Virtual Machine Placement in Cloud Environments Using a Hybrid Cuckoo Search and Bat Algorithm

Sifeddine Benflis, Sonia-Sabrina Bendib, Sedrati Maamar, Fatima Z. Cherhabil, and Hanane Merouani

University of Batna 2, Batna, Algeria

https://doi.org/10.26636/jtit.2025.4.2244

Abstract — The growing popularity of on-demand pay-as-you-go subscription models for online cloud computing requires increasing amounts of resources to ensure adequate quality of services. However, to satisfy the strong demand for these services, cloud infrastructure providers continue to scale up their data centers. This scaling often lacks an optimal resource management approach, thus leading to inefficiencies, excessive energy consumption, and higher costs. This creates challenges in the virtual machine placement (VMP) process focusing on identifying efficient ways for assigning virtual machines to physical hardware. This paper introduces a hybrid cuckoo search bat algorithm (HCS-BA) to solve VMP in heterogeneous cloud environments. The suitability of the cuckoo search algorithm for global searches is combined with the local refining capacity of the bat algorithm, therefore optimizing both energy consumption and resource utilization. The results of simulations carried out in Matlab and CloudSim for scalability testing demonstrate that HCS-BA outperforms both individual algorithms. It reduces energy consumption and improves resource utilization.

Keywords — bat algorithm, cloud computing, cuckoo search algorithm, virtual machine placement

1. Introduction

Cloud computing is an evolving technology that offers, over the Internet, a vast majority of services to a large number of clients. These services can be obtained on demand, without any interaction with the cloud provider, via the pay-as-you-go model which turns computing resources into business assets and allows to charge consumers for the capacity they have used.

As cloud services continue to evolve, the demand for larger and more efficient data centers is booming. These facilities require massive computing, networking, and data storage capacities to support various applications with the expected quality of services. Consequently, the growing reliance on cloud environments is driving the construction of new, more advanced data centers. However, this rapid expansion introduces significant challenges, including high energy consumption, resource usage inefficiencies, and network bandwidth limitations [1]–[3].

Virtualization has become the cornerstone of cloud computing, allowing physical resources to be efficiently shared among multiple users through virtual machines (VMs). These virtual machines run on physical hardware housed within data centers, leading to significant improvements in resource utilization [4]. However, this advancement also creates new challenges, most notably the virtual machine placement (VMP) problem. Therefore, addressing VMP has become essential for enhancing the performance and efficiency of cloud infrastructure.

The VMP problem involves mapping VMs to physical machines (PMs) within a cloud data center. It is a key consideration in how these data centers operate and plays a crucial role in cloud computing, as it significantly affects performance and costs. Placing virtual machines without optimization can result in significant inefficiencies such as underutilized servers, increased energy consumption from running unnecessary PMs, and idle servers that waste power.

As cloud services continue to grow and demand for computing resources increases, VMP has become a major area of interest for both researchers and practitioners. This challenge has been studied from various perspectives, with many approaches aiming to minimize energy use and improve overall system efficiency. Despite the often conflicting objectives, the diversity in strategies has led to a wide range of models, frameworks, and algorithms [5], [6].

Given the complexity of the VMP problem and its impact on efficiency, researchers have turned to heuristic and metaheuristic algorithms to find near-optimal solutions. Various approaches have been widely adopted, including ant colony optimization [2], whale optimization [7], cuckoo search algorithm (CSA) [8], and bat algorithm (BA) [9]. Furthermore, researchers have combined techniques such as hybridization of flower pollination with particle swarm optimization to take advantage of the features of multiple algorithms [4].

Over the years, both BA and CSA have been applied to a wide range of optimization problems, including VMP. The cuckoo search algorithm is known for its strong exploration capabilities, thanks to the Lévy flight mechanism [10], while BA effectively balances exploration and exploitation using adaptive parameters such as loudness and pulse rate [11]. These strengths have made both algorithms popular in the face of complex challenges. However, despite their proven

potential, there has been relatively little research focused on combining these two algorithms for VMP-related tasks. While each excels in different aspects (CSA in broad exploration and BA in fine-tuned exploitation), their hybridization remains largely unexplored.

Merging their complementary features could offer a promising solution to common issues, for instance premature convergence or getting caught in local optima. Exploring this hybrid approach could open new avenues to improve the efficiency and effectiveness of VMP in cloud environments.

In this paper, a hybrid algorithm has been proposed balancing the exploitation capabilities of BA with the exploration aptitude of the Lévy flight-relying CS algorithm (HCS-BA). This hybrid method is designed to optimize VMP in cloud computing environments while simultaneously reducing energy consumption and improving resource utilization.

The rest of this paper is organized as follows. Section 2 discusses related work. Problem formulation is introduced in Section 3 and the HCS-BA algorithm is described in Section 4. Section 5 presents and discusses the experimental results. The paper is concluded in Section 6.

2. Related Work

The process of selecting a VM that should be assigned to given physical machine (PM) is known as virtual machine placement. This task becomes challenging due to factors such as the scale and complexity of the cloud environment, which bring concerns such as efficient resource utilization, energy consumption, and overall system performance. To address these challenges, researchers have proposed a wide range of techniques, ranging from heuristic and metaheuristic methods to hybrid approaches, all aiming to find smarter and more efficient placement strategies.

Existing methods, such as flower pollination optimization (FPO) and particle swarm optimization (PSO), often struggle with local optima and result in inefficient resource utilization. To overcome the drawback of both algorithms, a novel multi-objective algorithm has been developed, known as hybrid particle swarm with Lévy flight flower pollination optimization (HPSOLF-FPO), being a hybrid of PSO with Levy flight and flower pollination, to minimize the time needed for the mapping of VM to PM, energy consumption, and resource waste [4]. It takes advantage of the exploitation strength of PSO and exploration strength of FPO, resulting in an improvement in all performance metrics. Unfortunately, HPSOLF-FPO still suffers from some limitations, mainly the congestion problem, meaning that too many VMs are allocated to a single PM. Local optima pose a challenge too, as they cannot be solved even with the help of Lévy flight.

In [12], an innovative method to address the high energy demands of cloud data centers has been introduced, i.e. a hybrid VMP algorithm that integrates a genetic algorithm for permutation-based optimization problems (IGA-POP) with a multidimensional resource-aware best fit (BF) allocation strategy. This combination reduces the number of active physical servers, leading to significant energy savings, and simultaneously reduced resource waste. IGA-POP balances exploration and exploitation within the optimization process, while BF strategy ensures the efficient use of critical resources such as CPU, RAM, and network bandwidth. The experimental results highlight its superiority in terms of energy efficiency and resource utilization, compared to traditional heuristic and metaheuristic approaches.

The problem of reducing energy consumption for dynamic virtual machine placement in cloud data centers has been tackled in [13]. The authors proposed an ant colony system (ACS) that was enhanced with designed heuristics. This approach dynamically adapts to workload fluctuations and prioritizes utilizing active PMs over activating new ones, thus minimizing total energy use. Improvements in precision and efficiency have been observed when a problem is defined as a constrained combinatorial optimization problem. Extensive simulations demonstrate that the proposed ACS outperforms traditional methods, such as first-fit decreasing (FFD) and existing ACS-based strategies across small- to large-scale data centers.

Paper [8] introduced an advanced optimization technique for efficient allocation of virtual physical machines. By improving the cuckoo search algorithm (CSA) from [14], with cost and perturbation functions, the study addresses resource utilization, reducing the number of active PMs and energy usage. The proposed approach ensures minimal resource waste while avoiding server overload. Experiments on benchmark datasets reveal its superiority to traditional methods, as it achieves reduced energy consumption and fewer active servers while maintaining fast task execution.

In the context of hybridization, the method combining flower pollination optimization (FPO) and the Pareto front module of the nondominated classification genetic algorithm (NSGA-II) was used to address the VMP problem in [15]. The proposed FP-NSO algorithm, along with a bio-VMP framework that helps allocate VMs to PMs, optimizes multiple objectives, including maximizing resource utilization and minimizing power consumption, hence reducing carbon emissions in cloud data centers.

Another proposal, the GATA algorithm [16], is a hybrid approach that combines the genetic algorithm (GA) and the tabu search (TS). The optimization objectives focus on reducing energy consumption and improving the balance of load across data center resources. To enhance GA's local search capabilities, TS has been used as a mutation operator, preventing premature convergence and increasing solution diversity. Experimental evaluations show that GATA outperforms the traditional GA, simulated annealing (SA) and ACS-based methods, achieving better energy efficiency, more balanced resource utilization, and competitive execution times.

A VMP method using an enhanced cuckoo search (ECS) was explored in [17]. The ECS integrates strategies such as Lévy flights for local and global exploration, "effective overload detection", "VM selection policies", and a status index for resource utilization. Optimization objectives are minimizing energy consumption, SLA violations, and VM migrations

while maximizing resource utilization. Experiments were conducted using CloudSim, and real workload traces demonstrated the superiority of ECS over other solutions, such as genetic, optimized firefly search (OFS) and ant colony (AC) algorithms, achieving significant energy savings, reduced SLA violations and optimized VM placements. However, even with Lévy flights, CS tends to over-explore, meaning that local optima exploitation may be weak.

An approach called multi-objective bat algorithm with decomposition (MOBA/D), presented in [9], addressed minimizing energy consumption and reducing network traffic. Unlike traditional methods, MOBA/D leverages a decomposition-based strategy to divide the VMP into smaller subproblems, solving them individually for improved efficiency with another technique that allows for the discretization of continuous values, meaning that it lets them deal with the VMP problem more efficiently.

The authors compared the performance of MOBA/D with that of established multiobjective algorithms such as MOEA/D, NSGA-II, and memetic MOEA across various scenarios. Experimental results demonstrate that MOBA/D outperforms these algorithms in terms of Pareto front solutions and execution time.

Despite significant advances in VMP algorithms, existing methods often suffer from limitations such as premature convergence to local optima, inefficient resource utilization, slow convergence speed, unstable exploitation, and weak exploration capabilities.

Study [18] introduced a hybrid approach that combines BA with the CSA to improve task scheduling in cloud settings. In this method, tasks are first ranked using BA and then CS is applied to assign these tasks to virtual machines in a way that balances the workload. The approach was tested in CloudSim and compared with other metaheuristics, showing that HB-CSA achieved better load distribution, improved scheduling efficiency, and more effective VM utilization.

However, the HB-CSA framework was designed specifically for task scheduling and does not address the broader challenge of VMP in heterogeneous cloud data centers. VMP brings additional complexity, as it involves diverse host resources, multidimensional constraints like CPU, memory, and storage, as well as energy consumption considerations. Importantly, heterogeneity and energy optimization were not central to the HB-CSA study, leaving room for approaches that explicitly address these issues. In this context, this paper aims to address these gaps by developing a hybrid algorithm that integrates BA with the CS algorithm for VMP. This combination leverages BA exploitation capabilities with efficient CSA exploration with Lévy flight.

3. Problem Formulation

We consider a cloud environment made up of a set of physical and virtual machines, each with varying resource capacities. The objective is to find the optimal placement of virtual machines in PMs in a way that maximizes resource utilization while minimizing energy consumption.

We have the following sets which define our model:

$PM = \{PM_1, PM_2, \ldots, PM_n\}$: set of physical machines

$VM = \{VM_1, VM_2, \ldots, VM_m\}$: set of virtual machines

$R_{VM_j} = \{\text{CPU}_j, \text{RAM}_j, \text{Storage}_j\}$: resource requirements of VM_j

$C_{PM_i} = \{\text{CPU}_i, \text{RAM}_i, \text{Storage}_i\}$: capacity of physical machine_i

$U_{\text{CPU}} =$ CPU usage of a physical machine

$U_{\text{RAM}} =$ RAM usage of a virtual machine

$U_{\text{Storage}} =$ storage usage of a virtual machine

3.1. Resource Model

The resource model describes how CPU, RAM, and storage usage rates are calculated and utilized in relation to their respective usages.

Resource utilization for each host is defined as:

$$\text{Total RAM usage} = \sum_{j=0}^{m} \text{RAM}_j \,, \tag{1}$$

$$\text{Total CPU usage} = \sum_{j=0}^{m} \text{CPU}_j \,, \tag{2}$$

$$\text{Total storage usage} = \sum_{j=0}^{m} \text{Storage}_j \,, \tag{3}$$

$$U_{\text{CPU}} = \left(\frac{\text{Total CPU usage}}{\text{CPU total capacity}} \right) \times 100 \,, \tag{4}$$

$$U_{\text{RAM}} = \left(\frac{\text{Total RAM usage}}{\text{RAM total capacity}} \right) \times 100 \,, \tag{5}$$

$$U_{\text{Storage}} = \left(\frac{\text{Total storage usage}}{\text{Storage total capacity}} \right) \times 100 \,. \tag{6}$$

The overall utilization across hosts is calculated as the average utilization of CPU, RAM, and storage is:

$$\begin{matrix}\text{Resource} \\ \text{utilization}\end{matrix} = \frac{\sum\limits_{i=0}^{n} U_{\text{CPU}_i} + \sum\limits_{i=0}^{n} U_{\text{Storage}_i} + \sum\limits_{i=0}^{n} U_{\text{RAM}_i}}{3} \,, \tag{7}$$

with the following constraints:

$$\sum_{j=0}^{m} \text{CPU}_j \leqslant C_{\text{PM}_i}^{\text{CPU}} \,, \tag{8}$$

$$\sum_{j=0}^{m} \text{Storage}_j \leqslant C_{\text{PM}_i}^{\text{Storage}} \,, \tag{9}$$

$$\sum_{j=0}^{m} \text{RAM}_j \leqslant C_{\text{PM}_i}^{\text{RAM}} \,. \tag{10}$$

3.2. Energy Model

The energy model estimates the power consumption of each host based on its utilization of resources. For every physical

machine, energy usage is calculated in the following way:

$$E_{\text{host}} = U_{\text{idle}} + (U_{\text{max}} - U_{\text{idle}}) \cdot (\alpha_{\text{cpu}} \cdot U_{\text{cpu}} + \beta_{\text{ram}} \cdot U_{\text{ram}} + \gamma_{\text{storage}} \cdot U_{\text{storage}}) , \quad (11)$$

where: U_{idle} is idle energy consumption, U_{max} is energy consumption in active state, α_{cpu}, β_{ram}, $\gamma_{storage}$ are weights for CPU, RAM, and storage contributions to energy consumption. Total energy consumption can be modeled as follows:

$$\text{Total energy} = \sum_{i=0}^{n} E_{\text{host}_i} . \quad (12)$$

3.3. Fitness Function

Energy consumption and resource utilization can be grouped into one single fitness function:

$$\text{Fitness function} = \alpha \cdot \text{energy} + \frac{1}{\beta \cdot \text{Resource utilization}} , \quad (13)$$

where α – weight for the energy component, β – weight for resource utilization.

4. Proposed Approach

In this investigation, both nests and bats are represented using a matrix **M**, which captures all potential solutions (Fig. 1). In this matrix, rows correspond to PMs and columns correspond to VMs. Each matrix represents a unique way to assign VMs to PMs, effectively illustrating different placement strategies. For each solution, $H \times V$ binary matrix is generated, where the solution $[i, j, k]$ $a \in A[0, 1]$ indicates VM k assigned to host j in the nest or bat i.

4.1. Overview of the Cuckoo Search Algorithm

The cuckoo search algorithm mimics the behavior of cuckoo birds. Cuckoo birds rely on other birds nests to lay their eggs rather than building their own. The host bird might identify the intruding egg, so once that happens, birds either build new nests or simply remove the egg. In a nest, each egg represents a solution, and the cuckoo egg represents a new and good solution. The host bird identifies the cuckoo egg with the probability of Pa, so the higher the probability, the higher the chance of the egg being removed [10], [19].

A Lévy flight is a random walk-in which step-lengths are calculated according to a heavy-tailed probability distribution [10], [19]. This helps prevent premature convergence and being stuck in local optima. This behavior is especially useful in optimization, because it allows the algorithm to explore the search space more efficiently.

M =

	VM1	*VM2*	*VM3*	*VM4*	*VM5*	*VM6*	*VM7*	*VM8*	*VM9*	*VM10*
PM1	1	0	1	0	0	0	1	0	0	0
PM2	0	1	0	0	0	1	0	1	0	0
PM3	0	0	0	1	0	0	0	0	0	0
PM4	0	0	0	0	1	0	0	0	0	0
PM5	0	0	0	1	0	0	0	0	1	1

Fig. 1. Example of VM-PM allocation matrix.

When applied to the VMP problem, each nest can be viewed as a possible allocation of virtual machines to physical hosts. Through the use of Lévy flights, CS is able to make occasional long jumps in the search space, which allows it to explore diverse placement possibilities and avoid being trapped in local optima. Over time, it discovers more effective VM placements by using a P_a probability to determine whether to keep or discard a given solution.

This strong exploratory ability makes the algorithm highly effective in discovering promising global solutions in complex and heterogeneous environments. However, because CS emphasizes global exploration, it often converges more slowly and lacks strong local exploitation capabilities, which means it may not always fine-tune solutions efficiently once good regions of the search space are found.

Algorithm 1 Cuckoo search using Lévy flight

Input: population size N_{pop}, iterations N_{iter}, hosts H, VMs V, abandonment rate P_a, fitness weights α, β, resource constraints.
Output: Best solution S_{best} and its fitness
Generate resource capacities for hosts
Assign random requirements to each VM
Initialize population:
Generate $H \times V$ binary matrix $nests[i, j, k] \in \{0, 1\}$
for each nest **do**
    Randomly assign each VM to one host
end for
Validate VM assignments and adjust overloaded hosts
for $iter = 1$ to N_{iter} **do**
    for each nest **do**
        Evaluate fitness via Eq. (13)
        Update S_{best} if improved
        if $rand > P_a$ **then**
            Apply Lévy flight for exploration
        else
            Replace P_a worst nests with new random ones
        end if
        Validate VM assignments and adjust overloaded hosts
    end for
end for
End

The Algorithm 1 with Lévy flight begins by initializing the problem parameters, such as the number of hosts and VMs, population size (nests), number of iterations, and fitness weights. Each nest represents a possible solution (a binary matrix), where each 1 indicates that a specific VM is placed on a specific host. The initial population of nests is created by randomly assigning VMs to hosts. The algorithm then checks for valid placements and adjusts any overloaded hosts to ensure feasibility. Adjusting overallocated hosts can be achieved by preventing PMs from hosting new VMs which exceed the total capacity of the PM using Eqs. (8)–(10).

During each iteration of the algorithm, the fitness of every candidate solution (or nest) is evaluated on fitness function with energy and resource utilization serving the role of param-

eters. If a solution outperforms the current best, it becomes the new reference point S_{best}. To maintain exploration and avoid premature convergence, the algorithm introduces randomness through two strategies governed by probability P_a.

With a chance higher than P_a, a Lévy flight is applied to explore new and potentially better placements by making large or small shifts in the VM-to-PM assignment. Otherwise, the algorithm abandons the least promising solutions and replaces them with new randomly generated ones to increase diversity in the population. After these updates, the algorithm ensures that all assignments are valid and adjusts any overloaded hosts accordingly. This process repeats over a set number of iterations, gradually refining the solutions, and finally returning the best one found.

4.2. Overview of the Bat Algorithm

Bats rely on a type of sonar called echolocation to figure the distance between them and their prey. They travel in the search space using a set of parameters [11]:

- Velocity: bats randomly move in the search space with V_i.
- Position: bats move from one position to another, represented by X_i.
- Frequency range: defines the lower and upper bounds of frequency and controls the range of velocity adjustments for bats.
- Loudness: represents the initial loudness of bats, controls step size and convergence behavior, and typically decreases over iterations as bats approach their prey.
- Pulse emission rate: defines the initial rate of pulse emission and increases as the algorithm progresses, promoting local exploitation.

In the BA applied to VMP, each bat represents a possible solution, essentially a way of assigning virtual machines to physical machines. Bats "fly" through the solution space by adjusting their positions (placements) based on jumps instead of velocity to make the problem discrete.

Instead of moving the bat in certain direction with a specific velocity, we use the *jump* variable which helps us move a VM

$\mathbf{M} =$

	VM1	*VM2*	*VM3*	*VM4*	*VM5*	*VM6*	*VM7*	*VM8*	*VM9*	*VM10*
PM1	1	0	1	0	0	0	1	0	0	0
PM2	0	1	0	0	0	1	0	1	0	0
PM3	0	0	0	1	0	0	0	0	0	0
PM4	0	0	0	0	1	0	0	0	0	0
PM5	0	0	0	1	0	0	0	0	1	1

Jump ↓

$\mathbf{M}' =$

	VM1	*VM2*	*VM3*	*VM4*	*VM5*	*VM6*	*VM7*	*VM8*	*VM9*	*VM10*
PM1	0	0	0	1	0	0	0	0	0	0
PM2	0	0	1	0	1	0	0	0	1	0
PM3	1	1	0	0	0	1	0	1	0	1
PM4	0	0	0	1	0	0	1	0	0	0
PM5	0	0	0	0	0	0	0	0	0	0

Fig. 2. Example of a discrete VM migration jump.

Algorithm 2 Bat algorithm flowchart

Input: population size N_{pop}, max iterations N_{iter}, weights α, β, loudness A, pulse rate r, f_{min}, f_{max}, maxHops, minHops, abandonment rate P_a, and resource constraints
Output: Best solution S_{best} and its fitness
Generate resource capacities for hosts
Assign random requirements to each VM
Initialize population:
Generate $H \times V$ binary matrix $bats[i, j, k] \in \{0, 1\}$
for each bat **do**
    Randomly assign each VM to one host
end for
Validate VM assignments and adjust overloaded hosts
for $iter = 1$ to N_{iter} **do**
    for each bat **do**
        Generate frequency $f \in [f_{min}, f_{max}]$
        Update VM assignments via frequency-guided jumps
        Evaluate fitness using Eq. (13)
        if fitness improved **then**
            Update global best solution
        end if
        if $rand > r$ **then**
            Perform local random walk around best solution
            Fine-tune using A-scaled adjustment
        end if
        if $rand < A$ **then**
            Decrease A and increase r
        end if
        Validate VM assignments and adjust overloaded hosts
    end for
end for
End

from one PM to another. Each bat is considered a candidate solution to the problem. The *jump* is calculated as follows:

$$jump = minHops + \text{random}(maxHops - minHops + 1)\,, \tag{14}$$

where $minHops$ represents the minimum number of hops, $maxHops$ represents the maximum number of hops, random picks a number between 0 and $(maxHops - minHops + 1)$. The bat algorithm also uses frequency, loudness, and pulse rate to control exploration and exploitation. These parameters control how much the bat explores new solutions or fine-tunes existing ones. At each step, a bat can explore globally or exploit locally, depending on how good the current solution is and how close it is to the best one found so far. Over time, bats naturally focus more on promising areas. The best solution discovered during this process is returned as the optimal placement.

Overall, BA is particularly effective in fine-tuning solutions. Its adaptive use of loudness and pulse emission rate allows it to gradually shift focus toward promising regions of the search space, making it well suited for refining virtual machine place-

ments once good candidates are found. Unfortunately, this strength comes with a limitation. Because the algorithm emphasizes local search, it can sometimes converge too quickly and miss better global solutions. In large and heterogeneous cloud environments, this tendency toward premature convergence highlights the need to complement the BA with stronger exploratory capabilities.

The BA shown as Algorithm 2 starts by setting up the cloud environment defining the resource capacities of each physical host and the demands of each virtual machine. Each bat in the population represents a possible way to place VMs onto hosts, initialized randomly using a binary matrix. VM placement is validated and overallocation is adjusted by preventing PMs from hosting new VMs that exceed the total capacity of the PM using the constraints (8)–(10). Then, the algorithm enters its main loop. Here, each bat updates its position using a frequency value, which guides how it shifts VM assignments.

Each updated solution is evaluated using fitness functions that consider energy efficiency and resource utilization. If a bat discovers a better solution, it becomes the new global best. Occasionally, if a random chance exceeds the bat's pulse rate, it performs a local search near the current best solution, fine-tuning it slightly based on its loudness. Then, with a probability based on A, the algorithm decides whether to accept the new solution. If accepted, it simulates a bat coming closer to its prey by decreasing the volume and increasing pulse rate, which gradually shifts the bat's behavior toward exploitation. Finally, it checks and adjusts the VM assignments to ensure that no host is overloaded.

4.3. Hybrid Cuckoo Search-Bat Algorithm (HCS-BA)

The major strength of the proposed hybrid cuckoo search-bat algorithm (HCS-BA) shown as Algorithm 3, is not just the combination of two metaheuristics, but the way in which this integration is structured to address the specific challenges of VMP in heterogeneous cloud environments.

While hybrid metaheuristics are common in the literature, most of them follow a sequential or operator-level fusion that often inherits the weaknesses of both methods. In contrast, HCS-BA employs a competitive, parallel design that preserves the independent search behavior of each algorithm and leverages their complementary strengths through a best-solution selection strategy. This ensures that the algorithm avoids premature convergence through explorations, while also accelerating refinement once promising solutions emerge through exploitation.

HCS-BA starts by generating the resource capacities (CPU, RAM, storage) for each physical host and assigning random requirements to the virtual machines. Two separate populations are initialized: one for CS (nests) and one for BA (bats), with each individual represented as a binary matrix indicating VM-to-host assignments.

The main loop runs for a set number of iterations. In the CS phase, the solution of each nest is evaluated using the fitness function. If a solution outperforms the current best, it becomes the new reference point S_{best} for nests. With probability P_a, the algorithm either applies a Lévy flight to explore new

Algorithm 3 HCS-BA scheme

Input: population size N_{pop}, max iterations N_{iter}, weights α, β, loudness A, pulse rate r, f_{min}, f_{max}, maxHops, minHops, abandonment rate P_a, and resource constraints
Output: Best solution S_{best} and its fitness
Generate host capacities (RAM, storage, CPU)
Assign random resource requirements to each VM
Initialize population:
Generate two $H \times V$ binary matrices:
    $bats[i,j,k], nests[i,j,k] \in \{0,1\}$
    (VM k assigned to host j)
for each nest and bat **do**
    Randomly assign each VM to one host
end for
Validate VM assignments and adjust overloaded
for $t = 1$ to N_{iter} **do**
                                ▷ cuckoo search phase
    for each nest **do**
        Evaluate fitness via Eq. (13)
        Update S_{best} if improved
        if $rand > P_a$ **then**
            Apply Lévy flight for exploration
        else
            Replace P_a worst nests with new random ones
        end if
        Validate VM assignments and adjust overloaded hosts
    end for
                                ▷ bat algorithm phase
    for each bat **do**
        Generate frequency $f \in [f_{min}, f_{max}]$
        Update VM assignments via frequency-guided jumps
        Evaluate fitness using Eq. (13)
        if fitness improved **then**
            Update global best solution
        end if
        if $rand > r$ **then**
            Perform local random walk around best solution
            Fine-tune using A-scaled adjustment
        end if
        if $rand < A$ **then**
            Decrease A and increase r
        end if
        Validate VM assignments and adjust overloaded hosts
    end for
                                ▷ Fitness comparison
    Compare best fitness of bats and nests
    Keep best individual for next iteration
end for
End

placements or replaces the worst-performing nests with fresh solutions. All placements are then validated to avoid host overloads.

Next comes the BA phase, where each bat adjusts its solution using frequency-guided jumps. If a solution outperforms the current best, it becomes the new reference point S_{best} for bats. Bats may also perform a local random walk around the best-known solution, fine-tuning it using their loudness parameter A. Over time, the noise decreases while pulse rate r increases, gradually shifting their behavior from exploration to exploitation. VM placements are re-validated after each update.

Finally, in the fitness comparison phase, the best solutions from both algorithms are compared. The one with the best fitness is retained for the next iteration, ensuring that the algorithm progresses by leveraging the strengths of both exploration (cuckoo) and exploitation (bat). This collaborative approach aims to balance diversity and refinement, guiding the search toward optimal placements of virtual machines.

5. Results and Experiments

To evaluate the proposed algorithm, we performed a series of simulations using both Matlab and CloudSim. Matlab experiments focused on implementing the core optimization logic and comparing the proposed hybrid approach with several well-known metaheuristic algorithms: bat algorithm (BA), cuckoo search algorithm (CSA), ant colony optimization (ACO), particle swarm optimization (PSO), and non-dominated sorting genetic algorithm (NSGA-II). Each experiment was carried out on a set-up consisting of 20 physical hosts, while the number of virtual machines was varied over four scenarios between 20, 50, 100, and 200.

To further test the scalability of the approach, we conducted additional experiments using CloudSim, in collaboration with the (HPC) laboratory at the University of Trento. Their cluster environment provided the computational power needed to simulate large-scale cloud data centers and compare the HCS-BA algorithm with its standalone variants (BA and CSA) under realistic load conditions. The experimental setup also incorporated heterogeneity at the VM level. Although all hosts shared identical configurations (each with 64 GB of RAM, 100 GB of storage, and 10 000 MIPS), the VMs were intentionally configured with CPU capacities of (1 000 MIPS), RAM (2 to 4 GB) and storage (5 to 10 GB).

The proposed HCS-BA algorithm handles heterogeneity naturally by evaluating the quality of each solution. Since every virtual machine has different resource requirements, the fitness function checks how well these demands are met by the available capacities of each host.

To assess the effectiveness of the HCS-BA algorithm, we measured the following key performance indicators:

- Energy consumption – the total energy usage of active hosts after VM placement,

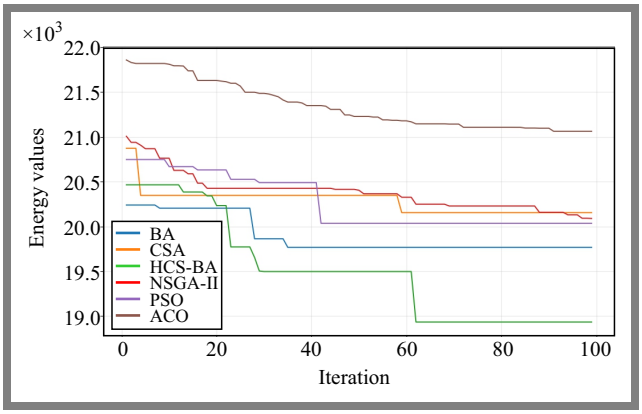

Fig. 3. Energy consumption over iterations for the 200 VM set.

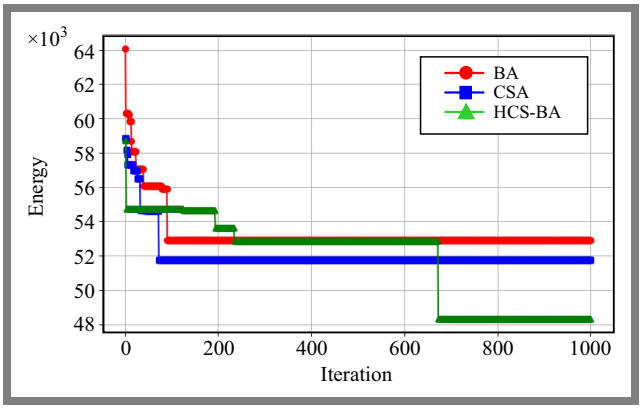

Fig. 4. Energy consumption over iterations for 1000 VMs.

- Resource utilization, the percentage of CPU, RAM, and storage utilization across all hosts,
- Fitness score – the optimized value that balances energy efficiency and resource utilization.

5.1. Energy Consumption

As illustrated in the results, the HCS-BA algorithm consistently outperforms all other compared methods, including CSA, BA, ACO, PSO, and NSGA-II, by achieving significantly lower energy consumption and faster convergence rates. Figure 3 highlights how HCS-BA rapidly stabilizes at an optimal energy level. Meanwhile, the CloudSim-based scalability test presented in Fig. 4 further confirms the robustness of the approach even as the system scales, since HCS-BA maintains its advantage over the standalone CSA and BA, proving both its adaptability and effectiveness in large-scale cloud environments.

5.2. Energy Consumption over Different VMs

As shown in Fig. 5, the HCS-BA algorithm provides strong and consistent performance across all VM set sizes. Although ACO slightly outperforms it in smaller configurations (20, 50, and 100 VMs), HCS-BA remains highly competitive, maintaining low energy consumption levels. However, as the environment scales to 200 VMs, HCS-BA clearly takes the lead, outperforming all other algorithms including CSA, BA, PSO, and NSGA-II. This demonstrates that the hybrid

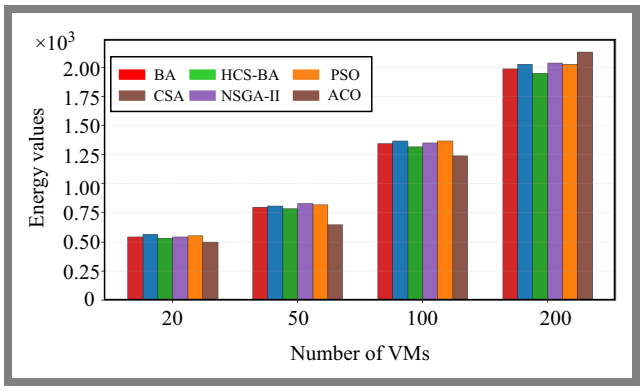

Fig. 5. Energy consumption on different VM set.

approach becomes increasingly effective as the system grows, showcasing its ability to efficiently handle larger and more complex cloud workloads.

5.3. Fitness Score

The fitness function analysis shown in Fig. 6 highlights how each algorithm converges over 100 iterations. CSA shows a quick initial drop in fitness values, but soon levels off at a higher point, suggesting that it settles for less optimal solutions due to its stronger focus on global exploration. In contrast, BA improves more steadily, showing good local search ability but slower overall convergence.

The hybrid HCS-BA, however, combines the strengths of both achieving the lowest fitness values by approximately iteration 60 and converging much faster. This demonstrates its effective balance between exploration and exploitation, leading to more accurate and energy-efficient virtual machine placement in cloud environments. Overall, HCS-BA not only outperforms standalone CSA and BA, but also surpasses other approaches like PSO, ACO, and NSGA-II, demonstrating its robustness, scalability, and superior optimization capability in large-scale scenarios.

Figure 7 presents the fitness values obtained from the CloudSim simulations used to evaluate the scalability of the algorithms. It clearly shows that the HCS-BA algorithm achieves lower fitness values compared to the standalone BA and CSA approaches, demonstrating its superior optimization capability. As the iterations progress, the HCS-BA converges more efficiently, maintaining stability even as the problem scale increases. This confirms that HCS-BA not only scales effectively in larger cloud environments, but also consistently delivers better fitness results.

Figure 7 also shows how well the algorithms scale. It is clear that the HCS-BA algorithm performs better, reaching better fitness values than BA and CSA. As the iterations progress, HCS-BA converges more smoothly and efficiently, showing that it can handle larger and more complex workloads without losing performance.

5.4. Fitness Score over Different VMs

As shown in Fig. 8, the HCS-BA algorithm maintains strong performance across all VM set sizes, achieving lower fitness

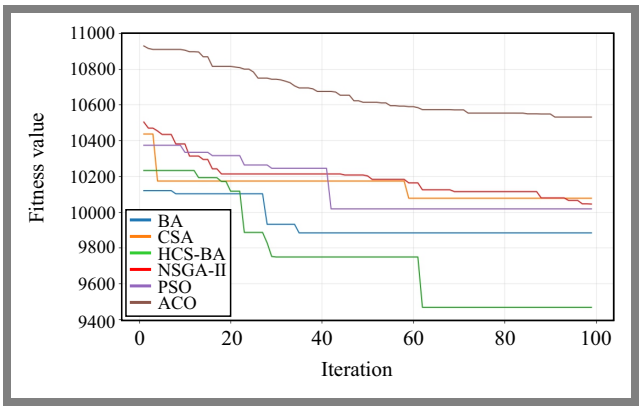

Fig. 6. Fitness score over iterations for a 200 VM set.

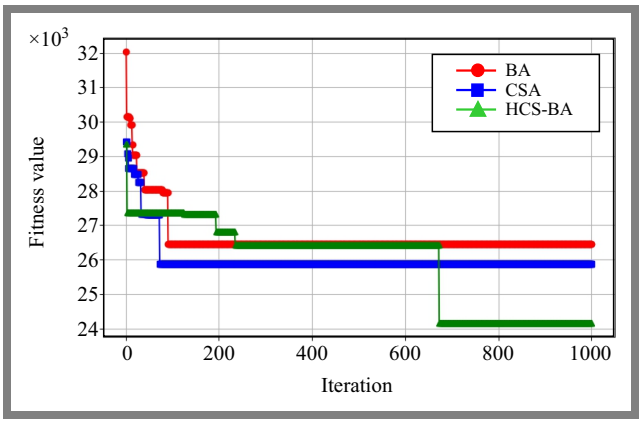

Fig. 7. Fitness score over iterations for a 1000 VM set.

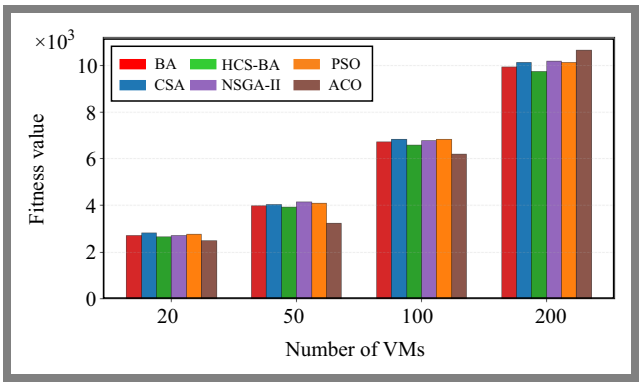

Fig. 8. Fitness score over different sizes of VM set size.

scores that reflect its efficient optimization behavior. Although ACO performs slightly better in smaller configurations (20, 50, and 100 VMs), HCS-BA remains highly competitive and stable. However, as the number of VMs increases to 200, HCSBA clearly stands out, outperforming all other algorithms, including CSA, BA, PSO, and NSGA-II. This demonstrates the impressive scalability and ability to deliver consistent high-quality results even as the complexity of the cloud environment increases.

5.5. Resource Utilization

Figure 9, generated using Matlab, compares resource utilization across different algorithms. Among them, NSGA-II achieves the highest resource utilization, outperforming all

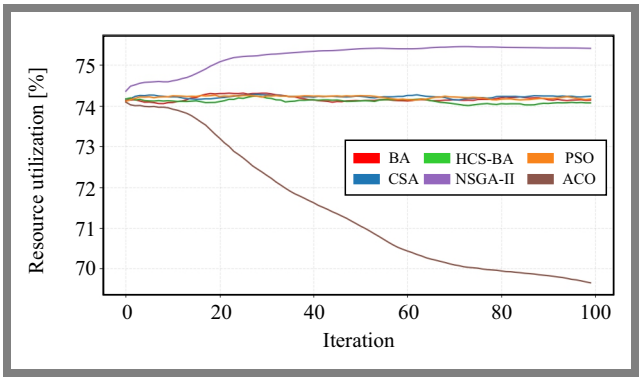

Fig. 9. Resource utilization over iterations for a 200 VM set.

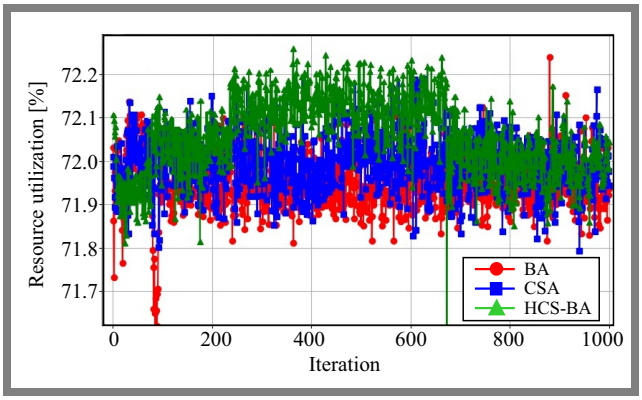

Fig. 10. Resource utilization over iterations for a 1000 VM set size.

other algorithms. HCS-BA, however, maintains a strong and stable performance, closely following PSO and its standalone variants (BA and CSA). On the other hand, ACO records the lowest utilization, indicating weaker efficiency in managing host resources. The CloudSim output presented in Fig. 10 shows that HCS-BA consistently achieves the highest resource utilization and remains stable throughout the simulation, outperforming both the standalone BA and CSA. This consistency demonstrates that HCS-BA scales effectively.

Several previous studies have shown notable energy savings. For example, in paper [13], a medium-scale data center achieved an energy improvement of 4.92%, while a large-scale one achieved 3.69%. Similarly, in [20], a reduction in energy consumption of 7% and 12% was reported using the non-dominated vector generation (ONVG) and spacing methods, respectively.

Another work [21] reported decreases of 28%, 27%, 24%, and 1% for four different algorithms. Although the mean resource utilization in some of these studies was slightly lower than the best performing baselines, they still outperformed many comparative methods.

In this work, the proposed HCS-BA shows that it consistently outperforms all other approaches in terms of energy efficiency. Compared to BA, CSA, NSGA-II, PSO, and ACO, HCS-BA achieves energy reductions of approximately 1.99%, 3.87%, 4.36%, 3.81%, and 8.64%, respectively. It also improves resource utilization by 4% compared to ACO. While HCS-BA shows performance comparable to BA, CSA, and PSO, it struggles when evaluated against NSGAII. Unlike previous studies that focused primarily on energy reduction, the proposed approach emphasizes maintaining high resource utilization efficiency, demonstrating a balanced improvement in both energy efficiency and system performance.

6. Conclusions

The proposed HCS-BA algorithm demonstrates significant improvements in optimizing VMP in heterogeneous cloud environments. Through comprehensive simulations, the hybrid algorithm consistently outperformed standalone CSA and BA and even recent VMP approaches in key performance metrics.

The HCS-BA approach achieved superior results in minimizing energy consumption, maximizing resource utilization, and converged faster to optimal solutions, validating its ability to effectively balance exploration and exploitation. While the proposed HCS-BA algorithm has demonstrated significant improvements in VMP, there are several avenues for future research to further enhance its effectiveness, such as scalability for large-scale cloud data centers, integration with machine learning techniques, and even turning it into a multi-objective optimization problem.

References

[1] P.D. Bharathi, P. Prakash, and V.K.K. Muppavarapu, "Virtual Machine Placement Strategies in Cloud Computing", *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, Vellore, India, 2017 (https://doi.org/10.1109/IPACT.2017.8244949).

[2] X.-F. Liu *et al.*, "An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing", *IEEE Transactions on Evolutionary Computation*, vol. 22, pp. 113–128, 2018 (https://doi.org/10.1109/TEVC.2016.2623803).

[3] M. Masdari, S.S. Nabavi, and V. Ahmadi, "An Overview of Virtual Machine Placement Schemes in Cloud Computing", *Journal of Network and Computer Applications*, vol. 66, pp. 106–127, 2016 (https://doi.org/10.1016/j.jnca.2016.01.011).

[4] S. Mejahed and M. Elshrkawey, "A Multi-objective Algorithm for Virtual Machine Placement in Cloud Environments Using a Hybrid of Particle Swarm Optimization and Flower Pollination Optimization", *PeerJ Computer Science*, vol. 8, art. no. 834, 2022 (https://doi.org/10.7717/peerj-cs.834).

[5] A. Alashaikh, E. Alanazi, and A. AlFuqaha, "A Survey on the Use of Preferences for Virtual Machine Placement in Cloud Data Centers", *ACM Computing Surveys*, vol. 54, pp. 1–39, 2020 (https://doi.org/10.1145/3450517).

[6] F. Lopez-Pires and B. Baran, "Virtual Machine Placement Literature Review", *arXiv*, 2015 (https://doi.org/10.48550/arXiv.1506.01509).

[7] M. Abdel-Basset, L. Abdle-Fatah, and A.K. Sangaiah, "An Improved Levy Based Whale Optimization Algorithm for Bandwidth-efficient Virtual Machine Placement in Cloud Computing Environment", *Cluster Computing*, vol. 22, pp. 8319–8334, 2019 (https://doi.org/10.1007/s10586-018-1769-z).

[8] H.O. Salami, A. Bala, S.M. Sait, and I. Ismail, "An Energy-efficient Cuckoo Search Algorithm for Virtual Machine Placement in Cloud Computing Data Centers", *The Journal of Supercomputing*, vol. 77, pp. 13330–13357, 2021 (https://doi.org/10.1007/s11227-021-03807-3).

[9] A. Gopu and N.N. Venkataraman, "Virtual Machine Placement Using Multi-objective Bat Algorithm with Decomposition in Distributed Cloud: MOBA/D for VMP", *International Journal of Applied Metaheuristic Computing*, vol. 12, pp. 62–77, 2021 (https://doi.org/10.4018/IJAMC.2021100104).

[10] M.A. Al-Abaji, "A Literature Review of Cuckoo Search Algorithm", *Journal of Education and Practice*, vol. 11, 2020 (https://doi.org/10.7176/JEP/11-8-01).

[11] X.-S. Yang, "Bat Algorithm: Literature Review and Applications", *International Journal of Bio-Inspired Computation*, vol. 5, pp. 141–149, 2013 (https://doi.org/10.1504/IJBIC.2013.055093).

[12] A.S. Abohamama and E. Hamouda. "A Hybrid Energy Aware Virtual Machine Placement Algorithm for Cloud Environments", *Expert Systems with Applications*, vol. 150, art. no. 113306, 2020 (https://doi.org/10.1016/j.eswa.2020.113306).

[13] F. Alharbi *et al.*, "An Ant Colony System for Energy-efficient Dynamic Virtual Machine Placement in Data Centers", *Expert Systems with Applications*, vol. 120, pp. 228–238, 2019 (https://doi.org/10.1016/j.eswa.2018.11.029).

[14] S. Walton, O. Hassan, K. Morgan, and M.R. Brown, "Modified Cuckoo Search: A New Gradient Free Optimisation Algorithm", *Chaos, Solitons & Fractals*, vol. 44, pp. 710–718, 2011 (https://doi.org/10.1016/j.chaos.2011.06.004).

[15] A.K. Singh, S.R. Swain, D. Saxena, and C.-N. Lee, "A Bio-inspired Virtual Machine Placement Toward Sustainable Cloud Resource Management", *IEEE Systems Journal*, vol. 17, pp. 3894–3905, 2023 (https://doi.org/10.1109/JSYST.2023.3248118).

[16] D.-M. Zhao, J.-T. Zhou, and K. Li, "An Energy-aware Algorithm for Virtual Machine Placement in Cloud Computing", *IEEE Access*, vol. 7, pp. 55659–55668, 2019 (https://doi.org/10.1109/ACCESS.2019.2913175).

[17] E. Barlaskar, Y.J. Singh, and B. Issac, "Enhanced Cuckoo Search Algorithm for Virtual Machine Placement in Cloud Data Centers", *International Journal of Grid and Utility Computing*, vol. 9, pp. 1–17, (https://doi.org/10.1504/IJGUC.2018.090221).

[18] P. Krishnmoorthy, "Performance Analysis of Hybrid Bat Algorithm and Cuckoo Search Algorithm HB-CSA for Task Scheduling in Mobile Cloud Computing", *SSRN Electronic Journal*, 2021 (https://doi.org/10.2139/ssrn.3997784).

[19] X.-S. Yang and S. Deb, "Cuckoo Search via Levy Flights", *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, Coimbatore, India, 2009 (https://doi.org/10.1109/NABIC.2009.5393690).

[20] A. Gopu *et al.*, "Energy-efficient Virtual Machine Placement in Distributed Cloud Using NSGA-III Algorithm", *Journal of Cloud Computing*, vol. 12, art. no. 124, 2023 (https://doi.org/10.1186/s13677-023-00501-y).

[21] X. Ye, Y. Yin, and L. Lan, "Energy-efficient Many-objective Virtual Machine Placement Optimization in a Cloud Computing Environment", *IEEE Access*, vol. 5, pp. 16006–16020, 2017 (https://doi.org/10.1109/ACCESS.2017.2733723).

Sifeddine Benflis, Ph.D. Student
Department of Computer Science
https://orcid.org/0009-0004-9075-7820
E-mail: sif.benflis@univ-batna2.dz
University of Batna 2, Batna, Algeria
https://univ-batna2.dz

Sonia-Sabrina Bendib, Assoc. Prof.
Department of Computer Science
https://orcid.org/0000-0003-4674-6185
E-mail: ss.bendib@univ-batna2.dz
University of Batna 2, Batna, Algeria
https://univ-batna2.dz

Sedrati Maamar, Assoc. Prof.
Department of Computer Science
https://orcid.org/0000-0003-0444-0399
E-mail: m.sedrati@univ-batna2.dz
University of Batna 2, Batna, Algeria
https://univ-batna2.dz

Fatima Z. Cherhabil, Ph.D. Student
Department of Computer Science
https://orcid.org/0000-0002-6937-5092
E-mail: f.cherhabil@univ-batna2.dz
University of Batna 2, Batna, Algeria
https://univ-batna2.dz

Hanane Merouani, Ph.D. Student
Department of Computer Science
https://orcid.org/0009-0008-0669-8442
E-mail: hanene.merouani@univ-batna2.dz
University of Batna 2, Batna, Algeria
https://univ-batna2.dz