Ninth International Conference on Intelligent Control and Information Processing
November 9-11, 2018; Wanzhou, China

Set-Based Comprehensive Learning Particle Swarm Optimization for Virtual Machine Placement Problem

Yue Weng, Wei-Neng Chen, An Song, and Jun Zhang
School of Computer Science and Engineering,
Provincial Key Lab of Computational Intelligence and Cyberspace Information,
South China University of Technology,
Guangzhou, China
Email: cwnraul634@aliyun.com

Abstract—**The virtual machine placement (VMP) is a significant technology in energy-saving field, which is an increasingly important problem of cloud computing centers. Most existing algorithms are difficult to handle the large-scale VMP problems with heterogeneous resources and large demand of virtual machines. In this paper, we propose the set-based comprehensive learning particle swarm optimization (SCLPSO) to solve the VMP problem. SCLPSO combines the set-based particle swarm optimization framework (S-PSO) with the comprehensive learning particle swarm optimizer. With the S - PSO framework, SCLPSO is able to solve the VMP problem which is defined on the discrete search space. With the redefined velocity updating rule in SCLPSO, each dimension of a particle can potentially learn from different exemplars. This strategy improves the exploration of the algorithm. The algorithm also introduces a heuristic factor to guide a virtual machine (VM) to be placed on a more suitable physical machine (PM), which improves the resource utilization of the PM. Based on the devised strategies, large-scale VMP problems with heterogeneous resources can be well resolved by SCLPSO. We conduct experiments on different instances and compare SCLPSO with other classical algorithms. The experimental results demonstrate that the proposed algorithm is promising.**

Keywords—Cloud computing, virtual machine placement (VMP), set-based comprehensive learning particle swarm optimization (SCLPSO), particle swarm optimization (PSO).

I. Introduction

The emergence of cloud computing satisfies people's increasing demand for computing resources. With virtualization technology, users can purchase various physical resources they need (e.g., CPU and memory) and need not care about the distribution and configuration of physical machine (PM). Cloud computing centers provide users with three types of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [1]. A virtual machine (VM) can be assigned to a physical machine in the cloud computing center according to users' requirements of resources.

With the expanding of cloud computing centers, the number of physical machines continues to increase, and the amount of consumed energy to power and cool physical machines is also increasing. Nearly half of the power consumption in cloud computing centers is used for the cooling of physical machines [2]. Even idle physical machines consume up to 66% of the peak power to maintain operating devices and other hardware resources [3]. Therefore, the key to solve the energy consumption problem is to reduce the number of active physical machines as much as possible and shut down idle physical machines.

Virtualization technology allows multiple virtual machines to be distributed on the same physical machine to share physical resources without affecting each other, which is called the virtual machine placement (VMP) problem. How to map all virtual machines to suitable physical machines is a NP-hard problem [4]. The solution space grows exponentially as the scale of the problem increases, making the enumeration impossible.

As the VMP problem is challenging, considerable amount of research efforts has been devoted to VMP. From the perspective of problem model, Aggarwal *et al.* [5] studied the important layout goals and constraints of VMP problem, and designed heuristics algorithms. As for disk placement, Hbaieb *et al.* [6] proposed a decomposition and local search method. Xia *et al.* [7] proposed a hierarchical decomposition and mixed integer programming (MIP) approach to solve the problem that restricts the VM virtual disk from being distributed over the PM physical disk. While some scholars proposed virtual machine migration technology to achieve server consolidation and minimize the use of low-utility servers [8]-[11]. During the placement process, the position of VM will be migrated to another PM for better assignment. On the other hand, from the perspective of the algorithm, Alboaneen *et al.* [12] introduced metaheuristic methods for VMP, including individual-based algorithms like simulated annealing (SA), reproductive population-based algorithms like genetic algorithm (GA), and non-reproductive population-based algorithms like biogeography-based optimization (BBO). Ajiro *et al.* [13] proposed another heuristic algorithm called First-Fit Decreasing (FFD) to solve VMP. Through these heuristics, a feasible solution can be found in a short time using the greedy search method. But as the problem scale increases, the quality of the solutions found by simple heuristics degenerates. As for meta-heuristics, Gao *et al.* [14] proposed a solution based on the ant colony algorithm to solve VMP problem, which is able to generate more promising solutions. Based on GA, Jamali *et al.* [15] proposed an improved grouping-based genetic algorithm (IGGA) to minimize the number of physical machines and maximize the utilization of resources as well.

In order to develop a more effective approach to VMP, this paper puts forward the SCLPSO algorithm combining both set-based discrete particle swarm optimization framework (S-PSO) [16] and the velocity updating rule in comprehensive learning PSO (CLPSO) [17] We also pay attention to the allocation and utilization of both CPU and memory resources. The main contribution of this work can be summarized as follows:

1) SCLPSO combines both the S-PSO framework and the velocity updating strategy of CLPSO. S-PSO framework extends the original PSO to discrete space, so that we can use PSO to solve discretization problems. Comprehensive learning strategy maintains a good searching diversity so that the proposed approach can achieve good performance on the considered problem.

2) In order to measure the connection of the VM and PM, we design the heuristic which can improve the utilization of different resources and provide a better assignment.

The remainder of the paper is organized as follows. Section II will introduce the background of the PSO, S-PSO and CLPSO. Section III will formulate the VMP problem. Section IV will introduce the proposed SCLPSO algorithm. Experimental studies on large-scale problems in both homogeneous and heterogeneous environments are presented in Section V. Conclusion is drawn in Section VI.

II. Background

A. The Original PSO

Particle swarm optimization (PSO) was proposed by Kennedy and Eberhart in 1995 [18]-[22]. It is a population-based heuristic algorithm and is often used to solve problems in continuous space. PSO sets *M* particles to find the globally optimal solution in a *n*-dimensional continuous space. The algorithm initializes a population of swarm randomly at the beginning and evaluates their fitness values. Then it uses (1) and (2) to update the position and velocity of each particle, where v_i^d is the *d*-th dimension of the velocity of the *i*-th particle, and x_i^d is the *d*-th dimension of the position of the *i*-th particle. With the guidance from the personal historically best position (*pbest*) and the globally historically best position (*gbest*) of the whole swarm, particles update their positions iteratively to seek for the optimal solution in the search space.

$$v_i^d = wv_i^d + c_1r_1(pbest_i^d - x_i^d) + c_2r_2(gbest^d - x_i^d), \quad (1)$$

$$x_i^d = x_i^d + v_i^d \quad . \quad (2)$$

B. The Set-Based PSO Framework

The original PSO cannot be used in discrete space directly because both (1) and (2) are defined in continuous space. To solve this problem, Chen *et al.* [16] proposed a novel set-based particle swarm optimization (S-PSO) method for discrete optimization problems [23]. S-PSO [24], [25] adopts the concept of crisp set and redefines the representation of "position" and "velocity" as well as other related operations, so that the updating rules (1) and (2) can be applied to solve discretization problems. The framework can be well applied to other PSO variants [16]. In S-PSO, the position of particle *i* is defined as a crisp set $\boldsymbol{X_i}$ and the velocity of particle *i* is defined as a set of probabilities $\boldsymbol{V_i}$, as shown in (3). The element in each velocity set is associated with a probability $p(e)$. $p(e)$ is the probability that determines whether particle *i* will learn from the element e to build a new position.

$$\boldsymbol{V_i} = \{e / p(e) \mid e \in E\} \ . \quad (3)$$

C. CLPSO

In order to enhance the searching diversity of PSO, Liang *et al.* [17] proposed a comprehensive learning PSO employing a novel learning strategy to update the velocity. Any particle can learn from other particles' previous best positions and each dimension of a particle can potentially learn from a different exemplar. The updating strategy is defined as follows:

$$V_i^d = \omega * V_i^d + c * rand_i^d * (pbest_{fi(d)}^d - X_i^d) \ , \quad (4)$$

where c is a parameter, $rand_i^d$ is a random number in [0,1], and $pbest_{fi(d)}^d$ is the *d*-th dimension of the $pbest$ position of particle $fi(d)$. $fi(d)$ is given as follows. First, a random number $ran \in [0,1]$ is generated. If *ran* is larger than a parameter *Pc*, then $fi(d) = i$. Otherwise, the algorithm applies the tournament selection on two randomly selected particles. The particle with better fitness will be selected. We do this process in each dimension for each particle, so that each particle and each dimension can learn from different particles. The parameter Pc for the *i*-th particle is defined as follows:

$$Pc_i = 0.05 + 0.45 * \frac{\left(\exp\left(\dfrac{10(i-1)}{POPSIZE-1}\right)-1\right)}{\left(\exp(10)-1\right)}. \quad (5)$$

CLPSO has shown excellent performance for complex multimodal optimization problems [17]. Therefore, in this paper, we combine this updating strategy and the S-PSO framework together to improve the performance of PSO in solving VMP.

III. Problem Definition

Virtualization technology is the core part of cloud computing. The cloud computing center receives user's requests and creates a virtual machine that meets user's requirements, including a specific number of CPUs, memory, hard disks, etc. Then the cloud computing center assigns VMs to PMs that have enough remaining physical resources. A reasonable allocation scheme can reduce the number of active physical machines, and thus can save energy consumption of cloud computing centers.

The purpose of the VMP problem is to find an optimal VM allocation scheme that uses the minimum number of PMs to place all the VMs, and ensures that the resource requirements of each VM are satisfied. We assume that the resource requirements of each VM will not exceed the resource capacity of one PM, and each VM is placed on only one PM, which means that VM cannot be further subdivided. VMs can share resources on the same PM without affecting each other. If one or more VMs are assigned to a PM, the PM needs to be turned on. In this paper, we consider the allocation of both CPU and memory resources. We assume that there are N PMs and N VMs. Let V = {1, 2…, N} be a set of VMs and P = {1, 2, …, N} a set of PMs. The notation is summarized in Table I.

We formulate the allocation scheme as a 0-1 matrix whose element x_{ij} represents the assignment of physical machines. The

TABLE.I. NOTATION OF THE VMP PROBLEM

Notation	Definition
PM_i $(1 \leq i \leq N)$	Physical machine i
pc_i	CPU of PM_i
pm_i	Memory of PM_i
VM_j $(1 \leq j \leq N)$	Virtual machine j
vc_j	CPU of VM_j
vm_j	Memory of VM_j
x_{ij}	Allocation matrix

row represents PM and the column represents VM. If VM_j is assigned to PM_i, then $x_{ij} = 1$, otherwise $x_{ij} = 0$, i.e.,

$$x_{ij} = \begin{cases} 1, \text{ if } VM_j \text{ is assigned to } PM_i \\ 0, \text{ otherwise} \end{cases} \quad \forall i \in P \text{ and } \forall j \in V. \tag{6}$$

We need to ensure that every VM should be assigned, i.e.,

$$\sum_{i=1}^{N} x_{ij} = 1 \quad \forall j \in V. \tag{7}$$

The objective is to optimize the number of required PMs, which are calculated as follows:

$$\text{Minimize } f(s) = \sum_{i=1}^{N} y_i, \tag{8}$$

where

$$y_i = \begin{cases} 1, \text{ if } \sum_{j=1}^{N} x_{ij} \geq 1 \\ 0, \text{ otherwise} \end{cases} \quad \forall i \in P. \tag{9}$$

The resource allocation should also follow the following constraints

$$\sum_{j=1}^{N} vc_j \cdot x_{ij} \leq pc_i \cdot y_i \quad \forall i \in P \ , \tag{10}$$

$$\sum_{j=1}^{N} vm_j \cdot x_{ij} \leq pm_i \cdot y_i \quad \forall i \in P. \tag{11}$$

Studies have shown that the energy consumption of PM is linearly related to the CPU usage of PM [20]. However, an open but idle PM can also consume 50%-70% of the energy consumption in full load condition [21]. Therefore, the energy consumption model in this paper is as follows:

$$P(u) = k \cdot P_{\max} + (1-k) \cdot P_{\max} \cdot u, \tag{12}$$

where $P_{\max}$ is the maximum energy consumption of a PM under full load conditions. k indicates the idle energy consumption ratio of an opened PM. u indicates the CPU usage of a PM.

IV. SCLPSO FOR VMP

A. Encoding of SCLPSO

The VMP problem is a combinatorial optimization problem defined in discrete space. Therefore, the encoding scheme needs to meet the discretization requirement and the constraints of VMP. According to the S-PSO framework, the set-based encoding scheme in the proposed SCLPSO is defined as follows.

1) Position

A position X represents a feasible assignment scheme. Each particle's position is an N-tuple $X = (X_1, X_2, \ldots, X_n)$ where n is the number of active PMs. Each element X_i in the N-tuple is a set which represents the VMs assigned on this PM. Different PMs may have different number of VMs on it. For example, a position X is given as $X = (\{VM_{i1}, VM_{i2}\}, \{VM_{i3}, VM_{i4}, VM_{i5}\}, \{VM_{i6}\}, \ldots)$. It means that VM_{i1}, VM_{i2} are assigned on PM_1, $VM_{i3}, VM_{i4}, VM_{i5}$ are assigned on PM_2, VM_{i6} is assigned on PM_3, and so on. If the element set X_i is empty, if means that no VM is assigned on the PM_i.

2) Velocity

In S-PSO, velocity V is also an N-tuple V = (V1, V2, …, Vn), where each element Vi is a set with probabilities. The probability determines the chance that the particle will learn from the element e to build a new position.

For example, the following velocity V is given

$V = (\{VM_{i1}/0.5, VM_{i2}/0.4\}, \{VM_{i3}/0.7, VM_{i4}/0.6\}, \{\}, \ldots)$. It means that VM_{i1} has the probability of 0.5 to be assigned to PM_1, VM_{i2} has the probability of 0.4 to be assigned to PM_1, VM_{i3} has the probability of 0.7 to be assigned to PM_2, and so forth.

For convenience, here we use V_i^j to describe the probability that VM_j will be assigned on PM_i.

B. The Velocity Updating

Following the S-PSO framework defined in [16] and the velocity updating rule of CLPSO [17] given in (4), the proposed SCLPSO updates velocities of particles using the following operators:

1) Coefficient × Velocity

The coefficient can be a constant or a random number which is a nonnegative real number. According to the S-PSO framework, the operator is defined as follows:

$$c \times V = \{e / p'(e) \mid e \in E\},$$
$$p'(e) = \begin{cases} 1, & \text{if } c \times p(e) > 1 \\ c \times p(e), & \text{otherwise} \end{cases}. \tag{13}$$

2) Position – Position

In S-PSO framework, a position is given by a crisp set, and the minus operator between two crisp set is the difference set. Given two position sets A and B, the operator is defined as follows:

$$\boldsymbol{A}-\boldsymbol{B}=\{e \mid e \in A \text{ and } e \notin B\}. \tag{14}$$

It should be noticed that the minus operation is executed in each element set. For example, we assume that the number of VMs and PMs is 5. Given $X=(\{1\},\{2,3\},\{4,5\},\{\},\{\})$ and $GBest=(\{1\},\{2,4\},\{3,5\},\{\},\{\})$, we then have $GBest-X=(\{\},\{4\},\{3\},\{\},\{\})$. In fact, the effect of $PBest-X$ or $GBest-X$ is to find out the elements used by better positions but not used by the current position. We figure out the difference set, and its elements are valuable to improve the current position. Just like the example, VM_4 should be assigned on PM_2, and VM_3 should be assigned on PM_3.

3) Coefficient × (Position - Position)

The result of "Position-Position" is also a crisp set. The multiplication operator between a coefficient and a crisp set is to generate a new velocity. The operation is defined as follows:

$$c \times E' = \{e / p'(e) \mid e \in E\},$$

$$p'(e)=\begin{cases}1, & \text{if } e \in E' \text{ and } c>1 \\ c, & \text{if } e \in E' \text{ and } 0 \le c \le 1 \\ 0, & \text{if } e \notin E'\end{cases}. \tag{15}$$

For example, suppose the coefficient $c=0.5$ and $GBest-X=(\{\},\{4\},\{3\},\{\},\{\})$, the $c*(GBest-X)$ will be $GBest-X=(\{\},\{4/0.5\},\{3/0.5\},\{\},\{\})$. It implies that $V_2^4=0.5$ and $V_3^3=0.5$.

4) Velocity + Velocity

Finally, we define the addition operator between two sets with possibilities.

$$V_1+V_2=\{e/\max(p_1(e), p_2(e)) \mid e \in E\}. \tag{16}$$

We choose the larger probability between two crisp sets to generate a new velocity.

C. Heuristic Factor

In traditional allocation strategies, algorithms seldom take the relationship between VM and PM into account. However, if such relationship can be utilized, we can better guide the distribution. For example, when a VM is placed on a PM, the current resource utilization of the PM can be greatly improved. Then we can choose this assignment scheme in priority. In this paper, we introduce the heuristic factors η to increase the resource utilization of PM and balance the usage of different physical resources of PM at the same time.

The heuristic factor adopts the greedy strategy to improve the assignment schemes. In order to reduce the number of final PMs, each PM needs to host as many VMs as possible, which can increase the utilization rate of PM resources. Balancing the use of various physical resources can avoid the situation that some resources are fully used while other resources are at idle. Based on this, heuristic factor is designed to increase the utilization of different resources and balance the use of different resources on the same PM, resulting in a higher and similar utilization of different resources.

The heuristic factor is related to VMs and PMs. It is calculated by the resource utilization that can be brought to the PM after the VM is assigned to the PM. The heuristic factor with a smaller value indicates that the VM is more suitable for placing on this PM. The heuristic factor in this paper is defined as follows:

$$\eta(i,j)=\frac{\left|\dfrac{pc_i-uc_i-vc_j}{pc_i}-\dfrac{pm_i-um_i-vm_j}{pm_i}\right|}{\left|\dfrac{pc_i-uc_i-vc_j}{pc_i}\right|+\left|\dfrac{pm_i-um_i-vm_j}{pm_i}\right|}. \tag{17}$$

In (17), pc is the CPU capacity of PM_i, uc is the CPU usage, pm and um is the information of memory. vc and vm are the CPU requirements and memory requirements for VM_j. The denominator of (17) measures the utilization in both CPU and memory resources. The numerator of (17) considers the remaining resources on the physical machine. If VM_j could increase the resource utilization of PM_i as well as balance the usage of different physical resources of PM_i, η will be smaller. This selection strategy will reduce the number of active PMs, which can further improve the quality of solutions.

D. Position Updating with the Step-by-Step Strategy

We can use the (4) and the operators defined on crisp sets to update velocities. When generating new positions, we use the step-by-step strategy, which considers each VM and generates the new position by placing VMs one by one.

Each VM has its own velocity subset, which represents the probabilities to be placed on every PM. Firstly, a random number $\alpha \in (0,1)$ is generated in each iteration, and then a cut set is generated according to (18).

$$cut_\alpha(V_i^j)=\{e \mid e/p(e) \in V_i^j \text{ and } p(e) \ge \alpha\}. \tag{18}$$

For each element in the cut set, (17) is used to calculate the heuristic factor between the current VM and the PM to find the most suitable assignment. The candidate PMs are sorted in ascending order according to the heuristic factor, and then the PM with smaller heuristic factor is selected. If a suitable PM can be found successfully during this process, the allocation of the VM is completed. If not, the VM is added to a reassignment set. This process is repeated for each VM until all VMs have been considered.

During reassignment, the elements in the reassignment set represent unsuccessfully allocated VMs, and then we use (17) to calculate the heuristic factors between it and the PMs that have been turned on. Then sort all the heuristic factors and select the PM with smallest heuristic factor. If there is a suitable PM, allocate and complete the allocation of this VM. If we cannot find a suitable PM, then turn on a new PM to place this VM. Repeat this process until the reassignment set is empty. The above is the process of building a solution step by step. When this process finishes, we can obtain a new feasible solution.

E. Complete SCLPSO Algorithm

The complete flowchart of SCLPSO is shown in Fig. 1. Firstly, we initialize the population and each VM is placed on a

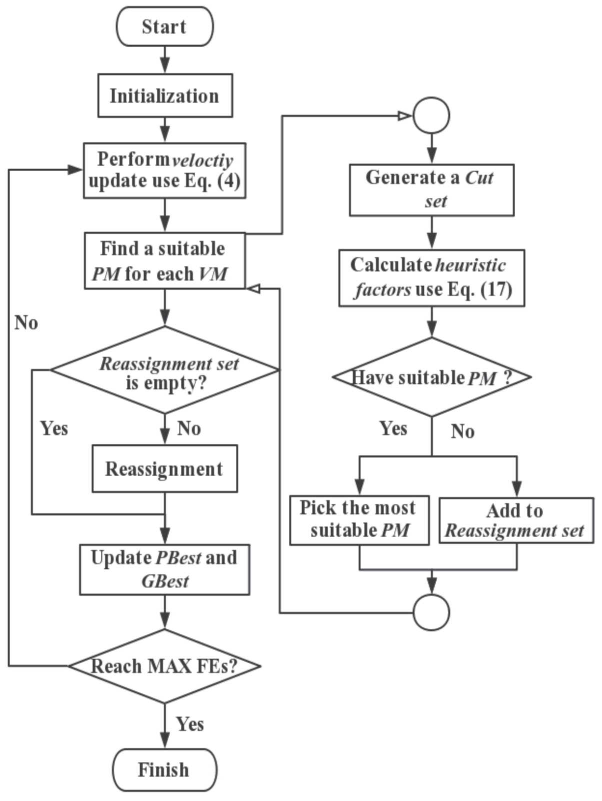
Start
Initialization
Perform velocity update use Eq. (4)
Find a suitable PM for each VM
Reassignment set is empty?
Reassignment
Update PBest and GBest
Reach MAX FEs?
Finish
Generate a Cut set
Calculate heuristic factors use Eq. (17)
Have suitable PM ?
Pick the most suitable PM
Add to Reassignment set
Yes
No

Fig. 1. Flowchart of the SCLPSO algorithm.

PM, which means $X = \{\{1\},\{2\},...\{N\}\}$ .The number of available PMs is equal to the number of VMs. Each element in the velocity set is assigned a random value in [0,1].

SCLPSO uses the updating rule of CLPSO in (4) and S-PSO to update velocities. Then we enter the process of solution construction. For each VM, we will generate a cut set using (18). And we will calculate the heuristic factors η for every PM in cut set and current VM using (17). If we can find the most suitable PM according to η , then we finish this VM assignment. Otherwise, we add it to the reassignment set. We will do this process for all VMs.

Then we check the reassignment set. If there still exist unassigned VMs, we go into the reassignment process and find suitable PMs to hold the VMs until the set is empty. Finally, we update the *PBest* and *GBest* .

We will repeat this process until the algorithm reaches the maximum number of fitness evaluations.

V. Experiments and Comparisons

We conduct experiments to test and verify the effectiveness of SCLPSO. All compared algorithms are implemented in C++ and the operating environment is CPU i7 and 4.0 GB RAM.

The experiment will use the standard test set given in [26] (i.e., Test A) and we design two other scenarios, Test B and Test C, to test the resource bottlenecks and the heterogeneity of physical machines. We compare SCLPSO algorithm with Ant Colony Optimization (ACO) based algorithm [27], [28] multi-objective algorithm (MACO) [29], HACOPSO [30] algorithm, and FFD [13] heuristic method. The ACO algorithm is suitable for solving discrete problems, [28] is based on max-min ant colony algorithm to solve the problem. [29] proposed multi-objective algorithm MACO and has certain advantages for solving multi-resource optimization in VMP problems. [30] proposed HACOPSO which combines ACO and PSO together. Its experimental results have shown superiority to most existing algorithms. The FFD sorts the VMs in descending order, first considering the CPU factor, and then considering the memory factor to be allocated on the first suitable PM. The FFD algorithm obtains better results than other deterministic algorithms [13]. Therefore, FFD is considered as a representative heuristic deterministic algorithm.

In the experiments, we set the number of populations *POPSIZE* to 10, and *MAX_FES* is the maximum number of iterations, which set to 150. Parameter c_1 and c_2 are both 2.0. As for ω , we use the linear decrement strategy in (19) ,

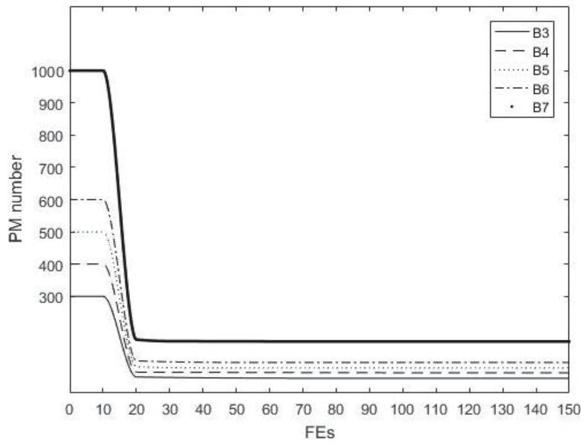

Fig. 2. The convergence curves of SCLPSO on instances B3-B7.

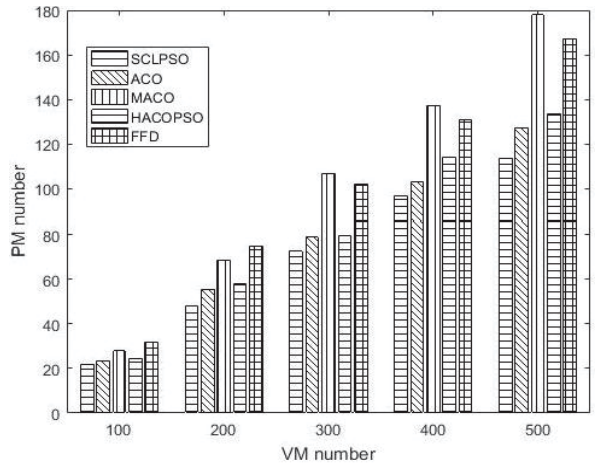

Fig. 3. The histogram of Test C.

TABLE.II . EXPERIMENTAL RESULT COMPARISONS IN TEST A

Instance	No. VM	optimum	SCLPSO	ACO	MACO	HACOPSO	FFD
A1	1363	112	**115.00**	116.50	130.26	117.56	133.00
A2	2314	191	**196.00**	199.03	222.16	204.03	229.00
A3	2639	216	**221.00**	225.33	251.13	238.93	258.00
A4	2972	241	**247.00**	251.43	282.00	263.93	288.00
A5	3289	267	**274.00**	278.76	312.26	322.83	320.00

TABLE.III . EXPERIMENTAL RESULT COMPARISONS IN TEST B

Instance	No. VM	SCLPSO	SCLPSO (without η)	ACO	MACO	HACOPSO	FFD
B1	100	**16.00**	**16.00**	**16.00**	**16.00**	17.00	18.00
B2	200	**32.00**	32.40	33.10	33.10	34.30	37.00
B3	300	**46.00**	46.70	46.73	47.20	47.10	53.00
B4	400	**63.40**	64.50	64.54	66.00	65.33	74.00
B5	500	**78.00**	80.86	79.10	82.23	81.50	92.00
B6	600	**94.80**	96.03	97.10	99.16	100.20	111.00
B7	1000	**160.00**	162.10	162.86	169.86	175.66	184.00

TABLE IV . EXPERIMENTAL RESULT COMPARISONS IN TEST C

Instance	No. VM	SCLPSO	ACO	MACO	HACOPSO	FFD
C1	100	**21.63**	23.33	28.35	24.56	32
C2	200	**48.10**	55.56	68.26	57.85	75
C3	300	**72.90**	79.44	106.90	79.40	102
C4	400	**96.86**	103.56	137.20	115.50	131
C5	500	**113.46**	127.56	178.50	134.60	167

$$\omega = \omega\_\max - \frac{(\omega\_\max - \omega\_\min)}{\text{MAX\_FES}} * T, \tag{19}$$

where $\omega\_\max$ is 0.9, $\omega\_\min$ is 0.4, and T is the current generation. The parameters of RGGA, ACO, MACO and HACOPSO are set according to their authors' recommendation. Each test runs 30 times and we compare the average results.

A. Test A: Large-Scale Homogeneous Test

The experimental data comes from [26] and the test scale is from 1000 to 3000. Each PM has 500 CPUs and 500 GB of RAM. The CPU demand for the VM is between [1,128] and the memory requirement is between [0,100]. The CPU and memory demand ratio are close to 1:1. The experimental results are shown in Table II.

From the experimental results, we can see that SCLPSO performs better than the other algorithms. The result of the solution is closer to the theoretical optimal solution. Even in the case of increasing data size, the good performance can be maintained. This is due to the use of comprehensive learning strategy to update the velocity. It can make particles have more exemplars to learn and avoid the premature. The rest of the algorithms are slightly inadequate, and the quality of solutions decreases as the data scale increases, the distance between the optimal solution and the theoretical solution continues to widen. FFD has a poor performance on large-scale solutions. With the expansion of scale, the performance of other algorithms continues to decline, and the efficiency of resource utilization for physical machines continues to decline as well.

B. Test B: Bottleneck Resource Test

In order to better test the performance of SCLPSO, we designed a bottleneck resource test. In this experiment, the data size ranged from 100 to 1000. We specify that each PM has 16 CPUs and 32 GB of RAM. The CPU requirement for each VM is between [1,4], the memory requirement is between [1,8], and the VM requirements follow a uniform distribution. The ratio of CPU demand to memory demand in the experiment is close to 1:2 with an obvious resource bottleneck. The results of this experiment are shown in Table III.

The data of this experiment is a resource bottleneck. SCLPSO has better performance than other compared algorithms. When the demand of VM is small, all algorithms can obtain good solutions. However, with the expansion of VM demand, how to properly deal with the bottleneck of virtual machine resources has become a major problem, and the quality of solutions from other algorithms has declined. To demonstrate the function of the heuristic factor, we design a SCLPSO version without heuristic factor and select PM randomly. As the demand increases, the gap with complete SCLPSO becomes more pronounced.

From the convergence curves in Fig. 2, we find that SCLPSO can converge to a high-quality at early iterations and have a slight improvement at later iterations. Combined with the heuristic factor, the step-by-step solution construction can find suitable PM for each VM and well use physical resources. This will increase the utilization of PMs and obtain high-quality solutions in a few iterations.

SCLPSO uses the heuristic factor η and cut sets to build a solution step by step, and dynamically calculates the relationship between the current VM and the available PMs. In constructing new solutions, SCLPSO tries to find the most suitable physical machine and balance the resource utilization of physical machines at the same time. This strategy can help converge to a high-quality solution in a few calculations.

C. *Test C: Hetergeneous Test*

Both of the above experiments are tested on the same PMs. In this experiment, we considered heterogeneous PMs, that is, PMs have different physical resources. The data size ranges from 100-500. 10% PMs are configured with 32 CPUs and 128 GB of memory, and 90% PMs are configured with 16 CPUs and 32 GB of memory. The CPU demand for VMs is in [1,8] and the memory requirement is in [1,32], which follows a uniform distribution. The experimental results are presented in Table IV.

From the histogram in Fig. 3, we can find that SCLPSO has a great advantage in solving heterogeneous physical machines. With the expansion of the data scale, the advantages become more obvious. This is because SCLPSO considers the relationship between the current VM and the PMs that can be placed when arranging each VM, including CPU and memory, so it can arrange virtual machines relatively well. Compared to FFD, sorting and prioritizing one dimension can easily lead to a situation where the utilization of one resource is high while the other resources are low.

The experimental results show that SCLPSO algorithm performs better than the other comparison algorithms for large-scale VM demand, resource bottleneck or server heterogeneity. S-PSO framework and comprehensive learning strategy can handle large-scale problems well. With the devised heuristic factor and cut set, the solution is constructed step by step. The relationship between current VM and available PMs can be dynamically calculated to ensure each VM is reasonably arranged on the appropriate PM.

VI. Conclusion

In this paper, based on the S-PSO framework and comprehensive learning strategy, we propose SCLPSO to solve the VMP problem which extends the original PSO to discrete searching space. The experimental results show that SCLPSO algorithm can solve large-scale optimization problems well, and it can also have better results on resource bottlenecks and heterogeneous situations. In future researches, we will try to apply SCLPSO in solving other discrete combinatorial problems, such as cloud workflow scheduling and insurance investment planning [31].

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant 61622206 and Grant 61332002, the Natural Science Foundation of Guangdong under Grant 2015A030306024, and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (Corresponding Author: Wei-Neng Chen).

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services & Applications*, vol. 1, no. 1, pp. 7-18, 2010.

[2] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: a survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732-794, 2017.

[3] G. Chen, S. Nath, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of* the *Usenix Symposium on Networked Systems Design and Implementation*, pp. 337-350, 2008.

[4] R. K. Gupta and R. K. Pateriya, "Energy efficient virtual machine placement approach for balanced resource utilization in cloud environment," *International Journal of Cloud-Computing and Super-Computing*, vol. 2, no. 1, pp. 9-20, 2015.

[5] A. Aggarwal and S. Malhotra, "Goals and constraints for devising efficient heuristics in virtual machine placement plan," in *Proceedings of the Second International Conference on Advances in Computing and Communication Engineering*, 2015, pp. 98-103.

[6] A. Hbaieb, M. Khemakhem, and M. B. Jemaa, "Using decomposition and local search to solve large-scale virtual machine placement problems with disk anti-colocation constraints," in *Proceedings of the 14th International Conference*, 2017, pp. 688-695.

[7] Y. Xia, M. Tsugawa, J. Fortes, and S. Chen, "Large-scale VM placement with disk anti-colocation constraints using hierarchical decomposition and mixed integer programming," *IEEE Transactions on Parallel & Distributed Systems*, vol. 28, no. 5, pp. 1361-1374, 2017.

[8] Y. Chang, C. Gu, and F. Luo, "Energy efficient virtual machine consolidation in cloud datacenters," in *Proceedings of the International Conference on Systems and Informatics*, 2017, pp. 401-406.

[9] R. Kashyap, S. Chaudhary, and P. M. Jat, "Virtual machine migration for back-end mashup application deployed on openstack environment," in *Proceedings of the International Conference on Parallel, Distributed and Grid Computing*, 2014, pp. 214-218.

[10] T. H. Duong-Ba, T. Nguyen, B. Bose, and T. T. Tran, "A dynamic virtual machine placement and migration scheme for data centers," *IEEE Transactions on Services Computing*, doi: 10.1109/TSC.2018.2817208, 2018.

[11] S. Telenyk, E. Zharikov, and O. Rolik, "An approach to virtual machine placement in cloud data centers," in *Proceedings of* the *Radio Electronics & Info Communications (UkrMiCo), 2016 International Conference*, pp. 1-6, 2016.

[12] D. A. Alboaneen, H. Tianfield, and Y. Zhang, "Metaheuristic approaches to virtual machine placement in cloud computing: a review," in *Proceedings of the International Symposium on Parallel and Distributed Computing*, 2016, pp. 214-221.

[13] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Proceedings of the International Computer Measurement Group Conference*, 2007, pp. 399-406.

[14] C. Gao, H. Wang, L. Zhai, Y. Gao, and S. Yi, "An energy-aware ant colony algorithm for network-aware virtual machine placement in cloud computing," in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, 2017, pp. 669-676.

[15] S. Jamali and S. Malektaji, "Improving grouping genetic algorithm for virtual machine placement in cloud data centers," in *Proceedings of the International Econference on Computer and Knowledge Engineering*, 2014, pp. 328-333.

[16] W. N. Chen, J. Zhang, H. S. H. Chung, W. L. Zhong, W. G. Wu, and Y. H. Shi, “A novel set-based particle swarm optimization method for discrete optimization problems,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 278-300, 2010.

[17] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, “Comprehensive learning particle swarm optimizer for global optimization of multimodal functions,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281-295, 2006.

[18] Q. Yang, W. N. Chen, J. D. Deng, Y. Li, T. L. Gu, and J. Zhang, “A level-based learning swarm optimizer for large scale optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp.578-584, 2018.

[19] Q. Yang, W.N. Chen, T. L. Gu, H. X. Zhang, J. D. Deng , Y. Li, and et al., “Segment-based predominant learning swarm optimizer for large-scale optimization,” *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2896-2910, 2017.

[20] X. Fan, W. D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer,” *ACM Sigarch Computer Architecture News*, vol. 35, no. 2, pp.13-23, 2007.

[21] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *ACM Sigcomm Computer Communication Review*, vol. 39, no. 1, pp. 68-73, 2008.

[22] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of the International Conference on Neural Networks*, 2002, pp. 1942-1948.

[23] X. Y. Wen, W. N. Chen, Y. Lin, T. L. Gu, H. X. Zhang, Y. Li, *et al.*, “A maximal clique based multiobjective evolutionary algorithm for overlapping community detection,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 3, pp. 363-377, 2017.

[24] Y. H. Jia, W. N. Chen, T. L. Gu, H. X. Zhang, H. Q. Yuan, Y. Lin, *et al.*, “A dynamic logistic dispatching system with set-based particle swarm optimization,” *IEEE Transactions on Systems, Men and Cybernetics: Systems*, vol. 48, no. 9, pp. 1607-1621, 2018.

[25] X. Yu, W. N. Chen, T. L. Gu, H. X. Zhang, H. Q. Yuan, S. Kwong, *et al.*, “Set-based discrete particle swarm optimization based on decomposition for permutation-based multiobjective combinatorial optimization problems,” *IEEE Transactions on Cybernetics*, vol. 48, no. 7, pp. 2139-2153, 2018.

[26] D. Wilcox, A. Mcnabb, and K. Seppi, “Solving virtual machine packing with a reordering grouping genetic algorithm,” *Evolutionary Computation*, vol. 30, pp. 362-369, 2011.

[27] Q. Yang, W. N. Chen, Z. T. Yu, T. L. Gu, Y Li, H. X. Zhang, *et al.*, “Adaptive multimodal continuous ant colony optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 2, pp. 191-205, 2017.

[28] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, “Virtual machine consolidation in cloud data centers using ACO metaheuristic,” in *Proceedings of the European Conference on Parallel Processing*, 2014, pp. 306-317.

[29] M. A. Tawfeek, A. B. EI-Sisi, A. E. Keshk, and F. A. Torkey, “Virtual machine placement based on ant colony optimization for minimizing resource wastage,” in *Proceedings of the International Conference on Advanced Machine Learning Technologies and Applications*, 2014, pp. 153-164.

[30] B. B. J. Suseela, “A multi-objective hybrid ACO-PSO optimization algorithm for virtual machine placement in cloud computing,” *International Journal of Research in Engineering and Technology*, vol. 3, no. 4, pp. 474-476, 2014.

[31] W. Shi, W. N. Chen, Y. Lin, T. L. Gu, S. Kwong, and J. Zhang, “An adaptive estimation of distribution algorithm for multi-policy insurance investment planning,” *IEEE Transactions on Evolutionary Computation*, doi: 10.1109/TEVC.2017.2782571, 2018.