



Accelerated computation of the genetic algorithm for energy-efficient virtual machine placement in data centers

Zhe Ding¹ · Yu-Chu Tian¹ · You-Gan Wang^{2,5} · Wei-Zhe Zhang³ · Zu-Guo Yu⁴

Received: 26 October 2021 / Accepted: 11 October 2022 / Published online: 4 November 2022
© The Author(s) 2022

Abstract

Energy efficiency is a critical issue in the management and operation of cloud data centers, which form the backbone of cloud computing. Virtual machine (VM) placement has a significant impact on energy-efficiency improvement for virtualized data centers. Among various methods to solve the VM-placement problem, the genetic algorithm (GA) has been well accepted for the quality of its solution. However, GA is also computationally demanding, particularly in the computation of its fitness function. This limits its application in large-scale systems or specific scenarios where a fast VM-placement solution of good quality is required. Our analysis in this paper reveals that the execution time of the standard GA is mostly consumed in the computation of its fitness function. Therefore, this paper designs a data structure extended from a previous study to reduce the complexity of the fitness computation from quadratic to linear one with respect to the input size of the VM-placement problem. Incorporating with this data structure, an alternative fitness function is proposed to reduce the number of instructions significantly, further improving the execution-time performance of GA. Experimental studies show that our approach achieves 11 times acceleration of GA computation for energy-efficient VM placement in large-scale data centers with about 1500 physical machines in size.

Keywords Genetic algorithm · Fitness function · Data center · Virtual machine placement · Energy efficiency

1 Introduction

Data centers are the backbone of cloud computing. The increasing global availability of cloud services requires the support of a massive number of data centers. As a result, an increasing demand for electricity becomes inevitable to power the data centers. Already, data centres use an estimated 200 terawatt hours (TWh) per annum, accounting for

1% of global electricity demand [21]. According to the Monthly Energy Review May 2016 from the U.S. Energy Information Administration (EIA), in 2013, the total cost of the energy consumption of the data centers in the U.S. was \$13 billion, representing 2.4% of the total electricity consumption. Detailed statistics showed that about 26% of the data center energy consumption were for servers and storage [9]. At current efficiency levels of hardware

✉ Yu-Chu Tian
y.tian@qut.edu.au

Zhe Ding
zhe.ding@hdr.qut.edu.au

You-Gan Wang
you-gan.wang@acu.edu.au

Wei-Zhe Zhang
wzzhang@hit.edu.cn

Zu-Guo Yu
yuzuguo@aliyun.com

² School of Mathematical Sciences, Queensland University of Technology, Brisbane, QLD 4001, Australia

³ School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

⁴ Key Laboratory of Intelligent Computing and Information Processing of the Ministry of Education of China and Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan 411105, Hunan, China

⁵ Institute for Learning Sciences and Teacher Education, Australian Catholic University, Brisbane, QLD 4000, Australia

¹ School of Computer Science, Queensland University of Technology, GPO Box 2434, Brisbane, QLD 4001, Australia

servers, the dramatically increasing demand of energy consumption in data centers would require to build 50 additional large power plants each year [22]. Thus, improving the energy efficiency of data centers becomes critical to the management and operation of data centers.

Cloud services are usually provisioned through virtualization. Thus, effective management of virtual resources such as virtual networks [37] and virtual machines (VMs) [3, 7] is an essential requirement in cloud data centers. Among a number of factors such as air cooling [15], VM placement to physical machines (PMs) plays an important role in the improvement of energy efficiency for large-scale data centers [3, 7]. As a large portion of the energy cost of data centers is to power PMs [4], it is beneficial to minimize their energy consumption through optimized VM placement [1, 20, 35]. Recent reports reveal that a significant energy saving of over 20% can be achieved from improved VM placement for data centers [3, 5]. This motivates our research and development of power-aware VM-placement strategies for further reducing energy consumption in data centers.

In spite of the improvement in the energy efficiency of data centers, existing solutions to VM placement are mainly derived from simple heuristic strategies, e.g., the First Fit Decrease (FFD) Algorithm and meta-heuristic evolutionary computation such as the Genetic Algorithm (GA) [5]. While FFD runs very fast to deliver a feasible solution, GA solves the VM-placement problem better in terms of the energy efficiency of data centers. GA is computationally demanding and runs too slowly, particularly for large-scale data centers. This limits its application in large-scale systems and scenarios where a fast VM-placement solution of good quality is required. Consequently, the energy efficiency of VM placement is compromised in the management and operations of data centers.

Our work in this paper presents our progress in accelerating GA computation for energy-efficient VM placement in data centers. The main contributions of the paper include:

- (1) An insight is developed into the understanding where the time is spent the most in the GA computation for energy-efficient VM placement. It reveals that the GA execution time is mostly spent on fitness evaluation. Thus, the most effective way to accelerate GA is to improve its fitness computation.
- (2) A new data structure is designed, which extends our previous study on GA acceleration [12], for energy-efficient VM placement. This reduces the complexity of the fitness evaluation from quadratic to linear one with respect to the input size of the VM-placement problem.
- (3) An alternative fitness function is proposed to further speed up GA computation without affecting the energy efficiency results of VM placement. It

improves the execution-time performance of GA through a significantly reduced number of instructions in GA computation. Experimental results show that our approach achieves 11 times acceleration of GA computation for energy-efficient VM placement in large-scale data centers with about 1, 500 PMs. This makes GA more practical in data center applications.

The paper is organized as follows: Notations used in the paper are listed in Table 1. Section 2 reviews related work and motivates the research of this paper. Section 3 reveals that GA consumes most of its time in fitness computation. Extending our previous study, Sect. 4 designs a new data structure for complexity reduction in fitness computation. An alternative fitness function is proposed in Sect. 5 for energy-efficient VM placement. Section 6 compares various fitness functions theoretically in terms of required CPU instructions. Simulation experiments are conducted in Sect. 7 to demonstrate our approach. Finally, Sect. 8 concludes the paper.

2 Related work

The problem of energy-efficient VM placement is a combinatorial optimization problem, which can be described as bin packing [3, 13]. Theoretically, an optimal solution to such a problem can be derived through an exhaustive search. However, such optimization problems are non-deterministic polynomial-time hard (NP-hard). They demand a significant computing effort for an optimal solution when the problem size is large. Consider placing 500 VMs to 100 PMs in a small-scale data center. The total number of combinations is 100^{500} . Assume 10 floating point operations are required in checking each of these combinations for energy optimization. Then, a total number of $10^{1,001}$ floating-point operations would be executed for an exhaustive search. If one of the fastest supercomputers in the world, the Sunway TaihuLight, is used for the computation at its Linpack Performance (Rmax) of 93, 014.6 Tflops, we would have to wait for more than 3.4×10^{976} years for a global optimal solution [13]. Thus, the exhaustive search technique is not practically viable for solving the energy-optimization problem, motivating various heuristic strategies for VM placement.

FFD is commonly used for heuristic VM placement. It is effective in dealing with general bin-packing problems like virtual resource management [30]. Recently, an advanced FFD algorithm is implemented for VM placement [38]. Although it does not aim at the energy efficiency of data centers, it can be easily adopted for energy-efficient VM placement when PMs are sorted in terms of energy efficiency. Similar to FFD, best-fit-decreasing (BFD) is another heuristic algorithm effective in dealing with bin-

Table 1 Notations and symbols

α	A parameter to characterize the shape of P_{cpu}
α_j	α of the j th PM
δ	Hypothesized value for statistical tests
C	The complexity of an algorithm
C_{alg1}	Complexity of Alg. 1 (old fitness on old data struc.)
C_{alg2}	Complexity of Alg. 2 (old fitness on new data struc.)
C_{alg3}	Complexity of Alg. 3 (new fitness on new data struc.)
E	The total energy consumption of a data center
f	CPU frequency
i, j	Indices for the i th VM and the j th PM, receptively
k	Index to indicate the k th time slot
n	The input size of a computing problem
N_{apm}	The number of active PMs
N_{fit}	The total number of times to compute fitness function
N_{fit1}	The number of times of fitness computing in a generation
N_{gen}	The number of generations
N_{ic-fit}	CPU instruction count on the fitness computation in GA
$N_{ic-fit1}$	CPU instruction count in a single fitness
$\hat{N}_{ic-fit1}$	Estimated $N_{ic-fit1}$
N_{ic}	CPU instruction count for calculation of a single fitness
N_{ic1}	Instruction count for Alg. 1 (old fitness on old data structure)
N_{ic2}	Instruction count for Alg. 2 (old fitness on new data structure)
N_{ic3}	Instruction count for Alg. 3 (new fitness on new data structure)
N_{pm}	The total number of PMs
N_{pop}	Population size
N_{slot}	The number of time slots (intervals)
N_{vm}	The number of VMs
\bar{N}_{vm-1pm}	The average number of VMs on a single PM
P	Power for a CPU
$p^{(max)}$	CPU power at full load
$p^{(min)}$	CPU power at idle state
$P_{apm}(0, k)$	Power of the last active PM in the k th time slot
$P_{pm}(j)$	Power for the j th PM
$P_{pm}(j, k)$	Power of the j th PM in the k th time slot
p	Probability value in statistical tests
T	Value of t -distribution in t -tests
T_{crit}	Critical value in Wilcoxon’s test
T_{exec}	Execution time of a GA
T_{fit}	Total computation time for the fitness function
t_k	The duration of the k th time slot (interval)
u_{cpu}	CPU utilization
$u_{pm}(j, k)$	Utilization of of the j th PM in the k th time slot
$u_{pm}(j)$	Utilization of the j th PM
$u_{vm}(i, j)$	Utilization of the i th VM in the j th PM
$u_{pm}(i, j, k)$	Utilization of the i th VM on the j th PM in the k th time slot
V	The set of all VMs, $V = \bigcup_{i=1}^{N_{vm}} v_i$
v_i	The i th VM, $i = 1, \dots, m$
w	The minimum of the sums of positive and negative ranks in Wilcoxon’s test

packing problems. It has been adopted in VM placement for energy optimization [25]. Its implementation has been based on a resource utilization ratio rather than real utilization measures.

With meta-heuristics, GA has also been investigated for optimized management of virtualized resources in data centers [23, 36]. Searching a bigger space than FFD and BFD in every step, GA gives a higher-quality solution than either FFD or BFD at the cost of an increased execution time. To improve the execution-time performance, the computational demand of GA has been trimmed while still maintaining the quality of solution [28].

As another meta-heuristic method, Ant Colony Optimization (ACO) has been studied for multi-objective, cloud-work flow scheduling [6]. Through an analysis of five types of real-world work flows, ACO has been shown to outperform Particle Swarm Optimization (PSO) and Non-dominated Sorting Genetic Algorithm-II (NSGA-II). Recently, ACO has also been investigated with a refined formulation and new heuristics for energy-efficient VM placement [2, 3]. As both ACO and GA belong to meta-heuristic methods, they share many similar features. For example, with iterations of many generations, both ACO and GA calculate a fitness function to evaluate whether or not a solution has improvement. Our work in this paper focuses on GA as a typical example of meta-heuristic optimization for energy-efficient VM placement.

Relevant to energy efficiency of data centers is VM consolidation. The work presented in [16] indicates that VM consolidation saves energy with the π -estimation benchmark program running. However, such energy-aware VM consolidation often changes the behaviors of task execution and VM operations. It needs to be considered together with Quality of Service (QoS) and Service Level Agreements (SLAs). Our work in this paper does not consider VM consolidation directly. The number of VM migrations can be reduced through a profile-guided, three-phase VM-placement framework [13].

2.1 Implementation of GA

GA is now implemented in multiple layers in the virtual resource management of data centers [32, 33]. It is designed for profile-guided application assignment to VMs. Similar to VM placement to PMs, application assignment to VMs is also a type of bin-packing problems. The concepts of both profiles and the meta-heuristic GA help reduce the energy consumption of data centers. Thus, GA is a promising tool for energy optimization in data centers through virtual resource management.

A critical problem of GA, however, is its slow execution for a quality solution with improved energy efficiency of data centers [14]. For example, GA runs much slowly than

FFD [10, 28]. For a VM-placement problem with N_{vm} VMs, the computational complexity of FFD is $O(N_{vm} \cdot \log N_{vm})$ [10]. In comparison, the initial sorting part of the Non-dominated Sorting GA (NSGA) with a population size N_{pop} has the computational complexity of $O(N_{vm} N_{pop}^3)$ [11]. Thus, the computation of GA components, e.g., crossover and mutation, over many generations for an energy-efficient VM placement in a data center is very time-consuming. Therefore, GA needs to be accelerated for its applications in more realistic data center scenarios.

Efforts have been made for decades to tune GAs [17, 29]. Reference [17] has discussed the control parameters of GA, e.g., population size, crossover rate, mutation rate, generation gap, scaling window, and selection strategy. The settings of these parameters are optimized for better performance [29], leading to an adaptive GA (AGA) with strengthened crossover and mutation. This inspires us to conduct a deep investigation into crossover and mutation for an accelerated GA in energy-efficient VM placement.

The population size of GA and its relationships with selection rate and chromosome length are investigated for the performance improvement of GA [18, 19]. From the analysis of these relationships, a compact GA is designed [19], which saves computational resources with a compressed selection rate and population size. These findings have been embedded into general GA implementations.

A recent work on GA-based virtual resource management of data centers focuses on GA's computational performance [28]. With energy cost as the fitness function, it uses the results from FFD as an initial (feasible) solution. Then, it employs the concept of decrease-and-conquer to simplify the GA computation. This leads to improved execution-time performance of GA for energy-efficient VM placement.

2.2 Technical gaps and motivation

GA has been applied to the VM-placement problem for energy-efficient data centers. Efforts have been made to enhance GA's performance, particularly in optimizing its parameters and choosing a better initial feasible solution. When a GA runs faster, an improved solution with more energy savings will be derived within a given period of time. However, a deep understanding about where the execution time is spent the most in the whole process of GA computation is yet to be developed. Such an understanding will enable further research and development of computationally efficient GA for energy-efficient VM placement to PMs in cloud data centers.

Our work in this paper reveals that the execution time of standard GA is mostly consumed in fitness computation. Therefore, our work focuses primarily on the computation

of fitness function. It is embedded with a new data structure extended from our previous study [12] to reduce the complexity of the fitness computation. Then, incorporating with this data structure, an alternative fitness function is proposed with much less computational demand. As a result, the execution-time performance of GA is significantly improved.

3 Insights into GA computation for energy-efficient VM placement

This section focuses on profiling the computation of standard GA, in terms of execution time, for energy-efficient VM placement. Through the profiling, we aim to develop some insights into how GA can be accelerated effectively.

3.1 The profiling of GA’s execution time

For energy-efficient VM placement, standard GA consists of a number of components. Some of these components execute sequentially, while some others run repeatedly. A general flowchart of standard GA is illustrated in Fig. 1. It is seen from this flowchart that GA starts with an initialization module, which includes data input, task assignment, and initial population. Then, it runs the selection module as tournament selection. If two parents are chosen successfully, go ahead to the crossover module, otherwise repeat the selection process. The crossover process is followed by mutation and individual validation. After that, check fitness

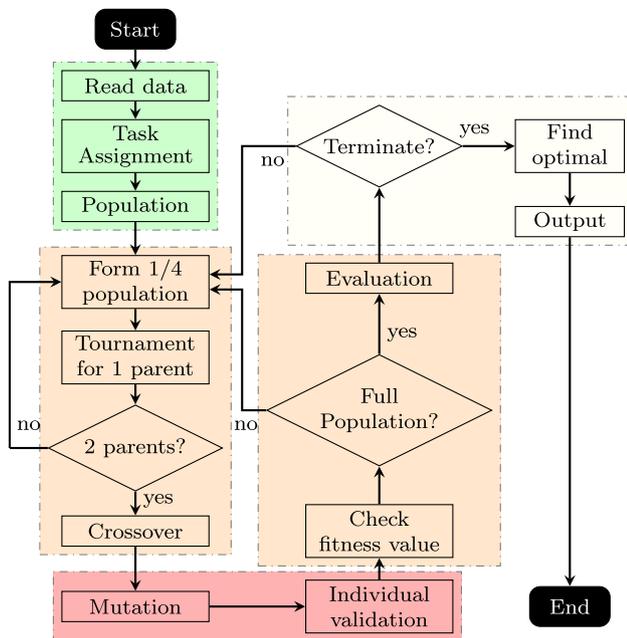


Fig. 1 The flowchart of implementing a standard GA for VM placement

value and population, and decide to go back to the population module or evaluation module. After evaluation, either go back to the population module for the next GA generation, or output results and terminate GA computation.

For the purpose of profiling GA’s execution time, the GA flowchart in Fig. 1 is simplified into Fig. 2. It is seen from Fig. 2 that the population component executes only once in the initialization of GA computation. The other three components, i.e., selection with crossover, mutation, and evaluation, execute repeatedly until the algorithm terminates with a satisfactory solution. Among these three components, Selection with Crossover, and Evaluation (SCE) are infused with a fitness function.

To understand where GA spends most of its time in its computation, we have profiled GA computation in terms of execution time for energy-efficient VM placement in data centers. For a typical scenario, which will be discussed later in our simulations, we have obtained the results depicted in the leftmost bar of Fig. 3.

A quantitative analysis of the results in the leftmost bar of Fig. 3 indicates that about 80% of the total execution time of GA are spent on SCE computation. Mutation consumes about 10% of the total execution time. Population consumes less than 10% of the total execution time.

Within SCE, fitness function-infused computation is dominant, consuming about 60% of the total execution time of GA as graphically indicated in leftmost bar of Fig. 3. Therefore, in order to improve the computational performance of GA in energy-efficient VM placement, we should focus on the simplification of fitness computation. From this understanding, we conduct a detailed analysis of fitness computation in GA.

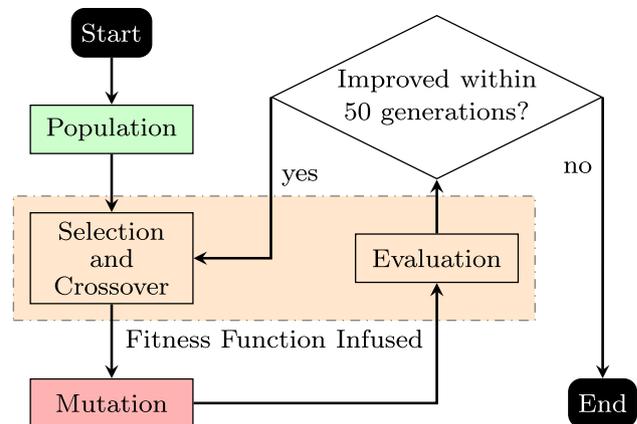


Fig. 2 The GA architecture infused with fitness function in Selection with Crossover, and Evaluation (SCE)

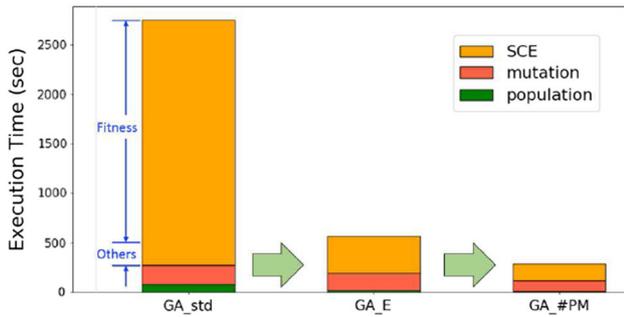


Fig. 3 An example of standard GA’s execution-time profiling, as shown in the leftmost bar. The data comes from a standard GA run of our experiments in Sect. 7. SCE means selection and crossover, and evaluation

3.2 Analysis of fitness computation

From the above analysis of GA profiling in terms of execution time, fitness computation dominates the computation in standard GA for energy-efficient VM placement. Let us investigate in detail how much computational effort the GA requires for fitness computation in the whole process of GA computation. Let N_{pop} denote GA’s population size. Then, for a new generation from N_{pop} individuals, $2N_{pop}$ parents need to be selected. If we use a tournament selection for crossover parents, each parent has to be selected among $N_{pop}/4$ randomly-chosen individuals from its previous generation. Each of the $N_{pop}/4$ individuals needs to calculate its fitness value. Therefore, the total number of fitness evaluations in the crossover of one generation is:

$$N_{fit1} = 2N_{pop} * N_{pop}/4 = N_{pop}^2/2 \tag{1}$$

Thus, for N_{gen} generations of GA, the total number of fitness evaluations in crossover is:

$$N_{fit} = N_{gen} * N_{fit1} = N_{gen} * N_{pop}^2/2 \tag{2}$$

For example, in a specific scenario, N_{pop} is set to be 64, and N_{gen} is about 500 on average. Then, Eq. (2) gives:

$$N_{fit} = 500 * 64^2/2 = 1,024,000$$

This is over a million fitness evaluations!

In Eq. (2), once selected, N_{pop} is a constant. Thus, N_{gen} is the only adjustable parameter to reduce N_{fit} . A faster convergence of GA is beneficial to reducing N_{fit} .

Next, we investigate how much computational effort the standard GA needs for fitness computation in terms of CPU instruction count N_{ic-fit} , i.e., the number of CPU instructions. Let T_{fit} denote the total execution time of the fitness computation in crossover. Also, denote the frequency and utilization of the CPU as f and u_{cpu} , respectively. Then, N_{ic-fit} of the overall fitness computation is:

$$N_{ic-fit} = f * T_{fit} * u_{cpu} \tag{3}$$

The CPU instruction count in each fitness evaluation is:

$$N_{ic-fit1} = N_{ic-fit}/N_{fit} \tag{4}$$

It follows from Eqs. (2) and (3) that:

$$N_{ic-fit1} = \frac{2 * f * T_{fit} * u_{cpu}}{N_{gen} * N_{pop}^2} \tag{5}$$

From Eq. (5), T_{fit} can be expressed as:

$$T_{fit} = \frac{1}{2} * N_{gen} * \frac{N_{pop}^2}{f * u_{cpu}} * N_{ic-fit1} \tag{6}$$

In Eq. (6), the value of the term $\frac{N_{pop}^2}{f * u_{cpu}}$ is determined by system configuration and hardware settings. Thus, N_{gen} and N_{fit1} are dominant factors for fitness computation. A faster convergence helps reduce N_{gen} , as mentioned previously. Another effective way to improve the computational performance is to reduce N_{fit1} . Consequently, the total execution time T_{exec} of GA can be reduced.

3.3 Insights into the GA computation

From the profiling of the GA computation in terms of execution time, it is understood that GA consumes the most of its computation time in fitness evaluation. Thus, the improvement of GA’s computational efficiency should focus on fitness computation. A simpler fitness function that is a good indicator of the energy efficiency of VM placement will help accelerate the GA computation. Meanwhile, an improved data structure that enables simplified GA computation will also help improve the computational efficiency of GA for energy-efficient VM placement.

Therefore, in the following, a new data structure that we recently proposed [12] is embedded into GA to reduce the complexity of fitness computation. Then, incorporating with this data structure, an alternative fitness function is proposed to reduce the CPU instruction count N_{fit1} . As a result, the GA computation is significantly accelerated for energy-efficient VM placement in data centers.

4 Embedding a new data structure into GA

4.1 Fitness computation in standard GA

In Standard GA for energy-efficient VM placement in data centers, the energy cost of a VM-placement plan is defined as the fitness function [13]. It is calculated from a power model of CPU with respect to CPU utilization [34]. The CPU power model is mathematically formulated as:

$$P = P^{(\max)} - \frac{P^{(\max)} - P^{(\min)}}{\exp(\alpha * u_{cpu})} \tag{7}$$

where P_{\max} represents the maximum power when the CPU is fully loaded (i.e., 100% utilization), P_{\min} represents the minimum power when the CPU is idle (i.e, 0% utilization), u_{cpu} represents the CPU utilization, and α is a constant that defines the shape of the power model curve. Figure 4 is a graphical demonstration of this power model [34].

Algorithm 1: Traditional fitness evaluation in standard GA for energy-efficient VM placement

Input: A VM-placement plan, including which PMs hosts which VMs
Output: The fitness value (energy consumption \mathbb{E})
Initialize: An empty PM utilization set

- 1 **foreach** *PM* in the given plan **do**
- 2 **foreach** *VM* in this *PM* **do**
- 3 Add the utilization of this VM to the located PM’s utilization;
- 4 Calculate the energy cost of this PM;
- 5 Add the energy consumption of this PM to this plan’s fitness value;
- 6 **return** A VM placement plan and its fitness value;

For the energy efficiency of data centers, we aim to minimize the total energy consumption \mathbb{E} over a certain period of time, subject to the constraints of CPU and memory capacity. Consider placing N_{vm} VMs to N_{apm} active PMs. For the k th time slot with length t_k , it follows from Eq. (7) that the power of the j th PM in the time slot is:

$$P_{pm}(j, k) = P_{pm}^{(\max)}(j) - \frac{P_{pm}^{(\max)}(j) - P_{pm}^{(\min)}(j)}{\exp(\alpha_j * u_{pm}(j, k))} \tag{8}$$

Thus, the constrained optimization problem over N_{slot} time slots is expressed as:

$$\left\{ \begin{array}{l} \min_{vm} \mathbb{E} = \sum_{k=1}^{N_{slot}} \sum_{j=1}^{N_{apm}} P_{pm}(j, k) * t_k \\ \text{s.t. } u_{pm}(j, k) = \sum_{i=1}^{N_{vm}} u_{pm}(i, j, k), \\ u_{pm}(i, j) \in \{u_{v_1}, u_{v_2}, \dots, u_{v_m}\}, \\ 0 \leq \forall u_{pm}(j) \leq 100\%, \\ \text{Memory constraints,} \end{array} \right. \tag{9}$$

where index j refers to each active PM ($j = 1, \dots, N_{apm}$); v_1, v_2, \dots, v_m refer to m VMs; and u_j is the CPU utilization of the j th PM.

In standard GA, the fitness function of this constrained optimization for the set $V = \bigcup_{i=1}^{N_{vm}} v_i$ of VMs hosted in N_{apm} active PMs over N_{slot} time slots is formulated as:

$$\mathbb{E} = \sum_{k=1}^{N_{slot}} \sum_{j=1}^{N_{apm}} P_{pm}(j, k) * t_k \tag{10}$$

The data structure currently used in the computation of standard GA is shown in Fig. 5. In this data structure, the PM list is used to directly collect VMs. Therefore, if we need to traverse all VMs, we have to traverse all PMs as well as the VMs within each PM. This leads to Algorithm 1. With two nested loops, Algorithm 1 has quadratic complexity.

4.2 Fitness update embedded with a new data structure

We design a new data structure that is extended from our recent study [12] to help reduce the (quadratic) complexity and problem size of traditional fitness computation in standard GA. The data structure is depicted in Fig. 6 [12]. In this data structure, VMs are linked with their addresses in PMs along with time and resource conditions. This VM-based data structure strictly matches the data structure of gene chromosome in GAs for VM-placement problems.

In the deployment of this data structure in GA, a VM placement plan is the gene of its relevant individual, and each VM with its corresponding address stands for a nucleotide of the gene. Then, during the crossover or mutation of a GA integrated with the this data structure, the program simply updates the address of each VM. Therefore, it becomes much more straight-forward and faster when deploying to a GA with this data structure than the standard data structure.

Now, the original two-dimensional data structure in Fig. 5 is replaced by two one-dimensional vectors in Fig. 6: a VM list whose elements are linked with PM IDs, and a PM list whose elements are linked with PM utilization. With this data structure, the process of fitness evaluation in GA is demonstrated in Algorithm 2.

Algorithm 2: Fitness evaluation under the adopted data structure [12] for energy-efficient VM placement

```

Input: A VM-placement plan, including each VM’s utilization and location
Output: The fitness value (energy consumption  $\mathbb{E}$ )
Initialize: An empty PM utilization set
1 foreach VM in the given plan do
2   | Add the utilization of this VM to the located PM’s utilization;
3 foreach PM utilization in the PM utilization set do
4   | if PM is not idle (i.e., utilization > 0) then
5     | | Calculate this PM’s energy consumption;
6     | | Add its energy consumption to this plan’s fitness value;
7 return A VM-placement plan and its fitness value;
    
```

Algorithm 2 has two sequential loops. The first loop calculates each PM’s utilization from all VMs that the PM hosts. The second loop computes the total energy cost from all active PMs. With these two sequential loops, the complexity of the algorithm is linear to the problem size.

It is worth mentioning that the calculation in Algorithm 1 includes all PMs in the data center (referring to N_{pm}). In comparison, the calculation in Algorithm 2 only includes active PMs (referring to N_{apm}), ignoring all idle PMs that are proposed to be powered off. In data centers, not all PMs are always kept on as $N_{apm} \leq N_{pm}$. Thus, the problem size is reduced, especially when we can switch off a large number of idle PMs for energy savings.

Consequently, Algorithm 2 reduces the complexity of Algorithm 1 from quadratic to linear one, and also reduces the problem size, adding the straight-forward deployment for genes, thus accelerating GA computation. This will be further discussed later in Sect. 6 from the theoretical perspective.

5 An alternative fitness function

General requirements for alternative fitness functions have been analyzed theoretically in a recent report [20]. Our work presented in this paper considers the requirements for an alternative fitness function from a different perspective:

- (1) First of all, an alternative fitness function is expected to be simpler in its expression than the standard one in Eq. (10);
- (2) Also, it should be a good indicator of the actual energy cost calculated from Eq. (10); and
- (3) Finally, it should be easily calculable from the adopted data structure shown in Fig. 6.

With these requirements in mind, let us start to search for a candidate of such an alternative fitness function.

We have observed from our comprehensive experiments that GA often pushes active PMs into high utilization (e.g.,

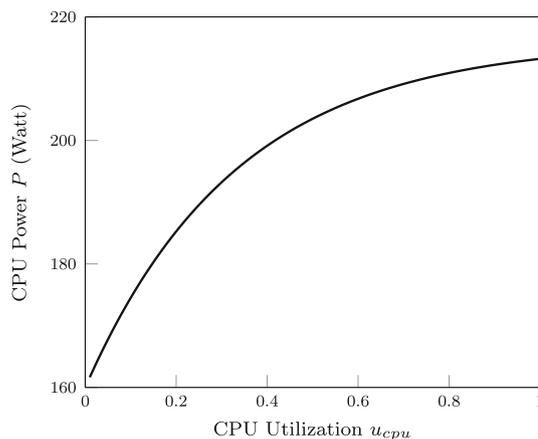


Fig. 4 A CPU power model for an Intel Xeon system with turbo boost on [34]

VM				VM45		...
		VM24		VM44		
	VM13	VM23	VM33	VM43		
	VM12	VM22	VM32	VM42	VM52	
	VM11	VM21	VM31	VM41	VM51	
PM	PM1	PM2	PM3	PM4	PM5	...

Fig. 5 Standard data structure for GA computation, in which VMs are stored in PMs

well above 95%) so that fewer active PMs are used, thus saving energy. The only exception is the last PM, which may not be fully filled with VMs. Fig. 7 depicts a plot of typical VM placement to over 1, 000 PMs. From this figure and Eq. (7), we have developed the following heuristic rule:

Heuristic Rule 1: In an optimized VM-placement plan, all active PMs are almost fully loaded, i.e.,

$$P_{pm}(j) \approx P_{pm}^{(max)}(j), j \in \{1, \dots, N_{apm}\} \tag{11}$$

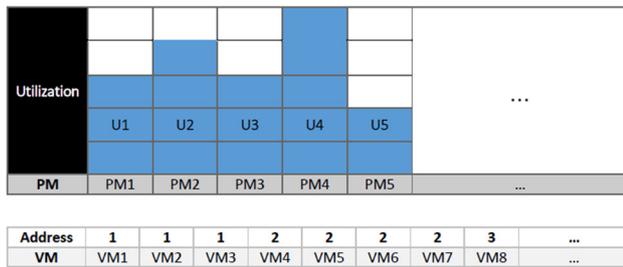


Fig. 6 New data structure extended from our previous study [12] for fitness computation, in which VMs are linked with their addresses in PMs along with time and utilization conditions

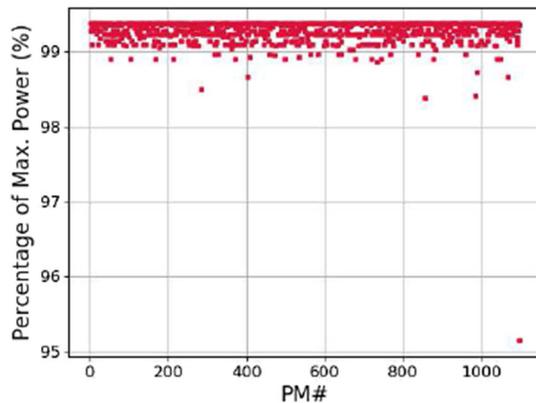


Fig. 7 A plot of typical GA VM placement to over 1, 000 PMs, all of which are almost fully loaded

Substituting Eq. (11) in Heuristic Rule 1 into Eq. (10) yields:

$$\mathbb{E} \approx \sum_{k=1}^{N_{slot}} \left[\sum_{j=1}^{N_{apm}-1} P_{pm}^{(max)}(j, k) + P_{pm}(0, k) \right] * t_k \tag{12}$$

where $P_{pm}(0, k)$ refers to the power of the last active PM that may not be fully loaded.

When all active PMs are almost fully loaded, the resulting \mathbb{E} from Eq. (12) is less related to the actual utilization of each PM. It is well characterized by the number of active PMs. In other words, \mathbb{E} is not strongly affected by actual $P(j, k)$ of each PM. Therefore, we have our second heuristic rule:

Heuristic Rule 2: In an optimized VM-placement plan, the resulting \mathbb{E} is characterized by the number of active PMs:

$$\mathbb{E} \underset{\sim}{\propto} N_{apm} \tag{13}$$

Using N_{apm} in Expression (13) as an alternative fitness function satisfies the three requirements stated at the beginning of this section: its expression is much simpler than the standard one in Eq. (10); it is a good indicator of the energy cost from Eq. (10); and finally it can be easily calculated from the adopted data structure shown in Fig. 6 simply by counting the number of active PMs.

With N_{apm} as the alternative fitness, Algorithm 3 is designed for fitness evaluation based on the embedded data structure in Fig. 6. We have the following observations:

- As in Algorithm 2, Algorithm 3 has two sequential loops. Thus, its complexity is also linear to the problem size.
- However, Algorithm 3 simply counts the number of active PMs. It uses much fewer CPU instructions than Algorithm 2, which calculates energy consumption from a nonlinear expression in Eq. (10).

Therefore, the execution-time performance of GA using our alternative fitness function with the adopted data structure is expected to be significantly improved. This improvement will be further analyzed in the next section. It will also be verified later through simulation experiments.

Algorithm 3: An evaluation of the alternative function N_{amp} for energy-efficient VM placement

Input: A VM-placement plan, including the corresponding PM list

Output: A fitness value (N_{apm} as an indicator of \mathbb{E})

Initialize: An empty PM counter N_{apm}

- 1 **foreach** *Planned VM* **do**
 - 2 Update corresponding active PM status;
 - 3 **foreach** *PM in the PM list* **do**
 - 4 **if** *PM utilization* $\neq 0$ **then**
 - 5 $N_{apm} = N_{apm} + 1$;
 - 6 **return** N_{apm} as this plan's fitness value;
-

6 Theoretical comparisons of fitness functions

This section provides theoretical comparisons among three types of fitness evaluations from Algorithms 1, 2 and 3:

- (1) The fitness evaluation from Algorithm 1 corresponds to traditional fitness in Equation (10) using the standard data structure shown in Fig. 5;
- (2) With the same traditional fitness in Eq. (10), Algorithm 2 uses the data structure in Fig. 6 adopted from [12]; and
- (3) The fitness computation from Algorithm 3 uses our alternative fitness N_{apm} in Eq. (13) and the adopted data structure in Fig. 6.

The two nested loops of Algorithm 1 imply quadratic complexity. Let N_{pm} denote the total number of all PMs. Also, let \bar{N}_{vm-1pm} denote the average number of VMs in each PM. Then, the complexity of Algorithm 1 is expressed as:

$$C_{alg1} \sim O(N_{pm} * \bar{N}_{vm-1pm}) \quad (14)$$

The two sequential loops of Algorithm 2 mean linear complexity. The first loop applies to N_{vm} VMs, while the second loop is for N_{apm} active PMs. Thus, the complexity of Algorithm 2 is expressed as:

$$C_{alg2} \sim O(N_{apm} + N_{vm}) \quad (15)$$

Algorithm 3 also has two sequential loops, each with the same purpose as that in Algorithm 2. Therefore, it has the same complexity as Algorithm 2, i.e.,

$$C_{alg3} \sim O(N_{apm} + N_{vm}) \quad (16)$$

Considering C_{alg1} , C_{alg2} and C_{alg3} , we have:

$$C_{alg1} > C_{alg2} \approx C_{alg3} \quad (17)$$

Both Algorithms 2 and 3 reduce the complexity of Algorithm 1 significantly from quadratic to linear one. A further analysis will show that Algorithm 3 is much simpler than Algorithm 2 in terms of the required number of CPU instructions.

For a single fitness evaluation, the instruction count is $N_{ic-fit1}$. Thus, the instruction count N_{ic} for an algorithm with complexity C is approximately:

$$N_{ic} \propto N_{ic-fit1} * C \quad (18)$$

Both Algorithms 1 and 2 need to execute a long list of CPU instructions for the floating-point computation of the non-linear energy function (10), while Algorithm 3 simply counts active PMs. Thus, conceptually, the $N_{ic-fit1}$ in

Algorithm 3 is much smaller than those from Algorithms 1 and 2.

Further considering the relationship in (17), we have the following relationships between instruction count N_{ic1} from Algorithm 1, N_{ic2} from Algorithm 2 and N_{ic3} from Algorithm 3:

$$N_{ic3} \ll N_{ic2} < N_{ic1} \quad (19)$$

Expression (19) clearly indicates that Algorithm 3 executes much faster than Algorithm 2 does. It also shows that both Algorithms 3 and 2 are faster than Algorithm 1, which is the Standard GA for energy-efficient VM placement. These results will be validated later with experimental simulations. Our simulations also show that, with Algorithm 3, the number N_{gen} of generations in GA is reduced, further speeding up GA computation.

7 Simulation experiments

Our experiments will test GA for energy-efficient VM placement under three types of fitness evaluations:

- Algorithm 1 for the Standard GA with the original fitness computation and data structure,
- Algorithm 2 for original fitness with the data structure adopted from [12], and
- Algorithm 3 for our alternative fitness embedded with this new data structure.

We give a VM-placement plan for each of the evaluated experimental scenarios. For each scenario, we evaluate the resulting energy consumption of the data centers with all idle PMs being powered off. We also record the execution time of the corresponding GA on our computation platform. Other parameters related to $N_{ic-fit1}$ and N_{gen} are evaluated as well.

An example, we take Google's Cluster-Usage Traces [8, 27] for our experimental tests. Because the data set is considerably huge, we extract a small part of the logs from the traces for demonstration. All data configurations are listed in Table 2.

Table 2 The input configuration from Google's data set

Scale	Input configuration			
	Start Time	Initial Job ID	Initial Task ID	#Items
Small	$9 * 10^8$	6251539840	379	680
Medium	$3.717 * 10^{11}$	6289303978	424	2242
Large	$1.8 * 10^9$	4028922835	429	5365

Table 3 Six types of VMs with normalized CPU capacity

CPU type	Huge	Large	Medium	Normal	Small	Tiny
CPU capacity	0.45	0.30	0.15	0.10	0.045	0.015

Three scales of data centers are tested in our experiments: small, medium, and large scales. Our small-scale data input refers to an private, enterprise-level cloud data center running hundreds of VMs on 100 PMs. Our medium-scale data input represents a private, university-level cloud data center running about two thousand VMs on 500 to 600 PMs. Our large-scale data input simulates a sector of a large, public cloud data center running thousands of VMs on over 1,000 PMs. All three scales of data inputs are extracted from Google’s cluster-usage traces.

It is worth mentioning that Google’s cluster traces only record tasks, not VMs. To use Google’s data set for VM-placement research, we assign these tasks to VMs prior to VM placement to PMs [26]. The task-assignment Algorithm 4 is taken from our previous work [12, 13]. It assigns each incoming task to a VM of proper size. The VMs in our experiments are designed with fixed sizes, which are listed

in Table 3. Fixing VM sizes is a common practice in the management of data centers, as in Amazon’s cloud.

After all tasks are assigned into VMs, we execute our GA for VM placement to PMs. Our GA implies a tournament selection strategy with a randomly-chosen 1/4 of the total population for each parent. It terminates when no further improvement is observed over 50 generations. Other GA parameters follow standard and popular configurations. For example, the GA population size is 64 individuals. The crossover and mutation rates are 50% and 0.15%, respectively. In addition, we use an First-Fit solution as an individual in the first generation of GA for a better initialization [28].

Our simulation experiments are conducted on a desktop computer. The computer is equipped with Intel Core 2 Q6700, 3.4 GHz CPU, and 16 GB DDR4 2666 MHz RAM. It runs Windows 10 Professional operating system. The input of our experiments is a comma separated-value (CSV) file extracted from Google’s Cluster-usage Traces as discussed previously. The outputs of our experiments include a VM-placement plan, the energy consumption of the data center, the number of GA generations, GA

Algorithm 4: Task assignment [12,13].

Input: All tasks (applications)
Output: A plan of application assignments to VMs
Initialize: An empty VM set
1 **foreach** *task* in all given tasks **do**
2 Assign this task to a VM of proper size;
3 Append this VM into the VM set;
4 Convert the VM set to the assignment plan;
5 **return** an application assignment plan;

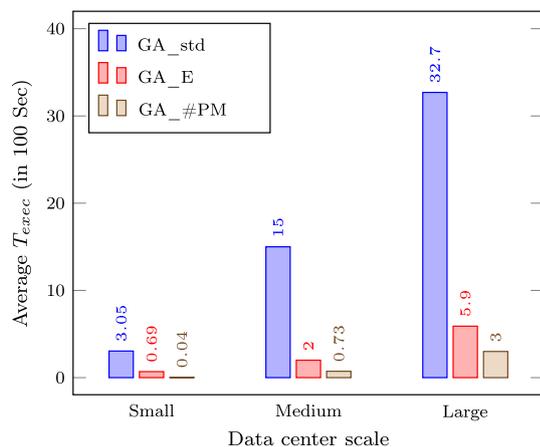


Fig. 8 The average execution time of GAs from Algorithm 1 (GA_std), Algorithm 2 (GA_E) and Algorithm 3 (GA_#PM) for small-, medium- and large-scale data centers

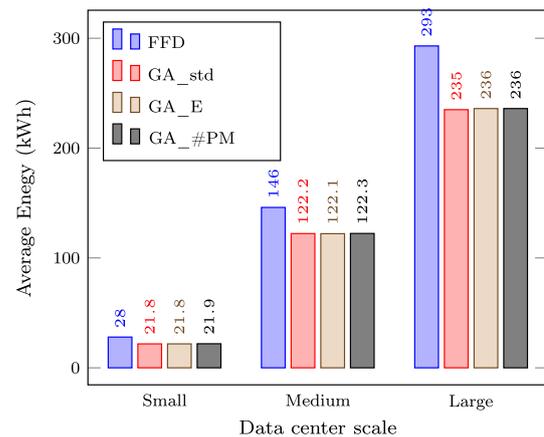


Fig. 9 The average energy-consumption performance in an hour from Algorithm 1 (GA_std), Algorithm 2 (GA_E) and Algorithm 3 (GA_#PM) together with FFD for small-, medium- and large-scale data centers

execution time, and the average execution time for each GA generation.

7.1 Execution-time performance

Figure 8 shows the execution-time performance of Algorithms 1, 2 and 3 in VM placement for small-, medium- and large-scale data centers, respectively. The Standard GA with Algorithm 1 using the original data structure consumes much more time than our improved Algorithms 2 and 3. For the simulated large-scale data center, our improved Algorithms 2 and 3 are about 5.5 times faster and about 11 times faster, respectively, than Algorithm 1. Furthermore, compared to Algorithm 2 (the standard fitness and the adopted new data structure), our improved Algorithm 3 (our alternative fitness embedded with the adopted new data structure) executes about 2 times faster.

Recalling the GA profiling in terms of execution time from Sect. 3, we have recorded the time consumption for each of the GA computation components for a specific experimental scenario in one GA run (see Fig. 3 in Sect. 3). A quantitative analysis of these results reveals that the SCE computation time has been significantly compressed from 52% to 23% by switching from Algorithm 1 (the Standard GA, GA_std) to Algorithm 2 (the standard fitness with the adopted new data structure, GA_E). This 23% is further reduced by about half by switching to Algorithm 3. This dramatic reduction in execution time verifies that the computation of GA for energy-efficient VM placement can be significantly accelerated through compressing the time consumption in SCE-infused computation.

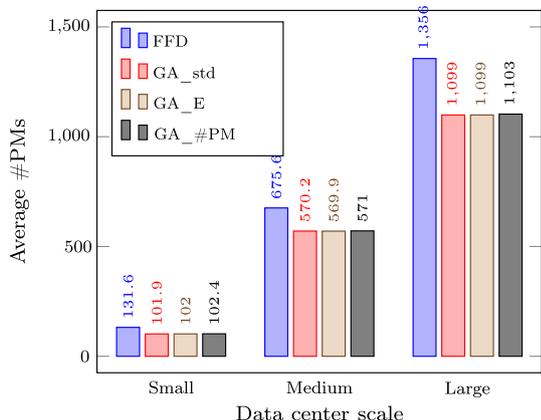


Fig. 10 The average numbers of PMs planned from Algorithm 1 (GA_std), Algorithm 2 (GA_E) and Algorithm 3 (GA_#PM) together with FFD for small-, medium- and large-scale data centers

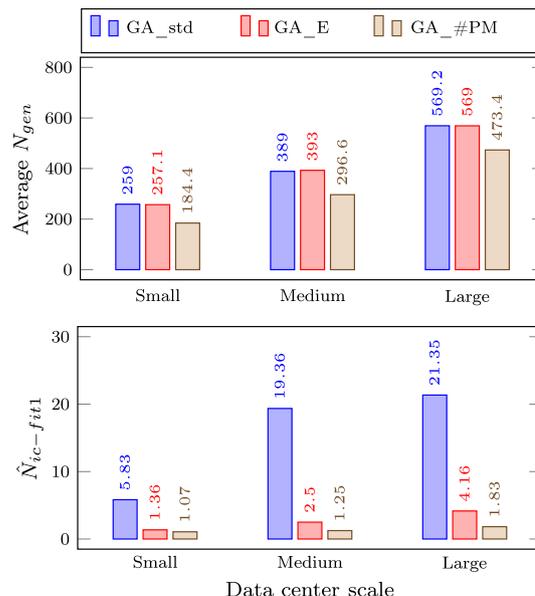


Fig. 11 Average N_{gen} (upper plot) and estimated $\hat{N}_{ic-fit1}$ (lower plot) from Algorithm 1 (GA_std), Algorithm 2 (GA_E), and Algorithm 3 (GA_#PM) for small-, medium-, and large-scaled data centers

7.2 Energy consumption performance

With significant improvement in the execution time of GA from our accelerated algorithms embedded with the adopted new data structure and/or our alternative fitness function, we would like to check whether or not the resulting energy consumption in data centers is also satisfactory. For small-, medium- and large-scale data centers, Fig. 9 shows the performance of energy consumption in an hour from Algorithms 1, 2, and 3. For comparison, the figure also depicts the energy consumption from FFD for each of the three data center scales.

Understandably, FFD leads to the largest energy consumption, as the other three algorithms start their iterations from the FFD solution (see Fig. 9). Also, Algorithm 1 (GA_std), Algorithm 2 (GA_E), and Algorithm 3 (GA_#PM) all give similar energy-consumption performance, indicating that the adopted new data structure and alternative fitness function achieve significant acceleration of GA computation without sacrifices in the energy-consumption performance.

7.3 The number of active PMs

Fig. 10 compares the average numbers of active PMs resulting from Algorithms 1 (GA_std), 2 (GA_E), and 3 (GA_#PM) in VM placement for small-, medium- and large-scale data centers. FFD’s results are also shown in the figure. For each of the three scales of data centers, FFD gives a VM-placement plan that requires the largest

Table 4 Featured results for GA_E and GA_#PM

Performance	Scale	Fitness	Max	Min	Avg	SD
E (kWh)	Large	E	237.7	233.8	236.0	0.853
		#PM	238.2	234.0	236.0	0.885
	Medium	E	123.2	121.1	122.1	0.487
		#PM	123.2	121.3	122.4	0.405
	Small	E	22.3	21.4	21.8	0.158
		#PM	22.3	21.6	21.9	0.166
#PM	Large	E	1109	1090	1099.0	4.10
		#PM	1111	1091	1103.0	4.26
	Medium	E	575	565	569.9	2.355
		#PM	575	566	571.2	1.956
	Small	E	104	100	102.0	0.768
		#PM	104	101	102.4	0.808
N_{gen}	Large	E	940	405	569.0	106.93
		#PM	669	370	473.4	56.38
	Medium	E	738	234	393.0	103.61
		#PM	439	227	296.6	43.93
	Small	E	464	154	257.1	68.44
		#PM	264	148	184.4	24.55

number of active PMs, implying the worst energy consumption performance. For small- and medium-scale data centers, Fig. 10 shows that all three algorithms (GA_std, GA_E and GA_#PM) lead to similar numbers of active PMs. The difference is about one PM only. For the simulated large-scale data center, the VM placement from GA_std and GA_E requires almost the same number of PMs (1099), while GA_#PM gives a plan of 1103 active PMs, indicating 0.36% more PMs than GA_std and GA_E do. However, this small sacrifice in the number of active PMs leads to an acceleration of 11 times in GA computation with the execution time being reduced from over 54 min (3270 s) to about 5 min (300 s) as shown in Fig. 8.

7.4 $N_{ic-fit1}$ and N_{gen}

To better understand how the adopted new data structure and alternative fitness accelerate GA computation for energy-efficient VM placement, we draw two bar graphs in Fig. 11 on average N_{gen} and estimated $\hat{N}_{ic-fit1}$ from GA_std, GA_E, and GA_#PM for small-, medium-, and large-scale data centers, respectively.

From many runs of GAs under simulated scenarios, the upper bar graph of average N_{gen} in Fig. 11 shows that GA_std and GA_E require similar numbers of generations. In comparison, GA_#PM uses an obviously smaller number of generations to get its solution.

The lower bar graph of Fig. 11 is $\hat{N}_{ic-fit1}$, which is estimated from the average duration of each generation. It shows that the adopted new data structure in GA_E reduces the CPU instruction count significantly from GA_std, and our alternative fitness in GA_#PM further reduces the CPU instruction count in fitness evaluation. For example, in the large-scale data center that we simulated, GA_E reduces the average $\hat{N}_{ic-fit1}$ from 21.35 (the result with GA_std) to 4.16, which is about a third of the original value. GA_#PM further reduces the average $\hat{N}_{ic-fit1}$ from 4.16 (the result with GA_E) to 1.83, which is a compression of over 50%. From the original GA_std, our GA_#PM compresses the average $\hat{N}_{ic-fit1}$ by over 91% from 21.35 down to 1.83.

7.5 Statistical analysis of results

For a deeper understanding of the GA acceleration presented in this paper, some quantitative simulation results are summarized in Table 4, which has been partially visualized in previous subsections. It is seen from this table, and also shown previously in Fig. 11, that the number of GA generations N_{gen} is reduced considerably in our GA_#PM over G_E, leading to a noticeable acceleration of GA computation (Fig. 8). It is also expected to know that this computational acceleration does not affect the energy efficiency of the resulting VM placement in data centers.

To verify that the accelerated GA computation presented in this paper with our new fitness function does not affect the energy efficiency of the VM-placement results, statistical tests are conducted on the simulation results from our GA_#PM and GP_E methods. The null hypothesis of the statistical tests between our GA_#PM and GP_E methods is:

H0 The energy efficiency from GA_#PM and GA_E has no obvious difference.

The alternative hypothesis is that the results of energy efficiency from GA_#PM and GA_E are statistically different.

Two types of statistical tests have been conducted:

- (1) two sample paired *t*-tests, and
- (2) Wilcoxon’s signed-rank tests.

These tests are carried out because the performance results from each of the two GA methods are evaluated using the same simulated data [24, 31]. For all statistical tests, the significance level is 0.05. To support the null hypothesis, it is required to have *p*-values greater than the significance level, and Wilcoxon’s tests additionally require that $w > T_{crit}$, where *w* stands for the minimum of the sums of

Table 5 Results of *t*-tests and Wilcoxon’s signed rank tests with alpha level 0.05 for GA_E and GA_#PM by using δ calculated from the quotient of GA_#PM and GA_E

Performance	Scale	δ	<i>t</i> -tests			Wilcoxon’s tests		
			t critical	<i>T</i>	<i>p</i>	<i>w</i>	<i>T_{crit}</i>	<i>p</i>
E (kWh)	Large	1.002	1.98	0.515	0.696	2303	1955	0.445
	Medium	1.002	1.98	0.152	0.560	2447	1955	0.789
	Small	1.004	1.98	0.296	0.384	2276	1955	0.392
#PM	Large	1.002	1.98	0.643	0.739	2303	1955	0.445
	Medium	1.002	1.98	0.288	0.613	2447	1955	0.789
	Small	1.004	1.98	0.170	0.567	2276	1955	0.392

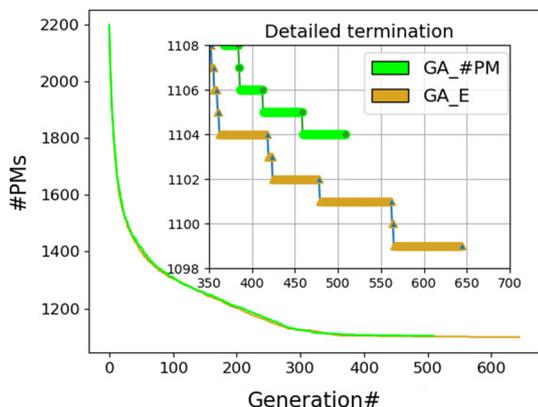


Fig. 12 A zoom-in of the convergence of GA_E from Algorithm 2 and GA_#PM from Algorithm 3

positive and negative ranks, and T_{crit} is the critical value in Wilcoxon’s tests.

Test samples in our statistical analysis are derived from the quotient of GA_E and GA_#PM. The quotient is denoted by δ . Test results from the *t*-tests and Wilcoxon’s tests are tabulated in Table 5.

It is seen from Table 5 that all *p* values are greater than the significance level 0.05. Additionally, it is also observed from the table that the relationship $w > T_{crit}$ holds in Wilcoxon’s tests. Therefore, the results from both *t*-tests and Wilcoxon’s tests support the acceptance of the null hypothesis, i.e., the results of energy efficiency from our GA_#PM and GA_E methods have no obvious difference. Actually, they also show that the numbers of PMs #PM from both GA_#PM and GA_E have no obvious difference. It is concluded that the accelerated GA computation with our new fitness function does not affect the energy efficiency of the resulting VM placement.

7.6 Convergence speed

When both N_{gen} and $N_{ic-fit1}$ are reduced, the convergence of GA computation becomes faster. Our quantitative analysis of the experimental results shows that, for the simulated large-scale data center, GA_#PM converges over 20 times faster than GA_E does. Figure 12 shows that

GA_#PM from Algorithm 3 (the green line) terminates earlier than GA_E from Algorithm 2. This further verifies a smaller N_{gen} value and an earlier termination from GA_#PM than GA_E.

In summary, according to Eq. (6) and Figs. 8 and 11, our improved Algorithm 2 (GA_E) reduces $N_{ic-fit1}$ from their values in the Standard GA in Algorithm 1 (GA_std). Since N_{pop} , *f*, and u_{cpu} are not the focus of this paper, they are set as constants in our simulation experiments. Our improved Algorithm 3 further compresses N_{gen} and $N_{ic-fit1}$ from their values in GA_E using Algorithm 2. As a result, the computation of GA is significantly accelerated for energy-efficient VM placement in data centers.

8 Conclusion

To make GA practically feasible for energy-efficient VM placement in large-scale data centers, this paper has investigated how to accelerate GA computation without sacrificing the quality of solution in terms of energy savings. Our analysis has revealed that fitness-related computation consumes most of the execution time in standard GA. Therefore, this paper has focused on speeding up the fitness evaluation in GA computation for energy-efficient VM placement. The speedup has been achieved from two main aspects: a new data structure for complexity reduction, and an alternative fitness function for instruction count reduction. The new data structure extended from our previous study reduces the complexity of the GA computation from quadratic to linear one with respect to the input size of the VM-placement problem, thus speeding up the GA computation. Along with this data structure, an alternative fitness function has been proposed, which is simply associated with the number of active PMs. It leads to a dramatic reduction in the number of instructions required in each instance of fitness evaluation, further accelerating the GA computation. Experimental studies have demonstrated that our accelerated GA under the alternative fitness function incorporating with the new data structure is about 11 times faster than the standard GA for energy-efficient

VM placement in large-scale data centers of about 1, 500 PMs without compromising energy efficiency performance of the data centers.

Our work presented in this paper focuses on accelerating GA computation through a new data structure and an alternative fitness function. It does not consider the impact of QoS and SLA requirements on the energy efficiency of data centers. However, QoS and SLAs are important in practical management and operations of real data centers. How to embed QoS and SLA requirements into the energy-efficient VM-placement problem deserves further research in the future.

Acknowledgements Author Y.-C. Tian would like to acknowledge the Australian Research Council (ARC) for its support through the Discovery Projects Scheme under Grants DP170103305 and DP220100580.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Data Availability The datasets used in the experimental tests of the current study are extracted from Google's Cluster-Usage Traces [8, 27]. The datasets generated and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Compliance with Ethical Standards The authors declare that they have no known competing financial interests, non-financial interests, or personal relationships that could have appeared to influence the work reported in this paper. This research does not involve Human Participants and/or Animals. Thus, informed consent does not apply.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abdessamia F, Zhang WZ, Tian YC (2020) Energy-efficiency virtual machine placement based on binary gravitational search algorithm. *Clust Comput* 23(3):1577–1588
- Alharbi F, Tian YC, Tang M, Ferdaus MH, Zhang WZ, Yu ZG (2021) Simultaneous application assignment and virtual machine placement via ant colony optimization for energy-efficient enterprise data centers. *Clust Comput* 24(2):1255–1275
- Alharbi F, Tian YC, Tang M, Zhang WZ, Peng C, Fei M (2019) An ant colony system for energy-efficient dynamic virtual machine placement in data centers. *Expert Syst Appl* 120:228–238
- Barr J (2022) Cloud computing, server utilization, and the environment (2015). <https://aws.amazon.com/es/blogs/aws/cloud-computing-server-utilization-the-environment/>. Accessed 29 Jul
- Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener Comput Syst* 28(5):755–768
- Chen ZG, Zhan ZH, Lin Y, Gong YJ, Gu TL, Zhao F, Yuan HQ, Chen X, Li Q, Zhang J (2019) Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach. *IEEE Trans Cybern* 49(8):2912–2926
- Dabbagh M, Hamdaoui B, Guizani M, Rayes A (2015) Energy-efficient resource allocation and provisioning framework for cloud data centers. *IEEE Trans Netw Serv Manag* 12(3):377–391
- Dabbagh M, Hamdaoui B, Guizani M, Rayes A (2018) Exploiting task elasticity and price heterogeneity for maximizing cloud computing profits. *IEEE Trans Emerg Top Comput* 6(1):85–96
- Dayarathna M, Wen Y, Fan R (2015) Data center energy consumption modeling: a survey. *IEEE Commun Surv Tutor* 18(1):732–794
- De La Vega WF, Lueker GS (1981) Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica* 1(4):349–355
- Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evol Comput* 6(2):182–197
- Ding Z, Tian YC, Tang M (2018) Efficient fitness function computation of genetic algorithm in virtual machine placement for greener data centers. In: 2018 IEEE 16th Int. Conf. Ind. Inform. (INDIN), Porto, Portugal, pp 181–186
- Ding Z, Tian YC, Tang M, Li Y, Wang YG, Zhou C (2020) Profile-guided three-phase virtual resource management for energy efficiency of data centers. *IEEE Trans Ind Electron* 67(3):2460–2468
- Elsayed S, Sarker R, Coello CAC (2019) Fuzzy rule-based design of evolutionary algorithm for optimization. *IEEE Trans Cybern* 49(1):301–314
- Fang Q, Zhou J, Wang S, Wang Y (2022) Control-oriented modeling and optimization for the temperature and airflow management in an air-cooled data-center. *Neural Comput Appl* 34:5225–5240
- Graubner P, Schmidt M, Freisleben B (2013) Energy-efficient virtual machine consolidation. *IT Prof* 15(2):28–34
- Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. *IEEE Trans Syst Man Cybern* 16(1):122–128
- Harik G, Cantú-Paz E, Goldberg DE, Miller BL (1999) The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evol Comput* 7(3):231–253
- Harik GR, Lobo FG, Goldberg DE (1999) The compact genetic algorithm. *IEEE Trans Evol Comput* 3(4):287–297
- Hormozi E, Hu S, Ding Z, Tian YC, Wang YG, Yu ZG, Zhang W (2022) Energy-efficient virtual machine placement in data centres via an accelerated genetic algorithm with improved fitness computation. *Energy* 252:123884
- Jones N (2018) How to stop data centres from gobbling up the world's electricity. *Nature* 561:163–166
- Kumar S, Pandey M (2016) Energy aware resource management for cloud data centers. *Int J Comput Sci Inf Secur* 14(7):844
- Lama P (2007) Autonomic performance and power control in virtualized datacenters. Ph.D. thesis, University of Colorado, Colorado Springs, CO USA
- Li F, Zhang X, Zhang X, Du XC, Xu Y, Tian YCT (2018) Costsensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets. *Inf Sci* 422:242–256

25. Liu Z, Xiang Y, Qu X (2015) Towards optimal CPU frequency and different workload for multi-objective VM allocation. In: 2015 12th Ann. IEEE Consumer Commun. Netw. Conf. (CCNC), Las Vegas, NV, pp 367–372
26. Panneerselvam J, Liu L, Antonopoulos N (2020) An approach to optimise resource provision with energy-awareness in datacentres by combating task heterogeneity. *IEEE Trans Emerg Top Comput* 8(3):762–780
27. Reiss C, Wilkes J, Hellerstein JL (2011) Google cluster-usage traces: format+ schema. Google Inc., White Paper, vol 1, pp 1–14
28. Sonkilm C, Tang M, Tian YC (2017) A decrease-and-conquer genetic algorithm for energy efficient virtual machine placement in data centers. In: 2017 IEEE 15th Int. Conf. Ind. Inform. (INDIN), Eden, Germany, pp 135–140
29. Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans Syst Man Cybern* 24(4):656–667
30. Thraves C, Wang L (2014) Power-efficient assignment of virtual machines to physical machines. In: First int. workshop adaptive resource manage. scheduling for cloud comput. (ARMS-CC), Paris, France, vol 8907, p 71
31. Ullah I, Paul S, Hong Z, Wang YG (2019) Significance tests for analyzing gene expression data with small sample sizes. *Bioinformatics* 35(20):3996–4003
32. Vasudevan M (2016) Profile-based application management for green data centres. Ph.D. thesis, Queensland University of Technology, Brisbane, Queensland, Australia
33. Vasudevan M, Tian YC, Tang M, Kozan E, Zhang W (2017) Profile-based dynamic application assignment with a repairing genetic algorithm for greener data centers. *J Supercomput* 73(9):3977–3998
34. Versick D, Waßmann I, Tavangarian D (2013) Power consumption estimation of CPU and peripheral components in virtual machines. *ACM SIGAPP Appl Comput Rev* 13(3):17–25
35. Wang L, Von Laszewski G, Chen D, Tao J, Kunze M (2010) Provide virtual machine information for grid computing. *IEEE Trans Syst Man Cybern Part A: Syst Hum* 40(6):1362–1374
36. Wu G, Tang M, Tian YC, Li W (2012) Energy-efficient virtual machine placement in data centers by genetic algorithm. In: Huang T, Zeng Z, Li C, Leung CS (eds) *ICONIP 2012: neural information processing, part III*, vol 7665. Lecture Notes in Computer Science. Springer, Berlin Heidelberg, pp 315–323
37. Yuan Y, Tian Z, Wang C, Zheng F, Lv Y (2020) A Q-learning-based approach for virtual network embedding in data center. *Neural Comput Appl* 32:1995–2004
38. Zhao C, Liu J (2015) A virtual machine dynamic consolidation algorithm based dynamic complementation and FFD algorithm. In: 2015 Fifth Int. Conf. Commun. Syst. Netw. Tech. (CSNT), Gwalior, India, pp 333–338

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