS loads* 37. Accumulate a repaired solution *X* in the set S_1. 38. ***end for***
C. Output	39. ***return*** the feasible candidate solution(s) of S_1.