Complex & Intelligent Systems (2022) 8:361–392
https://doi.org/10.1007/s40747-021-00368-z

ORIGINAL ARTICLE

Multi-objective fault-tolerant optimization algorithm for deployment of IoT applications on fog computing infrastructure

Yaser Ramzanpoor[1] · Mirsaeid Hosseini Shirvani[2] · Mehdi Golsorkhtabaramiri[1]

Received: 24 November 2020 / Accepted: 1 April 2021 / Published online: 6 May 2021

Abstract
Nowadays, fog computing as a complementary facility of cloud computing has attracted great attentions in research communities because it has extraordinary potential to provide resources and processing services requested for applications at the edge network near to users. Recent researchers focus on how efficiently engage edge networks capabilities for execution and supporting of IoT applications and associated requirement. However, inefficient deployment of applications' components on fog computing infrastructure results bandwidth and resource wastage, maximum power consumption, and unpleasant quality of service (QoS) level. This paper considers reduction of bandwidth wastage in regards to application components dependency in their distributed deployment. On the other hand, the service reliability is declined if an application's components are deployed on a single node for the sake of power consumption management viewpoint. Therefore, a mechanism for tackling single point of failure and application reliability enhancement against failure are presented. Then, the components deployment is formulated to a multi-objective optimization problem with minimization perspective of both power consumption and total latency between each pair of components associated to applications. To solve this combinatorial optimization problem, a multi-objective cuckoo search algorithm (MOCSA) is presented. To validate the work, this algorithm is assessed in different conditions against some state-of the arts. The simulation results prove the amount 42%, 29%, 46%, 13%, and 5% improvement of proposed MOCSA in terms of average overall latency respectively against MOGWO, MOGWO-I, MOPSO, MOBA, and NSGA-II algorithms. Also, in term of average total power consumption the improvement is about 43%, 28%, 41%, 30%, and 32% respectively.

Keywords Internet of things (IoT) · Fog computing · Fault tolerance · Traffic-aware deployment · Component deployment

Introduction

Recently, fog computing joint with cloud computing to cover its deficit such as intrinsic latency and to serve different industries. Since a fog server can process data gathered by IoT devices independently from cloud computing, it can efficiently save network communication bandwidth, cloud storage space, and reserving resources for mission-critical applications [1]. Also, fog supports unifying edge and cloud resources for customers. Fog computing facilitates deployment of IoT applications in vicinity of source data. Therefore, it reduces network load and guarantees on-time service delivery. However, deployment, management, and updating of IoT application lead new challenges in such layered environment. Fog computing in larger scale includes numerous heterogeneous computing nodes with separate processing, memory, and storage. In addition to, workload on each node is completely dynamic. Also, each IoT application has its own requirement in terms of sensitivity on latency, computing requirement, and privacy constraints. Therefore, the deployment of application components must be properly done on fog nodes; at the same time the application requirement, software and hardware features, bandwidth and tolerable latency between components on fog infrastructure must be taken into account [2]. Deployment of an application components on a single node yields maximize resource utilization, decrease in power consumption, and optimizing network bandwidth as well. Nevertheless, when

✉ Mirsaeid Hosseini Shirvani
mirsaeid_hosseini@yahoo.com;
mirsaeid_hosseini@iausari.ac.ir

1 Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran

2 Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

a fog node which hosts all of the components associated with an application crashes, the application cannot work properly in which it affects the reliability of customer applications. For this reason, it is clear-cut to take an efficient policy for a suitable and reliable components deployment scheme.

There are miscellaneous mapping possibilities in distribution of application components on fog nodes in which one of the most appropriate and optimal amongst them should be selected. For a small application with low number of components, there are several feasible solutions to deploy components on different fog nodes. Therefore, with the increase the number of application components and the number of fog nodes regarding to its heterogeneity, finding the optimal deployment scheme is computationally complex and there is not any exact solution for this. So, this problem belongs to NP-Hard class [3]. Recently, researches have been done in literature in regards to component distribution over fog and cloud computing nodes. A unified fog computing platform was proposed by Hong et al. [4] in year 2018 for dynamic component deployment on fog devices. In their proposed approach, it paid on distribution of component over more than one fog node to avoid single point of failure. Another algorithm for distribution of IoT application components with regards to application sensitivity on latency and efficient network resource usage viewpoints has been proposed by Taneja et al. [5] in year 2017. A general and extensible description model was proposed to specify QoS-aware IoT application deployment on fog infrastructure proposed by Brogi et al. [6] in 2017. Review on literature reveals that there are clear lack in component placement of IoT applications with two different viewpoints at the same time. In the other words, this paper presents power-aware and latency-aware algorithm for reliable component deployment on fog infrastructure. The former awareness is for provider as a prominent stakeholder and latter awareness is considered for service customer as another prominent stakeholder side viewpoints. To this end, this paper presents two new models in IoT-Fog environment in regards to application modules deployment viewpoint. The accurate models indicate whether the proposed algorithm is effective or not. So, after presenting two intricate new models namely power and reliability models for IoT components deployment on fog platforms, the multi-objective cuckoo search algorithm is extended which exploits Pareto dominance and crowding distance concepts for both gaining the set of non-dominated solutions and diversity in search space. Since the stated problem is a discrete optimization in nature, the CSA algorithm that permutes search space efficiently has been selected. Also, its operators are conducted in such a way that the good adjustment and balance between exploration and exploitation is achieved in which the final simulation results endorse it although there is no guarantee in stochastic approaches to reach optimal point.

Therefore, the main contributions of the current paper are as follow:

1. To reach the optimal power consumption, a *Fullmesh* sub networks is extracted from whole fog network by a proposed heuristic algorithm; among *Fullmesh* sub networks, the most appropriate one is selected for distribution of application components.
2. To mitigate the effect of single point of failure in application components deployment, the fault tolerance policy against failure is provided for each application to improve reliability; to this end, the minimum number of fog nodes for components deployment can be bounded to the maximum number of existing nodes in *Fullmesh* sub network.
3. The overall latency concept is modeled. In the process of application components deployment, efficient utilization of fog bandwidth resource is increased by minimizing overall latency. This can be potentially decrease resource wastage and power consumption.
4. The deployment of application components over fog nodes is formulated to a multi-objective optimization problem with minimization of both power consumption and overall latency viewpoints. To solve this combinatorial problem, a multi-objective cuckoo search optimization algorithm (MOCSA) is presented which compromises objectives and considers reliability in its constraints.

The rest of the paper is structured as follows. Related works are placed in Sect. "Related works". Some models associated to problem statement are presented in Sect. "Proposed framework and models". Section "Problem statement" states the problem under study. Proposed MOCSA is presented in Sect. "Proposed MOCSA algorithm for component deployment problem". This algorithm is validated in simulation and evaluation section which is placed in Sect. "Simulation and evaluation". Section "Conclusion and future direction" concludes this paper along with future direction.

Related works

This section investigates related works to find research gap in component deployment problem. A cloud service management standard named TOSCA was proposed for IoT component placement [7]. The main objective of this paper was to deploy components automatically by using application components description commensurate with fog nodes. The aid of this standard was to improve portability of applications in heterogeneous environment such as in cloud and fog environment. In proposed standard, a model for description of service structure and service process management was presented. In this model, placement of application components is automatically done by applying conceptual description of components topology and related application deployment.

An approach has been propounded in literature for latency-aware application component management in fog environment [8]. In this work, latency of service access, service delivery time, and internal communication latency have been considered. The objective was to guarantee the service delivery deadline and efficient resource utilization in fog environment. To optimize the number of utilized fog nodes for hosting application components, this exploits forward and reallocation strategy for application components. In addition, to cope with limitations of fog environment such as management overhead, single point of failure, redundant communications, and latency in decision, the decentralized organizing is proposed for substitution and forwarding the components.

A platform was proposed for a dynamic distribution of application components on fog sub networks [4]. In proposed approach, all requests are submitted to a server; then, the requests are registered in a database. Each request is split to multiple components which are encapsulated to a Docker or Container. Afterwards, a heuristic algorithm is run to determine components placement plan. The obtained plan is sent to fog platform for component distribution. The main goal is to maximize of generating successful placement plans for user applications.

A DIANE framework has been presented by Vogler et al. [9] in 2015 for producing optimal deployment topology of cloud-based IoT applications commensurate with existing infrastructures. To increase the flexibility of application that their deployment topologies undergo evolution during the time, separation of executing components is necessary. The application deployment topology changes may be for deployment requirement of new application, changes in edge network physical infrastructure such as add/remove sensors and gateways, environmental changes such as customer request patterns, and evolutionary changes in business logic during its life cycle. In production process of deployment topology, some parameters such as time needed for deployment, time and bandwidth request for application running, and exploitation of edge devices are evaluated.

A distributed programming interface was presented for colony of fog computing nodes so-called Foglets by Saurez et al. [10] in 2016. Foglets automatically detect fog computing resources in network hierarchy and deploys application components on fog nodes with tolerable latency requirement of each component.

An approach was devised for component deployment of IoT services on M2M platform to reduce traffic from the network to cloud datacenter because IoT application are made on M2M platforms [11].

A network-aware algorithm in regarding to optimal utilizing of resource was presented by Taneja et al. [5] in 2017. This algorithm detects fog nodes based on their capacity and application components requirement. If requirement is met, the mapping of components over fog nodes is done.

To facilitate deployment of applications on cloud2fog environment, a platform as a service (PaaS) architecture was propounded by Yangui et al. [12] in year 2016. In this architecture, engaging and execution of application components, SLA meeting evaluation and component migration via management interface are met. Accordingly, exploitation and execution of application components with regards to the objectives are detected, configured, and initiated.

Table 1 summarizes comparison of related works associated to IoT application component deployment on fog and a cloud infrastructure.

Review of literate illustrates that published works have been formulated to optimization problems with different viewpoints. Generally, optimization problems are categorized in two classes: single objective and multi-objective problems. Since the majority of optimization problems belong to NP-Hard category problems, the heuristics (or exact algorithms) and the meta-heuristic algorithms are engaged to solve these kind of problems. In single objective problems, only one objective function must be optimized. For instance, Refs. [13–17] were presented in literature to solve single objective engineering problems with heuristic and exact approaches. Some meta-heuristics GA-based [18–23], PSO-based [3, 24, 25], SA-based [26–28] have been developed to solve optimization problems in engineering domain. In addition, multi-objective optimization algorithms such as NSGA-II [29], MOPSO [30], MOGA [18, 31], MOBA [32], and MOGWO [33] among others have been extended in literature to solve multi-objective optimization problems which need to make a trade-off between conflicting objectives at the same time. In this line, several techniques were presented in literature to improve the quality of multi-objective optimization problems [34–38]. Specially, these methods were tested in some famous and applicable engineering benchmarks [34–38]. Since the modules placement associated to IoT application in fog environment is a discrete optimization problem, it urges to utilize an efficient discrete optimization algorithm this the reason to select CSA algorithm which permutes search space efficiently.

Overall investigation of reviewed literature also reveals that the majority of published works scarcely have paid on single point of failure avoidance and its effect on how to distribute application components over fog nodes and at the same time how to optimize bandwidth utilization. The distinction point of the current paper in comparison to other literatures revolves around the fact that the current paper strives in enhancement of user application's reliability in regards to tolerance against failure and to present

Table 1 Summary of the literature study

Author/Ref	Deployment aims						Advantages	Disadvantages
	Distributed	Fault tolerant	Resource aware	Latency Aware	Traffic aware	Energy efficient		
Mahmud et al. (2018) [8]	✓	✖	✓	✓	✖	✓	Deployed time-sensitive applications at proximity of source data	Lack of considering the chain of dependency during distribution process
Hong et al. (2016) [4]	✓	✖	✓	✓	✖	✓	Component distribution on the minimum number of computing nodes	It does not elaborate how to distribute components against one point of failure
Vögler et al. (2015) [9]	✓	✖	✓	✖	✖	✓	A framework presented for generating optimal deployment topology A descriptive model presented for component deployment	It does not elaborate how to distribute components
Saurez et al. (2016) [10]	✓	✖	✖	✖	✖	✖	A programming infrastructure for development and deployment of components An approach presented for components migration	It does not dependency challenges between components
Chen et al. (2017) [11]	✓	✖	✖	✖	✖	✖	Component distribution with minimum latency	QoS degradation with increase the number of components Lack of elaboration between components' dependency A single point of failure problem
Taneja et al. (2017) [5]	✓	✖	✓	✓	✖	✓	Supporting different netwrok topologies	Lack of elaboration between components' dependency A single point of failure problem
Yangui et al. (2016) [12]	✓	✖	✖	✓	✖	✖	Automated PaaS for componentt deployment	It does not guarantee optimal deployment A single point of failure problem
Current article	✓	✓	✓	✓	✓	✓	Reliability enhancement Traffic-aware deployment	Although it is not a weakness, it depends on sub full mesh derived from whole network

traffic-aware deployment to optimize network bandwidth utilization in component distribution process.

It is worth noting that presenting the accurate models indicate whether the proposed algorithm is effective or not. So, this paper presents two intricate new models namely power and reliability models for IoT components deployment on fog platforms to cover literature shortcomings. Then, it is formulated to multi-objective optimization problem.

Proposed framework and models

This section presents system framework and associated models. Then, all of them are engaged in problem statement. For the sake of simplicity, Table 2 illustrates utilized nomenclature in presented models.

Table 2 Nomenclature utilized in proposed models

Notation	Description
F	Fog network
Mega node	A fullmesh sub network including fog nodes
N	Number of fog nodes in Mega Node
M	Number of Applications Components
fn_i	Fog node i, where $i = 1, 2,..,N$
Id	Fog node identifier
H	Fog node hardware specification
S	Fog node software specification
HW_{fn}	Computing, memory, and storage capacity of a fog node
$\mathrm{SW}_{\text{Mega Node}}$	Software capacity of Mega Node
$\mathrm{S}_{\text{Mega Node}}$	Sensor capacity of Mega Node
sensorlist	Sensor list associated to a fog node
$\mathbf{B}$	Bandwidth of communication link
$\mathbf{L}$	Latency of communication link
B_{mn}	Communication Bandwidth between nodes m and n
L_{mn}	Communication latency between nodes m and n
d_{mn}	Distance between nodes m and n
n_L	Serving fog nodes to application i
UApp	User application
M_i	Number of components in Application i
$cmplist_i$	List of Components associated to application i
cmp_k	k-th component of an application
h	Hardware requested for a component
$\mathbf{s}$	Software requested for a component
m_i	A component of an application i deployed on a fog node L
b_{ij}	Favorite Communication Bandwidth between component i and j
l_{ij}	Favorite Communication latency between component i and j
hw_{cmp}	Computing, memory, and storage capacity requested for application components
sw_{cmp}	Software resource requested for application components
s_{cmp}	Sensor resource requested for application components
t_{ij}	traffic between component i and j
$x_{cmp,fn}$	Decision variable which determines a component is deployed on a fog node or not
y_{fn}	Decision variable which determines a fog node is active or not

System framework

The proposed target system framework is depicted in Fig. 1. As this figure shows, an organizer is placed in top level of fog layer. One of its most missions is to extract Fullmesh sub networks of fog nodes known as a Mega Node. The Mega Node architecture is similar to wireless mesh network (WMN) presented by Akyildiz et al. [39] in year 2005. Its computing pattern differs from traditional mesh networks in which it utilizes network of fog nodes such as switches and routers in distribution operation of inside the network. After the Mega Nodes extraction, the suitable Mega Node is adopted and organizer makes decision for component deployment in selected Mega Node in regards to application components features and requirements. Conceptually, the organizer is centralized, but it can be distributedly implemented for the sake of avoidance from the single point of failure phenomenon.

In the proposed framework, the high priority is to extract deployment plan based on selected Mega Node; then, the components are distributed based on extracted plan. Only the components which are not time-sensitive or are executed periodically for information processing are deployed on cloud infrastructure. In this regards, a deployment planner framework is used to manage and run suitable application components deployment regarding to system performance.

As Fig. 2 demonstrates, planner module contains application component manager and associated collaborative components. Beside deployment planner, some modules are placed for storage and retrieval of information associated to the network and other Mega Node's resources.

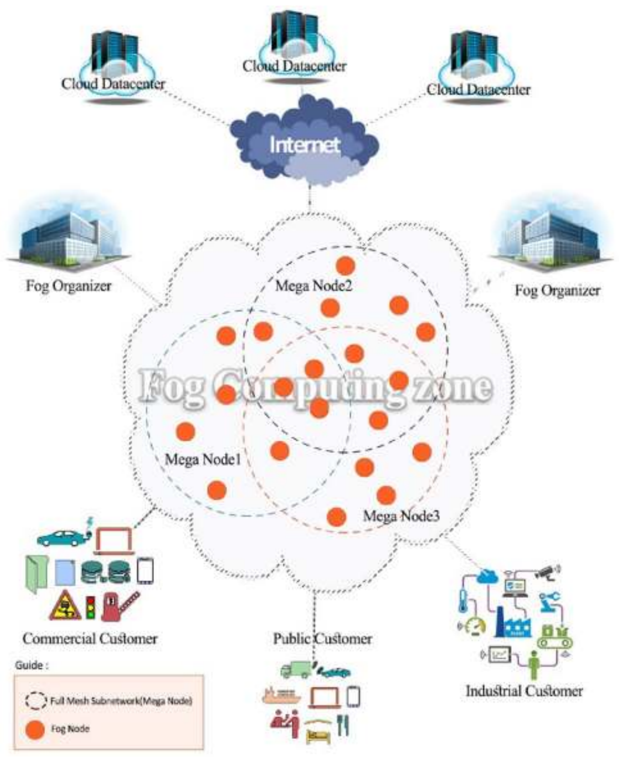


Fig. 1 Proposed system framework and associated mega nodes

The integrated information is used for management of application components and presenting favorite deployment plan via deployment planner. In the following, the proposed framework's modules are clarified.

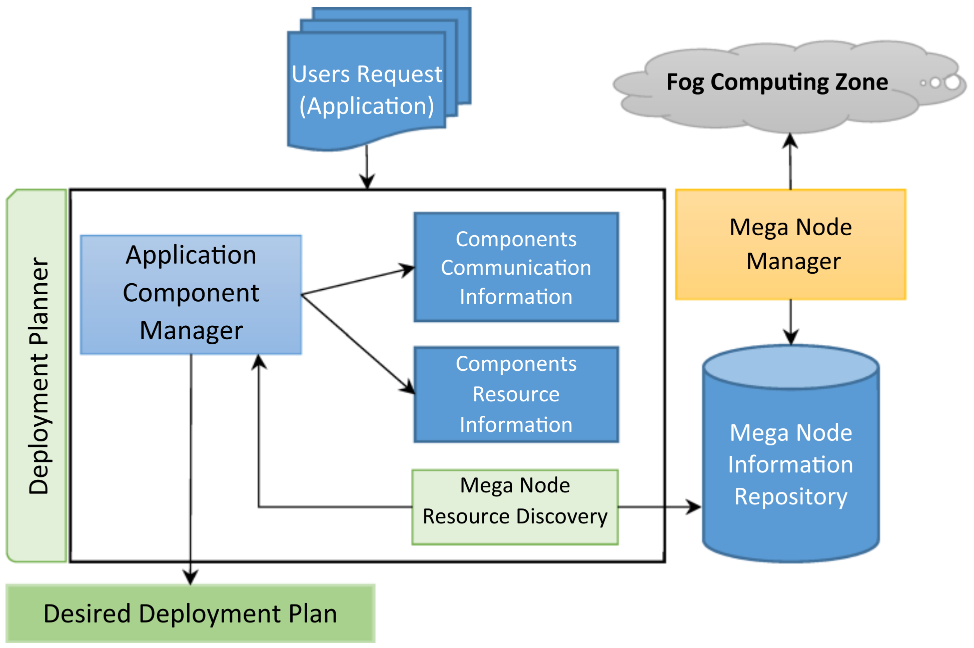
Application component manager This is a main module amongst others, which decides how to deploy application components on fog or cloud nodes. In a multi-component application, for the sake of dependency between its components, decision of deployment strongly depend on several issues such as resource availability, network structure, QoS requirement of applications, load sharing and etc. the deployment of components can be done based on objectives such as power consumption reduction, minimizing communication and reduction of overall traffic owing to running of applications.

Component resource information It extracts processing and memory requirement associated to application components from user submitted request. Then, it delivers this information to application component manager for decision making on deployment plan.

Components communication information Since communication plays a major role in resource consumption of fog nodes in running IoT applications, the management of application components on fog nodes includes optimizing usage of computing resources, memory, and communications at the same time. To this end, this section extracts communication information of application components from user requests and delivers it to application component manager.

Mega node resource discovery This module manipulates Mega Node's information repository which is obtained via application component manager. Then, it sends back the information of favorite Mega Node for application components deployment.

Fig. 2 Management framework for application components


Users Request (Application)
Fog Computing Zone
Deployment Planner
Application Component Manager
Components Communication Information
Components Resource Information
Mega Node Resource Discovery
Mega Node Manager
Mega Node Information Repository
Desired Deployment Plan

Mega node manager Based on information received from fog nodes, the Fullmesh sub networks of fog nodes are extracted; then, information of Fullmesh sub networks, known as Mega Nodes, are saved in a repository. In addition to, it validates status of existing Mega Nodes by periodically monitoring of fog infrastructure.

Fog model

This article assumes there exists a network of N number of fog nodes which are heterogeneous in terms of processing capacity and power consumption; all of them are enable to store and execute application components. These fog nodes belong to one or more Mega Node sets. Each node in a Mega Node can directly or indirectly access to different kind of sensors via wired or wireless connections. A fog node $fn \in F$ is introduced by a vector $(\mathrm{id}, \mathrm{mid}, H, S, sensorlist)$ where $\mathrm{id}, \mathrm{mid}, H, S$, and $sensorlist$ are fog node identifier, Mega Node id, hardware, software, and available sensors respectively. The components which are distributed among Mega Node's processors can avail to the software and sensors of that same Mega Node. In this regards, the communication link can be modeled by a vector (L,B) where L and B are latency and bandwidth respectively. The details of a Mega Node is elaborated in Fig. 3.

In this line, the communication network is modeled by a graph $\mathrm{G} = <FN,D>$ where $FN = \{fn_1, fn_2, \ldots, fn_N\}$ is a set of fog nodes and edge $d_{\mathrm{ij}} \in D$ shows distance between nodes fn_{i} and fn_{j}. Matrix D in Eq. (1) is dedicated for distance between each pair of fog nodes. In each Mega Node, if all components are placed on single node, then, $d_{ij} = 0$; otherwise $d_{ij} = 1$. In addition, the Fig. 4 illustrates a communication network in a Mega Node with three different fog nodes.

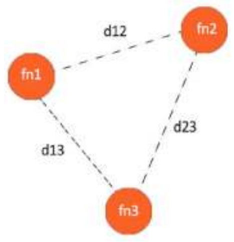
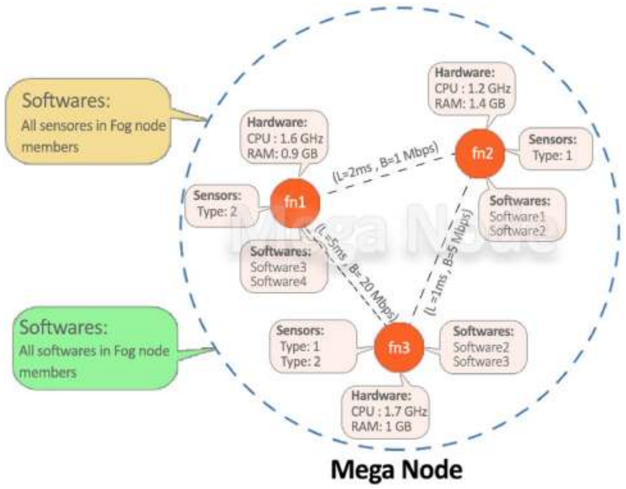


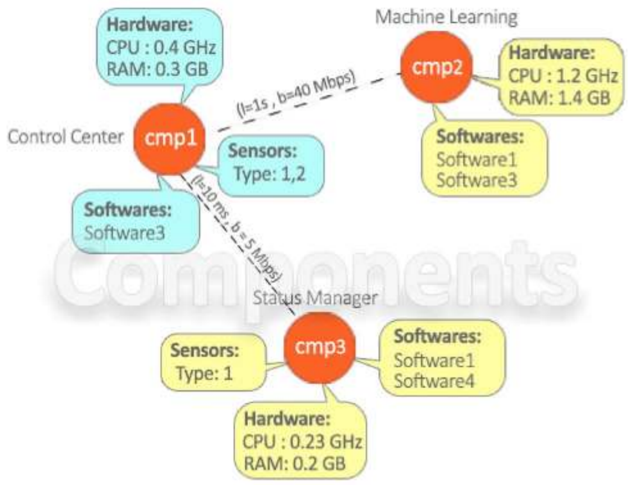
Fig. 4 Communication network in a mega node

$$D = \begin{array}{c|ccccc} & fn_1 & fn_2 & fn_3 & \cdots & fn_n \\ \hline fn_1 & d_{11} & d_{12} & d_{13} & \cdots & d_{1n} \\ fn_2 & d_{21} & d_{22} & d_{23} & \cdots & d_{2n} \\ fn_3 & d_{31} & d_{32} & d_{33} & \cdots & d_{3n} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ fn_n & d_{n1} & d_{n2} & d_{n3} & \cdots & d_{nn} \end{array} \quad (1)$$

Application model

In recent years, regarding to the nature of users requests and new expectations on internet-based services, the design of applications which manipulate users' data is constantly fluctuated based on changing requests; then, to meet user requirement, the multi-component structure approach is utilized [40]. So, application components are dependent and cooperate with each other to meet users' requirements. For instance, take a company that serves a smart health care service in a small IoT application for surveillance of aged people. This application includes three different components that Fig. 5 illustrates.


Softwares:
All sensores in Fog node members
Softwares:
All softwares in Fog node members
Hardware: CPU : 1.6 GHz RAM: 0.9 GB
Hardware: CPU : 1.2 GHz RAM: 1.4 GB
Sensors: Type: 1
Sensors: Type: 2
fn1
fn2
(L=2ms , B=1 Mbps)
Softwares: Software1 Software2
Softwares: Software3 Software4
(L=5ms , B= 20 Mbps)
(L=1ms , B=5 Mbps)
Mega Node
Sensors: Type: 1 Type: 2
fn3
Softwares: Software2 Software3
Hardware: CPU : 1.7 GHz RAM: 1 GB
Mega Node
Fig. 3 Mega node specification and its belonged fog nodes


Hardware: CPU : 0.4 GHz RAM: 0.3 GB
Machine Learning
cmp2
Hardware: CPU : 1.2 GHz RAM: 1.4 GB
(l=1s , b=40 Mbps)
Control Center
cmp1
Sensors: Type: 1,2
Softwares: Software1 Software3
Softwares: Software3
(l=10 ms , b = 5 Mbps)
Components
Status Manager
Sensors: Type: 1
cmp3
Softwares: Software1 Software4
Hardware: CPU : 0.23 GHz RAM: 0.2 GB
Fig. 5 Specification of application components

Status manager (cmp1) This component monitors aged and disabled people; it alarms the nearest medical and health-care center once it detects a disorder in physical or mental behavior.

Control center (cmp2) This component is used for interpret of integrated data and manual control of the system.

Machine Learning (cm3) This component is utilized to save data history of individuals and to estimate future wellbeing and health provided it is not latency-sensitive which can be deployed on cloud datacenter or fog infrastructure.

Figure 5 also depicts hardware resources along with software capabilities required for each component. Communication between components are drawn by special links. To manage on time status of aged people, component cmp1 must avail to needed sensors (physical state controller sensors) and an actuator which activates initial operation mechanism and announcement to medicine centers; this must be done during 10 ms. from deployed component cmp3 to the place of installed sensors and actuators. Furthermore, it is expected that the fog or cloud nodes can remotely access to existing neighbor things via APIs provided by fog middleware [41]. The problem that should be solved for application components deployment is how to place components so that the requested resources are met. Even for this simple example, different deployment plans must be evaluated for finding an optimal component mapping for this application because more than one component can be deployed on a fog node based on existing resources. Finding favorite and optimal deployment is impractical when the number of components and fog nodes are significantly increased. Then, this combinatorial problem must be solved by intricate meta-heuristic algorithms.

This paper assumes that there are R number of IoT applications each of $r \in R$ is shown by a vector $(M, cmplist)$. Each application has M number of components listed in *cmplist*. Also, each component is shown by a vector $(k, h, s, sensorlist)$ (see Fig. 5).

User applications are modeled by a graph $G = (cmplist, T)$ where $cmplist = \{cmp_1, cmp_2, \ldots, cmp_m\}$ and $T = t_{ij}$ shows the traffic matrix (TM) between components cmp_i and cmp_j. Equation (2) demonstrates traffic matrix and the Fig. 6 illustrates components communication graph.

$$TM = \begin{array}{c|ccccc} & cmp_1 & cmp_2 & cmp_3 & \cdots & cmp_m \\ \hline cmp_1 & t_{11} & t_{12} & t_{13} & \cdots & t_{1m} \\ cmp_2 & t_{21} & t_{22} & t_{23} & \cdots & t_{2m} \\ cmp_3 & t_{31} & t_{32} & t_{33} & \cdots & t_{3m} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ cmp_m & t_{m1} & t_{m2} & t_{m3} & \cdots & t_{mm} \end{array} \quad (2)$$

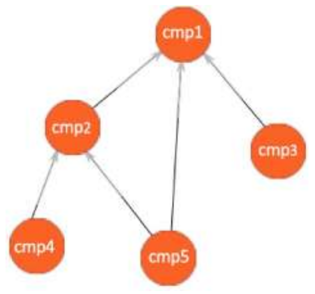


Fig. 6 Components communication graph

Reliability model

Deployment of an application's components on the minimum number of fog nodes leads to reach the goals such as reduction in power consumption and efficient utilization of cloud computing resources, but one of the confronting challenges is the acceleration of the single point of failure phenomenon in users' applications. Therefore, for the sake of meeting both optimization objective functions of cloud computing owners and to decrease the degree of applications' vulnerability in centralized distribution in fog infrastructure, the threshold parameter is considered for the number of fog nodes in distribution of applications' components. To this end, in the worst case, at most number of needed nodes for components distribution is bounded to the number of available nodes in selected Mega Node. In the other words, the best effort is bounded to Mega Node capacity.

Deployment model

To deploy components, one of the Mega Nodes regarding to claimed requirement is selected among the list of extracted Mega Nodes. In each Mega Node, if all components are placed on single node, then, $d_{ij} = 0$; otherwise $d_{ij} = 1$. Fog nodes in a Mega Node meet all of components resource requirements in terms of latency, bandwidth, and sensors. In this paper, we assume that all of sensors or software request for application components cab be shared by fog nodes associated to Mega Node. In distribution process of application components on fog nodes, the computing resources, fog nodes distance, and QoS parameter requested for application components must be taken into consideration. To reduce traffic load, the distance matrix which is used for each pair of fog nodes in network graph and also traffic pattern matrix between each pair of components must be calculated. Note that, communication links between fog nodes fn_m and fn_n have constant capacity in terms of latency and bandwidth. Therefore, traffic rate between application components is bounded to fog nodes' capacity. So, this limitation is shown in Eq. (3).

$$\sum_{cmp_i \in fn_m} \sum_{cmp_j \in fn_n} b_{ij} \times l_{ij} < B_{mn} \times L_{mn} \tag{3}$$

where b_{ij} and l_{ij} are favorite bandwidth and latency between components cmp_i and cmp_j. Also, parameters B_{mn} and L_{mn} are bandwidth and latency between fog nodes fn_m and fn_n respectively. Note that, a component can be deployed on a fog node provided this node is active. For this reason, decision variable y_{fn} is set to one when fog node fn is an active node to adopt a component. Equation (4) shows this decision variable.

$$x_{cmp,fn} \leq y_{fn}, \forall cmp \in UApp,\ fn \in F \tag{4}$$

Furthermore, the requested hardware associated to components cannot exceed the capacity of underlying fog nodes. Therefore, Eq. (5) is used to show this constraints.

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot \mathrm{hw}_{cmp} \leq \mathrm{HW}_{fn},\ \forall fn \in F \tag{5}$$

In Eq. (5), parameter HW_{fn} is relevant to fog node capacity in term of hardware and hw_{cmp} is requetsed resources relevant to components.

As assumed all software resources are available for each node in Mega Node, the software limitation is drwan in Eq. (6).

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot \mathrm{sw}_{cmp} \leq \mathrm{SW}_{\text{Mega Node}}, \forall \text{Mega Node} \in F \tag{6}$$

where the term $\mathrm{SW}_{\text{Mega Node}}$ is software capacity of Mega Node and sw_{cmp} is the requested software by application components. Also, another constraint on requetsed sensors for application components cannot exceed from Mega Node's capacity in term of number of its availabe sensors. This is elaborated in Eq. (7).

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot \mathrm{s}_{cmp} \leq S_{\text{Mega Node}}, \forall \text{Mega Node} \in F \tag{7}$$

A decision variable $x_{cmp,fn}$ is used to determine whether component cmp is placed on fog node fn or nor. Equation (8) is dedicated to this issue.

$$x_{\mathrm{cmp},fn} = \begin{cases} 1 & \text{application's } cmp \text{ is placed on fog node } fn \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Furthermore, each component is only placed on one fog node in which Eq. (9) depicts.

$$\sum_{fn \in F} x_{cmp,fn} = 1,\ \forall cmp \in UApp \tag{9}$$

Problem statement

In this paper, deployment of IoT application components is formulated to a multi-objective optimization problem. To address the issue, two objective functions and problem formulation are presented.

Overall latency

One of the most prominent objective functions of deployment problem is to minimize system overall latency which has drastic impact on average QoS degradation. So, the amount of latency owing to dependent components of an application which are placed on two different fog nodes in a Mega Node, is calculated via Eq. (10).

$$\text{Latency}_{mn} = \sum_{cmp_i \in fn_m} \sum_{cmp_j \in fn_n} L_{mn} \tag{10}$$

The latency between each pair of dependent components depends on latency between fog nodes which are hosting separate components. Note that, the amount of latency is ignored when two dependent components are placed on the same node. The overall latency of the system, owing to deployment of all applications and related components, is measured via Eq. (11).

$$UApp_{\text{latency}} = \sum_{fn_{m,n} \in \text{Mega Node}} \text{Latency}_{mn} \tag{11}$$

Power consumption

The effective subjects on fog nodes' power consumption are load of computation, communication technology, the transfer data traffic volume, distance between nodes and etc. To calculate the power consumption of a fog node, power consumption owing to both application's components processing and data transfer between nodes should be taken into account. Literature review proves that the power consumption of a processing node has linearly relation to its resource utilization [42]. So, the average normalized resource utilization associated to each fog node is measured via Eqs. (12).

$$U_{fn_i}^{\text{Res}} = \frac{W_1 \cdot \sum_j^{fn_i} \frac{R_{\text{Com}_j}^{\text{CPU}}}{R_{fn_i}^{\text{CPU}}} + W_2 \cdot \sum_j^{fn_i} \frac{R_{\text{Com}_j}^{\text{RAM}}}{R_{fn_i}^{\text{RAM}}}}{2} \tag{12}$$

where parameters W_1 and W_2 are two coefficients that show the importance of them in fog node's power consumption. Note that, their values are $0 \leq W_1 \leq 1$, $0 \leq W_2 \leq 1$, and $W_1 + W_2 = 1$. Since the power consumption of processing units outwieghts versus the main memory, the processor utilization is taken for power consumption; consequently, the parameters are set as $W_1 = 0.9$ and $W_2 = 0.1$ [42]. The Eq. (13) measures the power consumption owing to utilized resources relevant to each node that hosts different components.

$$P_{fn}^{\mathrm{Res}} = y_{fn} \times \left(P_{\max} - P_{\min}\right) \times U_{fn}^{\mathrm{Res}} + P_{\min} \tag{13}$$

where parameters $P_{\min}$ and $P_{\max}$ are used to indicate the minimum and maximum power consumption of each processing node in the minimum and maximum utilization conditions respectively. In addition to, decision binary varibale y_{fn} is used to show whether the processing node is active or not. Moreover, the power consumption owing to data transfer via communication links are obtained by Eq. (14).

$$P_{fn}^{\mathrm{Tr}} = \sum_{fn_i \neq fn_j} t_{\mathrm{Com}_i, \mathrm{Com}_j} \times P_{\mathrm{Tr}} \tag{14}$$

The parameter P_{Tr} is of prower consumption unit for traffic trasfer. Note that, this power is taken in case the components are placed on different computing nodes. Cosequently, the total power consumption is obtained via Eq. (15). The first section is for resource utilization and the second section is for traffic transfering power consumption.

$$P_{fn} = P_{fn}^{\mathrm{Res}} + P_{fn}^{\mathrm{Tr}} \tag{15}$$

Problem formulation

The deployment of IoT application components by distributing over fog nodes is formulated to a multi-objective optimization problem. After definition of objective functions, this formulation is brought in Eqs. (16)–(24).

$$\min \mathrm{TPC} = \mathrm{Min} \sum_{fn \in F} P_{fn} \tag{16}$$

$$\min UApp_{\mathrm{latency}} = \mathrm{Min} \sum_{fn_{m,n} \in \mathrm{Mega\ Node}} \mathrm{Latency}_{mn} \tag{17}$$

Subject to:

$$\sum_{cmp_i \in fn_m} \sum_{cmp_j \in fn_n} b_{ij} \times l_{ij} < B_{mn} \times L_{mn} \tag{18}$$

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot \mathrm{hw}_{cmp} \leq \mathrm{HW}_{fn}, \quad \forall fn \in F \tag{19}$$

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot \mathrm{sw}_{cmp} \leq \mathrm{SW}_{\mathrm{Mega\ Node}}, \quad \forall \mathrm{Mega\ Node} \in F \tag{20}$$

$$\sum_{cmp \in UApp} x_{cmp,fn} \cdot s_{cmp} \leq S_{\mathrm{Mega\ Node}}, \quad \forall \mathrm{Mega\ Node} \in F \tag{21}$$

$$x_{cmp,fn} \leq y_{fn}, \forall cmp \in UApp,\ fn \in F \tag{22}$$

$$\sum_{fn \in F} x_{cmp,fn} = 1, \quad \forall cmp \in UApp \tag{23}$$

$$x_{cmp,fn} \in \{0,1\}, \quad y_{fn} \in \{0,1\} \tag{24}$$

In the aforementioned problem formulation, the Eqs. (16, 17) are objective functions to be minimized at the same time the constraints drawn in Eqs. (18–24) must be met. To solve this combinatorial optimization problem, an intricate multi-objective optimization algorithm is presented.

Proposed MOCSA algorithm for component deployment problem

As the stated problem is a multi-objective optimization problem, we extend a multi-objective optimization algorithm in regards to two equal important objectives. A multi-objective optimization algorithm differs from a single objective optimization algorithm because in multi-objective optimization algorithm a trade-off between objectives must be done. To this end, the dominance concept is utilized [24, 31, 42]. The multi-objective optimization algorithm must be conducted in search space to find non-dominated solutions known as Pareto front [31]. Regarding to the discrete nature of the search space associated to stated problem, the cuckoo search algorithm (CSA) is adopted for the sake of its performance and adaptation with discrete search space. The CSA was firstly introduced in literature by Yang and Deb [43] at year 2009. It had successful outcome in different optimization domains such as in [44–46]. To solve deployment problem, a multi-objective version of CSA known (MOCSA) is extended which inherits strength of both CSA and NSGA-II algorithms [29].

The CSA mimics its behavior from cuckoo birds. This kind of bird has an aggressive attitude in which it even lays eggs in the other birds' nests along with throwing away their eggs. In CSA, every egg in a nest is a candidate solution. When a cuckoo lays one egg in a nest in fact it produces a new solution. In this regards, a single objective CSA utilizes three rules:

At first, each cuckoo lays one egg in a randomly selected nest.

Secondly, better nests holding eggs (solutions) with better quality remain for next generation.

Thirdly, number of existing nests are fix; and a host nest, a cuckoo can detect strange egg with the probability $p_a \in [0,1]$; in this case, the host bird can either smash the egg or leave the nest for constructing completely new nest in the new place.

To construct MOCSA with *k* objective functions, three mentioned rules of canonical CSA needs to be customized in regards to objective functions. New rules are:

In each iteration, each cuckoo lays *k* eggs in a randomly selected nest in which the *i*-th egg is representative of the *i*-th objective function. In regard to similarity and discrepancy between eggs, each nest is left with probability p_a and the new nest is constructed with *k* new eggs. In addition to, some operations can be defined to permute search space efficiently. Mathematically, the first rule can utilize Random Walk or Levy flight approaches (c.f. Eqs. (25, 26) to uniformly permute (traverse) search space for generating new solutions. The second rule is an elitism based approach so that better solutions remain in next generation. In this line, selection of better solutions generates the suitable convergence of algorithm. The third rule can be taken as a mutation approach so the worse solutions are probabilistically omitted and the new solutions are generated in regards to similarities the solutions with other solutions. This mutation approach is done by vector operator via combined Levy flight and quality differential of solutions. Figure 7 draws block diagram of proposed algorithm.

This algorithm receives problem specifications and execution's settings as input such as information about requested resources for applications, number of components and their communication details, number of fog nodes and associated network information, number of initial solutions, and number of maximum iterations. Then, it returns a set of non-dominated solutions as deployment plans.

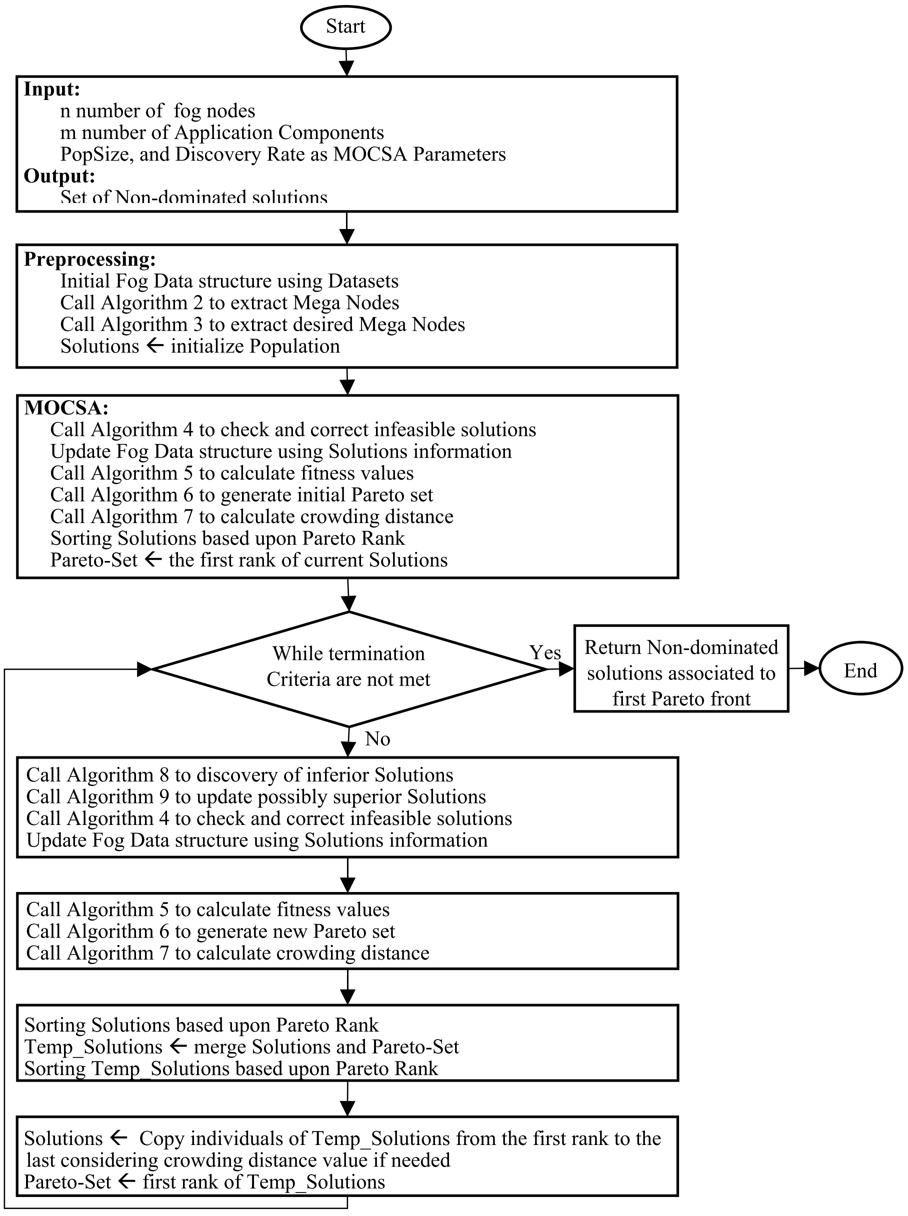

Start
Input:
n number of fog nodes
m number of Application Components
PopSize, and Discovery Rate as MOCSA Parameters
Output:
Set of Non-dominated solutions
Preprocessing:
Initial Fog Data structure using Datasets
Call Algorithm 2 to extract Mega Nodes
Call Algorithm 3 to extract desired Mega Nodes
Solutions ← initialize Population
MOCSA:
Call Algorithm 4 to check and correct infeasible solutions
Update Fog Data structure using Solutions information
Call Algorithm 5 to calculate fitness values
Call Algorithm 6 to generate initial Pareto set
Call Algorithm 7 to calculate crowding distance
Sorting Solutions based upon Pareto Rank
Pareto-Set ← the first rank of current Solutions
While termination Criteria are not met
Yes
Return Non-dominated solutions associated to first Pareto front
End
No
Call Algorithm 8 to discovery of inferior Solutions
Call Algorithm 9 to update possibly superior Solutions
Call Algorithm 4 to check and correct infeasible solutions
Update Fog Data structure using Solutions information
Call Algorithm 5 to calculate fitness values
Call Algorithm 6 to generate new Pareto set
Call Algorithm 7 to calculate crowding distance
Sorting Solutions based upon Pareto Rank
Temp_Solutions ← merge Solutions and Pareto-Set
Sorting Temp_Solutions based upon Pareto Rank
Solutions ← Copy individuals of Temp_Solutions from the first rank to the last considering crowding distance value if needed
Pareto-Set ← first rank of Temp_Solutions

Fig. 7 Block diagram of proposed algorithm

Problem encoding

One of the most important issues in CSA algorithm is the concept of *nest* which is a candidate solution. Encoding on nest has intensive impact on algorithm performance. There are miscellaneous encoding viewpoint for different problems. The art is to find the most appropriate one. Each nest is a possible solution for IoT application components deployment on fog nodes. A nest contains |*M*| number of eggs each of which is representative of a component. The number assigned to each egg is drawn from [1...|*N*|] interval which indicates the fog node number hosting that component. Figure 8 depicts encoding of an example for deployment of 10 components on 3 fog nodes.

Proposed MOCSA

In single objective optimization cuckoo search algorithm, the population is partitioned into two superior and inferior nests with predetermined probability based on their fitness value. In the other words, the determined parameter P_a is the fraction of population which are placed in the inferior nests whereas the rest are placed in the superior nests after sorting population based on their fitness values. In each generation, iteration, the algorithm works in two stages. At first stage, for each individual of inferior nests, each new position is generated by Levy Flight distribution; then, the old individual is directly constituted by the new generated one. At the second stage, for each individual in superior nests, each new position is generated by Levy Flight distribution; if the new generated individual is better than the old version in term of fitness value, the old version is substituted by the new generated one. Since the multi-objective optimization algorithm differs from a single objective, we have customized CSA to MOCSA algorithm to gain non-dominated solutions. The general behavior is the same, but the differences are in the ranking and partitioning processes. For ranking, we utilize non-dominated and crowding distance concepts. Once it is needed to partition population into two parts, we utilize non-dominated sorting strategy based on Algorithm 6; then from the worst ranking to best ranking, the solutions are directly copied to inferior nests; in this direction according to the probability P_a, if the solutions associated to the *k*-th ranking value overflows the inferior nests, the crowding distance values are considered. In the other words, the rest individuals with the worst crowding distance values are selected to be copied to fulfill the rest of inferior nests. Afterwards, the rest populations are copied to superior nests. It is worth mentioning that, in the second stage when the new individual is generated for each individual in the superior nests, if the new individual dominates the old version in regards to two objective functions, the old individual is substituted by the new generated solution.

The proposed MOCSA algorithm is elaborated in Algorithm 1 which deploys IoT application components efficiently on fog nodes in regards to objective functions. As mentioned earlier, Algorithm 1 receives the problem specifications as input and returns non-dominated solutions in regards to two prominent objective functions. It is iterated until the termination criterion is met. Here, the condition of termination is to execute MaxIteration times. Before the Algorithm 1 starts in its main loop which is between lines 14 through 27, it performs preprocessing stages. Algorithms 2 and 3 are dedicated to extract Mega Nodes and desired Mega Nodes which are explained in preprocessing stages. New solutions are generated in line 5 from extracted desired Mega Nodes. In line 7, Algorithm 4 is called to check and correct infeasible solutions. Then, the associated Data Structure is updated in line 8. Algorithm 5 is called to assign two fitness values to each individual based on Eqs. (16, 17) since it is a multi-objective problem. The main loop of proposed MOCSA starts in line 14 and ends in line 27. In the proposed algorithm in each generation the population is partitioned into two inferior and superior nests. As explained earlier, the main loop runs two stages. At first, the worst solutions in inferior nests are updated and at the second stage the better solutions in superior nests are updated provided the new generated solutions dominate the old version otherwise no update is done. In line 9, all fitness values associated to all solutions are assigned by calling Algorithm 5. In lines 10–11 the Algorithms 6–7 are called to make Pareto fronts and crowding distance for current solutions. In the main loop, P_a percent of solutions associated to the worst ranking is copied in the inferior nests by utilizing Pareto front and crowding distance values and the rest is copied to superior nests. Before algorithm plummets into the main loop, in line 12 the current solutions are sorted based on ranking concepts. Then, the first ranking solutions are kept in Pareto-Set repository in line 13. As mentioned

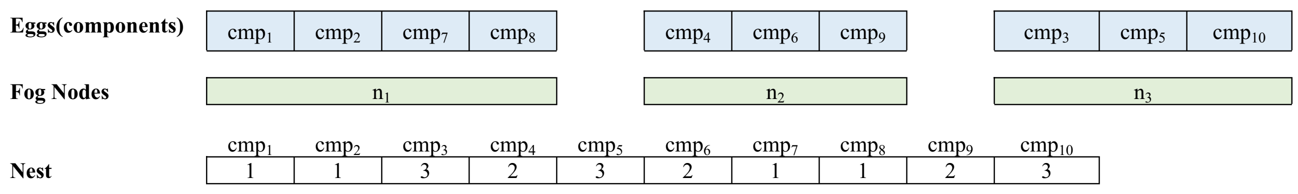


Fig. 8 An example for deployment encoding and associated Nest

earlier, in line 15, the Algorithm 8 is called to update solutions in inferior nests; afterwards, the second stage is started where the solutions pertained to superior nests are to be updated. If the new changes dominate the old version, the old version is substituted by the new generated solution in superior nests. This change is done by calling Algorithm 9 in line 16. In line 17, Algorithm 4 is called to check and correct infeasible solutions. Then, the associated Data Structure is updated in line 18. In line 19, the fitness values of all updated solutions are calculated by calling Algorithm 5; then, the non-dominated solutions and crowding distance are calculated by calling Algorithms 6 and 7 respectively. The current solution is then sorted by their rank values. The temporary solutions are made by merging the current solutions and the last Pareto-Set values. The temporary solutions are sorted based on rank values. From the first ranking to the last are copied to the current solutions variable by considering crowding distance values if needed. In addition, the first rank is directly copied in Pareto-Set variable. After the last iteration is done. The final values in Pareto-Set containing the first ranking solutions of the last operation is return as final non-dominated solutions.

Algorithm 1. The Proposed MOCSA Algorithm for Component Deployment Problem

```
        Input Parameters:
        ComponentsDataset, Fog Network Dataset, Fog Nodes and Components Latency and Bandwidth Matrix :
        Parameters of Applications And Fog Network;
        N : Number of Fog Nodes ;
        M: Number of Components;
        (*Parameters of MOCSA *)
        PopSize: As number of Solutions in Population;
        $P_a$: Discovery Rate;
        MaxIteration : Maximum number of Iterations;
        Output:
        First Pareto front in final Solutions in regards to objective functions
 1:     (* Preprocessing *)
 2:     Fog_DS = Initial Fog Data Structure using Datasets;
 3:     Mega Nodes ← Call Algorithm 2 to Exploit Mega Nodes;
 4:     Candidate_Mega_Nodes ← Call Algorithm 3 to find desired Mega Nodes;
 5:     Solutions [1..PopSize] ← Initial Random Population (PopSize,componentsDataset,
        Candidate_Mega_Nodes);
 6:     (* End of Preprocessing *)
 7:     Solutions[1..PopSize] ← Call Algorithm 4 to Check And Correct Infeasible Solutions
 8:     Update Fog_DS using Solutions[1..PopSize] Information;
 9:     Call Algorithm 5 for Calculate Solutions cost of individuals in Solutions [1..PopSize] based on Eqs (16-17);
10:     Call Algorithm 6 for None Dominated Sorting Solutions of Solutions [1..PopSize];
11:     Call Algorithm 7 for calculate Crowding Distance of Solutions in Solutions [1..PopSize];
12:     Sorting Solutions [1..PopSize] based upon Pareto Rank;
13:     Pareto-Set ← The first rank of current Solutions
14:     While termination criteria is not met do
15:         Call Algorithm 8 for discovery of  inferior Solutions; direct updating in Solutions with probability $P_a$;
16:         Call Algorithm 9 to update possibly superior nests with probability (1-$P_a$)
17:         Call Algorithm 4 to Check And Correct Infeasible Solutions
18:         Update Fog_DS using Solutions information;
19:         Call Algorithm 5 to Calculate Solutions' cost in Solutions based on Eqs (16-17);
20:         Call Algorithm 6 for None Dominated Sorting Solutions in Solutions;
21:         Call Algorithm 7 to calculate Crowding Distance of Solutions in Solutions;
22:         Sorting Solutions in Solutions based upon Pareto Rank;
23:         Temp_Solutions = merge Solutions and Pareto-Set;
24:         Sorting Temp_Solutions based upon Pareto Rank;
25:         Solutions= Copy individuals of Temp_Solutions from the first rank to the last considering crowding
        distance value if needed
26:         Pareto-Set= first rank of Temp_Solutions
27:     End While
28:     Return first rank as Pareto-Set; //a near optimal solution
```

Preprocessing

In this stage, the preprocessing is performed to extract desired Mega Nodes. Algorithm 2 selects different Mega Nodes from input fog network. The Mega Node characteristics was clarified earlier which is abstracted to clique in graph theory. It returns all cliques with K-nodes. Mega Node extraction brings some merits; firstly the search space reduction for finding optimal deployment plan; secondly, providing common sensors and software associated to Mega Node for requested components. In Algorithm 2, in the while-loop between lines 3 through 11, firstly all nodes which are connected are extracted; each pair of connected nodes is placed in a row in Mega_Nodes array. In lines 13 through 20, in the for-loop, each fog node *i* is compared with each row in Mega_Nodes array that does not containing node *i*. If node *i* is connected with all nodes in that row, then the node *i* is added to that row. In each iteration, the repeated row is omitted. The main loop is iterated until the last array of Mega_Nodes which contains the set of Mega Nodes is delivered.

Algorithm 2. Mega Nodes Exploitation

```
Input Parameters:
Fog Network Communication Matrix(i×j) As Fog_Net, where i and j=1,..,N
and n is number of fog nodes
K : number of desired clique nodes;
Output:
Mega_Nodes is vertical array of fullmesh subnetworks
Initialization:
define Mega_Nodes = {};
define Fog_Net = upper triangular(Fog_Net);
1:  While the termination criteria not met do
2:    If Mega_Nodes is empty then
3:      R = 0;
4:      For Each Fognode_i in Fog_Net do
5:        For Each Fognode_j= Fognode_{i+1} in Fog_Net do
6:          If Fognode_j connected to Fognode_i in Fog_Net then
7:            R = R+1;
8:            Mega_Nodes[row R]=[Fognode_i , Fognode_j];
9:          End If
10:       End For Each
11:     End For Each
12:   Else
13:     For Each Fognode_i do
14:       For Each R = all row of Mega_Nodes do
15:         Current_row = Mega_Nodes[row R];
16:         If Fognode_i Not in Current_row and Fognode_i connected to all
                    node in Current_row in Fog_Net then
17:           Mega_Nodes[row R]=[Mega_Nodes[row R], Fognode_i];
18:         End If
19:       End For Each
20:     End For Each
21:   End If
22:   Remove duplicate rows in Mega_Nodes;
23: End While
24: Return Mega_Nodes
```

After Mega Nodes extraction, some Mega Nodes are selected by Algorithm 3 in regards to meeting of constraints in Eqs. (18–21) in the stated problem. In this algorithm, if latency and bandwidth are provisioned by the Mega Node in the current row, then, Latency_BW_status variable is set to true. In addition to, if hardware, software, and sensors can be provided by the current Mega Node, the amount of HW_Status, SW_Status, and S_Status are set to true. If a current Mega Node can fulfill all required resources, it is added to selected Mega Node list.

The termination criterion of Algorithm 2 is the number of desired clique size (K). In the other words, the main loop is iterated K times. Since the effective statements of Algorithm 2 are in the while-loop, its time complexity is O ($K \bullet N^2$) where $K < N$. Also, Algorithm 3's time complexity is O ($N + M$) because the main work is done in the for-loop between lines 1 through 9.

Algorithm 3. Find Candidate Mega Nodes

```
    Input Parameters:
      define fn=(1,..,N) is fog node list;
      define Appcmp=(1,..,M) is Application components list;
      define FN_BW=array(N×N)is bandwidth between fog nodes;
      define FN_Latency=array(N×N)is Latency between fog nodes;
      define Appcmp_BW=array(M×M)is bandwidth between components;
      define Appcmp_Latency=array(M×M)is Latency between components;
      define AppcmpDataset is Applicatin components resource requirements;
      define FNDataset is Fog nodes resources;
      define Mega_Nodes is Mega Nodes list(Algorithm2 output);
    Output:
    Candidate_Mega_Nodes
 1: For each   Mega_Nodes as row_i do
 2:   Latency_BW_status =
        Check_Latency_BW(Appcmp_Latency,Appcmp_BW,row_i,
        FNDataset,FN_BW,FN_Latency); //constrain in Eq. (18)
 3:   HW_status = HW_chk(AppcmpDataset, FNDataset,row_i); //constrain in Eq. (19)
 4:   SW_status = SW_chk(AppcmpDataset, FNDataset,row_i); //constrain in Eq. (20)
 5:   S_status  = S_chk(AppcmpDataset, FNDataset,row_i);  //constrain in Eq. (21)
 6:   If Latency_BW_status & HW_status & SW_status & S_status are true then
 7:     Candidate_Mega_Nodes = [Candidate_Mega_Nodes;row_i];
 8:   End if
 9: End For
10: Return Candidate_Mega_Nodes;
```

Initialization step

Similar to other meta-heuristic algorithms, the CSA starts with initialization phase in which line 5 of Algorithm 1 performs this. It randomly generates individuals from search space. To reduce MOCSA's time complexity, the value domain of eggs are confined to the proposed encoding approach. Since some solutions may violate problem constraints during the individual productions, the Check&Correct algorithm is designed which Algorithm 4 shows. Indeed, Algorithm 4 is presented to exploit maximum benefit from produced population for utilizing them in optimal solutions.

Algorithm 4. Check&Correct

```
Input Parameters:
define Fog_Data_Structure is Fog nodes status in each solution;
define Mega_Node is Candidate Nodes taken from Algorithm 3;
define Appcmp=(1,..,M) is Application component list;
define AppcmpDataset is Application component Specification;
define Solutions is all solution in a population;
Output:
 Feasible Solutions and Fog_Data_Structure
1:  For Each Solutions As solution do
2:    For Each Fog node in solution as sourceFN do
3:      If sourceFN Resource overload is true then
4:        violation = true;
5:        While violation is true do
6:          find Appcmp with minimum dependency to other component on
7:          sourceFN and migrate it to other fog node in
8:          Mega_Node with enough resource. Candidate Appcmp has
9:          maximum dependency to components on destination Fog node;
10:         Update solution;
11:         Update Fog_Data_Structure;
12:         If sourceFN resource overload is false then
13:           violation = false;
14:         End If
15:       End While
16:     End If
17:   End For Each
18: End For Each
19: Return Solutions and Fog_Data_Structure;
```

Time complexity of Algorithm 4 is O (N•PopSize) because two nested for-loop are the most effective statements.

Fitness function

Generally, one of the most important things in evolutionary computation is to evaluate solutions. This is done by fitness functions in regards to problem's objective functions. In this paper, fitness function is adjusted based on total power consumption and overall latency which are in Eqs. (16) and (17). The proposed fitness function is depicted in Algorithm 5.

Algorithm 5. Fitness Function

```
Input Parameters:
define Solutions is a Population;
Output:
Updated each solution in Solutions with solution cost
1: For Each solution in Solutions as solution [i] do
2:   Energy = calculate computing and network resource energy consumption
       of all fog node in solution_i;
3:   allLatency = calculate overall Latency of fog nodes in solution_i;
4:   Solution[i]'s Cost in Solutions = [Energy,allLatency];
5: End For Each
6: Return Updated Solutions;
```

It is clear-cut that its time complexity of Algorithm 5 is *O* (PopSize).

Non-dominated sorting

In multi-objective optimization algorithms the goal is to omit unfavorable solutions and to select superlative solutions with special strategy in such a way that solutions in lower levels are omitted at the same time the better solutions are remained until the final solution is obtained step by step. In the proposed MOCSA, we apply non-dominated sorting algorithm to find Pareto front. This algorithm investigates the state of current solutions in term of dominance concept regarding to objective functions. In fact, it classifies solutions in different Pareto levels so that all solutions in the same ranking level cannot dominate each other whereas the solutions in upper levels dominate solutions in downer level. The favorable non-dominated solutions belong to the first ranking level. Algorithm 6 finds non-dominated solutions.

Algorithm 6. None-dominated Sorting Strategy

```
        Input Parameters:
        define Solutions is a Population;
        define Dominate_Matrix=array(n×n) default cells value is 0, n is
          number of solutions(a temp Matrix with 0,1 and infinite values,
          used for Subgrouping solutions in population);
        define k=1 is first subgroup;
        define F{k}=array() is array of solution index in each subgroup k;
        Output:
        Updated Solutions with subgrouped solutions;
1:      For Each solution in Solutions as $solution_i$ do
2:          If $solution_i$ is $solution_n$ then
3:              F{k}=[F{k},i];
4:              $solution_i$ Rank in Solutions = k; break;
5:          End If
6:          For Each $solution_j$ = $solution_{i+1}$ do
7:              If $solution_i$ Cost Dominates $solution_j$ Cost then
8:          Dominate_Matrix(i,j) = 1;
9:              Else If $solution_j$ Cost Dominates $solution_i$ Cost then
10:                 Dominate_Matrix(j,i) = 1;
11:             End If
12:         End For Each
13:         //First Subgroup or pareto Front 1
14:         If sum all values in column i of Dominate_Matrix is equal to 0 then
15:             F{k}=[F{k},i];
16:             $solution_i$ Rank in Solutions = k;
17:         End If
18:     End For Each
19:     s= all solution index in subgroup F{k};
20:     all cell values in Dominate_Matrix rows s = 0;
21:     all cell values in Dominate_Matrix columns s = infinite;
22:     While true do
23:       k=k+1;
24:       s = all zero columns index in Dominate_Matrix;
25:         If s is empty then
26:             Break;
27:         End If
28:         F{k}=s;
29:       all cell values in Dominate_Matrix rows s = 0;
30:       all cell values in Dominate_Matrix columns s = infinite;
31:         For Each solution index in s as i do
32:             $solution_i$ Rank in Solutions = k;
33:         End For Each
34:     End While
35:     Return updated Solutions;
```

Since the effective statements of Algorithm 6 are in nested For-loop, its time complexity is O (PopSize^2).

Crowding distance

Finding efficient solutions strongly depends on the strategy that the algorithm takes. The best strategy must be conducted in such a way that explore search space efficiently. More distribution in search space, more contingent to gain better and logical solutions. Diverse solutions in larger district are preferable against denser solutions in smaller region the reason why we apply crowding distance algorithm to investigate solutions in term of density in a district search area. This way avoids to integrate solutions locally. Algorithm 7 elaborates crowding distance procedure.

Algorithm 7. Crowding Distance

```
      Input Parameters:
      define Solutions is a Population;
      Output:
      Updated each solution in Solutions with crowding distance value
1:    For Each solution in Population as solution_i do
2:       Subgroup = Find all solution in Solutions with Rank equal to
            solution_i Rank;
3:       Crowding distance solution_i = 0;
4:       For Each Objective value in solution Cost As Obj_j do
5:          Sort  Subgroup based upon Obj_j in ascending order;
6:          If  solution_i is first or last element in Subgroup then
7:             Crowding distance solution_i = infinite;
8:          Else
9:             A= Absolute value(solution_{i+1} Obj_j-solution_{i-1} Obj_j)/
                  (max Subgroup Obj_j - Subgroups min Obj_j);
10:            Crowding distance solution_i = Crowding distance solution_i+ A;
11:         End If
12:      End For Each
13:   End For Each
14:   Return Updated Solutions;
```

It is clear that the time complexity of Algorithm 7 is O (PopSize).

Inferior nests update

In this process, the fraction of worse solutions by probability P_a are detected and amended. This operation is similar to mutation in GA [43–46]. Since our algorithm works in multi-objective domain, the worst solutions are selected from the worst ranking frontier; also, the crowding distance is called where needed. The modification of worse solutions are done by walking around approach. Algorithm 8 is dedicated to do so. In line 4, the invalid solutions are amended. Then, updated solutions as new solutions are returned.

Algorithm 8. Discovery Inferior Solutions and Update

```
      Input Parameters:
      define Solutions_new is new Population of Solutions;
      define P_a is Discovery Rate
      Output:
      updated Solutions_new;
1:    Worse_solutons = a fraction of Solutions based upon P_a Probability;
2:    For Each Worse_solutons in Solutions_new as ws do
3:      ws_new= modify ws using random walk();
4:      Boundary_update(ws_new);
5:      Replace ws in nest_new with ws_new
6:    End For Each
7:    Return updated Solutions_new;
```

Table 3 Different scenarios of simulation

Scenarios #	1	2	3	4	5	6	7	8
Fog nodes #	10	15	20	25	40	55	70	100
Appcmp #	20	25	30	40	60	75	100	150

Table 4 Fog nodes resources

FN#	1	2	3	4	5
CPU(GHz)	1.02	1.15	1.38	1.46	1.06
RAM(GB)	1.3	1.6	1.2	1.4	1.3
CPU_Thr	0.98	0.93	0.96	0.94	0.92
RAM_Thr	1.00	0.99	0.91	0.92	0.98
P_min	94	82	99	81	91
P_max	133	132	133	147	142
Sensor	1.2	1.2	1.2	2	0
Software	0	1.2	2	0	1.2
P_tr	0.2	0.2	0.2	0.1	0.1

Table 5 Bandwidth between fog nodes

FN#	1	2	3	4	5
1	1	0.98	0.80	0.89	0.97
2	0.84	1	0.82	0.94	0.92
3	0.93	0.94	1	0.97	0
4	0.88	0.92	0.91	1	1.00
5	0.92	0.99	0	0.90	1

Table 6 Latency between fog nodes

FN#	1	2	3	4	5
1	0	0.17	0.19	0.10	0.10
2	0.12	0	0.10	0.11	0.18
3	0.18	0.14	0	0.12	1
4	0.16	0.19	0.14	0	0.14
5	0.11	0.14	1	0.16	0

Table 7 Resource requested for application components

Appcmp#	1	2	3	4	5
CPU	0.15	0.19	0.24	0.26	0.29
RAM	0.2	0.2	0.1	0.2	0.1
Sensors	1	0	2	0	2
Software	0	1	0	1.2	1

Time complexity of Algorithm 8 is θ (P_a•PopSize); therefore is O (PopSize) because of its only one for-loop and the fact that $P_a < 1$.

Table 8 Bandwidth requested for application components

Appcmp#	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0.33	0	0.32
3	0	0	1	0.31	0.20
4	0	0	0	1	0.39
5	0	0	0	0	1

Table 9 Latency requested for application components

Appcmp#	1	2	3	4	5
1	0	1	1	1	1
2	0	0	0.20	1	0.28
3	0	0	0	0.24	0.21
4	0	0	0	0	0.20
5	0	0	0	0	0

Superior nests possibly updates

To produce next generation solutions, the elitism mechanism is applied so the better solutions are transferred to the next generation. The favorable trait of each meta-heuristic algorithm is how to make balance between exploration and exploitation in search space, but some of them fail to make a balance; for instance, PSO suffers from earlier convergence [24, 25] or simulated annealing (SA) suffers from not to be strong in exploration phase [26–28]. Fortunately, our proposed MOCSA makes a good adjustment between exploitation and exploration. Once it exchanges a random solution with the best so far if it is better, it tries in exploitation phase such as in Algorithm 9. For exploration, it utilizes uniform distribution in search space to explore search space globally such as in Algorithm 8. A prominent part of CSA is to utilize Levy Flight for both local and global searching; it uses random walk which is characterized by probabilistically instantaneous jumping in search space [47]. To do so, by utilizing Levy Flight approach [44], the new generation individuals are produced in line 2; if each new generated individual dominates the previous generation individual then the old generation is substituted by new one. It is well depicted in lines 4–6 of Algorithm 9. As the obtained values in new solutions are continuous, these values are amended commensurate with the problem conditions in line 3 of Algorithm 9.

Table 10 Setting parameters of different algorithms in simulation

	Specific parameters				Number of objective	Population size	Max iterations
MOCSA	Pa:	0.25			2	100	100
MOGWO	Archive size	100	Alpha	0.1			
	nGrid	10	Beta	4			
			Gamma	2			
MOPSO	nGrid	20	*W*	0.4			
	maxvel	5	C1	2			
	u_mut	0.5	C2	2			
MOBA	AL	0.9	minf	0			
	r	0.9	maxf	1			
	Alpha1	0.9					
	Gamma1	0.9					
NSGA-II	pCrossover	0.7					
	pMutation	0.4					

Pa, discovery rate of alien eggs/solutions; C2, Swarm confidence factor; Alpha, grid inflation parameter; AL, loudness parameter update in Iterations; Beta, lleader selection pressure parameter; *r*, pulse emission rate parameter update in Iterations; Gamma, extra repository member selection pressure; alpha1, constant used to update AL; archive size, repository size; gamma1, constant used to update *r*; nGrid, number of grids per each dimension; minf, frequency used to velocity update; maxvel, maxmium velocity in percentage(search space percentage); maxf, frequency used to velocity update; u_mut, uniform mutation percentage; pCrossover, crossover percentage; *W*, inertia weight; pMutation, mutation percentage; C1, Individual confidence factor

Then, the new obtained solution is added to the list of next generation solutions.

Algorithm 9. Superior Nests Possibly Updates

```
     Input Parameters:
     define Solutions from topmost current Population of Solutions with (1-Pa) probability;
     Output:
     Solutions_new is Next generation of population;
1:    For Each Solution [i] in Solutions do
2:    S_new= random walk or levy flight from solution [i] for each of its meme by  Eq. (25)
     and Eq. (26);
3:    Boundary_update(S_new);
4:    If S_new dominates solution [i] Then
5:       Solutions_new = [Solutions_new , S_new];
6:    End-IF
7:   End For Each
8:   Return Solutions_new;
```

In line 2, Algorithm 9 produces a number y as a random nest number from Levy distribution based on Eq. (25).

$$y = (1 - u)^{-\frac{1}{\alpha}} \tag{25}$$

where the variable u is a uniform variable in [0...1] interval and the parameter α is obtained by Eq. (26).

$$\alpha = G^{1/6} \tag{26}$$

where the parameter G is the generation number [44]. After that, line 3 updates the obtained solutions according to boundary of problem domain. Time complexity of Algorithm 9 is O(PopSize) because of its only for-loop.

Simulation and evaluation

To assess the effectiveness of proposed MOCSA algorithm in solving multi-objective optimization problem of components deployment on fog nodes, experiments are defined, executed and evaluated. To reach concrete results different scenarios are conducted. Also, the performance of proposed MOCSA is compared with four prominent and successful multi-objective optimization algorithms, namely, MOGWO [33], MOPSO [30], MOBA [32] and NSGA-II [29]. In this comparison, the evaluation metrics are total power consumption and overall latency which are relevant to stated problem's objective functions. As mentioned earlier, the total power consumption is sum of processing power consumption owing to resource utilization and power consumption owing to data transfer between fog nodes via communication links. In addition to, the overall latency is sum of latency obtained from communication components which are placed on different fog nodes. Furthermore, the Pareto front relevant to each algorithm are compared. Also, final deployment that MOCSA gives is dawn.

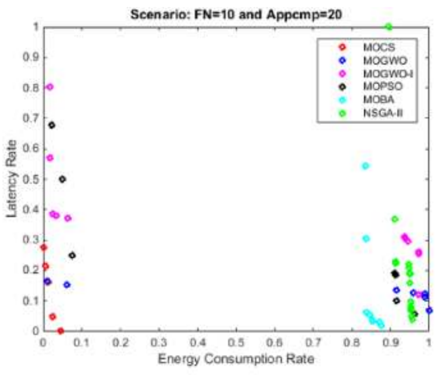
Note that, Mirjalili et al. [33] in year 2016 added two new modules to canonical GWO algorithm to make multi-objective version of GWO algorithm. The first is Archive module that is used to save non-dominated solutions so far and the second is for leader wolf to select alpha, beta, and delta wolves; this is used for updating position of omega wolves in the course of optimization. The aforementioned features are utilized to keep current solutions and gradually update them toward final Pareto front. In this line, Coello et al. [30] proposed MOPSO which utilizes history record for saving the best solution experienced by an particle and save it for non-dominated solutions of previous rounds. This mechanism works similar to elitism of evolutionary computation. It also use a global repository so that each particle keeps experience during its flight. This repository is used for leader selection to guide other particles in search space. Accordingly, each particle can select different leaders. The MOPSO works based on generating different hypercube which divide search space in several sections [30]. One of the most successful meta-heuristic algorithm is bat optimization algorithm (BOA) which was firstly introduced by Yang [48] in 2010. Afterwards, in 2011, he proposed multi-objective bat optimization algorithm by incorporating dominance concepts to solve multi-objective optimization problems [32]. One of the famous and applicable multi-objective optimizer which is based on genetic algorithm is NSGA-II that was firstly introduced by Deb et al. [29] in 2002. NSGA-II generates population then calls fast non-dominated sorting algorithm to place solutions in different ranks. All solutions in the same rank cannot dominate each other, but they can dominate the solutions placed in lower ranks. By utilizing canonical crossover and mutation, the new generated solutions may dominate the solutions associated to previous solutions. In this case, the dominated solutions are omitted. This procedure is repeated until the termination criteria is met. Finally, the non-dominated solution of the first rank is returned.

Experimental settings

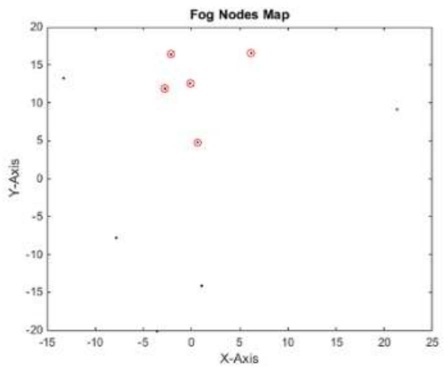
To evaluate the proposed approach, different scenarios are conducted in which the number of requested components and fog nodes increase gradually. Table 3 elaborates scenarios in details. Note that, the scenarios (5–8) are defined for scalability testing of comparative algorithms where the size of inputs are significantly increased. All experiments are executed on a dual core Intel Corei3 380 M platform with 2.53 GHZ clock rate, four logical processors, and 8 GB as main memory.

Since fog computing is ad-hoc and there is not abundant datasets in literature, we produce dataset by uniform distribution fashion such as in Tables 4, 5, 6, 7, 8 and 9. In addition to, the fog is completely heterogeneous in terms of resources and their speed the reason why we consider fluctuations in produced dataset. Tables 4, 5 and 6 gives underlying fog computing specifications for an example with 5 fog nodes. In this regards, Table 4 shows fog nodes specifications in terms of CPU clock rate, main memory and their threshold, minimum and maximum power consumption (idle vs full-loaded), kind of supported sensors and software, and power consumption of data transfer. In this table, the zero value indicates lack of support. The value 1 and 2 indicates the type of sensors. Tables 5 and 6 show bandwidth and latency between direct communications of fog nodes. The values was normalized in [0...1] interval. In Table 5, the value zero means that there is not any connection between nodes whereas the value one indicates the nodes are the same; this concept is reverse in Table 6.

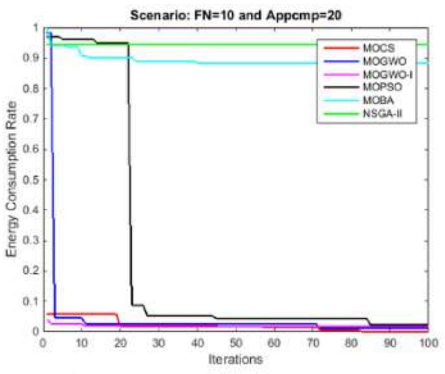
In this regards, Tables 7, 8, and 9 draw an example of resources requested for applications containing 5 different components. Table 7 is used for CPU, RAM, kind of sensors, and software requests for components. Table 8 is used for bandwidth requested for each pair of components. Also,



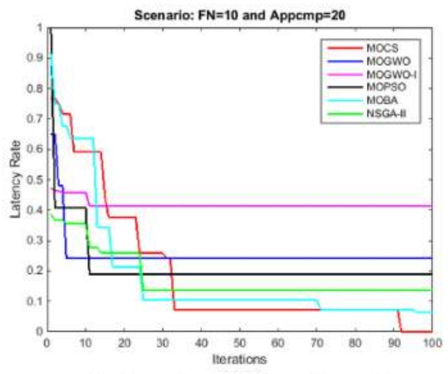
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 9 Performance comparison of different algorithms in scenario with 20 components on 10 fog nodes

Table 9 is utilized for the least latency tolerable between each pair of components.

For simulations and comparisons, parameter settings of algorithms MOCSA, MOGWO, MOPSO, MOBA, and NSGA-II are brought in Table 10.

Experimental results

In this section, the comparison between proposed MOCSA and other algorithms are based on Pareto front, two objective functions values, and elapsed time. Also, we utilize another versions of MOGWO algorithms known as MOGWO-I. In the second version, two operators crossover and mutation of genetic algorithm are applied for exploring the search space. In addition to, optimal deployment plan is drawn and the hosting node of application components is drawn in red color.

First scenario: 10 fog nodes and 20 application components

Figure 9 demonstrates performance comparison of different algorithms in a scenario with 20 requested components to be placed on 10 underlying fog nodes. Figure 9a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 9b; it shows the optimal deployment plan and the selected fog nodes are 1, 4, 5, 9, and 10. In addition to, Fig. 9c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

Table 11 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the second place after MOPSO that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others.

Second scenario: 15 fog nodes and 25 application components

Figure 10 demonstrates performance comparison of different algorithms in a scenario with 25 requested components to be placed on 15 underlying fog nodes. Figure 10a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 10b; it shows the optimal deployment plan and the selected fog nodes are 3, 5, 9, 10, and 13. In addition to, Fig. 10c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

Table 12 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the second place after MOPSO that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others. In term of execution time, the proposed MOCSA competes marginally with NSGA-II that is in the third place.

Table 11 Performance comparison of algorithms in term of elapsed time

MOCSA	238.41 s	MOGWO	266.4 s	MOBA	548.98 s
MOPSO	210.2 s	MOGWO-I	985.56 s	NSGA-II	276.07 s

Third scenario: 20 fog nodes and 30 application components

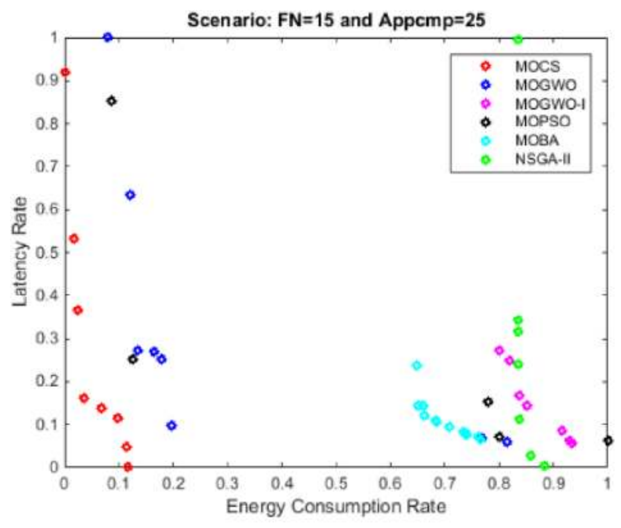
Figure 11 demonstrates performance comparison of different algorithms in a scenario with 30 requested components to be placed on 20 underlying fog nodes. Figure 11a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 11b; it shows the optimal deployment plan and the selected fog nodes are 6, 7, 11, 16, and 17. In addition to, Fig. 11c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

Table 13 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the third place after MOPSO and NSGA-II that are the fastest and the second fastest between all, but the quality of non-dominated solutions of MOCSA are better than others.

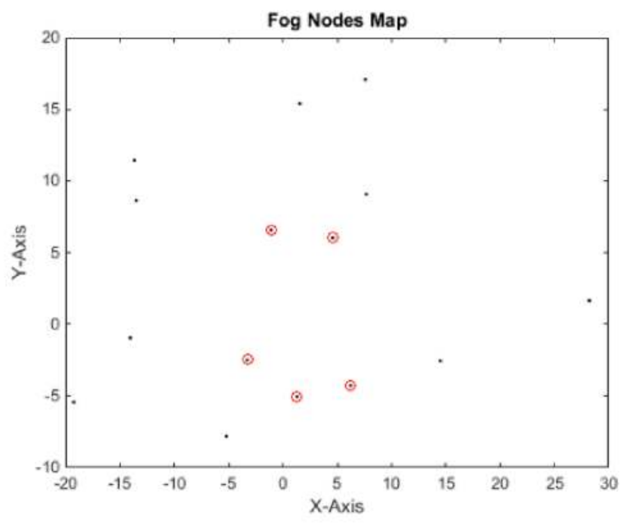
Fourth scenario: 25 fog nodes and 40 application components

Figure 12 demonstrates performance comparison of different algorithms in a scenario with 40 requested components to be placed on 25 underlying fog nodes. Figure 12a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 12b; it shows the optimal deployment plan and the selected fog nodes are 1, 4, 7, 9, 11, 16 and 21. In addition to, Fig. 12c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

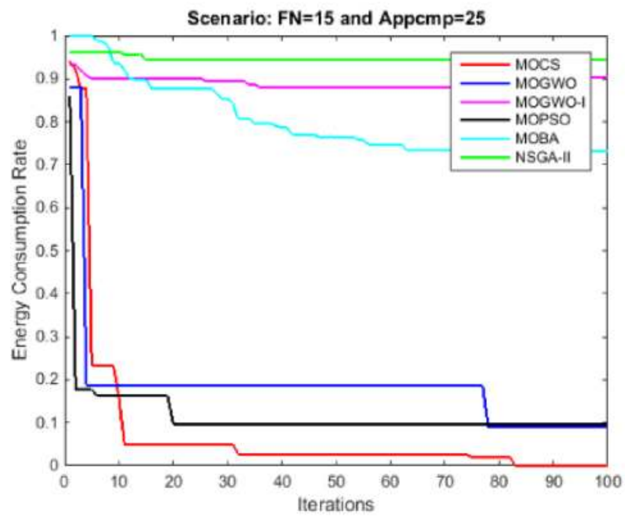
Table 14 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the third place after NSGA-II and MOPSO that is the fastest and second fastest between all, but the quality of non-dominated solutions of MOCSA are better than others.



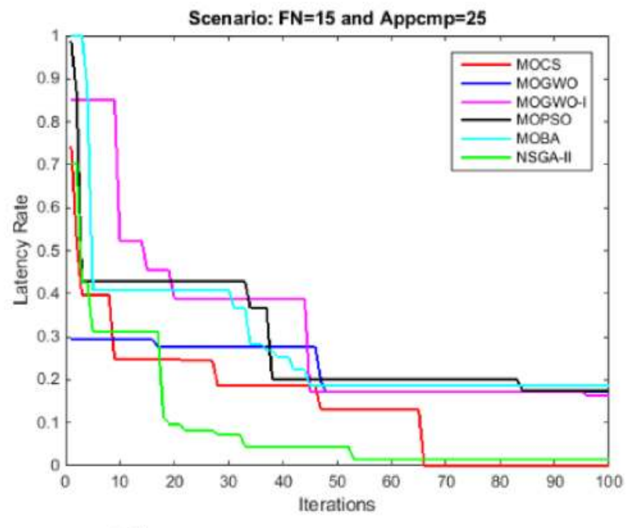
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 10 Performance comparison of different algorithms in scenario with 25 components on 15 fog nodes

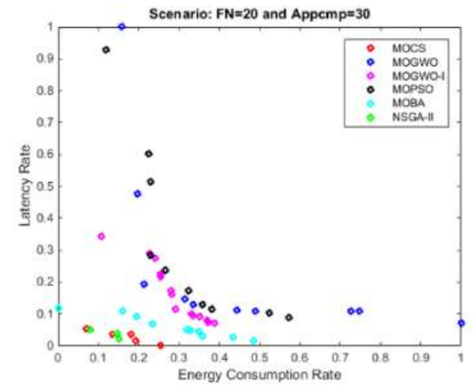
Fifth scenario: 40 fog nodes and 60 application components

Figure 13 demonstrates performance comparison of different algorithms in a scenario with 60 requested components to be placed on 40 underlying fog nodes. Figure 13a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 13b; it shows the optimal deployment plan and the selected fog nodes are 1, 2, 3, 7, 9, 12, 14, 15, 16, 19, 24, 28, 31, 33 and 37. In addition to, Fig. 13c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

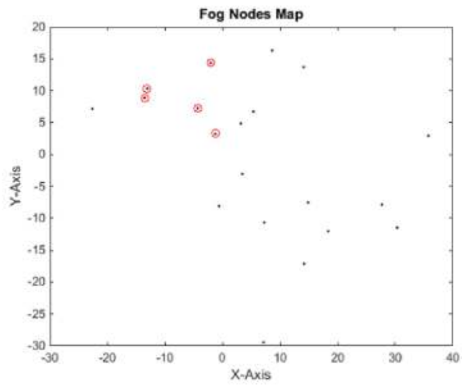
Table 15 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the

Table 12 Performance comparison of algorithms in term of elapsed time

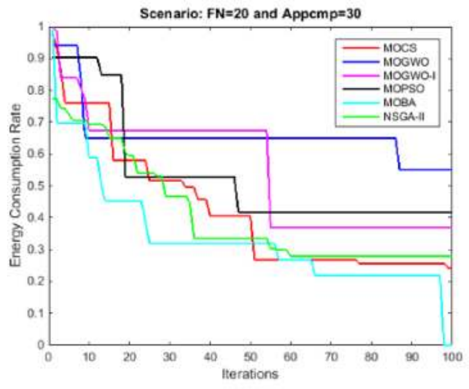
MOCSA	269.77 s	MOGWO	293.16 s	MOBA	301.02 s
MOPSO	241.55 s	MOGWO-I	1156.3 s	NSGA-II	270.01 s



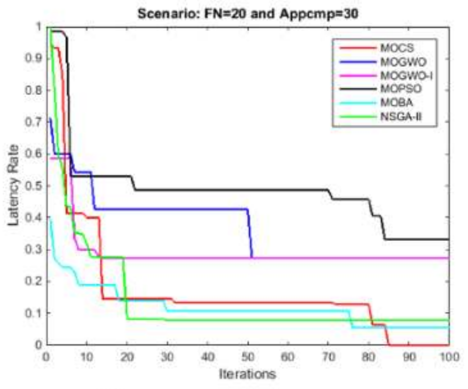
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 11 Performance comparison of different algorithms in scenario with 30 components on 20 fog nodes

Table 13 Performance comparison of algorithms in term of elapsed time

MOCSA	298.03 s	MOGWO	332.02 s	MOBA	342.39 s
MOPSO	275.62 s	MOGWO-I	1264.3 s	NSGA-II	290.87 s

second place after MOPSO that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others. In term of execution time, the proposed MOCSA competes marginally with NSGA-II that is in the third place.

Sixth scenario: 55 fog nodes and 75 application components

Figure 14 demonstrates performance comparison of different algorithms in a scenario with 75 requested components to be placed on 55 underlying fog nodes. Figure 14a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 14b; it shows the optimal deployment plan and the selected fog nodes are 3, 6, 8, 9, 22, 24, 28, 29, 31, 32, 39, 43, 45 and 55. In addition to, Fig. 14c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

Table 16 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the second place after NSGA-II that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others. In term of execution time, the proposed MOCSA competes marginally with MOPSO that is in the third place.

Seventh scenario: 70 fog nodes and 100 application components

Figure 15 demonstrates performance comparison of different algorithms in a scenario with 100 requested components

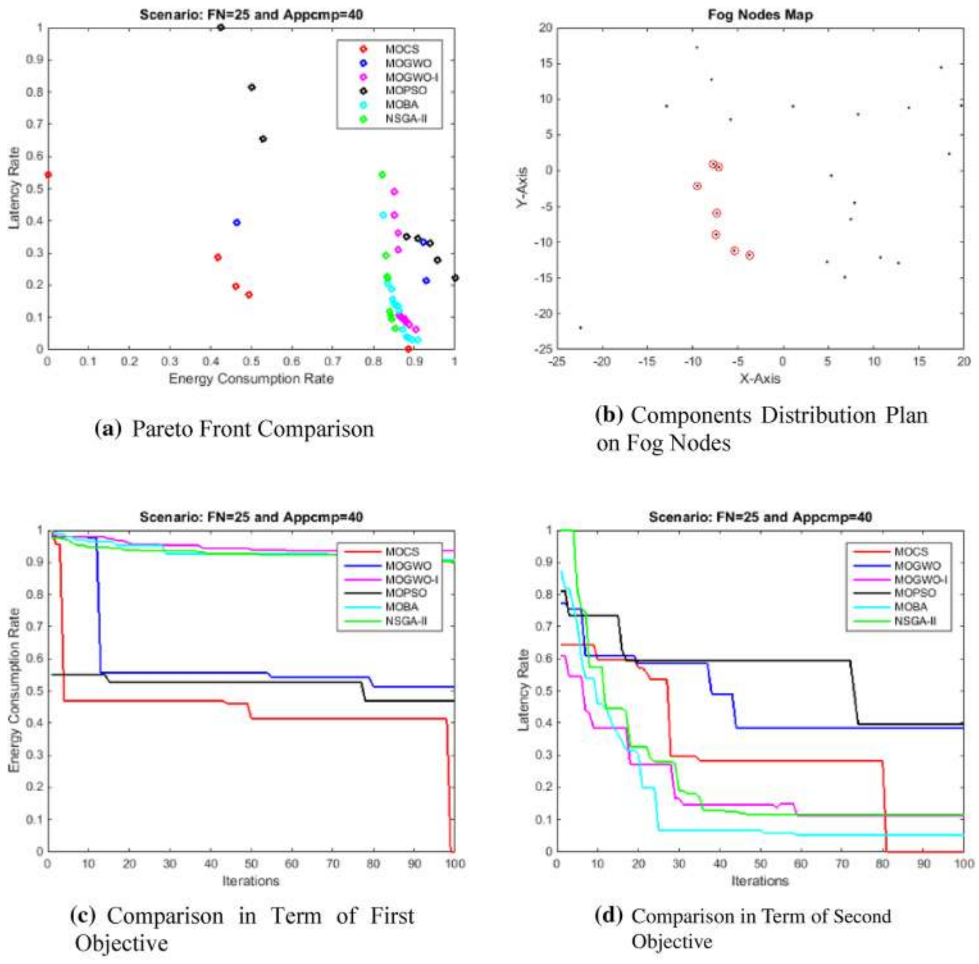


Fig. 12 Performance comparison of different algorithms in scenario with 40 components on 25 fog nodes

Table 14 Performance comparison of algorithms in term of elapsed time

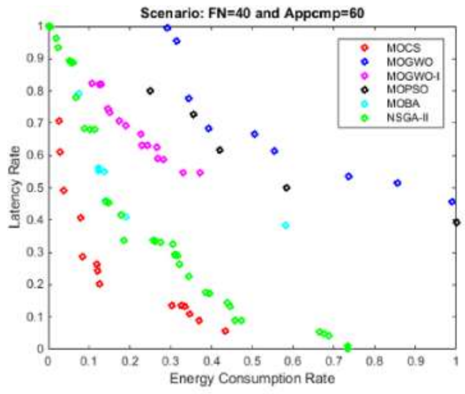
MOCSA	446.99 s	MOGWO	463.06 s	MOBA	469.16 s
MOPSO	401.48 s	MOGWO-I	1804.02 s	NSGA-II	379.46 s

to be placed on 70 underlying fog nodes. Figure 15a draws Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 15b; it shows the optimal deployment plan and the selected fog nodes are 8, 10, 13, 17, 18, 19, 20, 24, 30, 35, 38, 39, 40, 42, 45, 49, 51, 52, 53, 58, 64 and 66. In addition to, Fig. 15c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

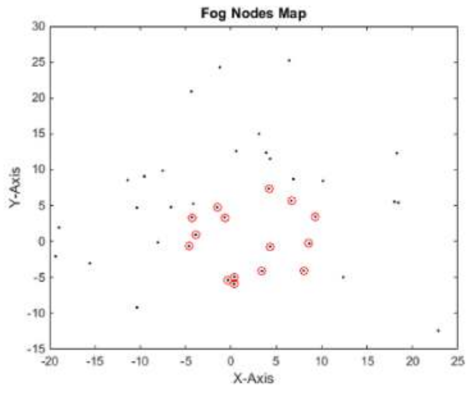
Table 17 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the second place after NSGA-II that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others. In term of execution time, the proposed MOCSA competes marginally with MOPSO that is in the third place.

Eighth scenario: 100 fog nodes and 150 application components

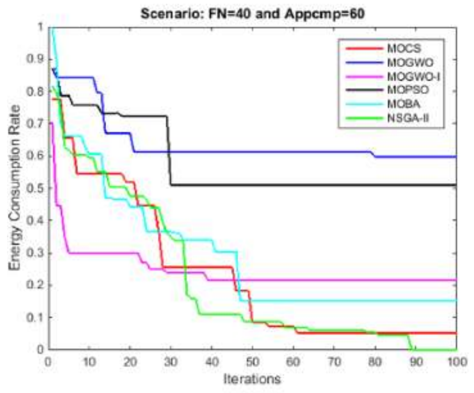
Figure 16 demonstrates performance comparison of different algorithms in a scenario with 150 requested components to be placed on 100 underlying fog nodes. Figure 16a draws



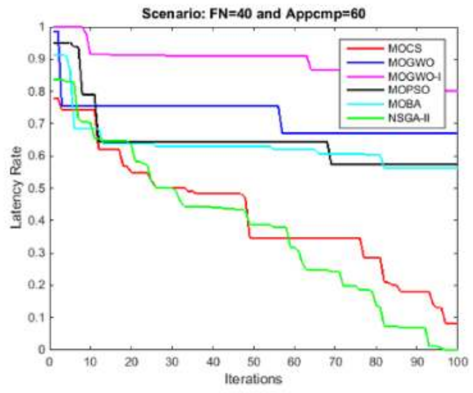
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 13 Performance comparison of different algorithms in scenario with 60 components on 40 fog nodes

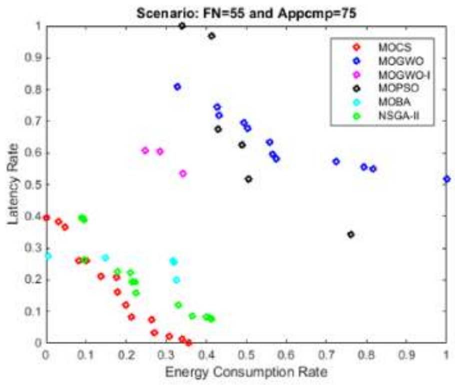
Table 15 Performance comparison of algorithms in term of elapsed time

MOCSA	666.82 s	MOGWO	760.19 s	MOBA	743.96 s
MOPSO	659.77 s	MOGWO-I	3401.5 s	NSGA-II	676.16 s

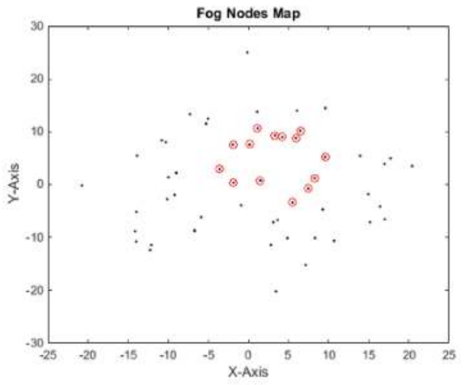
Pareto frontiers derived from different algorithms. As this figure shows, MOCSA outperforms against others. Mega Node which MOCSA extracts is depicted in Fig. 16b; it shows the optimal deployment plan and the selected fog nodes are 1, 9, 16, 17, 19, 21, 24, 28, 39, 40, 51, 53, 56, 57, 59, 60, 62, 63, 65, 72, 73, 78, 84, 86, 88, 89 and 93. In addition to, Fig. 16c and d depict comparison of different algorithms' performance in terms of the first objective (total power consumption based on Eq. (16)) and the second objective (overall latency based on Eq. (17)).

Table 18 compares algorithms' performance in term of elapsed time. This Table shows that MOCSA falls in the second place after NSGA-II that is the fastest between all, but the quality of non-dominated solutions of MOCSA are better than others. In term of execution time, the proposed MOCSA competes marginally with MOBA that is in the third place.

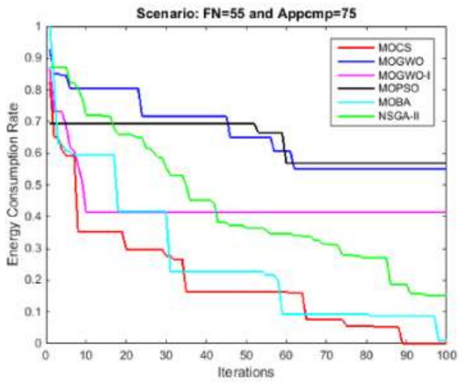
For the sake of data analysis statistically, the proposed MOCSA outperforms 43%, 28%, 41%, 30% and 32% improvement against MOGWO, MOGWO-I, MOPSO, MOBA and NSGA-II in term of average reduction in power consumption; also, in the minimum value gained by solutions, the proposed MOCSA outperforms 26%, 36%, 23%, 39% and 43% improvement against MOGWO, MOGWO-I, MOPSO, MOBA and NSGA-II in term of minimum



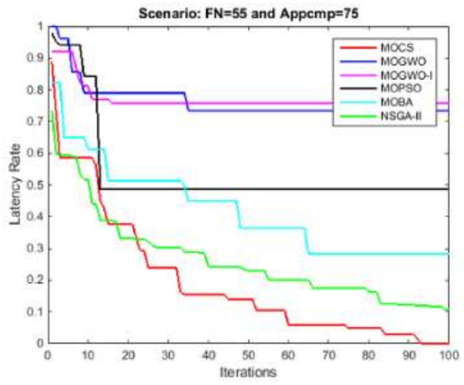
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 14 Performance comparison of different algorithms in scenario with 75 components on 55 fog nodes

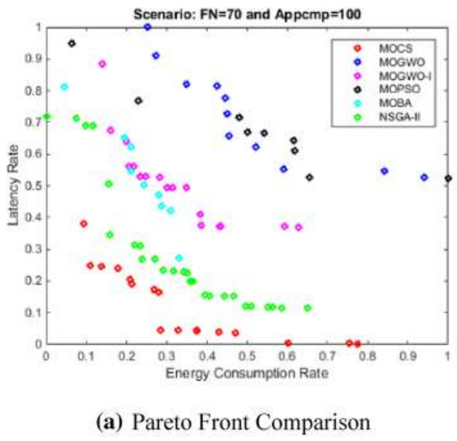
Table 16 Performance comparison of algorithms in term of elapsed time

MOCSA	835.23 s	MOGWO	1024.2 s	MOBA	1072.8 s
MOPSO	867.71 s	MOGWO-I	3690.06 s	NSGA-II	770.96 s

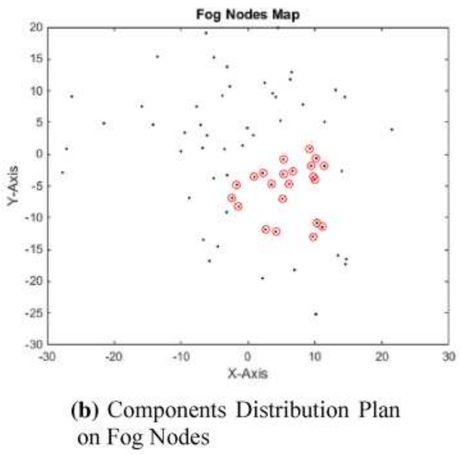
value of power consumption. In addition to, the proposed MOCSA outperforms 42%, 29%, 46%, 13% and 5% improvement against MOGWO, MOGWO-I, MOPSO, MOBA and NSGA-II in term of average reduction in overall latency; also, in the minimum value gained by solutions, the proposed MOCSA outperforms 40%, 33%, 37%, 17% and 6% improvement against MOGWO, MOGWO-I, MOPSO, MOBA and NSGA-II in term of minimum value of overall latency.

Time complexity

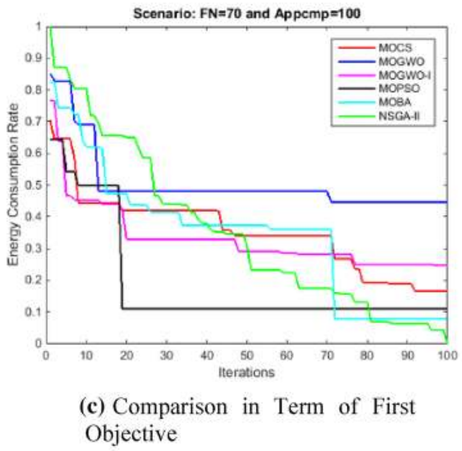
Now that, time complexity of all sub algorithms have been determined, the time complexity of Algorithm 1 is now calculated. The preprocessing takes $K{\bullet}N^2{+}M+N$ which belongs to $O(M+K{\bullet}N^2)$. Also, the main loop iterates Max-Iteration times. For the main loop, we have MaxIteration $\times(N{\bullet}\text{PopSize}+\text{PopSize}^2)$. If we consider $N<\text{PopSize}$, Algorithm 1's time complexity is $\text{O}(M+K.N^2+\text{Max-Iteration}{\bullet}\text{PopSize}^2)$ which is relatively acceptable time complexity.



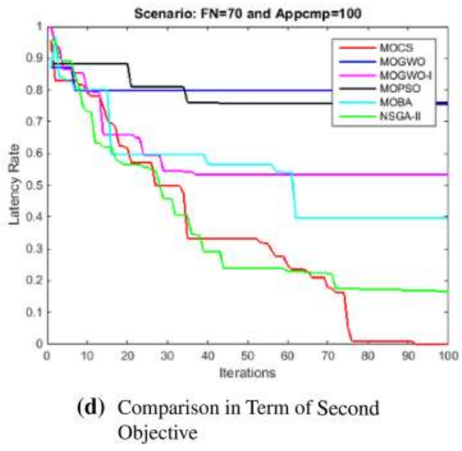
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

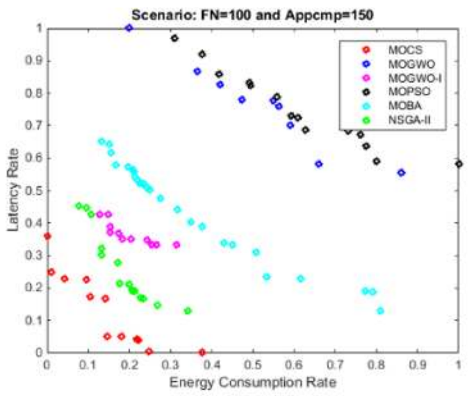
Fig. 15 Performance comparison of different algorithms in scenario with 100 components on 70 fog nodes

Conclusion and future direction

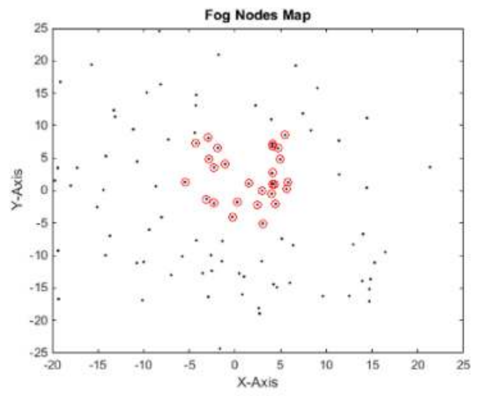
In this paper, an algorithm for the deployment of IoT application components on fog nodes has been presented to meet reliable deployment for user requests. To address this issue, this deployment problem was modeled to a multi-objective optimization problem with total power consumption and overall latency perspectives. To solve this combinatorial optimization problem, a multi-objective optimization algorithm based on cuckoo search meta-heuristic algorithm known MOCSA was extended. To reach concrete results, different scenarios were conducted and the effectiveness of proposed MOCSA was compared with well-reputed meta-heuristic algorithms MOGWO, MOPSO, MOBA, and NSGA-II in fair experimental conditions. The results obtained from simulations prove the significant superiority of proposed algorithm in terms of average overall latency and average total power consumption against other state-of-the-arts in objective functions. The merit of the current paper is to deliver users reliable services along with meeting objective functions. Also, the simulation proved the proposed MOCSA is potentially scalable. The limitation of the current work is to know the resource request in advance. For future work, we intend to present a dynamic model for mobile IoT applications in chain of fog computing nodes with QoS and economic perspectives to reach equilibrium in desired objectives.

Table 17 Performance comparison of algorithms in term of elapsed time

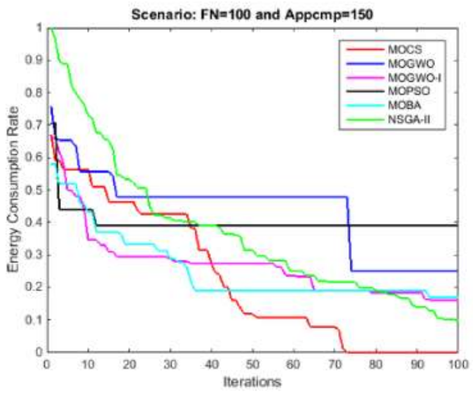
MOCSA	1139.4 s	MOGWO	1769.6 s	MOBA	1299.9 s
MOPSO	1236.2 s	MOGWO-I	4552.2 s	NSGA-II	1082.1 s



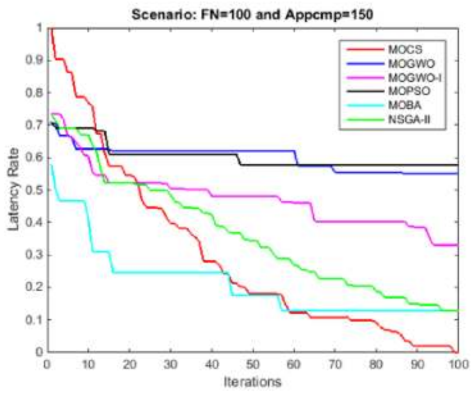
(a) Pareto Front Comparison



(b) Components Distribution Plan on Fog Nodes



(c) Comparison in Term of First Objective



(d) Comparison in Term of Second Objective

Fig. 16 Performance comparison of different algorithms in scenario with 150 components on 100 fog nodes

Table 18 Performance comparison of algorithms in term of elapsed time

MOCSA	2199 s	MOGWO	2392.4 s	MOBA	2294.5 s
MOPSO	2434 s	MOGWO-I	7104.7 s	NSGA-II	2038s

Declarations

Conflic of interest There is not any conflict of interest.

References

1. Foukalas F (2020) Cognitive IoT platform for fog computing industrial applications. Comput Electr Eng 87:1–13. https://doi.org/10.1016/j.compeleceng.2020.106770
2. OpenFog. An OpenFog Architecture Overview (2017) https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. Accessed Feb 2017
3. Azimi SH, Pahl C, Hosseini-Shirvani M (2020) Particle swarm optimization for performance management in multi-cluster IoT edge architectures. Int Cloud Comput Conf CLOSER. 2020:328–337. https://doi.org/10.5220/0009391203280337
4. Hong HJ, Tsai PH, Hsu CH (2016) Dynamic module deployment in a fog computing platform. In: 18th Asia-Pacific network operations and management symposium (APNOMS), pp 1–6. https://doi.org/10.1109/APNOMS.2016.7737202
5. Taneja M, Davy A (2017) Resource-aware placement of IoT application modules in fog-cloud computing paradigm. In: Proc. of the IFIP/IEEE symposium on integrated network and service

management, IM ’15, IEEE, pp 1222–1228. https://doi.org/10.23919/INM.2017.7987464
6. Brogi A, Forti A (2017) QoS-aware deployment of IoT applications through the fog. IEEE Internet Things J 4:1185–1192. https://doi.org/10.1109/JIOT.2017.2701408
7. Li F, Vogler M, Claeßens M, Dustdar S (2013) Towards automated IoT application deployment by a cloud-based approach. In: 6th international conference on service-oriented computing and applications, IEEE, pp 61–68. https://doi.org/10.1109/SOCA.2013.12
8. Mahmud R, Ramamohanarao K, Buyya R (2018) Latency-aware application module management for fog computing environments. ACM Trans Internet Technol 2018:1–21. https://doi.org/10.1145/3186592
9. Vögler M, Schleicher JM, Inzinger C, Dustdar S (2015) DIANE—Dynamic IoT Application Deployment. In: IEEE international conference on mobile services, pp 298–305. https://doi.org/10.1109/MobServ.2015.49
10. Saurez E, Hong K, Lillethun D, Ramachandran U, Ottenwalder B (2016) Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: DEBS, pp 258–269. https://doi.org/10.1145/2933267.2933317
11. Chen BL, Huang SC, Luo YC, Chung YC, Chou J (2017) A dynamic module deployment framework for M2M platforms. In: IEEE 7th international symposium on cloud and service computing (SC2). IEEE, pp 194–200. https://doi.org/10.1109/SC2.2017.37
12. Yangui S, Ravindran P, Bibani O, Glitho R. H, Hadj-Alouane NB, Morrow MJ, Polakos PA (2016) A platform as-a-service for hybrid cloud/fog environments. In: 2016 IEEE international symposium on local and metropolitan area networks (LANMAN), pp 1–7. https://doi.org/10.1109/LANMAN.2016.7548853
13. Babu R, Bhattacharyya B (2019) Strategic placements of PMUs for power network observability considering redundancy measurement. Measurement 134:606–623. https://doi.org/10.1016/j.measurement.2018.11.001
14. Babu R, Bhattacharyya B (2018) An approach for optimal placement of phasor measurement unit for power network observability considering various contingencies. Iran J Sci Technol Trans Electr Eng 42(2):161–183. https://doi.org/10.1007/s40998-018-0063-7
15. Babu R, Bhattacharyya B (2016) Optimal allocation of phasor measurement unit for full observability of the connected power network. Int J Electr Power Energy Syst 79:89–97. https://doi.org/10.1016/j.ijepes.2016.01.011
16. Babu R, Bhattacharyya B (2017) Weak bus-oriented installation of phasor measurement unit for power network observability. Int J Emerg Electr Power Syst 18:5. https://doi.org/10.1515/ijeeps-2017-0073
17. Babu R, Bhattacharyya B (2020) Optimal placement of PMU for complete observability of the interconnected power network considering zero-injection bus. Int J Appl Power Eng 9(2):135–146. https://doi.org/10.11591/ijape.v9.i2.pp135-146
18. Hosseini Shirvani M (2018) Web service composition in multi-cloud environment: a bi-objective genetic optimization algorithm. In: 2018 IEEE (SMC) international conference on innovations in intelligent systems and applications. https://doi.org/10.1109/INISTA.2018.8466267
19. Hosseini Shirvani M, Gorji AB (2020) Optimization of automatic web services composition using genetic algorithm. Int J Cloud Comput 9(4):397–411. https://doi.org/10.1504/IJCC.2020.112313
20. Hosseini-Shirvani M (2018) A new shuffled genetic-based task scheduling algorithm in heterogeneous distributed systems. J Adv Comput Res 2018:19–36
21. Hosseinzadeh S, Hosseini SM (2015) Optimizing energy consumption in clouds by using genetic algorithm. J Multidiscipl Eng Sci Technol 2(6):1431–1434
22. Razavi F, Zabihi F, Hosseini SM (2016) Multi-layer perceptron neural network training based on improved of stud GA. J Adv Comput Res 7(3):1–14
23. Javadian Kootanaee A, Poor Aghajan A, Hosseini SM (2021) A hybrid model based on machine learning and genetic algorithm for detecting fraud in financial statements. J Optim Ind Eng 14(2):180–201. https://doi.org/10.22094/joie.2020.1877455.1685
24. Hosseini-Shirvani M (2020) Bi-objective web service composition problem in multi-cloud environment: a bi-objective time-varying particle swarm optimisation algorithm. J Exp Theor Artif Intell 2020:1–24. https://doi.org/10.1080/0952813X.2020.1725652
25. Hosseini-Shirvani M (2019) A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems. Eng Appl Artif Intell 2019:90. https://doi.org/10.1016/j.engappai.2020.103501
26. Saeedi P, Hosseini SM (2021) An improved thermodynamic simulated annealing-based approach for resource-skewness-aware and power-efficient virtual machine consolidation in cloud datacenters. Soft Comput. https://doi.org/10.1007/s00500-020-05523-1
27. Noorian Talooki R, Hosseini Shirvani M, Motameni H (2021) A Hybrid Meta-heuristic scheduler algorithm for optimization of workflow scheduling in cloud heterogeneous computing environment. J Eng Design Technol Emerald Publ (In Press)
28. Tanha M, Hosseini Shirvani M, Rahmani AM (2020) GATSA: a hybrid meta-heuristic task scheduling algorithm based on genetic and thermodynamic simulated annealing algorithms in cloud computing environment. Neural Comput Appl Springer Publ (In Press)
29. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multi objective genetic algorithm: Nsga-II. IEEE Trans Evol Comput 6(2):182–197. https://doi.org/10.1109/4235.996017
30. Coello CAC, Lechuga MS (2002) MOPSO: a proposal for multiple objective particle swarm optimization. In: Proceedings of the 2002 congress on evolutionary computation (CEC’02). USA: IEEE Publications. https://doi.org/10.1109/CEC.2002.1004388
31. Hosseini-Shirvani M, Rahmani AM, Sahafi A (2018) An iterative mathematical decision model for cloud migration: a cost and security risk approach. Softw Pract Exp Homepage 48(3):449–485. https://doi.org/10.1002/spe.2528
32. Yang XS (2011) Bat algorithm for multiobjective optimization. Int J Bio-Inspired Comput 3(5):267–274. https://arxiv.org/abs/1203.6571v1
33. Mirjalili S, Saremi S, Mirjalili SM, Coelho LDS (2016) Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. J Expert Syst Appl Elsevier 47:106–119. https://doi.org/10.1016/j.eswa.2015.10.039
34. Wang Z, Ong Y, Ishibuchi H (2019) On scalable multiobjective test problems with hardly dominated boundaries. IEEE Trans Evol Comput 23(2):217–231. https://doi.org/10.1109/TEVC.2018.2844286
35. Wang Z, Ong Y, Sun J, Gupta A, Zhang Q (2019) A generator for multiobjective test problems with difficult-to-approximate pareto front boundaries. IEEE Trans Evol Comput 23(4):556–571. https://doi.org/10.1109/TEVC.2018.2872453
36. Wang Z, Zhang Q, Zhou A, Gong M, Jiao L (2016) Adaptive replacement strategies for MOEA/D. IEEE Trans Cybern 46(2):474–486. https://doi.org/10.1109/TCYB.2015.2403849
37. Wang Z, Zhang Q, Li H, Shibuchi H, Jiao L (2017) On the use of two reference points in decomposition based multiobjective evolutionary algorithms. Swarm Evol Comput 34:89–102. https://doi.org/10.1016/j.swevo.2017.01.002
38. Ali LB, Helaoui M, Naanaa W (2019) Pareto-based soft arc consistency for multi-objective valued CSPs. ICAART. 2019:294–305
39. Akyildiz IF, Wang X, Wang W (2005) Wireless mesh networks: asurvey. Comput Netw 47(4):445–487. https://doi.org/10.1016/j.comnet.2004.12.001

40. Arcangeli JP, Boujbel R, Leriche S (2015) Automatic deployment of distributed software systems: definitions and state of the art. J Syst Softw 3:198–218. https://doi.org/10.1016/j.jss.2015.01.040
41. Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: a platform for internet of things and analytics. In: Big data and internet of things: a roadmap for smart environments, Springer, pp 169–186. https://doi.org/10.1007/978-3-319-05029-4_7
42. Farzai S, Hosseini-Shirvani M, Rabbani M (2020) Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters. Sustain Comput Inf Syst 2020:28. https://doi.org/10.1016/j.suscom.2020.100374
43. Yang XS, Deb S (2009) Cuckoo search via Levy flights. In: Proceedings of world congress on nature & biologically inspired computing, pp 210–214. https://doi.org/10.1109/NABIC.2009.5393690
44. Sait SM, Bala A, El-Maleh AH (2016) Cuckoo search based resource optimization of datacenters. Appl Intell 44:489–506. https://doi.org/10.1007/s10489-015-0710-x
45. Tavana M, Shahdi-Pashaki S, Teymourian E, Santos-Arteaga FJ, Komaki M (2017) A discrete cuckoo optimization algorithm for consolidation in cloud computing. Comput Ind Eng 115:495–511. https://doi.org/10.1016/j.cie.2017.12.001
46. Hosseini Shirvani M, Farzai S (2020) Optimal deployment of IoT application components on hybrid fog2cloud infrastructure for reduction of power consumption toward green computing by cuckoo search algorithm. In: The first national conference of New Development in Green Studies, Computations, Applications, and Challenges, NGIS01
47. Walton S, Hassan O, Morgan K, Brown MR (2011) Modified cuckoo search: a new gradient free optimisation algorithm. Chaos Solitons Fractals 44(9):710–718. https://doi.org/10.1016/j.chaos.2011.06.004
48. Yang XS (2010) A new metaheuristic bat-inspired algorithm, in nature inspired cooperative strategies for optimization. Stud Comput Intell 284:65–74. https://doi.org/10.1007/978-3-642-12538-6_6