Energy-aware and carbon-efficient VM placement optimization in cloud datacenters using evolutionary computing methods

Tahereh Abbasi-khazaei1 · Mohammad Hossein Rezvani2

Accepted: 9 May 2022 / Published online: 30 June 2022 © The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract

One of the most critical concerns of cloud service providers is balancing renewable and fossil energy consumption. On the other hand, the policy of organizations and governments is to reduce energy consumption and greenhouse gas emissions in cloud data centers. Recently, a lot of research has been conducted to optimize the virtual machine placement on physical machines to minimize energy consumption. Many previous studies have not considered the deadline and scheduling of Internet of Things (IoT) tasks. Therefore, the previous modelings are mainly not well-suited to the IoT environments where requests are time-constraint. Unfortunately, both the sub-problems of energy consumption minimization and scheduling fall into NP-hard issues. This study proposes a multi-objective virtual machine placement to jointly minimize energy costs and scheduling. After presenting a modified Memetic algorithm, we compare its performance with baseline methods and state-of-the-art ones. The simulation results on the CloudSim platform show that the proposed method can reduce energy costs, carbon footprints, service-level agreement violations, and the total response time of IoT requests.

Keywords Cloud computing · Virtual machine placement · Energy consumption · Renewable energy · Carbon footprint · Cost optimization

1 Introduction

A big challenge with cloud data centers is high power consumption. In general, electricity and maintaining a cloud data center are two of the high costs imposed. The cost of electricity is a more critical factor. Studies show that an idle server consumes about 50–70% of its maximum power. Also, according to Amazon estimates, the energy cost of a data center accounts for 40% of its total Operational Expenditures (OPEX) (Khosravi et al. 2017). The rate of carbon generated in a cloud data center depends on the type of energy source it consumes. It varies for different energy sources such as oil, coal, and renewable energy. Thus, even a slight reduction in energy consumption leads to significant cost savings and a carbon footprint. Much of this problem can be alleviated through the dynamic placement of Virtual Machines (VMs). For this purpose, the real-world parameters affecting the Virtual Machine Placement (VMP) must be considered in the modeling. Optimizing energy consumption directly affects carbon footprint and carbon taxation, which, in turn, can reduce the cost of cloud service providers.

In the past, a lot of research has been conducted on the VMP. The main challenge in this regard is to perform the VMP in such a way that minimizes energy consumption and, at the same time, satisfies users’ Service Level Agreement (SLA) and Quality of Experience (QoE). Some researchers have used classical mathematics methods (Ahvar et al. 2019; Thiam and Thiam 2019; Xu and Buyya 2020; López et al. 2019). The VMP falls into bin-packing problems, an NP-hard issue (Parvizi and Rezvani 2020; Aboutorabi and Rezvani 2020). Therefore, some researchers have used heuristic methods to reduce allocation time (Khosravi et al. 2017; Tavakoli-Someh and...
The emergence of the Internet of Things (IoT) has recently generated big data volumes. Many of these data come from delay-sensitive applications. Some of the most critical applications are agriculture (Khoobkar et al. 2022), vehicular traffic management (Esfandiari and Rezvani 2020; Babazadeh Nanehkaran and Rezvani 2021), e-health (Karuppiah and Gurunathan 2021), and so on. Each IoT request is assigned to a VM to be executed. Then each VM must, in turn, be mapped to a suitable physical server using VMP approaches. Here, it is essential to pay attention to the deadline of tasks. Unfortunately, many previous studies have merely focused on optimizing green energy consumption and have by no means considered task scheduling. Therefore, previous modelings are unsuitable for IoT environments where requests are time-constraint. Scheduling optimization, like energy efficiency, falls into the category of NP-hard problems.

This study proposes a multi-objective VMP algorithm for joint minimization of energy costs and scheduling. After presenting a modified Memetic algorithm, we compare its performance with baseline methods such as the Genetic Algorithm (GA) and state-of-the-art techniques. Roughly speaking, we seek to answer how to optimize the VMP concerning the SLA and the QoE of users while considering both energy and scheduling constraints. Some of our most important assumptions are:

- Each IoT task has a deadline.
- We have information about changes in the outdoor temperature of the servers in a certain period.
- Solar energy is considered a significant type of renewable energy.
- Datacenters are considered geographically distributed.
- Energy costs vary for data centers that are geographically dispersed.

The most important innovations of this research are as follows:

- We formulate the two sub-problem of energy cost optimization and IoT task scheduling optimization. Then, we solve them jointly with the deterministic weighted sum method, which is one of the well-known methods of nonlinear optimization. To the best of our knowledge, no research has been conducted on the joint optimization of green energy and task scheduling.
- Because the problem is NP-hard, we solve it on a large scale using a modified MA method. Then, we evaluate the performance of the proposed method against the baseline method, such as GA and state-of-the-art techniques.

The remainder of this paper is organized as follows: Sect. 2 reviews the most critical studies on energy-efficient VMP; Sect. 3 explains the problem formulation; Sect. 4 describes the proposed method; Sect. 5 evaluates the results of simulating the proposed method on the CloudSim simulator (Calheiros et al. 2011); Finally, Sect. 6 concludes the study and provides suggestions for future research.

2 Related work

So far, there has been a great deal of literature on energy cost minimization. Before reading this section, interested readers are highly recommended to refer to Ahvar et al. (2019) for a comprehensive survey regarding the architecture. After studying previous research, we divided it into three categories: exact (classic mathematical), heuristic, and meta-heuristic. For each of these categories, we present the purpose, limitations, methodology, and evaluation criteria in Tables 1 and 2.

Some researchers address the issue of VMP. To solve the carbon dioxide problem, Xu and Buyya (2020) proposed a strategy for managing the greenhouse gas and renewable energy effects for several data centers in California, Virginia, and Dublin located in different time zones. Their results show that the proposed algorithm could reduce carbon emissions by nearly 40% compared to state-of-the-art data centers while guaranteeing an average response time to user requests. Other related research can be found in Khosravi et al. (2017). Primary research using classical mathematics and heuristic methods is shown in Table 1.

Another group of studies has used meta-heuristic methods. Gao et al. (2013) developed an algorithm inspired by Ant Colony Optimization (ACO). The goal is to obtain a set of non-dominant solutions (Pareto Optimal set) that simultaneously minimizes the waste of resources and energy consumption. Parvizi and Rezvani (2020) solved the VMP problem using a Non-dominated Sorting Genetic Algorithm (NSGA-III) to minimize total resource loss and minimize energy consumption and also the number of active physical machines. For this purpose, a multi-objective optimization problem was designed, and after introducing a nonlinear convex optimization solution, it is solved by the NSGA-III method. Also, in Rashida et al. (2019), the optimization of VMP in heterogeneous multi-cloud systems by considering the peak time of demand and...
the geographical location of allocated resources was presented, aiming to reduce energy costs. The authors also presented a dynamic energy model designed for physical machines and cloud communication components. The GA has been used in some research, such as (Tavakoli-Someh and Rezvani 2019; Misra and Kuila 2022; Keshavarznejad et al. 2020). Besides the methods mentioned above, other methods such as the artificial bee algorithm (Aboutorabi and Rezvani 2020) and the Cuckoo algorithm (Mohammadi and Rezvani 2019) have been used.

Iwendi et al. (2021) developed a method to minimize the energy consumption of sensors in the IoT network by combining the Whale Optimization Algorithm (WOA) with the Simulated Annealing (SA). They tried to select the most suitable Cluster Heads (CH) in the IoT network by considering several criteria such as the number of nodes, load, temperature, and residual energy. Maddikunta et al. (2020) conducted a study with the same objectives, but this time combining the WOA and a Butterfly Flame Optimization (MFO) algorithm. RM et al. (2020) proposed a new architecture to minimize energy consumption in the IoT. Their proposed method first minimizes energy consumption by clustering different IoT networks using a wind-driven optimization algorithm. Next, an optimized CH is selected for each cluster using the Firefly algorithm. Their method reduces data traffic compared to other non-clustered state-of-the-art designs. The CH selection is a complex issue in almost all types of networks. For example, it is common for wireless sensor networks to use evolutionary methods to select CHs to minimize energy consumption. To see how to model such systems, you can refer to Chauhan et al. (2021a).

Optimization methods are also widely used in other fields of engineering. Vashishtha and Kumar (2022) proposed a deep-learning scheme to identify bucket defects in the Pelton wheel. Initially, the raw vibration signal is transmitted through a time-varying filter. The filter parameters are optimized by the Amended Grey Wolf Optimization (AGWO). They use the Convolutional Neural Network (CNN) model to train the dataset. Another example is the minimization of passband, stopband, and transition band errors for designing a two-channel quadrature mirror filter bank (Chauhan et al. 2021b). Interested readers can refer to Vashishtha and Kumar 2021a, b and Vashishtha et al. (2021) to see more engineering applications of such optimization methods.

### Table 1  Research done with exact (classic mathematical) and heuristic methods

| Refs.          | Years | Objective                                      | Constraints                                                                                   | Evaluation criteria                      | Method      |
|----------------|-------|------------------------------------------------|---------------------------------------------------------------------------------------------|------------------------------------------|-------------|
| Ahvar et al.   | 2019  | Energy consumption estimates                   | Heterogeneity of VM in size (CPU and RAM)                                                   | Total energy consumption                 | Exact       |
|                |       |                                                | Execute all VMs during the period                                                           | Dynamic energy consumption               |             |
|                |       |                                                | Uniform distribution of VMs                                                                 |                                          |             |
|                |       |                                                | Number of computer nodes for all architectures                                             |                                          |             |
| Thiam and Thiam| 2019  | Reducing energy consumption                    | VM migration limit                                                                          | Energy consumption                      | Exact       |
|                |       |                                                | Select the host where the VM migrated                                                       | VM migrations                            |             |
|                |       |                                                | Refrain from overloading/underloading the new host                                          | The standard deviation of the execution time |             |
| Xu and Buyya   | 2019  | Reduce total greenhouse gas emissions          | Do not exceed the capacity of the VM                                                        | Comparison of brown energy consumption (such as fossil energy) for several algorithms | Exact       |
|                |       | Reduce the response time                       | The utilization of the VM to execute requests should not exceed the capacity of the physical machine | Average execution time                   |             |
| Laganà et al.  | 2018  | Reducing the amount of energy required by the power grid | PUE                                                                                          | Energy price                             | Heuristic   |
|                |       |                                                | Datacenter battery capacity                                                                 | Hourly production of renewable energy    |             |
|                |       |                                                | The capacity of solar panels                                                                | Energy consumption                       |             |
|                |       |                                                |                                                                                              | Energy cost                              |             |
|                |       |                                                |                                                                                              | Electric energy demand                   |             |
In summary, the most important differences between this study and the above research are as follows:

- Some studies (Iwendi et al. 2021; RM et al. 2020; Maddikunta et al. 2020; Chauhan et al. 2021a) have tried to balance the load while minimizing response time. Although load balancing may reduce response time, it does not necessarily reduce energy consumption. In this research, our focus is not on load balancing and the selection of CHs.

- Many previous studies (Khosravi et al. 2017; Parvizi and Rezvani 2020; Tavakoli-Someh and Rezvani 2019; Mohammadi and Rezvani 2019) focus exclusively on minimizing energy consumption within the data center and pay no attention to the outside. This study considers both energy consumption during offloading from the end device to the base station and renewable energy consumption within the data center.

- None of the above research considers satisfying the time constraint for IoT tasks. This makes modeling inconsistent with real-world requirements, especially for delay-sensitive tasks. This study finds the joint minimization of brown energy consumption and task scheduling. Although optimizing these two conflicting goals may slightly increase energy consumption, it improves the overall acceptance ratio of users. The results of our simulation confirm this claim.

### 3 System model

In this section, we formulate the energy consumption optimization sub-problem and the scheduling optimization problem, respectively. Then, by combining them, we present a joint optimization problem.

#### 3.1 Main parameters

In this study, the total cost of the data center, $c_{\text{total}}$, consists of two components: the cost of energy consumption of the system, $c_{\text{energy}}$, and the cost of carbon dioxide produced, $c_F$. Also, we assume that three types of energy sources can be used to supply power to each data center. First, we try to use green or renewable energy (for example, solar energy) if available. This will reduce the cost incurred by carbon dioxide. If renewable energy is not available, off-site grid energy can be used. Finally, brown fuels, such as diesel generators, can be used in the cloud data center if these two energies are not available. Several geographically distributed data centers are interconnected through the same network infrastructure. The notations used in this research are shown in Table 3.
Table 3 Notations used in this study

| Symbol | Symbol description |
|--------|---------------------|
| $D$    | Total number of data centers |
| $D = \{d_1, d_2, \ldots, d_D\}$ | Data centers set |
| $H$    | The total number of physical servers in a data center |
| $H = \{h_1, h_2, \ldots, h_H\}$ | Set of physical servers in a data center |
| $E$    | The set of energy resources |
| $V$    | Total number of virtual machine requests |
| $V = \{v_1, v_2, \ldots, v_V\}$ | A set of virtual machine applications |
| $X_{V \times D}$ | A matrix with binary elements to show the virtual machine mapping status to the data center |
| $x_{vi}^{d_j}$ | Each element of the matrix $X_{V \times D}$ to show the mapping of a virtual machine $v_i$ to a data center $d_j$ |
| $Y_{1 \times 3}$ | A row vector that indicates the virtual machine $v_i$ is mapped to the brown/green/off-site power supply network |
| $y_{vi}^{br}$ | The $v_i$ element of the row vector $Y_{1 \times 3}$ that indicates the virtual machine $v_i$ is mapped to the brown power supply network |
| $y_{vi}^{gr}$ | The $v_i$ element of the row vector $Y_{1 \times 3}$ that indicates the virtual machine $v_i$ is mapped to the green power supply network |
| $y_{vi}^{off}$ | The $v_i$ element of the row vector $Y_{1 \times 3}$ that indicates the virtual machine $v_i$ is mapped to the off-site power supply network |
| $Z_{V \times H}$ | A matrix with binary elements to show the virtual machine mapping status to the server |
| $z_{hv}$ | Each element of the matrix $Z_{V \times H}$ that shows the mapping status of the virtual machine $v_i$ to the server $h_j$ |
| $T_{v}$ | Maintenance time of the virtual machine $v_i$ |
| $c_T$ | The total cost of energy and carbon |
| $c_{vi}$ | The cost of running virtual Machine $v_i$ on the data center $d_j$ |
| $c_E$ | Energy cost |
| $c_{FP}$ | The cost of carbon footprint |
| $c_{E_{vi}}$ | The cost of server energy to run a virtual machine $v_i$ |
| $c_{E_{vi}}^{\text{O}}$ | Overhead energy cost to run a virtual machine $v_i$ |
| $c_{E_{vi}}^{\text{E}}$ | The cost of energy to run a virtual machine $v_i$ |
| $E_{vi}^{\text{T}}$ | Total energy to run a virtual machine $v_i$ |
| $E_{vi}^{\text{S}}$ | Server energy to run the virtual machine $v_i$ |
| $E_{vi}^{\text{O}}$ | Overhead energy for running a virtual machine $v_i$ |
| $E_{vi}^{\text{br}}$, $E_{vi}^{\text{gr}}$, $E_{vi}^{\text{off}}$ | Power consumed in the form of local brown / local green/offsite to run a virtual machine on the server |
| $E_{vi}^{\text{br}}$, $E_{vi}^{\text{gr}}$, $E_{vi}^{\text{off}}$ | The cost of local brown/local green/offsite energy used to run the virtual machine $v_i$ on the server |
| $\mu_{h_i}$ | Power consumption by a physical server $h_i$ at peak |
| $\mu_{idle}$ | Power consumption by a physical server $h_i$ at idle state |
| $u_{h_i}(t)$ | Utilization of a physical server $h_i$ at a time $t$ |
| $P_{h_i}(u, t)$ | Power consumption of a physical server $h_i$ at a time $t$ with the utilization $u$ |
| $P_{h_i}(u, t^+)$ | Power consumption of a physical server $h_i$ with the execution of a new virtual machine and with the utilization $u$ |
| $\theta_{d_j}$ | Overhead energy |
| $\theta_{d_j}(t)$ | Utilization of a data center $d_j$ at time $t$ |
| $h_{d_j}(t)$ | Outside temperature of a data center $d_j$ at time $t$ |
| $E_{vi}^{\text{br}}$, $E_{vi}^{\text{gr}}$, $E_{vi}^{\text{off}}$ | Local brown/local green / offsite energy overhead used to run a virtual machine $v_i$ |
| $c_{br_{vi}}^{\text{FP}}$, $c_{gr_{vi}}^{\text{FP}}$, $c_{off_{vi}}^{\text{FP}}$ | Cost of consuming local brown / local green/offsite energy used to run a virtual machine $v_i$ |
| $r_{br_{vi}}$, $r_{gr_{vi}}$, $r_{off_{vi}}$ | The carbon production rate in local brown / local green/offsite energy used to run a virtual machine $v_i$ |
| $c_{tax}$ | Carbon tax on energy resources in the form of local brown / local green/offsite |
| $\text{cpu}_{vi}$ | Required processing capacity of a virtual machine $v_i$ |
| $\text{mem}_{vi}$ | The required memory capacity of a virtual machine $v_i$ |
3.1.1 The datacenter energy efficiency

We use the Power Usage Effectiveness (PUE) metric to report the performance of a data center, introduced by Green Grid (a non-profit organization of IT professionals) in 2007. This metric has become the most common metric for reporting “energy efficiency.” Organizations such as the Uptime Institute track and report the average PUE of data centers using numerous surveys (Brady et al. 2013). Conceptually, PUE is the power used by IT devices in a data center divided by the total amount of fuel consumed (Fawaz et al. 2019). The higher the PUE, the more energy is wasted in data centers to maintain servers (Khosravi et al. 2017). Accordingly, companies such as Google and Facebook have significantly reduced their PUE in recent years by focusing on the utilization and design of custom hardware. Thus, a value of 1 for PUE is ideal for a data center and indicates that 100% of the energy provided by each piece of equipment is used. The coefficients of this equation have already been calculated by Khosravi et al. (2017) based on the results in Rasmussen et al. (2009) as follows:

\[
PUE(\frac{u_{dc}(t)}{h_{dc}(t)}, h_{dc}(t)) \approx 1 + 0.2 + 0.01 \frac{u_{dc}(t)}{h_{dc}(t)} + 0.01 \frac{u_{dc}(t)}{h_{dc}(t)}h_{dc}(t)
\]

Interested readers can refer to Khosravi et al. (2017) to see how the coefficients of the above equation are calculated.

3.1.2 Overhead power

We denote the overhead energy to run the virtual machine \( v_i \) by \( E_{\text{overhead}} \). Today, server cooling systems are essential elements that have a remarkable effect on overhead energy costs.

3.1.3 Server energy

A previous study by Gao et al. (2013) has shown that there is a linear relationship between the power of a server \( h_i \) and its CPU utilization \( u_h(t) \):

\[
P_{h_i}(u, t) = P_{h_i}^{\text{idle}} + \left( P_{h_i}^{\text{peak}} - P_{h_i}^{\text{idle}} \right) u_h(t)
\]

where \( P_{h_i}^{\text{idle}} \) and \( P_{h_i}^{\text{peak}} \) are the average energy values of the server \( h_i \) at idle and peak loads, respectively.

3.1.4 Energy cost

Let denote the cost of brown, renewable, and off-site energies consumed to run a virtual machine \( v_i \) on the server by \( c_{\text{brown}} \), \( c_{\text{green}} \), and \( c_{\text{offsite}} \), respectively. Here, energy consumption is calculated in cents per kilowatt-hour (cent/KWh). Although the cost of renewable energy, \( c_{\text{green}} \), is negligible, the cost of the other two fuels is high. Depending on the geographical distribution of data centers, these costs depend on governmental policies in different countries.
3.1.5 Density/taxation of carbon dioxide

The density of carbon dioxide produced by data centers can vary depending on the energy source and location. Three energy sources have been used in this research, and the carbon density for renewable energy has been considered zero. The rate of carbon dioxide produced is calculated in tons per Megawatt-hour (Ton/MWh). Due to environmental concerns, especially global warming, some countries calculate the tax on carbon dioxide produced in dollars per tonne (Dollars/Tonne) to implement environmental sustainability. This study considers the density of carbon produced from a renewable energy source zero, and the relevant tax cost is also zero.

3.1.6 Temperature

The following primary variable significantly affecting the PUE is the temperature outside the data center. Let \( h_{dcj}(t) \) denote the outside temperature of a data center \( dcj \) at time \( t \). The higher the outdoor temperature of a server, the higher the cooling system’s energy consumption. This, in turn, will lead to a significant increase in the PUE.

3.2 Cost optimization sub-problem

Here, we formulate the objective function and constraints of the energy efficiency sub-problem.

3.2.1 Objective function

Our goal is to reduce the total cost of the system, \( c^T \), which is the summation of energy cost, \( c^E \), and the cost of producing carbon dioxide \( c^{FP} \):

\[
c^T = c^E + c^{FP}
\]

We write

\[
c^T = \sum_{v \in V} \sum_{d_j \in D} c^E_{v_j} \cdot x^d_{v_j}
\]

We denote each element of the matrix \( X_{V \times D} \) by \( x^d_{v_j} \). If the decision variable \( x^d_{v_j} \) takes 1, it means assigning \( v_j \) to the data center \( d_j \). Now, we describe each of the components of the above equation:

(a) Energy cost

The cost of energy, \( c^E \), can be divided into two components. The first component, \( c^E_{v_j} \), is the cost of energy consumed to execute a request by the server. The second component, \( c^O_{v_j} \), is the cost of overhead energy to perform the virtual machine \( v_j \)'s request:

\[
c^E = \sum_{v \in V} \left( c^S_{v_j} + c^O_{v_j} \right)
\]

Note that the cost of the carbon tax is calculated separately. Also, three different energy sources are used in geographically dispersed areas, each with a foreign financial policy. Therefore, the costs incurred to perform the server request can vary dramatically. This energy is calculated as follows:

\[
c^S_{v_j} = \sum_{\varepsilon \in \{br, gr, off\}} E^S_{v_j} \cdot c^{E,E}
\]

Also, the power consumption of the server to run a virtual machine \( v_j \) is as follows:

\[
E^S_{v_j} = y^e_{v_j} \cdot E^S_{v_j}
\]

Depending on which of the three energy sources \( e = \{br, gr, off\} \) is used, the decision variable \( y^e_{v_j} \) takes different values from the vector \( Y^e \). If the decision variable \( y^e_{v_j} \) has a value of 1, the corresponding energy source is allocated to the virtual machine \( v_j \).

The energy used by the server at time \( t \) to execute the request of the virtual machine \( v_j \) is calculated from the following equation:

\[
E^S_{v_j} = \Delta P_{h_l} \cdot t_{v_j}^{old}
\]

\[
\Delta P_{h_l} = (P_{h_l}(u, t^+)-P_{h_l}(u, t))
\]

\[
\Delta P_{h_l} = (P_{\text{peak}} - P_{\text{idle}})(u_{h_l}(t^+)-u_{h_l}(t))
\]

We now calculate the second term of Eq. (5), i.e., the overhead energy cost. Note that the overhead cost varies depending on which energy source we use:

\[
c^O_{v_j} = \sum_{\varepsilon \in \{br, gr, off\}} E^O_{v_j} \cdot c^{E,E}
\]

We now calculate the overhead energy to run the virtual machine \( v_j \). Here, similar to Eq. (7), we use the energy resource mapping matrix to calculate the overhead energy:

\[
E^O_{v_j} = y^e_{v_j} \cdot E^O_{v_j}
\]

To calculate the overhead energy \( E^O_{v_j} \), we will have a procedure similar to Eq. (8):

\[
E^O_{v_j} = \Delta P^O \cdot t_{v_j}^{old}
\]

To calculate Eq. (13), we need to calculate the power overhead \( P^O \). From the definition of PUE, we can write:
Renewable energy sources have the highest priority, of the problem are as follows:

3.2.2 Constraints

By replacing Eq. (14) in Eq. (13), we obtain

\[ E^O_v = \Delta P_{hi} \cdot t_{vi}^{hdd} (PUE(u_{dj}(t), h_{dj}(t)) - 1). \]  

(b) Cost of carbon dioxide production

The cost of carbon production and the carbon tax in Eqs. (3) to (4) are formulated as follows:

\[ c^{FP} = \sum_{v_i \in V} \sum_{cE_v} c^{E_v} \cdot r_{FP} \cdot c_{lux}. \]  

Similar to what we did in Eqs. (7) and (12), we can write the following relationships using resource mapping vectors to virtual machines:

\[ c^{E_v} = y^{br}_v \cdot c^{E_v}, \quad c^{E_v} = y^{gr}_v \cdot c^{E_v}, \quad c^{E_v} = y^{off}_v \cdot c^{E_v}. \]  

We have already mentioned that the tax on carbon by renewable energy sources is zero. If the other two types of energy are not available, brown energy can be used when necessary. Based on this, Eq. (16) can be rewritten as follows:

\[ c^{FP} = \sum_{v_i \in V} c^{E_v} \cdot r_{FP} \cdot c_{lux}. \]  

3.2.2 Constraints

The constraints to be considered for the objective function of the problem are as follows:

- Renewable energy sources have the highest priority, and if green energy is limited or unavailable, data centers may use external grid energy.
- The total capacity of the processor, RAM, and storage for virtual machines running on a server \( h_i \) should not exceed its capacity. So we write

\[
\sum_{v_i \in V} \sum_{h_j \in H} cpu_{v_i} \cdot z_{v_i}^{h_j} \leq cpu_{h_j} \\
\sum_{v_i \in V} \sum_{h_j \in H} mem_{v_i} \cdot z_{v_i}^{h_j} \leq mem_{h_j} \\
\sum_{v_i \in V} \sum_{h_j \in H} str_{v_i} \cdot z_{v_i}^{h_j} \leq str_{h_j}
\]  

The decision variable \( z_{v_i}^{h_j} \) is a binary variable of the matrix \( Z_{V \times H} \). If the value of this variable is equal to 1, it means that the virtual machine \( v_i \) is located on the server \( h_j \); otherwise, it will have a value of 0.
- Every virtual machine \( v_i \) should use only one power source at a time. So,

\[ y_v^{br} + y_v^{gr} + y_v^{off} = 1, \quad \forall v_i \in V \]  

- Each element of the virtual machine mapping matrix must be non-negative:

\[ y_v^{br}, y_v^{gr}, y_v^{off} \geq 0, \quad \forall v_i \in V \]  

- The total local energy consumption to run the virtual machine on the server should not exceed the whole green energy capacity \( E^{T, gr}_{v_i} \) and total brown energy \( E^{T, br}_{v_i} \) available to the data center. So, we write

\[ \sum_{v_i \in V} \left( E^{S, gr}_{v_i} + E^{O, gr}_{v_i} \right) \leq E^{T, gr}_{d_j} \]  

\[ \sum_{v_i \in V} \left( E^{S, br}_{v_i} + E^{O, br}_{v_i} \right) \leq E^{T, br}_{d_j} \]  

- Each virtual machine must be placed exclusively on one server. So,

\[ \sum_{h_j \in H} z_{v_i}^{h_j} = 1, \quad \forall v_i \in V \]  

Finally, based on the above constraints and the objective function that we defined in Eqs. (3) and (4), the optimization problem is defined as follows:

\[ P_1 : \min_{(X,Y,Z)} f_1 : c^T \]  

s.t. Eqs. (19) to (23)
priority based on several parameters. VM requests are then sorted in descending order based on the required processor cycles. If this parameter is equal for the requests, the processor’s speed is used. If the above parameters are the same, the requests are sorted based on the memory parameter. Finally, if all of the above parameters are equal for requests, the hard drive capacity required to execute the VM request is used as the sorting parameter. If no suitable host is found in the current data center, the next will be explored. In this way, the most relevant data center and its hosts are selected.

The FFD method may require a lot of execution time for longer lists. Therefore, for large-scale problems, it is better to use evolutionary algorithms.

### 3.3 Scheduling optimization sub-problem

It is important to note that each IoT request must be executed on a VM instance. Therefore, we will use the terms “IoT request” and “VM instance” interchangeably in the following. The response time of a request $i$ sent by an IoT object consists of the sum of (a) the transmission latency to offload an object from the IoT instance $i$ to the relevant fog instance, $t_{o\text{load}}^i$, (b) the processing latency of the request $i$.

---

**Algorithm 1 - FFD**

| Input: DatacenterList, VmRequestList |
|--------------------------------------|
| Output: destinations                 |
| destinations = empty                 |
| sortedVmRequestList = sort VmRequestList based on cpu, Ram, Storage |
| for i in sortedVmRequestList.size:   |
|   vm$_i$ = sortedVmRequestList.get(i) |
|   destination = greedyAllocation(vm$_i$, DatacenterList) |
|   if (destination != null)           |
|     destinations.put (vm$_i$, destination) |
|   else                               |
|     reject vm$_i$                    |
| return destinations                 |

**Algorithm 2 - greedy_Allocation**

| Input: vm, datacenterList |
|----------------------------|
| Output: destination       |
| destinations = empty      |
| sortedVmRequestList = sort VmRequestList based on cpu, Ram, Storage |
| for j in DatacenterList.size: |
|   d$_i$ = DatacenterList.get(i) |
|   hosts$_{d_i}$ = d$_i$.getHosts( ) |
|   for k in hosts$_{d_i}$.size: |
|     h$_k$ = hosts$_{d_i}$.get(k) |
|     if (isHostFeasibleForVm(h$_k$, vm$_i$) == true) |
|       destinations.put (vm$_i$, d$_i$, h$_k$) |
|     else |
|       continue |
| end of for          |
| return null         # there is no feasible host for this VM at this time, and we have to reject this VM request |
| end of for           |
inside a fog instance $t_{ij}^{\text{proc}}$. Total response time in the physical host $h_j \in H$ for the request $v_i \in V$ is obtained as follows:

$$t_{ij}^{\text{resp}} = t_{ij}^{\text{offload}} + t_{ij}^{\text{proc}}, \quad \forall v_i \in V, \exists h_j \in H$$  \hspace{1cm} (26)

We now proceed to calculate $t_{ij}^{\text{offload}}$ in Eq. (26). We write:

$$t_{ij}^{\text{offload}} = L_{v_i} \cdot t_{\text{trans}},$$

where $t_{\text{trans}}$ denotes the transmission delay of each byte of request from an IoT device to the physical server. Note that here we have ignored the propagation delay. We now obtain $t_{ij}^{\text{proc}}$ in Eq. (26):

$$t_{ij}^{\text{proc}} = L_{v_i} \cdot \frac{\tau_{h_j}}{\theta_{h_j}},$$

where $\tau_{h_j}$ is the time duration of each instruction inside a physical server (in seconds). Also, $\theta_{h_j}$ is the size of each instruction on a physical server (in bytes). Therefore, $\tau_{h_j}/\theta_{h_j}$ denotes the time duration of each byte of the IoT request to be executed on the physical server. Multiplying this value by the request size $L_{v_i}$, gives us the processing time $t_{ij}^{\text{proc}}$ of the IoT request $v_i$ on the physical server $h_j$.

As mentioned earlier, a value of 1 for the binary decision variable $z_{h_j}^{v_i}$ indicates that a virtual machine $v_i \in V$ is placed on a physical host $h_j \in H$; otherwise, it is zero. According to Eq. (26), the response time function for an IoT request running on a virtual machine $v_i \in V$ can be defined as follows:

$$t_{v_i}^{\text{resp}} = \sum_{h_j \in H} t_{ij}^{\text{resp}} \cdot z_{h_j}^{v_i}, \quad \forall v_i \in V,$$

and the aggregate response time of all requests is as follows:

$$t_R^{\text{resp}} = \sum_{v_i \in V} t_{v_i}^{\text{resp}}.$$  \hspace{1cm} (30)

The response time of each IoT request running on a virtual machine $v_i \in V$ should not exceed its deadline. That is,

$$t_{v_i}^{\text{resp}} \leq \text{deadline}_v, \quad \forall v_i \in V$$

The sub-problem of scheduling optimization is defined as follows:

$$\text{P}_2: \min_{(Z)} f_R^{\text{proc}}, \quad \text{s.t. } \text{Eqs. (19), (23), and (31)}$$

Our goal is to minimize the overall response time of the system, taking into account the deadline of each task.

### 3.4 Joint cost and scheduling optimization problem

As mentioned earlier, our goal is to find the best candidates from the physical host nodes to place any IoT requests to. The placement operation must be performed to lead to a trade-off between response time and energy cost in the system.

Finally, the optimization problem is formulated as follows:

$$\text{P}_3: \min_{(X,Y,Z)} \left\{ f_1: c^T & f_2: t^{R} \right\}$$

s.t. \text{Eqs. (19) – (23), and (31)}

The above multi-objective problem can be converted to a single-objective form as follows (Jafari and Rezvani 2021):

$$\text{P}_4: \min_{(X,Y,Z)} F = w_1 \cdot \text{Ln}(f_1) + w_2 \cdot \text{Ln}(f_2)$$

s.t. \text{Eqs. (19) – (23), and (31)}

$$w_1 + w_2 = 1$$

$$0 \leq w_1 \leq 1$$

$$0 \leq w_2 \leq 1$$

The use of the convex function of the logarithm in Eq. (36) guarantees the existence of the optimal solution (Parviz and Rezvani 2020). Unfortunately, as mentioned earlier, both sub-problems of task scheduling and energy efficiency fall into NP-hard problems. Therefore, deterministic methods fail to obtain a solution. In the next section, we solve this problem using a metaheuristic method.

In this research, we solve the above problem for the first time using two meta-heuristic methods of Genetic Algorithm (GA) and Memetic Algorithm (MA).

### 4 Proposed metaheuristic methods

We consider the method presented in Khosravi et al. (2017) as the baseline method to solve the sub-problem of Eq. (24). We call this method “Cost and Renewable Energy-Aware Dynamic PUE” or in abbreviation CRA-DP. Then, we solve the joint cost and scheduling optimization problem and propose two metaheuristic algorithms to overcome the NP-hardness and time complexity of the problem.

#### 4.1 Genetic algorithm

In this section, we present a genetic-based approach to the VMP problem of Eq. (24). We call this method “Cost and
Renewable Energy-Aware Dynamic PUE with Genetic Algorithm'' or in abbreviation CRA-DP-GA. In this method, there is a population that consists of several chromosomes. Each chromosome is in the form of an array. The number of cells in this array is equal to the number of VMs, \( N_v \). The value stored inside each cell is the identifier of a physical machine. Each cell of the chromosome is called a gene. Figure 1 shows a possible solution to the problem of Eq. (24).

To better understand the structure of Fig. 1, let us give an example. If we place six VMs on four physical machines, the chromosomes may take any of the following forms:

\[
\{1, 1, 3, 2, 4, 1\} \quad \{3, 1, 2, 2, 4, 4\} \quad \{2, 4, 2, 2, 3, 1\} \quad \ldots
\]

The pseudo-code of the gene and the CRA-DP-GA are presented in Algorithm 3 and Algorithm 4, respectively.

### Algorithm 3- newGene

**Input:** vmRequest
**Output:** gene

1. init empty gene
2. gene.setHost(null)
3. gene.setDatacenter(null)
4. gene.setVm(vmRequest)
5. return gene

### Algorithm 4- Genetic Algorithm(CRA_DP_GA)

**Input:** DatacenterList, HostList, VmRequestList, PopulationSize, \( P_{crossover} \), \( P_{mutation} \)
**Output:** FittestChromosome

```python
p_0 = initialPopulation(DatacenterList, HostList, VmRequestList, PopulationSize);
current = p_0
while(convergence criterion not reach)
    current = fitnessFunction(current);
    current = generateNextGeneration(current, \( P_{crossover} \), \( P_{mutation} \), PopulationSize);
end
return current.getFittestChromosome();
```

The CRA-DP-GA flowchart is shown in Fig. 2. After collecting the user needs in the form of VM requests, they are placed on physical machines using a greedy algorithm, and the initial population is created. Then, a new population is created by applying the crossover operator. This operator selects several chromosomes from among the chromosomes in a population to reproduce. More graceful chromosomes are more likely to be chosen. During crossover, parts of chromosomes are randomly exchanged. This makes the children not precisely like their parents. Now, the mutation operator is applied to the chromosomes. This operator randomly selects a gene from a chromosome and then changes its content. This action causes the search scope to be extended to untouched and newer spaces. It can be concluded that the most crucial benefit of the mutation operator is to avoid getting stuck in the local optimal points. Calculating the fitness of each chromosome is the end of a loop in a genetic algorithm called a “generation.” This cycle continues until the chromosome’s fitness does not change or the maximum number of iterations is reached. At this point, the best solution is reported.

4.1.1 Initial population

The method of selecting the initial population may significantly affect the speed of convergence (Esfandiari and Rezvani 2020). The random method is used if no information is available (Rashida et al. 2019). Accordingly, we use a random way to generate the initial population in this study. A pseudo-code for the initial population is provided in Algorithm 5.
4.1.2 Selection operator

The selection process causes the chromosomes to have a better fit. For the selection operator, different methods are used, such as the roulette wheel or the binary tournament. In the roulette wheel method, the whole population is divided into sections representing an individual. The probability of choosing an individual for the next generation is determined by the ratio of the individual’s fitness value to the total fitness values of the entire population. In contrast, the individual with the highest fitness value is selected in the binary tournament method for the next generation. Here, there are no arithmetic calculations based on the value of proportionality. The selection of an individual for the next generation is based solely on several comparisons based on fitness value. Usually, the speed of convergence of the tournament strategy is much faster than the roulette wheel. Algorithm 6 provides the pseudo-code for selecting individuals based on tournament strategy. First, a subset of the attributes of a population is determined. Then, members of the population compete with each other, and ultimately only one attribute from each subgroup is selected. The tournament method provides fair conditions for selecting all solutions and thus preserves diversity.

Algorithm 5- initialPopulation

| Input: DatacenterList, HostList, VmRequestList, PopulationSize |
| Output: initialPopulation |
| initialPopulation = empty; |
| While (initialPopulation.size < PopulationSize) |
| newChromosome = empty #new chromosome itself is a list of the gene; size is VmRequestListSize |
| for all vmRequest in VmRequestList: |
| newChromosome.add(newGenom(vmRequest)); |
| end of for |
| for i in newChromosome.size: |
| gene_i = newChromosome.getGene(i); |
| d_random = select random datacenter from DatacenterList |
| h_random = select random host from d_random |
| gene_i.setDatacenter(d_random); |
| gene_i.setHost(h_random); |
| end of for |
| if validateChromosome(newChromosome) == true: |
| add newChromosome to initialPopulation |
| end of while |
| return initialPopulation |

4.1.2.1 Algorithm 6

∀i ∈ {1, ..., N_D * N_H}

| pm(i) | pm(i) | pm(i) | ... | pm(i) | pm(i) |
|-------|-------|-------|-----|-------|-------|
| vm(1) | vm(2) | vm(3) | ... | vm(N_Y - 1) | vm(N_Y) |

Fig. 1 The structure of each chromosome in the CRA-DP-GA
Before definitively producing each chromosome, we check the limitations in Eqs. (19) to (20). Pseudo-codes of chromosome validation are provided in Algorithms 7 and 8.

### Algorithm 6 - tournamentSelection

**Input:** population  
**Output:** pair  

```
tournamentPopulation = empty  
while (pair.size < 2)  
    select 2 random individual and add them to tournament  
    Select the best individual from the tournament  
    add best individual to pair  
end  
return pair
```

4.1.3 Cross-over operator

During the cross-over process, parts of the parent chromosomes are randomly swapped. This allows each offspring (child) to inherit a combination of its parents’

### Algorithm 7 - validateChromosome

**Input:** Chromosome  
**Output:** boolean  

```
for i in chromosome.size;  
    gene_i = Chromosome.getGene(i);  
    h_i = gene_i.getHost();  
    vm_i = gene_i.getVm  
    feasible = isHostFeasibleForVm(h_i, vm_i)  
    if h_i is not busy and feasible == true  
        return true  
    else  
        return false  
end of for
```

### Algorithm 8 - isHostFeasibleForVm

**Input:** vm, host  
**Output:** boolean  

```
if (vm.getCpu <= host.getCpu && vm.getRam <= host.getRam && vm.getStorage <= host.getStorage) && \( v_{\text{resp}} \leq t_{\text{deadline}} \)  
    return true  
else  
    return false
```
attributes. The chromosomes selected for the cross-over operation and the resultant chromosomes are called “parent” and “child,” respectively. According to Fig. 3, based on experimental observations, we considered the appropriate cross-over rate to be 0.7. In this study, we use a two-point method for the cross-over operation. This operator randomly selects two points from the parent chromosomes and swaps the genes of those points. The pseudo-code of the cross-over operation is shown in Algorithm 9.

---

Algorithm 9 - crossover
---

**Input:** pair  
**Output:** offspring chromosomes  
`generate two random positions`  
`swap data beyond the two points between chromosome in pair`  
`return pair`

---
4.1.4 Mutation operator

A mutation operator is used to increase genetic diversity in a population and, consequently, to increase the chances of achieving an optimal solution in a shorter time. This operator also causes the algorithm to avoid getting stuck in the optimal local solutions. This operator modifies one or more genes in a chromosome with a small mutation probability. Like the crossover operator, we assume a mutation probability of 0.1 based on the experimental observations shown in Fig. 3. Selecting this value resulted in a more stable response than other values. As shown in Fig. 3, setting this value also causes fewer fluctuations in the cost function. The pseudo-code of the mutation routine is demonstrated in Algorithms 10 and 11. As shown in the pseudo-code of Algorithm 12., the crossover operation is performed with a probability of $P_{\text{crossover}}$ and the mutation operation is performed with a probability of $P_{\text{mutation}}$, and ultimately the population size reaches $\text{populationSize}$.

```
Algorithm 10. mutation
Input: pair
Output: modified pair
while (true)
    chromosome_1 = pair.getFirstChromosome
    chromosome_2 = pair.getSecondChromosome
    gene_random_1 = randomly select a gene from chromosome_1
    gene_random_2 = randomly select a gene from chromosome_2
    d_random_1 = randomly select a datacenter from datacenterList
    d_random_2 = randomly select a datacenter from datacenterList
    h_random_1 = randomly select a host from d_random_1
    h_random_2 = randomly select a host from d_random_2
    if (gene_1.getDatacenter! = d_random_1 && gene_1.getHost!
    = h_random_1) && (gene_2.getDatacenter! = d_random_2 && gene_2.getHost!
    = h_random_2)
        gene_random_1.setDatacenter(d_random_1)
        gene_random_2.setDatacenter(d_random_2)
        gene_random_1.setHost(h_random_1)
        gene_random_2.setHost(h_random_2)
    return pair
end of while
```

```
Algorithm 11. pairFeasibility
Input: pair
Output: boolean
feasibility = false
chromosome_1 = pair.getFirstChromosome
chromosome_2 = pair.getSecondChromosome
if (validateChromosome(chromosome_1) == true && validateChromosome(chromosome_2) ==
    true)
    return true
else
    return false
```
4.1.5 The fitness function

In the GA, the idea is to remove the N-worst possible solutions (chromosomes) and leave the most appropriate ones. To measure the quality of solutions, we use a function called the fitness function. The larger the value of this function, the higher the quality of the solution. The fitness function was previously shown in Eq. (24), in which our goal is to minimize the cost. The pseudo-code for calculating chromosome fitness is shown in Algorithms 13 and 14.

---

**Algorithm 12. generateNextGeneration**

Input: currentGeneration, \( P_{crossover}, P_{mutation}, PopulatioSize \)

Output: nextGeneration

nextGeneration = empty

random = generate random number between [0,1]

while(nextGeneration < populationSize)

    pair = tournamentSelection(current)

    if (random < \( P_{crossover} \)) then

        pair = crossover(pair)

    end

    if (random < \( P_{mutation} \)) then

        pair = mutation(pair)

        if (pairFeasibility(pair) == true) then

            add pair to nextGeneration

        end

end

return nextGeneration

---

Fig. 3 Impact of cross-over rate and mutation rate on overall costs
As shown in Algorithm 14, the server utilization at an upcoming round $t^+$ is the sum of server utilization and required CPU. We have already obtained server power consumption in Eq. (2) as a function of CPU utilization. Also, according to Eq. (5), the total consumed energy is equal to the total energy consumed by the server and the energy consumption overhead. The PUE metric mentioned in Eq. (1) is used to calculate the overhead. In the final rows of Algorithm 14, the amount of carbon dioxide is checked. Then, the amount of costs imposed on the system is calculated according to the type and the amount of consumed energy and tax costs. If the amount of solar energy is less than the required energy of the VM, the server will use all available solar energy and consumes the rest of its energy needs from an offsite energy source. As stated in Eqs. (3) to (4), the energy cost of each data center
includes two parts: the cost of consumed energy and the cost that is incurred due to carbon production. The total cost of the data center is obtained by adding the total amount of consumed energy, operational costs, the amount of carbon dioxide, and the cost of the tax. For each chromosome, this value is added to the fitness value. The above calculations are performed for all genes.

4.2 Memetic algorithm

4.2.1 Main algorithm

The GA requires a lot of time to process different generations to achieve the desired result (Rashida et al. 2019). We present a GA-inspired algorithm called the Memetic Algorithm (MA) to overcome this shortcoming. We call this method “Cost and Renewable Energy-Aware Dynamic PUE with Memetic Algorithm” or in abbreviation CRA-DP-MA. The structure of each possible solution is similar to what was shown previously in Fig. 1. Unlike the GA, each chromosome in the MA improves its fitness by using a local search method, such as Hill Climbing (Russell and Norvig 2002). The MA flowchart is similar to what we showed earlier in Fig. 2, except that in the final stage, each chromosome (a possible solution) finds its neighbor solutions through minor modifications in its current state. It then replaces the best neighbor chromosome with the worst solution in the population (Rashida et al. 2019). Our goal is to minimize the cost function in Eq. (24). Since the MA’s selection, crossover, and mutation routines are the same as the GA, we do not show them again. Also, we consider the climbing rate to be 0.8 based on the experiments shown in Fig. 4. The pseudo-code of the CRA-DP-MA is shown in Algorithm 15, and its flowchart is depicted in Fig. 5. Note that the probability of crossover, mutation, and local search is denoted by \( P_{\text{crossover}} \), \( P_{\text{mutation}} \), and \( P_{\text{localSearch}} \), respectively.

**Algorithm 15 - The Proposed Memetic Algorithm (CRA-DP-MA)**

**Input:** List of Host and vms
**Output:** Best solution

\[ p_0 = \text{initialPopulation(DatacenterList, HostList, VmRequestList, PopulationSize)}; \]

**while** (convergence standard not reach)

Evaluate the fitness of population \( p_0 \);
Select two individuals from \( p_0 \) as parents by applying 2 - tournament selection operator;
\( \text{pair} = \text{tournamentSelection(current)} \)
Generate two offspring by applying the crossover operator to parents with probability \( P_{\text{crossover}} \);
Select one individual from \( p_0 \) as \( X \) by applying 2 - tournament selection operator;
Generate new solution by applying the mutation operator to \( X \), respectively, with probability \( P_{\text{mutation}} \);
Select one individual, \( Y \), from \( p_0 \) by applying 2 - tournament selection operator;
Improve \( Y \) via local search (the proposed hill climbing method) with probability \( P_{\text{localSearch}} \);
Replace new solutions into \( p_0 \) and in turn remove the worst individuals from \( p_0 \);
Update the best solution

**End While**

**End**

4.2.2 Local search

As explained in Sect. 4.2.1, the significant difference between the MA and the GA is local search, for example, a hill-climbing method. As shown in Fig. 6, hill-climbing is
an iterative technique that starts with a candidate solution and then moves on neighbors to achieve a better solution (Russell and Norvig 2002). If this movement leads to a better solution, another move will be made on this new solution. This process continues until further improvement in the solution space is not possible. It can be said that the hill-climbing method tries to minimize the objective function of Eq. (24), namely \( c_{\text{total}} \). In each iteration, it selects an element of the state space, i.e., vector \( Z_{N_V \times N_H} \), and examines whether this change improves the value of the objective function. This process continues until no change leads to an improvement in the value of \( c_{\text{total}} \). At this point, the vector \( Z_{N_V \times N_H} \) will be the "local minimum" and the algorithm’s output. The most significant reasons for terminating the algorithm are the existence of a local minimum, flat, and ridges. Since each candidate solution can have multiple neighbors, the heuristic by which neighbors are selected is crucial.

We propose Host-based Hill-climbing with the pseudocode shown in Algorithm 16. The only factor that limits the performance of the hill-climbing algorithm is the number of neighbors. If the number of neighbors is large, the cost of this method may be very high (Rashida et al. 2019). To solve this problem, we have defined a method called Host-based Neighborhood (HbN) in Algorithm 17. In Algorithm 16, for each gene, we call Algorithm 17. Finally, Algorithm 16 generates all possible neighbors with the FFD algorithm without placement. Simply speaking, in finding neighbors, if the host for a gene is already selected, there is no chance of re-election. Eventually, the best neighboring chromosome is replaced by the current chromosome.

### Algorithm 16- Host_based_Hill_Climbing

**Input:** pair

**Output:** modifiedPair

```
chromosome = randomlySelectChild(pair)
allPossibleHost = emptyListOfHosts
for (i = 0; i < chromosome.length; + + i)
    currentGene = chromosome.getGene(i)
    allPossibleHosts.set(i) = Host_based_Neighborhood(currentGene.getVm( ))
end
allNeighbours = generate All Possible neighbour for chromosome whit FFD algorithm without replacement
bestChromosome = hill_Climbing(chromosome,Top(allNeighbours))
modifiedPair = replace bestChromosome with chromosome in pair
return modifiedPair
```

### Algorithm 17- Host_based_Neighborhood

**Input:** vm

**Output:** list of hosts

```
resultList = empty
datacenterId = vm.getHost().getDatacenter().getId()
resultList.addAll(getAllHosts(dataCenterId))
utilization = vm.getHost().getUtilization()
for (i = 0; i < resultList.length; + + i)
    if not (utilization * 0.7 \leq resultList.get(i).getUtilization \leq utilization * 1.3)
        resultList.removeHostAtIndex(i)
end
return resultList
```
Fig. 4 Amount of objective function improvement in the CRA-DP-MA against the CRA-DP-GA for different values of climbing rates.

Fig. 5 Flowchart of the CRA-DP-MA.
We now proceed to describe the decisions made at each step of Algorithm 17.

Step 1: As shown in Fig. 7, the algorithm must first find all available hosts in the same data center for each gene. For example, if gene #1 on chromosome #X is located inside datacenter #4, we need to inspect all the hosts inside datacenter #4. We check only the hosts in the current data center because the locality of the solution must be met. Using the hosts inside other data centers leads to diversity in temperature conditions and, consequently, the availability of renewable energy.

Step 2: Once the permissible hosts for each gene have been found, those with the same utilization should be selected as the current host. Based on the experimental results shown in Fig. 8, we set the threshold for utilization similarity to be 30%. Based on this, a host with at least 30% similarity with the current host is selected. Otherwise, the host will be removed from the neighbors’ list. This decision balances the load and prevents allocations from being biased towards empty or overloaded hosts.

Step 3: In this step, for each gene, several candidate hosts are obtained, each of which can identify the candidate neighbors. We then randomly select a maximum of 10 different neighbors using the FFD algorithm without replacement. Experimental results show that the optimal value for the number of neighbors in a local search routine is 10. If we consider the number of neighbors more than 10, the algorithm’s efficiency decreases. Conversely, if we set the number of neighbors to more than 10, there is no significant improvement in the solution.

Step 4: In this step, the local search algorithm is applied to 10 neighbors obtained from the previous step, and the best solution is selected.

4.3 CRA-DPSCH-DDMPEA method

We compare our proposed method with one of the state-of-the-art methods by Chauhan et al. (2021c). The method is called the diversity-driven multi-parent evolutionary algorithm (DDMPEA) with adaptive non-uniform mutation (ANUM). In the diagrams and algorithms, the implementation of the joint cost and scheduling optimization problem (Eq. (36)) using this method is abbreviated as CRA-DPSCH-DDMPEA. For a fair comparison with state-of-the-art techniques, we will use the DDMPEA later in Sect. 5 for the sub-problem of Eq. (24). In the CRA-DPSCH-DDMPEA, non-uniform mutations are used to maintain diversity in solutions (Chauhan et al. 2021d). Here, fitness variance is used to detect premature convergence of the population to local optima. Also, more than two parents are used for cross-over operations. After selecting several parents and cross-over, a non-uniform adaptive mutation occurs, resulting in a reduction in the variance of the candidate solutions. Hence, the algorithm is driven according to population diversity and does not get stuck in the local optima. The steps of CRA-DPSCH-DDMPEA are described in detail as follows:

4.3.1 Population initialization

Initially, $N_P$ members are randomly generated according to the lower and upper bounds of the search space using a uniform distribution as follows:

$$x_{ij} = x^{\min}_j + \frac{r_j}{D}(x^{\max}_j - x^{\min}_j), \quad i = 1, 2, ..., N_P; \quad j = 1, 2, ..., D, \quad (41)$$

where $r_j$ is a random number with uniform distribution for member $i$ and dimension $j$. Here, $N_P$ is the population size and $D$ is the search space dimension. Also, $x^{\min}_j$ and $x^{\max}_j$ are the minimum and maximum for the variable.

4.3.2 Multi-parent cross-over

Unlike GA, here, three parents are used for cross-over. For this purpose, parents are first selected from the best $p\%$ of the total population. After selecting three random members $x_{r1}, x_{r2},$ and $x_{r3}$ from this set, an offspring $O'_{ij}$ is created with the following formula:

$$O'_{ij} = x^{r1}_{r1,j} + x^{r2}_{r2,j} - x^{r3}_{r3,j} + \frac{1}{3} \cdot N_P \cdot \frac{P}{100}; \quad j = 1, 2, ..., D, \quad (42)$$

where $x_j$ is a random weight that follows a normal distribution with a mean of 0.7 and a variance of 0.1. Also, $t$ represents the number of generations produced so far.
This scheme leverages the Differential Evolution (DE) capabilities (Jafari and Rezvani 2021).

The control operation is now performed on the generated offspring to ensure that it is within the allowable range:

\[ O_{ij}^t = \begin{cases} O_{ij}^t + \frac{x_{ij}^{\text{min}}}{2}, & \text{if } O_{ij}^t < x_{ij}^{\text{min}} \\ O_{ij}^t + \frac{x_{ij}^{\text{max}}}{2}, & \text{if } O_{ij}^t \geq x_{ij}^{\text{max}} \end{cases} \]  

(43)

Now, the fitness of the created offspring is assessed by placing the answer in Eq. (43).

### 4.3.3 Adaptive mutation strategy

As is common in evolutionary computing, mutations maintain diversity from one generation to the following (Besharati et al. 2021). If the algorithm gets stuck in a local optimum, then the variance of the population fitness will become zero. To cope with this situation and also to create faster convergence, we use an adaptive non-uniform mutation operator as follows:

\[ \text{Om}_{ij} = O_{ij}^t (1 + 0.5\eta), \quad i = 1, 2, ..., \frac{1}{3}NP \cdot \frac{p}{100}; \quad j = 1, 2, ..., D, \]  

(44)

where \( \eta \) denotes the weight coefficient obtained from a normal distribution with a mean 0 and a variance of 1. The mutation occurs when \( r_i(0, 1) < p_m \), where \( r_i(0, 1) \in U(0, 1) \) is a uniform random number in the interval (0, 1) for the \( i \)-th offspring. Also, \( p_m \) is the probability of mutation that results from the following equation:

\[ p_m = \begin{cases} e^{-h}, & \text{if } \sigma^2 < \sigma_1 \\ 0, & \text{if } \sigma^2 \geq \sigma_1 \end{cases}, \]  

(45)

where \( \sigma_i \) is the variance threshold, which is calculated from the range of variables as follows:

\[ \sigma_i = \frac{1}{100} (x_{ij}^{\text{max}} - x_{ij}^{\text{min}}), \quad j = 1, 2, ..., D. \]  

(46)

Also, \( \sigma^2 \) denotes the variance of population fitness. Here, the degree of diversity in the iteration \( t \) is obtained as follows:

\[ h = \max_{1 \leq i \leq NP} \left\{ \max_{1 \leq j \leq D} \left( x_{ij}^{t} - x_{ij}^{\text{best2}} \right) \right\}, \]  

(47)

where \( x_{ij}^{\text{best2}} \) is the best fitness value position in the iteration \( t \). When the value of \( h \) is high, individuals become more scattered in the search space, so a small amount of mutation is needed. Conversely, when the value of \( h \) is small, individuals become more crowded in the search space, and therefore, a large amount of mutation is required. Roughly speaking, when the CRA-DPSCH-DDMPEA gets stuck in local minima, it tries to escape this undesired situation by leveraging the fitness variance. This parameter is calculated as follows:

\[ \sigma^2 = \sum_{i=1}^{NP} \left( \frac{f_i - \overline{f}}{f} \right)^2, \]  

(48)

where \( f_i \) is the fitness of \( i \)-th individual and \( \overline{f} \) is the mean of the fitness values. Also, \( f \) is the returning factor used to control the individuals’ fitness variance and is calculated as follows:

\[ f = \begin{cases} \max \left| f_i - \overline{f} \right|, & \text{if } \left| f_i - \overline{f} \right| < 1 \\ 1, & \text{otherwise} \end{cases} \]  

(49)

The fitness variance \( \sigma^2 \) indicates population density. The smaller it is, the closer individuals are assembled. Large amounts of fitness variance indicate that individuals are randomly dispersed. Once the mutation is done, the new individual is added to the population. The population is then ranked according to the fitness value of the individuals. Finally, \( NP \) best individuals are selected and create the next-generation population. Before proceeding, it should be noted that researchers have used various methods to achieve diversity. Interested readers can refer to Chauhan et al. 2021e for further study.

As previously stated in Eqs. (36), to (40) the optimization problem in this study has both inequality constraints (denoted by \( g_i(X) \)) and equality constraints (denoted by \( h_j(X) \)). Let us denote the number of inequality and total constraints by \( G \) and \( M \), respectively. Thus, they can be shown as follows:

\[ g_i(X) \leq 0, \quad i = 1, ..., G \]  

(50)

\[ h_j(X) = 0, \quad j = G + 1, ..., M \]  

(51)

For the sake of convenience, we can combine the above two constraints and rewrite them as follows:

\[ G_i(X) = \begin{cases} \max \{0, g_i(X)\}, & i = 1, ..., G \\ \max \{0, |h_i(X)| - \delta\}, & i = G + 1, ..., M \end{cases} \]  

(52)

To obtain the total constraint violation \( V(X) \) for an infeasible solution, we calculate the weighted average of all constraints as follows:

\[ V(X) = \sum_{i=1}^{M} G_i(X) \]  

(53)

Decision theory manages constraints in the optimization problem (Naghdehforoushha et al. 2022). Here, there are many methods, one of the most common of which is fuzzy
decision-making. In this method, the input variables are fuzzified into a membership function $\mu$. Each input variable $X$ is assigned a membership value in the range $[0,1]$. A value of $\mu = 0$ for an input variable indicates that it is deterministic, not fuzzy. Simply speaking, the larger the membership value $\mu$ for an input variable $X$, the fuzzier it is. The fuzzy membership function $\mu$ is plotted in Fig. 9 and is expressed as follows:

$$
\mu(F) = \begin{cases} 
1, & \text{if } F_i \leq F_{i}^\text{min} \\
\frac{F_i^\text{max} - F_i}{F_{i}^\text{max} - F_{i}^\text{min}}, & \text{if } F_{i}^\text{min} < F_i < F_{i}^\text{max} (i = 1, 2) \\
0, & \text{if } F_i \geq F_{i}^\text{max}
\end{cases}
$$

(54)

It is important to note that here we have two objective functions $F_1$, and $F_2$. The objective function $F_1$ is the same as our principal objective, as stated in Eq. (36). The objective function $F_2$, imposed by fuzzy decision-making, indicates the degree of constraint violation. The value of the membership function indicates the level of satisfaction of the objective function $F_i$. Let us denote the number of all non-dominated solutions by $K$. We now rank the achievement of a non-dominated solution $k$ concerning all solutions. To do this, we must first divide the achievement of this solution by the sum of the achievements of all non-dominated solutions. By performing such normalization, the function $\mu_D^k$ can be calculated as a membership function for non-dominated solutions as follows:

$$
\mu_D^k = \frac{\sum_{i=1}^{M} \mu(F_i)}{\sum_{k=1}^{K} \sum_{i=1}^{M} \mu(F_i)}
$$

(55)

The best solution $O_{\text{best}}$ is an offspring that gets the maximum membership score $\mu_D^k$. It is obtained as follows:

$$
O_{\text{best}} = \max\{\mu_D^k : k = 1, 2\}
$$

(56)

The pseudo-code of the CRA-DPSCH-DMPEA is shown in Algorithm 18.
**Algorithm 18- Pseudo-code of the CRA-DPSCH-DDMPEA method**

**Input:** number of individuals $N_p$, the search space dimension $D$, principal objective function $F_i$,
inequality constraints $g_i(X)$, equality constraints $h_i(X)$, a ratio for accepted solutions $p$

**Output:** The best solution $O_{best}$

Create the initial population containing individuals like $x_{ij}$ using Eq. (41).
Evaluate the objective function $F_i(x_{ij})$
Evaluates the constraint violation $F_2 = V(X)$ using Eq. (53).
Evaluate the fuzzy membership function $\mu(F_i)$ using Eq. (54).
Evaluate normalized membership function $\mu_D^{k}$ using Eq. (55).
Identify the best solution $O_{best}$ using Eq. (56).

for $t = 1 \rightarrow T$

Evaluate the objective function $F_i(x_{ij})$
Evaluate the constraint violation $F_2 = V(X)$ using Eq. (53).
Evaluate the fuzzy membership function $\mu(F_i)$ using Eq. (54).
Evaluate normalized membership function $\mu_D^{k}$ using Eq. (55).
Sort the whole population based on $\mu_D^{k}$ values and select the best $p\%$.

for $i = 1 \rightarrow (1/3)N_p.(p/100)$

Select three random members $x_{r1}$, $x_{r2}$, and $x_{r3}$
Select the random weight $\alpha_{ij}$ using a normal distribution $N(0.7, \sqrt{0.1})$.
Create an offspring $O_{ij}^{t}$ using the cross-over operation of Eq. (42).
Perform the control operation on the new offspring $O_{ij}^{t}$ using Eq. (43).

end for

Evaluate the objective function $F_i(O_{ij}^{t})$
Evaluate the constraint violation $F_2(O_{ij}^{t}) = V(O_{ij}^{t})$ using Eq. (53).
Calculate the fitness variance of the population using Eq. (48).
Calculate the degree of diversity, $h_i$, using Eq. (47).
Calculate the probability of mutation $p_m$ using Eq. (45).

if $r_i (0,1) < p_m$ then

Create $O_{ij}^{t}$ by performing mutation on $O_{ij}^{t}$ using Eq. (44).

end if

Evaluate the objective function $F_i(O_{ij}^{t})$
Evaluate the constraint violation $F_2(O_{ij}^{t}) = V(O_{ij}^{t})$ using Eq. (53).
Add $O_{ij}^{t}$ and $O_{ij}^{t}$ to the total population
Sort the whole population based on $\mu_D^{k}$ values and select the best $N_p$ individuals for the next-generati
Update the best solution $O_{best}$ using Eq. (56).

end for

return the best solution $O_{best}$
5 Performance evaluation

In this section, we first describe the settings used in the simulation. Then, we compare the performance of the modified MA and the GA with the baseline and state-of-the-art methods.

5.1 Experimental setting

We used the CloudSim tool (version 3.03) to simulate the proposed method (Calheiros et al. 2011). The CloudSim was developed at the University of Melbourne, Australia. Also, parts of the analysis of results were done with SPSS software. The experiments were done on a 64-bit Intel® Core™ i5-8269U Processor with 6 MB Cache, 4 Cores, 4.20 GHz CPU frequency, and 8 GB RAM.

To locate data centers, we have selected four cities in the United States, with different time zones and the same network infrastructure: Hampton (in Virginia state), Houston (in Texas state), Salem (in Oregon state), and San Francisco (in California state). The number of hops for each packet to reach the destination is estimated between 12 to 14. So, the side effects of different distances in the communication network can be ignored (Khosravi et al. 2017). Each data center has 130 physical machines that are configured in 5 different types, according to Table 4.

5.1.1 Solar energy

We used data reported from the PVGIS-NSRDB data center on the European Commission portal (Commission 2014) concerning the solar energy of the four cities mentioned above. These data result from a joint venture between PVGIS-NSRDB and the National Solar Radiation Database (NREL). In this study, each PhotoVoltaic (PV) solar panel, obtained from solar cell assembly, has modules installed in a fixed position and does not change during the day/year. Using this data, we have the amount of solar energy per day in terms of W/m². Also, we have considered the area of each flat solar absorber plate equal to 2684 m². This number is based on the configuration used in Khosravi et al. (2017). Therefore, by having the energy produced for each solar panel, we can calculate the total power of a solar power plant consisting of 2648 solar panels. The solar energy specifications are shown in Fig. 10.

5.1.2 Outdoor temperature

We used the European Commission portal data to obtain the ambient temperature per hour (Amazon https://aws.amazon.com/ec2/instance-types/). For this purpose, ambient temperature data regarding the above four cities have been collected from May 20, 2014, to May 30, 2014. Figure 11 shows these data.

5.1.3 PUE and server power consumption

We use Eq. (1), described earlier, to calculate the PUE. Also, it was previously stated in Eq. (2) that power has a linear relationship with CPU utilization (Gao et al. 2013). We consider the values of $P_{idle}$ and $P_{peak}$ to be 162 and 215 watts, respectively. Therefore, Eq. (2) can be simplified as follows:

$$P_{hl}(u, t) = 162 + 53u_{hl}(t)$$  \hspace{1cm} (57)

5.1.4 Carbon dioxide production rate and carbon tax

The rate of carbon dioxide emission in each data center is obtained from the website of the US Department of Energy (Administration https://www.eia.gov/electricity/state/archive/2014/) in Tons/MWh. We have also calculated the tax on carbon dioxide emissions, which is applied in some countries (Khosravi et al. 2017). The carbon dioxide tax is calculated in dollars per tonne (Dollars/Tonne). These values are shown in Table 5.

5.1.5 Energy cost

The cost of consumed electricity is obtained from the US Energy Information Administration (EIA) (Workload https://www.cs.huji.ac.il/labs/parallel/workload/) concerning the four cities. The information is shown in Table 5. We estimate the cost of solar energy to be zero. In this regard, note that installation and maintenance costs are incurred only once and are not a function of the consumed solar energy. Thus, even if more solar energy is consumed, it will not impose more costs on a cloud data center.

Fig. 9 Fuzzy membership function in this research (Chauhan et al. 2021c)
5.1.6 Workload data

The workloads produced by users arrive at VMs. We select VMs’ specifications based on what Amazon EC2 offers (Amazon https://aws.amazon.com/ec2/instance-types/). The characteristics of the applications are shown in Table 6. The workload is generated using the Lublin-Feitelson model (workload, p. xxxx), including two types: the bag of the task (in which there is no interaction with the user) and the web request. These two types of workloads have different distributions. The only difference between the two groups is holding time. Generally, web requests take longer than the bag of task requests. By modifying a few parameters from workload, p. xxxx, we created a proportional distribution of requests to generate two different user requests. According to the approach (Khosravi et al. 2017) to make the bag of task requests, we changed the first parameter of the gamma distribution to 20.4. In this way, we generate requests with a longer holding time. We also changed the holding time distribution to Hypergamma (with an average of 73 and a variance of 165) to generate web requests.

In this research, 30 different workloads are produced, each of which is tested separately. In this way, we can make more confidence in the accuracy of the test results.

5.2 Execution time of algorithms

For the sake of fairness, we compare our proposed method with one of the state-of-the-art methods called DDMPEA. The details of this method have already been described in Sect. 4.3. We first use this method to solve the cost optimization sub-problem of Eq. (24) and call it CRA-DP-DDMPEA. Then, we use it to solve the joint cost and scheduling optimization of Eq. (36) and name it CRA-DPSCH-DDMPEA. So far, none of the previous studies have addressed the issue of joint optimization. Similarly, we abbreviate the solution of the cost optimization sub-problem by the MA with the symbol CRA-DP-MA. Also, solving the problem of joint cost and scheduling optimization is abbreviated with CRA-DPSCH-MA.

We now proceed to analyze the execution time of all algorithms. Roughly speaking, we want to examine the extent to which different metaheuristic methods could make a significant difference with the baseline algorithm CRA-DP.

One of the prerequisites for most statistical analyzes concerning random variables is the normality of the data. As a result, checking the normality of the data is a crucial point, especially when the distribution is not transparent and the sample size is small. One of the criteria for data normality is the Quantile-Quantile (Q-Q) plot. There are many statistical tests such as Shapiro–Wilk, Dagostino, Anderson–Darling, and Kolmogorov–Smirnov that can check the normality of the data. In this study, we use the Shapiro–Wilk and Kolmogorov–Smirnov methods to inspect the hypothesis of normality of the data (H0). Because we assume a 95% confidence interval, the significance level is 0.05. Assumption H0 is rejected if the value of the sig (significance level) is less than 0.05. Table 7 shows the result of the normality test concerning the execution time of algorithms. Since the Sig value in both Shapiro–Wilk and Kolmogorov–Smirnov tests for all VMP algorithms is less than 0.05, the hypothesis H0 is rejected. In other words, the data obtained from the execution time of the algorithms are not normal. There are two ways to continue: (1) use non-parametric tests to compare the results, (2) try to normalize the data to use parametric methods. Note that parametric methods are more accurate than non-parametric ones. So, we choose the second way. Due to space limitations, we do not provide details on data normalization. Table 8 shows the execution time after normalization.

The Q-Q plot for the execution time of algorithms is shown in Figs. 12, 13, and 14. As mentioned earlier, the Q-Q plot determines how close the runtime value is to the normal distribution.

Now, according to the results obtained from Table 8 and the value of the Sig parameter for each algorithm, hypothesis H0 (data normality) is accepted. Therefore, to continue the analyses, we can use any parametric methods, for example, the Analysis of Variance (ANOVA) (Babazadeh Nanekaran and Rezvani 2021). The ANOVA test hypotheses are as follows:

H0: All algorithms’ average convergence time is equal (no significant difference).

| Configuration type | Core number | Core frequency (GHz) | Memory (MB) | Storage (GB) |
|--------------------|-------------|-----------------------|-------------|--------------|
| Type 1             | 2           | 1.7                   | 16          | 2000         |
| Type 2             | 4           | 1.7                   | 32          | 6000         |
| Type 3             | 8           | 1.7                   | 32          | 7000         |
| Type 4             | 8           | 2.4                   | 64          | 7000         |
| Type 5             | 8           | 2.4                   | 128         | 9000         |
The average convergence time is not equal for all algorithms (a significant difference).

Table 9 shows the results of the ANOVA test. Since the value of the $\text{Sig}$ is less than 0.05, it can be concluded that hypothesis $H_0$ is rejected. In other words, there is a significant difference among algorithms (groups) in terms of convergence time. Apart from the normality, another prerequisite for the ANOVA test is that the data variance is homogeneous. The Levin test checks this. Due to space limitations, we do not provide details of this test. Because the data variance is not homogeneous, the Games-Howell post hoc test can be used for the pairwise comparison of algorithms. The results of this test are shown in Table 10.

**H1:** The average convergence time is not equal for all algorithms (a significant difference).

Table 9 shows the results of the ANOVA test. Since the value of the $\text{Sig}$ is less than 0.05, it can be concluded that hypothesis $H_0$ is rejected. In other words, there is a significant difference among algorithms (groups) in terms of convergence time. Apart from the normality, another prerequisite for the ANOVA test is that the data variance is homogeneous. The Levin test checks this. Due to space limitations, we do not provide details of this test. Because the data variance is not homogeneous, the Games-Howell post hoc test can be used for the pairwise comparison of algorithms. The results of this test are shown in Table 10.

The value of $\text{Sig}$ in all rows of the table is less than 0.05, which indicates a significant pairwise difference among algorithms. The analysis of this table is based on the upper and lower limit values with the following rules:

1. If both the upper limit and the lower limit are positive, the difference between the mean of the two algorithms is more significant than zero, and the mean of the first algorithm is greater than that of the second algorithm.
2. If both the upper limit and the lower limit are negative, the difference between the mean of the two algorithms is less than zero, and the mean of the first algorithm is less than that of the second algorithm.

---

**Table 5** Data center specifications (Khosravi et al. 2017; Administration [https://www.eia.gov/electricity/state/archive/2014/](https://www.eia.gov/electricity/state/archive/2014/))

| Data center specifications | Hampton | Houston | Salem | San Francisco |
|---------------------------|---------|---------|-------|--------------|
| Carbon density rate (tons/MWh) | 0.43 | 0.580 | 0.138 | 0.28 |
| Carbon tax rate (dollars/ton) | 22 | 24 | 48 | 11 |
| Energy cost (cent/kWh) | 9.17 | 8.94 | 8.68 | 15.15 |
If the upper limit is positive and the lower limit is negative, the difference between the means of the two algorithms is not significant; hence, the equality of the mean of the two algorithms is not rejected.

The Mean Difference column in Table 10 shows that the means of the CRA-DP-GA and CRA-DP-MA differ by 0.0256 and 0.0376, respectively, compared to the CRA-DP algorithm. Given the random nature of metaheuristic algorithms, this amount of discrepancy seems plausible.

Also, the average time to find the optimal solution is shown in Fig. 15. As can be seen in the figure, there is a significant difference between the average time of the CRA-DP and other algorithms. The CRA-DP has the highest average execution time, followed by the CRA-DP-GA and CRA-DP-DDMMPEA, respectively. We expect that with the increase in the number of data centers and physical machines, the time to find the optimal solution for the CRA-DP algorithm will increase significantly. The reason is that this algorithm searches most of the data centers and physical machines, which in turn makes VMP longer, while the search space in the other two metaheuristic algorithms is much more limited.

As shown in Fig. 15, combined optimization methods (CRA-DPSCH-DDMMPEA and CRA-DPSCH-MA) have a longer convergence time than methods in which only cost optimization is performed. Note that the multi-objective optimization problem presented in Eq. (36) has more constraints than the single-objective case of Eq. (24). Also, the values obtained for each function in multi-objective problems are usually less than those obtained in single-objective cases. This is because of the tradeoff that can be made between potentially conflicting goals. This causes exploration to be slower in the search space, leading to long execution times. Although this time overhead is small, it has impressive achievements explained later in Figs. 17 and 22. Also, Fig. 15 shows that the convergence time in multi-objective methods never increases by more than 4% compared to single-objective cases.

5.3 Energy consumption

As shown in Fig. 16, the FFD algorithm consumes more fossil fuel for VM placement than other algorithms. This is because the FFD method places the VMs only based on the remaining capacity of physical servers, and no other parameter is involved in this decision. However, the CRA-DP-MA algorithm has the lowest brown energy consumption because it involves significant parameters such as PUE and data center availability. One of the most critical parameters that affect the energy consumption of fossil fuels is the PUE. This parameter affects the VMP by considering both the outside temperature and the workload. The availability of renewable energy sources and prioritizing data center power supply also significantly impact metaheuristic VMP algorithms. As shown in Fig. 16, there is no significant difference between metaheuristic algorithms. In other words, the brown energy consumption of the proposed meta-heuristic algorithms is not much higher.

| VM type       | Core number | Core frequency (GHz) | Memory (MB) | Storage (GB) | Probability of user type request |
|---------------|-------------|----------------------|-------------|--------------|----------------------------------|
| **Standard instances** |             |                      |             |              |                                   |
| M1Small       | 1           | 1                    | 1740        | 160          | 0.25-BT                         |
| M1Large       | 2           | 4                    | 7680        | 850          | 0.12-WR                         |
| M1XLarge      | 4           | 8                    | 15,360      | 1690         | 0.08-WR                         |
| **High memory instances** |       |                      |             |              |                                   |
| M2XLarge      | 2           | 6.5                  | 17,510      | 420          | 0.12-WR                         |
| M22XLarge     | 4           | 13                   | 35,020      | 850          | 0.08-WR                         |
| **High CPU instances** |       |                      |             |              |                                   |
| C1Medium      | 2           | 5                    | 1740        | 320          | 0.1-BT                          |

| Tests of normality | Kolmogorov-Smirnova | Shapiro–Wilk |
|--------------------|---------------------|--------------|
|                     | Statistic | df  | Sig  | Statistic | df  | Sig  |
| FFD                 | .484      | 30  | .000 | .349      | 30  | .000 |
| CRA-DP              | .167      | 30  | .033 | .852      | 30  | .001 |
| CRA-DP-GA           | .297      | 30  | .000 | .569      | 30  | .000 |
| CRA-DP-MA           | .293      | 30  | .000 | .650      | 30  | .000 |

*Lilliefors significance correction
the figure shows that the energy consumption of methods that use joint cost and scheduling optimization is slightly higher than that of single-objective methods. The general reason behind this behavior has already been stated in Fig. 15. The brown energy consumption in multi-objective scenarios never increases by more than 3% compared to single-objective cases.

As shown in Fig. 17, the FFD algorithm has the lowest energy consumption regarding renewable sources. Given that the order and availability of physical resources are the only parameters for VMP in this algorithm, such a result is reasonably expected. The interesting point in Fig. 17 is that there is very little difference between solar energy consumption in the CRA-DP-GA and the CRA-DP-MA algorithms. This behavior can be found in the neighborhood selection method in these algorithms. The proposed neighborhood algorithm has a severe limitation on

| Tests of normality       | Kolmogorov–Smirnov | Shapiro–Wilk |
|--------------------------|--------------------|--------------|
|                          | Statistic  | df  | Sig  | Statistic | df  | Sig  |
| CRA-DP-MA                | .174      | 24  | .059 | .945      | 24  | .208 |
| CRA-DP-GA                | .132      | 24  | .200*| .939      | 24  | .159 |
| CRA-DP                  | .151      | 24  | .164 | .957      | 24  | .384 |

*Lilliefors significance correction

Fig. 12 Normalized Q-Q plot of CRA-DP-MA

Fig. 13 Normalized Q-Q plot of CRA-DP-GA
choosing a neighbor for a solution. We consider this limitation to generate neighboring solutions with a higher chance. The fitness function in the CRA-DP-MA assigns the highest importance for solar energy. So, if we do not apply the data center constraints, we will obtain solutions with entirely different fitness. Figure 17 shows that the green energy consumption in multi-objective methods never increases by more than 4% compared to single-objective cases. Also, the ANOVA test result is shown in Table 12. The value of $\text{Sig.} = 0.963$ indicates the acceptance of hypothesis $H_0$. This suggests that statistically, the green energy consumption by metaheuristic algorithms is not significantly different.

5.4 Carbon footprint emission

As shown in Fig. 18, the FFD algorithm has the highest carbon dioxide production. According to the explanations provided in Sect. 5.3 and considering that fossil fuel consumption is directly related to carbon dioxide emissions, such good performance is entirely predictable for the FFD algorithm. Similar to Figs. 16 and 18 does not show a significant difference among metaheuristic methods. They

| Table 9 One-way analysis of variance (ANOVA) test for execution time |
|----------------|----------------|----------------|----------------|----------------|
| Sum of squares | $df$           | Mean square    | $F$            | $\text{Sig.}$ |
| Between groups | .000           | 2              | .000           | 104.980        | .000           |
| Within groups  | .000           | 79             | .000           | .000           | .000           |
| Total          | .000           | 81             |                |                |                |

| Table 10 Games-Howell post hoc test |
|-------------------------------------|
| Multiple comparisons                |
| Games-Howell                        |
| (I) VAR    | (J) VAR   | Mean difference (I-J) | SE     | Sig   | 95% CI |
|-----------|-----------|-----------------------|--------|-------|--------|
|           |           |                       | Lower bound | Upper bound |
| CRA-DP    | CRA-DP-MA | $-.00256^*$           | .00022  | .000  | $-.0031$ | $-.0020$ |
| CRA-DP    | CRA-DP-GA | $-.00376^*$           | .00026  | .000  | $-.0044$ | $-.0031$ |
| CRA-DP-GA | CRA-DP    | $-.00256^*$           | .00022  | .000  | .0020   | .0031   |
| CRA-DP-GA | CRA-DP-MA | $-.00120^*$           | .00032  | .001  | $-.0020$ | $-.0004$ |
| CRA-DP-MA | CRA-DP    | $-.00376^*$           | .00026  | .000  | .0031   | .0044   |
| CRA-DP-MA | CRA-DP-GA | $-.00120^*$           | .00032  | .001  | .0004   | .0020   |

$^*$The mean difference is significant at the 0.05 level
do not have a considerable effect on reducing the carbon footprint compared to the CRA-DP algorithm. However, they can prevent an increase in greenhouse gas emissions by reducing the convergence time. Figure 18 shows that the carbon footprint in multi-objective methods never increases by more than 3% compared to single-objective cases. Also, the ANOVA test is shown in Table 13. The value of Sig. = 0.543 indicates the acceptance of hypothesis $H_0$. This indicates that the amount of carbon dioxide produced by metaheuristic algorithms is not significantly different.
5.5 Cost

According to Eqs. (3) to (4), the energy cost is the sum of the green energy (renewable) cost and the brown energy (fossil fuels) cost. As mentioned earlier, since the cost of renewable energy is low in the long-term run, we set it at zero. As a result, the energy cost is only a function of consumed fossil fuel. In Fig. 19, the CRA-DP-MA, with a negligible difference, consumes less energy than the CRA-DP-GA, CRA-DP-DDMPEA, and the CRA-DP algorithms. The reason for this behavior was previously described in Fig. 16.

Figures 20 and 21 show the carbon cost and the total cost, respectively. According to Eq. (18), the carbon cost is a function of the rate of produced carbon dioxide and the associated tax. As shown in the figure, the amount of carbon produced by metaheuristic methods is never significantly worse than the baseline method, CRA-DP. Figure 20 shows that the cost of carbon in multi-objective plans never increases by more than 4% compared to single-
objective cases. Similarly, Fig. 21 indicates that the total cost in multi-objective methods never increases by more than 5% compared to single-objective cases.

### 5.6 SLA violations

By the SLA violation, we mean the number of VM requests rejected due to the lack of physical resources in the data center. We conducted a stress test on the system and changed the number of VM requests from 1000 to 1700. Figure 22 shows the percentage of SLA violations for different algorithms. As seen in the figure, the more traffic intensity the system exerts, the lower the SLA violations. This, in turn, degrades the QoE of users. However, the amount of degradation in methods that use joint cost and scheduling optimization is far less than single-objective ones. The results show that the proposed method can reduce SLA violations by almost twice in heavy traffic.

| ANOVA     | Sum of squares | df  | Mean square | F    | Sig  |
|-----------|----------------|-----|-------------|------|------|
| Between groups | 3,834,114.771  | 2   | 1,917,057.386 | .614 | .543 |
| Within groups   | 271,418,811.035 | 87  | 3,119,756.449 |     |      |
| Total           | 275,252,925.806 | 89  |             |      |      |

Fig. 19 Energy cost for different VMP algorithms

Fig. 20 The cost of produced carbon for different VMP algorithms
This confirms our primary claim in this study about the need to pay attention to the scheduling of tasks.

6 Conclusion and future trends

In this paper, we targeted one of the most practical issues of cloud computing, namely carbon-efficient VM placement. To meet real-world requirements, we incorporated the delay-sensitive nature of IoT requests into modeling.

After formulating each sub-problems, we provided a joint energy cost and scheduling optimization for the problem. Since the problem is NP-hard, we solved it on a large scale using the modified Memetic Algorithm (MA). Then, we evaluated the performance of the proposed method against the baseline method, such as the Genetic Algorithm (GA) and state-of-the-art techniques.

Solar energy was also considered one of the energy supply sources to reduce energy and carbon consumption costs. Our solution minimizes the energy consumption of...
cloud service providers and considers the users’ QoE and SLA requirements. The simulation results revealed the superiority of the proposed approaches in terms of significant criteria such as VM allocation time, energy cost, energy consumption, Power Usage Effectiveness (PUE), and SLA violation. The results showed that proposed metaheuristics algorithms could significantly decrease the convergence time compared to GA-based and the heuristic method and, at the same time, do not cause a significant increase in energy consumption and cost.

For the sake of fairness, a comparison was made between multi-objective and single-objective optimization modes. As is common in optimization theory, we expect the value of the functions obtained in multi-objective optimization to be less than those of a single-objective case. However, the amount of degradation in various criteria was never more than 5%, which is quite promising. Also, the results showed that in heavy traffic, the proposed method could reduce SLA violations by almost twice. When most tasks do not have a firm deadline, the competitive advantage of the proposed method over the base method may not be as significant. However, it will be as efficient as the base method in the worst case.

There are several possible lines of research for the future. To find a neighborhood in the MA, both the outside temperature and renewable energy significantly impact data center utilization. This study assumed that hosts in the same geographical locations are more likely to have similar temperatures and solar energy. As one of the future works, the effect of these two parameters can be examined regardless of the geographical location. We also ignored the impact of communication network components. One future research could consider the impact of these components on energy consumption and the VMP cost. Machine learning methods, especially reinforcement learning, can predict future carbon-efficient placements.

**Funding** The authors did not receive support from any organization for the submitted work.

**Data availability** The datasets generated during and analyzed during the current study are available in the [MENDELEY] repository. [http://data.mendeley.com/datasets/2g7dy8bnfj/1].

**Declarations**

**Conflict of interest** On behalf of all authors, the corresponding author states no conflict of interest.

**References**

Aboutorabi SJS, Rezvani MH (2020) An optimized meta-heuristic bees algorithm for players’ frame rate allocation problem in cloud gaming environments. Comput Games J 9(3):281–304

Ahvar E, Orgerie AC, Lebre A (2019) Estimating energy consumption of cloud, fog and edge computing infrastructures. IEEE Trans Sustain Comput 7:277

Amazon. Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/

Babazadeh Nanekharaan A, Rezvani MH (2021) An incentive-compatible routing protocol for delay-tolerant networks using second-price sealed-bid auction mechanism. WIREs Commun 121(3):1547–1576

Besharati R, Rezvani MH, Sadeghi MMG (2020) An Incentive-compatible offloading mechanism in fog-cloud environments using second-price sealed-bid auction. J Grid Comput 19(3):1–29

Brady GA, Kapur N, Summers JL, Thompson HM (2013) A case study and critical assessment in calculating power usage effectiveness for a data centre. Energy Conver Manage 76:155–161

Burke EK, Hyde MR, Kendall G (2006) Evolving bin packing heuristics with genetic programming. In: Parallel problem solving from nature-PPSN IX. Springer, Berlin, pp 860–869

Chauhan S, Vashishtha G (2021) Mutation-based arithmetic optimization algorithm for global optimization. In: 2021d International conference on intelligent technologies (CONIT), IEEE, pp 1–6

Chauhan S, Singh M, Aggarwal AK (2021a) Cluster head selection in heterogeneous wireless sensor network using a new evolutionary algorithm. In: Wireless personal communications, pp 1–32

Chauhan S, Singh M, Aggarwal AK (2021b) Design of a two-channel quadrature mirror filter bank through a diversity-driven multi-parent evolutionary algorithm. Circuits Syst Signal Process 40(7):3374–3394

Chauhan S, Singh M, Aggarwal AK (2021c) Diversity driven multi-parent evolutionary algorithm with adaptive non-uniform mutation. J Exp Theor Artif Intell 33(5):775–806

Chauhan S, Vashishtha G, Kumar A (2021d) A symbiosis of arithmetic optimizer with slime mould algorithm for improving global optimization and conventional design problem. J Supercomput 78:1–41

Chekuri C (1998) Approximation algorithms for scheduling problems. Stanford University

Commission E (2014) Available from European Commission

Esfandiari S, Rezvani MH (2020) An optimized content delivery approach based on demand–supply theory in disruption-tolerant networks. In: Telecommunication systems, pp 1–25

Fawaz AH, Mohammed AFY, Laku LIY, Alnazi R (2019) PUE or GPUE: a carbon-aware metric for data centers. In: 2019 21st international conference on advanced communication technologies (ICACT), IEEE, pp 38–41

Gao Y, Guan H, Qi Z, Hou Y, Liu L (2013) A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J Comput Syst Sci 79(8):1230–1242

Iwendi C, Maddikunta PKR, Gadekallu TR, Lakshmanna K, Bashir AK, Piran MJ (2021) A metaheuristic optimization approach for energy efficiency in the IoT networks. Softw Practice Exp 51(12):2558–2571

Jafari V, Rezvani MH (2021) Joint optimization of energy consumption and time delay in IoT-fog-cloud computing environments
using NSGA-II metaheuristic algorithm. J Ambient Intell Human Comput. https://doi.org/10.1007/s12652-021-03388-2

Karupiah SV, Gurunathan G (2021) Secured storage and disease prediction of E-health data in cloud. J Ambient Intell Humaniz Comput 12(6):6295–6306

Keshavarznejad M, Rezvani MH, Adabi S (2020) delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. Clust Comput 24:1825–1853

Khoobkar MH, Dehghan Takht Fooladi M, Rezvani MH, Gilian Sadeghi MM (2022) BLMDP: Partial offloading with stable equilibrium in fog-cloud environments using replicator dynamics of evolutionary game theory. In: Cluster computing (to be appear)

Khosravi A, Andrew LL, Buyya R (2017) Dynamic VM placement method for minimizing energy and carbon cost in geographically distributed cloud data centers. IEEE Trans Sustain Comput 2(2):183–196

Laganà D, Mastroianni C, Mei M, Renga D (2018) Reducing the operational cost of cloud data centers through renewable energy. Algorithms 11(10):145

López J, Kushik N, Zeghlache D (2019) Virtual machine placement quality estimation in cloud infrastructures using integer linear programming. Softw Qual J 27(2):731–755

Maddikunta PKR, Gadekallu TR, Kaluri R, Srivastava G, Parizi RM, Khan MS (2020) Green communication in IoT networks using a hybrid optimization algorithm. Comput Commun 159:97–107

Misra SK, Kuila P (2022) Energy-efficient task scheduling using quantum-inspired genetic algorithm for cloud data center. In: Advanced computational paradigms and hybrid intelligent computing. Springer, Singapore, pp 467–477

Mohammadi A, Rezvani MH (2019) A novel optimized approach for resource reservation in cloud computing using producer–consumer theory of microeconomics. J Supercomput 75(11):7391–7425

Naghdheforoushha M, Fooladi DT et al (2022) BLMDP: a new bi-level markov decision process approach to joint bidding and task-scheduling in cloud spot market. Turk J Elec Eng Comp Sci. https://doi.org/10.3906/elk-2108-89

Parvizi E, Rezvani MH (2020) Utilization-aware energy-efficient virtual machine placement in cloud networks using NSGA-III meta-heuristic approach. In: Cluster computing, pp 1–23

Rashida SY, Sabaei M, Ebadzadeh MM, Rahmani AM (2019) A memetic grouping genetic algorithm for cost efficient VM placement in multi-cloud environment. In: Cluster computing, pp 1–40

Rasmussen N, American Power Conversion Denmark ApS (2009) Electrical efficiency measurement for data centers. U.S. Patent Application 12/260448

RM SP et al (2020) Load balancing of energy cloud using wind driven and firefly algorithms in internet of everything. J Parallel Distrib Comput 142:16–26

Rodrigo NC, Rajiv R, Anton B, César AFR, Rajkumar B (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Practice Exp J 41(1):23–50

Russell S, Norvig P (2002) Artificial intelligence: a modern approach

Tavakoli-Someh S, Rezvani MH (2019) Multi-objective virtual network function placement using NSGA-II meta-heuristic approach. J Supercomput 75(10):6451–6487

Thiam C, Thiam F (2019) Optimizing electrical energy consumption in cloud data center

Vashishtha G, Kumar R (2021a) Autocorrelation energy and aquila optimizer for MED filtering of sound signal to detect bearing defect in Francis turbine. Meas Sci Technol 33(1):015006

Vashishtha G, Kumar R (2021b) Pelton wheel bucket fault diagnosis using improved shannon entropy and expectation maximization principal component analysis. J Vib Eng Technol 10:1–15

Vashishtha G, Kumar R (2022) An amended grey wolf optimization with mutation strategy to diagnose bucket defects in Pelton wheel. Measurement 187:110272

Vashishtha G, Chauhan S, Singh M, Kumar R (2021) Bearing defect identification by swarm decomposition considering permutation entropy measure and opposition-based slime mould algorithm. Measurement 178:109389

Workload, p. parallel workload Archive. https://www.cs.huji.ac.il/labs/parallel/workload/

Xu M, Buyya R (2020) Managing renewable energy and carbon footprint in multi-cloud computing environments. J Parallel Distrib Comput 135:191–202

Zhang X, Wu T, Chen M, Wei T, Zhou J, Hu S, Buyya R (2019) Energy-aware virtual machine allocation for cloud with resource reservation. J Syst Softw 147:147–161

Zhao H, Wang J, Liu F, Wang Q, Zhang W, Zheng Q (2018) Power-aware and performance-guaranteed virtual machine placement in the cloud. IEEE Trans Parallel Distrib Syst 29(6):1385–1400

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.