Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 396582, 20 pages
http://dx.doi.org/10.1155/2015/396582

Research Article

DRSCRO: A Metaheuristic Algorithm for Task Scheduling on Heterogeneous Systems

Yuyi Jiang,[1] Zhiqing Shao,[1] Yi Guo,[1,2] Huanhuan Zhang,[1] and Kun Niu[1]

[1]*College of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China*
[2]*School of Information Science and Engineering, Shihezi University, Shihezi 832003, China*

Correspondence should be addressed to Zhiqing Shao; zshao@ecust.edu.cn

Received 12 August 2015; Revised 13 November 2015; Accepted 23 November 2015

Academic Editor: Ching-Ter Chang

An efficient DAG task scheduling is crucial for leveraging the performance potential of a heterogeneous system and finding a schedule that minimizes the *makespan* (i.e., the total execution time) of a DAG is known to be NP-complete. A recently proposed metaheuristic method, Chemical Reaction Optimization (CRO), demonstrates its capability for solving NP-complete optimization problems. This paper develops an algorithm named Double-Reaction-Structured Chemical Reaction Optimization (DRSCRO) for DAG scheduling on heterogeneous systems, which modifies the conventional CRO framework and incorporates CRO with the variable neighborhood search (VNS) method. DRSCRO has two reaction phases for super molecule selection and global optimization, respectively. In the molecule selection phase, the CRO as a metaheuristic algorithm is adopted to obtain a super molecule for accelerating convergence. For promoting the intensification capability, in the global optimization phase, the VNS algorithm with a new processor selection model is used as the initialization under the consideration of scheduling order and processor assignment, and the load balance neighborhood structure of VNS is also utilized in the ineffective reaction operator. The experimental results verify the effectiveness and efficiency of DRSCRO in terms of *makespan* and convergence rate.

1. Introduction

A large application can be decomposed into several smaller models (i.e., tasks) processed in parallel on heterogeneous computing systems. An efficient task scheduling is crucial for leveraging the performance potential of a heterogeneous system. The problem of the task scheduling on heterogeneous system can be stated as assigning the processors to the tasks for minimizing the *makespan* (i.e., the total execution time). As one task is required only after all of its predecessors are executed, these tasks with precedence constraints can be modeled as directed acyclic graphs (DAGs), where the nodes and the directed edges represent the tasks and the communications between the tasks, respectively. Finding a schedule that minimizes the execution time of a parallel program is known to be NP-complete [1]. Therefore, two scheduling strategies, heuristic and metaheuristic, are developed for searching a suboptimal solution with lower execution time.

Heuristic scheduling strategies focus on identifying a solution by exploiting the heuristics, an important class of algorithms based on which is list scheduling [2–12], such as heterogeneous earliest finish time (HEFT) [3]. List scheduling consists of two basic phases, constructing a scheduling list of tasks order by priority of each task and mapping each task to a processor in priority order according to greedy approach (i.e., a task with the highest-priority is assigned to a processor that allows the earliest finish time). The performance of heuristic-based algorithms relied on the effectiveness of the heuristics in a tremendous manner.

Metaheuristic scheduling strategies such as Ant Colony Optimization (ACO) [13], Genetic Algorithms (GA) [14–21], Tabu Search (TS) [22, 23], and Simulated Annealing (SA) [24] search the solution spaces in a direct manner and produce consistent and high quality results on the wide range problems while, in comparison with heuristic-based algorithms, these strategies always cost much more time. The Chemical Reaction Optimization (CRO) is a new metaheuristic method and has shown its efficiency in solving NP-complete problem [25–29]. There are only two CRO-based algorithms [27, 30]

for DAG scheduling on heterogeneous system so far according to our knowledge. These two algorithms both focused on the DAG scheduling with the objective of minimizing the *makespan*. However, as metaheuristic scheduling strategies, CRO-based algorithms for DAG scheduling still have very high time cost and the convergence rates of them also need to be improved. In [30], the concept of super molecule is applied for accelerating convergence and the super molecule is selected by heuristic scheduling strategies. However, the performance of this kind of super molecule selection method is affected by the range of problems.

This paper proposes an algorithm, Double-Reaction-Structured CRO (DRSCRO), for DAG task scheduling on heterogeneous systems to aim at obtaining schedules with better quality. In this paper, the conventional CRO framework scheme is modified and two reaction phases, one for super molecule selection and another for global optimization, are developed in DRSCRO. CRO as a metaheuristic algorithm is utilized in the molecule selection phase to obtain a super molecule [31] for better convergence rate. And the variable neighborhood search (VNS) algorithm [32] method with a new processor selection model, as well as its neighborhood structure, is also utilized to promote the intensification capability in the global optimization phase.

There are three major contributions of this work:

(1) Developing DRSCRO by modifying the conventional CRO framework and utilizing a metaheuristic method to obtain a super molecule for accelerating convergence.

(2) Utilizing the VNS [32] algorithm with a new processor selection model as the global optimization phase initialization, which takes into account the optimization of the scheduling order and processor assignment, and applying one of its neighborhood structures in the reaction operator to promote the intensification capability of DRSCRO.

(3) Conducting simulation experiments to prove the efficiency and effectiveness of DRSCRO in terms of *makespan* and convergence rate.

The next section introduces relevant research works on the DAG scheduling problem on heterogeneous systems. Section 3 describes the models of the studied problem as formal statement. Section 4 presents the design of the proposed DRSCRO for DAG scheduling. In Section 5, the simulation performance of DRSCRO is analyzed and compared with some existing scheme algorithms. Section 6 draws the conclusions of this paper and the suggestions for future research.

2. Literature Review

The DAG scheduling problem, which has been proven to be NP-hard in general [1], can be formulated as the search for an optimal solution to the assignment of the tasks in DAG onto a set of processors, to minimize the total scheduling length (i.e., *makespan*). There are two main categories, heuristic (deterministic) and metaheuristic (nondeterministic), of the various scheduling algorithms proposed over the last decade. As metaheuristic methods, CRO-based algorithms for DAG scheduling on heterogeneous systems are based on Chemical Reaction Optimization (CRO) algorithm, which was proposed very recently and has shown its power to deal with NP-complete problems.

2.1. Heuristic and Metaheuristic Methods. The heuristic methods are on the basis of the heuristics which are extracted from intuitions, and the most important class of them is list scheduling algorithms [2–12]. The HEFT algorithm, which was proposed by Topcuoglu et al. [3], utilizes the information of execution cost on average of each task as an upward-ranking heuristic to calculate the task priority. At each step of HEFT, the task with the highest value of upward rank is selected and mapped to the processor with a greedy approach (i.e., the assigned processor minimizes the earliest finish time of the selected task). Experimental results prove that HEFT obtains better performance on schedule quality and computational cost than the other list scheduling algorithms. The performance of heuristic-based algorithms heavily relied on the effectiveness of the heuristics. The higher complexity DAG scheduling problems have, the harder greedy heuristics produce consistent results on a wide range of problems. In particular, GA has been widely used to evolve solutions for many task scheduling problems as the most representative metaheuristic method [21]. Different from heuristic-based algorithms, the metaheuristic methods use a guided-random-search-based process for solution searching. They typically require sufficient sampling of candidate solutions in the search space and have shown robust performance on a variety of scheduling problems. For solving DAG scheduling problem successfully, many metaheuristic algorithms have been utilized such as GA [14–21], ACO [13], SA [24], TS [22, 23], CRO [27, 30], VNS [21], and energy-efficient stochastic [33].

According to No-Free-Lunch Theorem [34], all well-designed metaheuristic methods have the same performance on searching for optimal solutions when averaged over all possible fitness functions. In comparison with the heuristic methods, the metaheuristic methods, which always have much higher computational cost, can obtain better performance in terms of schedule quality, because the metaheuristic methods can search a wider area of the solution space with the guided-random-search-based processes for solution searching, while the search of the heuristic-based algorithms are narrowed down to a very smaller portion by means of the heuristics.

2.2. CRO-Based Algorithms for DAG Scheduling on Heterogeneous Systems. CRO was proposed by Lam and Li very recently [25], and, as far as we know, as metaheuristic methods, Double Molecular Structure-Based Chemical Reaction Optimization (DMSCRO) [27] and Tuple-Based Chemical Reaction Optimization (TMSCRO) [30] are the only two CRO-based algorithms for DAG scheduling on heterogeneous systems. CRO-based algorithms mimic the chemical reaction process, which accords with energy conservation, in a closed container. The molecules with two kinds of energy, potential energy (PE) and kinetic energy (KE), in CRO-based

Table 1: Parameters used in CRO.

Parameters	Definition
PE	Current potential energy of a molecule
KE	Current kinetic energy of a molecule
InitialKE	Initial kinetic energy of a molecule
θ	Threshold value guides the choice of on-wall collision or decomposition
ϑ	Threshold value guides the choice of intermolecule collision or synthesis
Buffer	Initial energy in the central energy buffer
KELossRate	Loss rate of kinetic energy
MoleColl	Threshold value to determine whether to perform a unimolecule reaction or an intermolecule reaction
PopSize	Size of the molecules
iters	Number of iterations

algorithms are the solutions to DAG scheduling problem. The PE value of a molecule is calculated by fitness function, which is equal to the objective value, *makespan*, of the corresponding solution. And KE is for helping the molecule escape from local optimums and its value is nonnegative. A buffer is also used in CRO-based algorithms for energy interchange and conservation. Moreover, to find the solution with the global minimal *makespan*, four types of elementary chemical reactions, on-wall ineffective collision, decomposition, intermolecular ineffective collision, and synthesis, are applied for the intensification and the diversification searches. The typical execution flow of CRO framework adopted in DMSCRO and TMSCRO is as proposed in [25] and the parameters used in CRO are presented in Table 1.

As metaheuristic methods, DMSCRO and TMSCRO have better performance in terms of schedule quality than heuristic methods and the reason is as presented in the last paragraph of Section 2.1. The experimental results in [27, 30] prove that both of DMSCRO and TMSCRO outperform GA. DMSCRO is the first algorithm by applying CRO proposed by Lam and Li in [25] to solve the DAG scheduling problem, and it enjoys the advantages of both GA and SA. On the one hand, the intermolecular collision and on-wall collision designed in DMSCRO have similar effect to the crossover operation and the mutation operation in GA, respectively. On the other hand, the energy conservation requirement in DMSCRO is able to guide the searching of the optimal solution similarly to the way the Metropolis Algorithm of SA guides the evolution of the solutions in SA. Two additional operations, decomposition and synthesis, give DMSCRO more opportunities to jump out of the local optimum and explore the wider areas in the solution space. This benefit enables DMSCRO to find good solutions faster than GA, which has been widely used to evolve solutions for many task scheduling problems. DMSCRO are not compared with SA in [27, 30], because the underlying principles and philosophies between DMSCRO and SA differ a lot [27]. Typically, metaheuristic algorithms like CRO-based algorithm of GA-based algorithms operating on a population of solutions are able to find good solutions faster than that operating on a single solution like SA-based algorithms. Comparing with DMSCRO, TMSCRO applies constrained earliest finish time algorithm to data pretreatment to take the advantage of the super molecule and constrained critical paths [35], which is, as heuristic information, for accelerating convergence. Moreover, the molecule structure and elementary reaction operators design in TMSCRO are more reasonable than those in DMSCRO on intensification and diversification of searching the solution space.

However, for solving the NP problem of DAG scheduling on heterogeneous systems, CRO-based algorithms, TMSCRO and DMSCRO, still have very large time expenditure as metaheuristic scheduling strategies; therefore, the searching capabilities and convergence rates of them need to be improved. There are three deficiencies of TMSCRO and DMSCRO. First, in [30], the concept of super molecule is applied for accelerating convergence and the super molecule is selected by heuristic scheduling strategies, but the performance of this kind of super molecule selection method is affected by the range of problems. Second, in both TMSCRO and DMSCRO, the initial molecules, which are very important for the whole searching process, are randomly created, and the uncertainty of this kind of initialization undermines the searching capabilities of TMSCRO and DMSCRO. Moreover, the intensification capabilities of CRO-based algorithms for DAG scheduling also need to be improved, to obtain better performances of the average results when the iteration stopping criterions are satisfied.

Therefore, this paper proposes an algorithm, Double-Reaction-Structured CRO (DRSCRO), for DAG task scheduling on heterogeneous systems to aim at obtaining schedules with better quality. In this paper, the conventional CRO framework scheme is modified and two reaction phases, one for super molecule selection and another for global optimization, are developed in DRSCRO. CRO as a metaheuristic algorithm is utilized in the molecule selection phase to obtain a super molecule [31] for better convergence rate. Moreover, in the global optimization phase, the variable neighborhood search (VNS) algorithm method [21, 32, 36], which is an effective metaheuristic with the utilizations of neighborhood structures and a local search to change the neighborhood systematically, is used to optimize the initial molecule, and one of its neighborhood structures is also adopted in the reaction operator to promote the

intensification capability. And there is a new model proposed for processor selection utilized in the neighborhood structures of the VNS algorithm for better effectiveness.

Moreover, in [21], VNS was incorporated with GA for DAG scheduling, but the task priority was unchangeable in the VNS algorithm in [21], which reduces the efficiency of VNS to obtain a better solution. So, different from [21], to promote the intensification capability of the whole algorithm, the VNS in DRSCRO is modified under the consideration of the optimization of the scheduling order and the processor assignment both.

3. Problem Formulation

The DAG scheduling problem is typically with two inputs: a heterogeneous system for task computing in parallel and a parallel program of application (i.e., DAG). In this paper, the heterogeneous system is assumed as a static computing system model presented by $P = \{p_i \mid i = 1, 2, 3, \ldots, |P|\}$, which is a fully connected network of processors. The heterogeneity level in this paper is formulated as $(1+\text{hl}\%)/(1-\text{hl}\%)$, where the parameter $\text{hl} \in (0, 1)$. In this paper, $\text{EcCost}_{p_j}(v_i)$ represents the computation cost of a task v_i mapped to the processor p_j and the value of each $\text{EcCost}_{p_j}(v_i)$ is randomly chosen within the scope of $[1 - \text{hl}\%, 1 + \text{hl}\%]$.

In general, $\text{DAG} = (V, E)$ consists of a task (node) set V and an edge set E. $\text{EcCost}_{p_j}(v_i)$ is as defined in the first paragraph of this section, and the same processor executes a task in the DAG without preemption. The constraint between tasks v_i and v_j is denoted as the edge $e_{i,j}$ ($e_{i,j} \in E$), which means that the execution of task v_j only after the execution result of task v_i has been transmitted to task v_j. Each edge $e_{i,j}$ has a nonnegative weight $\text{comm}(v_i, v_j)$ denoting the communication cost between v_i and v_j. Each task in a DAG can only be executed on one processor and the communication can be performed simultaneously by the processors. In addition, when two communicating tasks are mapped to the same processor, the communication cost of them is zero. Predecessor (v_i) represents the set of the predecessors of v_i, while successor (v_i) represents the set of the successors of v_i. The task with no predecessor is denoted as v_{entry} while the task with no successor is denoted as v_{exit}.

Consider that there is a DAG with $|V|$ tasks to be mapped to a heterogeneous system with $|P|$ processors. Assuming the highest-priority ready task v_i on the processor p_j, the earliest start time of v_i, $T_{\text{ESTime}}(v_i, p_j)$, can be formulated as

$$T_{\text{ESTime}}\left(v_i, p_j\right) = \max\left\{T_{\text{avail}}\left(p_j\right), T_{\text{ready}}\left(v_i, p_j\right)\right\}, \quad (1)$$

where $T_{\text{avail}}(p_j)$ can be defined as (2). $T_{\text{avail}}(p_j)$ is the time when processor p_j is available to the execution of the task v_i:

$$T_{\text{avail}}\left(p_j\right) = \max_{v_k \in \text{exec}(p_j)} T_{\text{AFTime}}\left(v_k\right), \quad (2)$$

where $\text{exec}(p_j)$ represents all the tasks which have already been scheduled on the processor p_j and $T_{\text{AFTime}}(v_k)$ denotes the actual finish time when the task v_k finishes its execution. $T_{\text{ready}}(v_i, p_j)$ in (1) represents the time when all the data needed for the process of v_k have been transmitted to p_j, which is formulated as

$$T_{\text{ready}}\left(v_i, p_j\right) = \max_{v_k \in \text{predecessor}(v_i)} \left\{T_{\text{AFTime}}\left(v_k\right) + \text{comm}\left(v_k, v_i\right)\right\}, \quad (3)$$

where $T_{\text{AFTime}}(v_k)$ has the same definition in (2) and predecessor (v_i) denotes the set of all the immediate predecessors of task v_i. $\text{comm}(v_k, v_i)$ is 0 if the task v_k and task v_i are mapped to the same processor p_j.

If task v_i is mapped to the processor p_j with nonpreemptive processing approach, the earliest finish time of task v_i, $T_{\text{EFTime}}(v_i, p_j)$, is formulated as

$$T_{\text{EFTime}}\left(v_i, p_j\right) = T_{\text{ESTime}}\left(v_i, p_j\right) + \text{EcCost}_{p_j}\left(v_i\right). \quad (4)$$

After the task v_i is executed by the processor p_j, $T_{\text{EFTime}}(v_i, p_j)$ is assigned to $T_{\text{AFTime}}(v_i)$. The *makespan* of the entire parallel program is equivalent to the actual finish time of exit task v_{exit}:

$$makespan = \max_{v_i \in V}\left\{T_{\text{AFTime}}\left(v_i\right)\right\} = T_{\text{AFTime}}\left(v_{\text{exit}}\right). \quad (5)$$

The computation of the communication-to-computation ratio (CCR) can be formulated as in

$$\text{CCR} = \frac{\sum_{e_{i,j} \in E} \text{comm}\left(v_i, v_j\right)}{\sum_{v_i \in V} \overline{W\left(v_i\right)}}, \quad (6)$$

where $\overline{W(v_i)}$ is the average computation cost of task v_i and it can be calculated as follows:

$$\overline{W\left(v_i\right)} = \sum_{k=1}^{|P|} \frac{\text{EcCost}_{p_k}\left(v_i\right)}{|P|}. \quad (7)$$

A simple four-task DAG and a heterogeneous computation system with three processors are shown in Figures 1(a) and 1(b), respectively. The definition of the notations can be found in Table 2.

4. Design of DRSCRO

DRSCRO imitates the molecular interactions in chemical reactions based on the concepts of atoms, molecule, molecular structure, and energy of a molecule. In DRSCRO, a molecule corresponds to a scheduling solution in DAG scheduling, with a unique molecular structure representing the atom positions in a molecule. We utilize the molecular structure of TMSCRO in our work, under the consideration of its capability to represent the constrained relationship between the tasks in a molecule (solution). In addition, the energy of each molecule corresponds to the fitness value of a solution. The molecular interactions try to reconstruct more stable molecular structure with lower energy. There are four kinds of basic chemical reactions, on-wall ineffective

Table 2: Definitions of notations.

Notations	Definitions
DAG = (V, E)	Input directed acyclic graph with $\|V\|$ nodes representing tasks and $\|E\|$ edges representing constrained relations among the tasks
$P = \{p_i \mid i = 1, 2, 3, \ldots, \|P\|\}$	Set of heterogeneous processors in target system
$\text{EcCost}_{p_j}(v_i)$	Execution cost of task v_i using processor p_j
$\text{comm}(v_k, v_i)$	Communication cost from task v_k to task v_i
$T_{\text{ESTime}}(v_i, p_j)$	The earliest start time of task v_i which is mapped to processor p_j
$T_{\text{EFTime}}(v_i, p_j)$	The earliest finish time of task v_i which is mapped to processor p_j
$T_{\text{avail}}(p_j)$	The time when processor p_j is available
$T_{\text{ready}}(v_i, p_j)$	The time when all the data needed for the process of v_i have been transmitted to p_j
$T_{\text{AFTime}}(v_k)$	Actual finish time when task v_k finishes its execution
$\text{predecessor}(v_i)$	Set of the predecessors of task v_i
$\text{successor}(v_i)$	Set of the successors of task v_i
$\text{exec}(p_j)$	Set of the tasks which have already been scheduled on the processor p_j
$\overline{W(v)}$	Average computation cost of task v
CCR	Communication-to-computation ratio
hl	Parameter for adjusting the heterogeneity level in a heterogeneous system

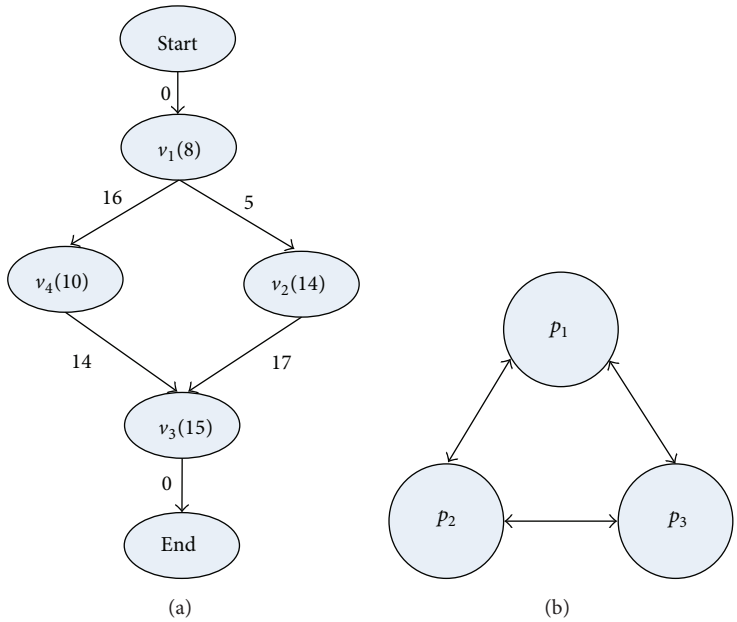

Figure 1: (a) DAG model. (b) Heterogeneous computation system model.

collision, decomposition, intermolecular ineffective collision, and synthesis, for molecular interactions in DRSCRO, and each kind of reaction contains two subclasses. These two subclasses of reaction operators are applied in the phase of super molecule selection and the phase of global optimization, respectively.

4.1. Framework of DRSCRO. The framework of DRSCRO to schedule a DAG job is as shown in Figure 2 with two basic phases, the phase of super molecule selection and the phase of global optimization. In each phase, DRSCRO first initializes the process of a phase, and then the phase process enters iteration.

In this framework, DRSCRO first executes the phase of super molecule selection to obtain the super molecule (i.e., just the molecule with the global minimal *makespan*), SMole, with other output molecules as the input of the next global optimization phase for the first time (the input of VNS algorithm is the population with SMole after each iteration in the global optimization phase in the other times), and then DRSCRO performs the phase of global optimization to approach the global optimum solution. The VNS algorithm

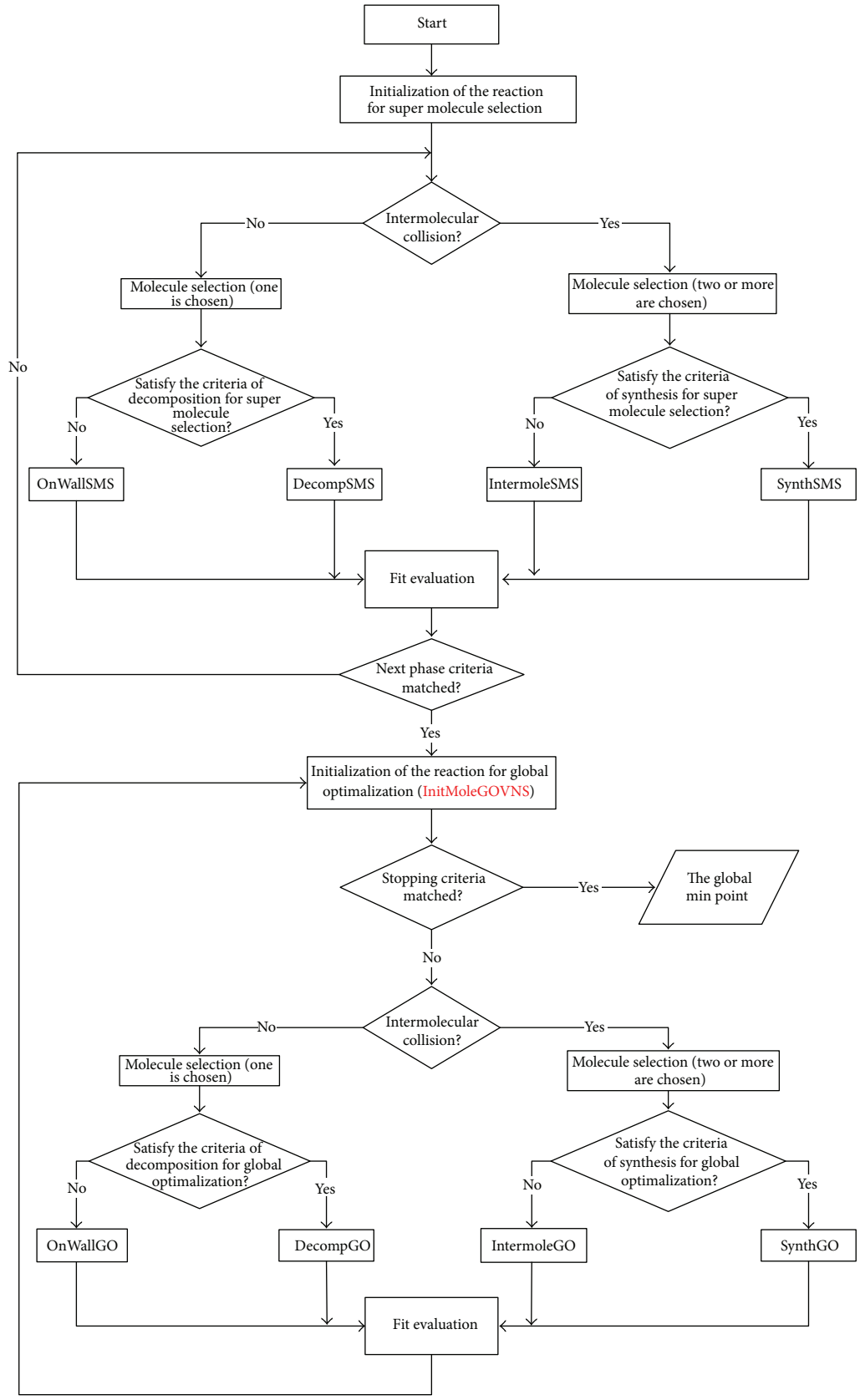

Figure 2: Framework of DRSCRO.

(1) *makespan* = 0;
(2) **for** each node v_k in $\mathbf{m} = ((v_1, f_1, p_1), (v_2, f_2, p_2), \ldots, (v_{|V|}, f_{|V|}, p_{|V|}))$ do
(3) calculate the actual finish time of v_k (i.e. $T_{\text{AFTime}}(v_k)$)
(4) **if** *makespan* < $T_{\text{AFTime}}(v_k)$
(5) update *makespan*
(6) *makespan* = $T_{\text{AFTime}}(v_k)$
(7) **end if**
(8) **end for**
(9) **return** *makespan*;

Algorithm 1: Fit(**m**) calculating the fitness value of a molecule and the processor allocation optimization.

with a new model for processor selection is adopted as the initialization of the global optimization phase, and it is also utilized as a local search process to promote the intensification capability of DRSCRO. There are four kinds of elementary chemical reaction in DRSCRO, on-wall collision, decomposition, intermolecular collision, and synthesis. And each kind of reaction contains two types of operators which are, respectively, utilized in two phases of DRSCRO. In each iteration, one of the elementary chemical reaction operators is performed to generate new molecules and the PEs of the newly generated molecules (i.e., the fitness function values of the newly generated molecules) will be calculated. In addition, SMole will be tracked and only participates in on-wall ineffective collision and intermolecular ineffective collision in the global optimization phase to explore as much as possible the solution space in its neighborhoods and the main purpose is to prevent the super molecule from changing dramatically. The iteration of each phase repeats until the stopping criteria (or next phase criteria) are met, and SMole and its fitness function value are just the final solution and *makespan* (i.e., global min point), respectively. In the implementations of the experiments in this paper, the next phase criteria and the stop criteria of DRSCRO are set as when there is no *makespan* improvement after 10000 consecutive iterations in the search loop.

4.2. Molecular Structure and Fitness Function. This subsection presents the encoding of scheduling solutions (i.e., the molecular structure) and the statement of the fitness function in DRSCRO.

4.2.1. Molecular Structure. In this paper, an atom with three elements can be denoted as a tuple (v_i, f_i, p_i) and the molecular structure M with an array of tuples can be formulated as in (8) to represent a solution to the DAG scheduling problem. The order of the tuples in M represents the priority of each DAG task v_i with the allocated processor p_i, and $\mathbf{V} = (v_1, v_2, \ldots, v_{|V|})$ is a topological sequence of DAG, which is with the hypothetical entry task (with no predecessors) v_1 and exit task (with no successors) $v_{|V|}$, respectively, representing the beginning and end of execution. Moreover, if tuple **A** is before tuple **B** and $v_{\mathbf{A}}$ is the predecessor of $v_{\mathbf{B}}$ in DAG, the second integer of tuple **B**, $f_{\mathbf{B}}$, will be 1, and vice versa

$$\mathbf{m} = \left((v_1, f_1, p_1), (v_2, f_2, p_2), \ldots, (v_{|V|}, f_{|V|}, p_{|V|})\right). \quad (8)$$

4.2.2. Fitness Function. Potential energy (PE) is defined as the fitness function value of the corresponding solution represented by *S*. The overall schedule length of the entire DAG, namely, *makespan*, is the largest finish time among all tasks, which is equivalent to the actual finish time of the exit node in DAG. In this paper, the goal of DAG scheduling problem by DRSCRO is to obtain the scheduling that minimizes *makespan* and ensure that the precedence of the tasks is not violated. Hence, each fitness function value is defined as

$$\text{Fit}(\mathbf{m}) = \text{PE}_{\mathbf{m}} = \textit{makespan}. \quad (9)$$

Algorithm 1 presents how to calculate the value of the optimization fitness function Fit(**m**).

4.3. Super Molecule Selection Phase

4.3.1. Initialization. There are two kinds of initial molecule generator, one used in the phase of super molecule selection and the other used in the phase of global optimization, to generate the initial solutions for DRSCRO to manipulate. The tuples of the first molecule **m** used in the initialization of the phase of super molecule selection are ascendingly ordered by the upward rank value [27] of their v_i, and element three p_i of each tuple is generated by a random perturbation. The upward rank value can be calculated by

$$\begin{aligned}&\text{Rank}(v_i)\\ &\quad = \overline{W(v_i)}\\ &\qquad + \max_{v_j \in \text{successor}(v_i)} \left\{\text{comm}(v_i, v_j) + \text{Rank}(v_j)\right\}.\end{aligned} \quad (10)$$

A detailed description of the initial molecule generator of the super molecule selection phase is given in Algorithm 2. For the first input molecule **m**, p_x in each tuple in **m** is set as p_1.

4.3.2. Elementary Chemical Reaction Operators. In DRSCRO, the operators for super molecule selection just randomly change p_i of each tuple in a molecule as the intensification searches or the diversification searches [25] to optimize the processor mapping of a solution. Figures 3, 4, 5, and 6, respectively, show the examples of four operators for super

(1) $MoleN = 1$;
(2) **while** $MoleN \leq PopSize$ do
(3) **for** each p_i in molecule **m** to randomly change;
(4) change p_i randomly
(5) **end for**
(6) generate a new molecule $\mathbf{m}'$;
(7) $MoleN = MoleN + 1$;
(8) **end while**

Algorithm 2: InitMoleSMS(**m**) generating the initial population for the super molecule selection phase.

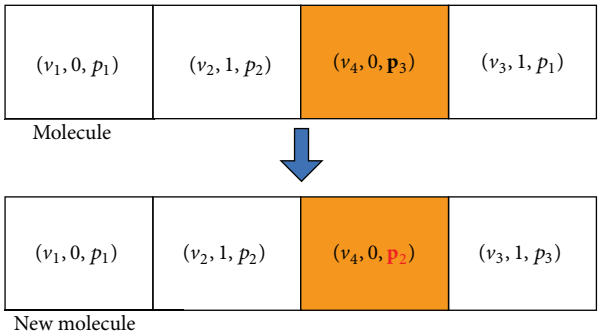

Figure 3: Example of OnWallSMS.

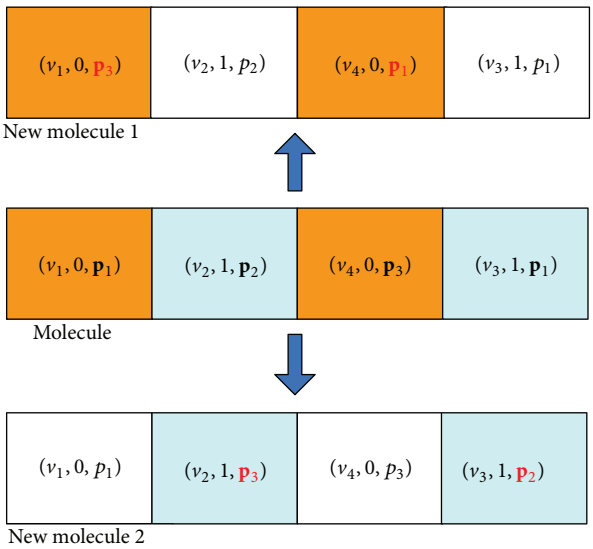

Figure 4: Example of DecompSMS.

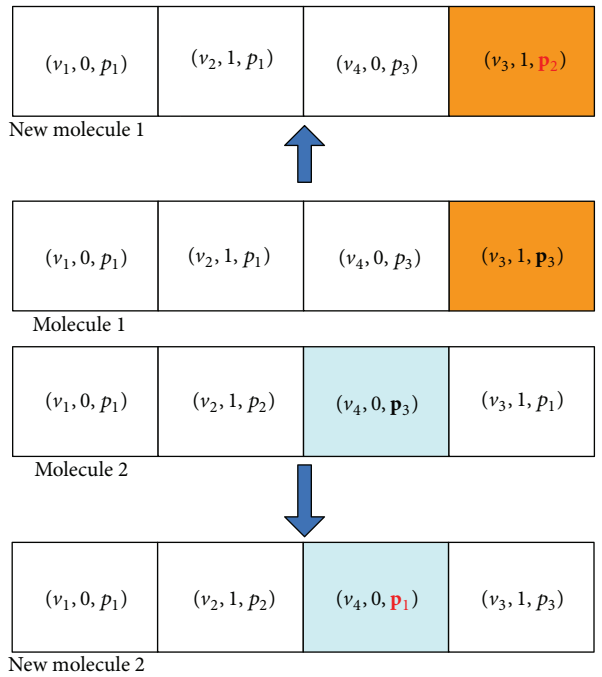

Figure 5: Example of IntermoleSMS.

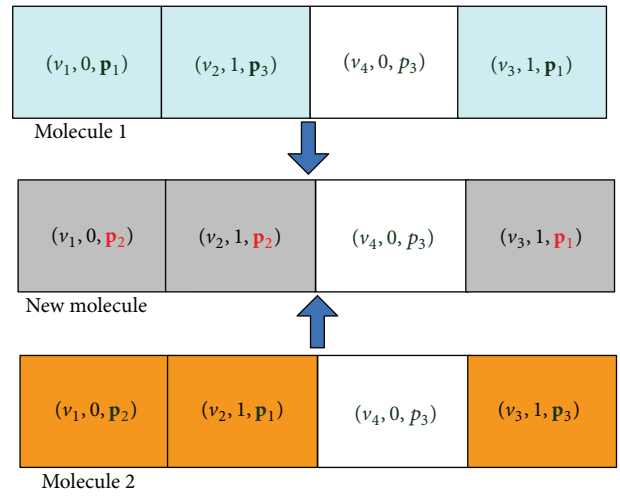

Figure 6: Example of SynthSMS.

molecule selection, in which the molecules correspond to the DAG as shown in Figure 1(a). And the white blocks in these examples denote the tuples that do not change during the reaction operation calculations.

As shown in Figure 3, the operator, OnWallSMS, is used to generate a new molecule $\mathbf{m}'$ from a given reaction molecule **m** for optimization. OnWallSMS works as follows: (1) The operator randomly chooses a tuple (v_i, f_i, p_i) in **m**. (2) The operator changes p_i randomly. In the end, the operator generates a new molecule $\mathbf{m}'$ from **m** as an intensification search.

As shown in Figure 4, the operator, DecompSMS, is used to generate new molecules $\mathbf{m}'_1$ and $\mathbf{m}'_2$ from a given reaction molecule **m**. DecompSMS works as follows: (1) The operator generates two molecules $\mathbf{m}'_1 = \mathbf{m}$ and $\mathbf{m}'_2 = \mathbf{m}$. (2) The operator keeps the tuples in $\mathbf{m}'_1$, which is at the odd position in **m** and then changes the remaining p_x's of tuples in $\mathbf{m}'_1$, randomly. (3) The operator retains the tuples in $\mathbf{m}'_2$, which is at the even position in **m**, and then changes the remaining p_x's of tuples in $\mathbf{m}'_2$ randomly. In the end, the operator generates two new molecules $\mathbf{m}'_1$ and $\mathbf{m}'_2$ from **m** as a diversification search.

(1) *tempSet* = *pop_set*;
(2) *pop_subset* = ∅;
(3) **if** *pop_set* is the input of the VNS algorithm for the first time (i.e. the output of the super molecule selection phase)
(4) **for** each **tempS** in *tempSet* except **SMole**
(5) choose a tuple (v_i, f_i, p_i) in **tempS**, where $f_i = 0$, randomly;
(6) generate a random number $rnd \in (0, 1)$;
(7) **if** $rnd \geq 0.5$
(8) find the first predecessor $v_j = \text{Pred}(v_i)$ from v_i to the begin in molecule **tempS**;
(9) interchanged position of (v_i, f_i, p_i) and $(v_{j+1}, f_{j+1}, p_{j+1})$ in molecule **tempS**;
(10) update f_i, f_{i+1} and f_{j+1} as defined in the last paragraph of Section 4.2.1.
(11) **end if**
(12) **for** each p_i in molecule **tempS** to randomly change;
(13) change p_i randomly
(14) **end for**
(15) **if** Fit(**tempS**) < Fit(**SMole**)
(16) **SMole** = **tempS**;
(17) **end if**
(18) **end for**
(19) **end if**
(20) *pop_subset* adds **SMole**;
(21) *pop_subset* adds the molecules in *tempSet* with tuple order different from **SMole**;
(22) **while** |*pop_subset*| ≠ *pop_subset_num* **do**
(23) *pop_subset* add a molecules in *pop_set* which do not exist in *pop_subset*;
(24) **end while**

Algorithm 3: InitVNS(*pop_set*, *pop_subset_num*) initializing the subset of the population *pop_set* for undergoing the VNS algorithm.

As shown in Figure 5, the operator, IntermoleSMS, is used to generate new molecules $\mathbf{m}'_1$ and $\mathbf{m}'_2$ from given molecules $\mathbf{m}_1$ and $\mathbf{m}_2$. This operator first uses the steps in OnWallSMS to generate $\mathbf{m}'_1$ from $\mathbf{m}_1$, and then the operator generates the other new molecule $\mathbf{m}'_2$ from $\mathbf{m}_2$ in the similar fashion. In the end, the operator generates two new molecules $\mathbf{m}'_1$ and $\mathbf{m}'_2$ from $\mathbf{m}_1$ and $\mathbf{m}_2$ as an intensification search.

As shown in Figure 6, the operator, SynthSMS, is used to generate a new molecule $\mathbf{m}'$ from given molecules $\mathbf{m}_1$ and $\mathbf{m}_2$. SynthSMS works as follows: The operator keeps the tuples in $\mathbf{m}'$, which is at the same position in $\mathbf{m}_1$ and $\mathbf{m}_2$ with the same p_x's, and then changes the remaining p_y's in $\mathbf{m}'$, randomly. As a result, the operator generates $\mathbf{m}'$ from $\mathbf{m}_1$ and $\mathbf{m}_2$ as a diversification search.

4.4. Global Optimization Phase

4.4.1. Initialization. VNS is utilized by our proposed algorithm as the initialization of the global optimization phase and it is also as a local search process to promote the intensification capability of DRSCRO during the running of the whole algorithm.

Algorithms 3 and 4, respectively, present the subset generator of the phase output and the main steps of the whole VNS algorithm (i.e., the initialization of the global optimization phase). In DRSCRO, the VNS algorithm only processes the subset of the population with the super molecule, SMole, after each iteration in the global optimization phase (the output of super molecule selection phase is the input of VNS for the first time). As presented in Algorithm 3, if the *pop_set* (i.e., the set of population) is the output of the super molecule selection phase, the tuple orders and p_is of its elements will be adjusted. *pop_subset* is the subset of population and *pop_subset_num* is the number of the elements in *pop_subset*, which is set as $PopSize \times 50\%$ in this paper.

In Algorithm 4, different from the VNS proposed in [21], the task priority was changeable in the VNS algorithm used in DRSCRO, the reason for which is that the unchangeable task priority in the VNS reduces its efficiency to obtain a better solution. Therefore, under the consideration of the optimization of the scheduling order and the processor assignment both, the input molecules of VNS can be with different tuple order (i.e., task priority) as presented in Algorithm 3 in each iteration. $d_{\max}$ is set to 2 as presented in [37]. As the essential factor of VNS, two neighborhood structures, load balance and communication reduction neighborhood structures, which demonstrate their power in solving DAG scheduling problem on heterogeneous systems as presented in [21], are adopted by the VNS algorithm in DRSCRO for their high efficiency. In this paper, a new model is also proposed for processor selection of these two neighborhood structures. As presented in [21], there are two intuitions used to construct the neighborhood structures. One is that balancing load among various processors usually helps minimizing the *makespan*, especially when most tasks are allocated to only a few processors; the other is that reducing communication overhead and idle waiting time of processors always results in a more effective schedule, especially given a relatively high unit communication. However, there is a contradiction between these two intuitions, because reducing communication overhead and idle waiting time of processors always means that some processors are with most tasks.

(1) *pop_subset* = InitVNS(*pop_set*);
(2) Select the set of neighborhood structures $Neigh_d$ ($d = 1, 2, 3, \ldots, d_{\max}$);
(3) **for** each individual **m** in the *pop_subset* **do**
(4) $d = 1$;
(5) **while** $d < d_{\max}$ **do**
(6) Randomly generate a molecule $\mathbf{m}_1$ from the dth neighborhood of **m**;
(7) Apply some local search method with $\mathbf{m}_1$ as the initial molecule (the local optimum presented by $\mathbf{m}_2$);
(8) **If** $\mathbf{m}_2$ is better than **m**
(9) $\mathbf{m} = \mathbf{m}_2$;
(10) $d = 1$;
(11) **else**
(12) $d = d + 1$;
(13) **end if**
(14) **end while**
(15) **until** the termination condition is satisfied
(16) **end for**
(17) execute the combination strategy;

Algorithm 4: InitMoleGOVNS(*pop_set*) generating the initial population of the global optimization phase.

(1) **for** each processor p_i in the solution ω **do**
(2) Compute Tend(p_i);
(3) **end for**
(4) Choose the processor $p_{\max}$ with the largest Tend($p_{\max}$);
(5) Randomly choose a task v_{random} from exec($p_{\max}$);
(6) Randomly choose a processor p_{random} different from $p_{\max}$;
(7) Reallocate v_{random} to the processor p_{random};
(8) Encode and reschedule the changed solution $\boldsymbol{\omega}'$;
(9) **return** $\boldsymbol{\omega}'$;

Algorithm 5: GenNeighborhoodofLoadBalance($\boldsymbol{\omega}$).

So, different from the original ones in [21], we develop a new model for processor selection. Let $\text{TC}_{\text{load}}(p_i)$ be all the task execution cost of processor p_i, and $\text{TC}_{\text{comm}}(p_i)$ is the communication cost overhead of processor p_i as defined in [21]. The values of $\text{TC}_{\text{load}}(p_i)$ and $\text{TC}_{\text{comm}}(p_i)$ are the tendencies of load balancing and communication reducing, respectively (i.e., the tendency of task reducing or increasing). The greater $\text{TC}_{\text{load}}(p_i)$ is the stronger tendency of reducing tasks on p_i is, and the greater $\text{TC}_{\text{comm}}(p_i)$ is the stronger tendency of increasing tasks on p_i is. Therefore, a parameter Tend(p_i) is developed to measure the tendency with the combination of $\text{TC}_{\text{load}}(p_i)$ and $\text{TC}_{\text{comm}}(p_i)$ as (11). The neighborhood structure computation processes of load balance and communication reduction are as presented in Algorithms 5 and 6, respectively. The proposed model is under the comprehensive consideration of both intuitions and can make the VNS algorithm more effective than the original one:

$$\text{Tend}(p_i) = \frac{1}{1 + e^{-(\text{TC}_{\text{load}}(p_i)/\text{TC}_{\text{comm}}(p_i))}}. \tag{11}$$

The VNS algorithm in DRSCRO utilizes the dual termination criteria. The termination criterion 1 sets the upper bound of the local search iterations to 20, and the termination criterion 2 sets the maximum iteration number without improvement to 3. The VNS algorithm will stop if either criterion is satisfied. To form a new initial population, a combination strategy is utilized for combining the current population and the VNS output after the VNS algorithm outputs the subset of the population. The current population and the VNS output are first merged and sorted by increasing *makespan*; then the first *PopSize* molecules are selected to generate the new initial population.

4.4.2. Elementary Chemical Reaction Operators. The operators for global optimization not only vary p_i of each tuple but also interchange the positions of the tuples in a molecule as the intensification searches or the diversification searches [25] to optimize the whole solution.

On-wall ineffective collision (as an intensification search), decomposition (as a diversification search), and synthesis (as a diversification search) are as presented in [30], and we do not repeat them here to focus on our main work. In [30], the function of the ineffective collision operator is similar to that of the on-wall ineffective collision operator. Therefore, different from [30], a modified ineffective collision operator is proposed in this paper, and it utilized the load balance neighborhood structure used in the VNS mentioned

(1) **for** each processor p_i in the solution $\boldsymbol{\omega}$ **do**
(2) Compute Tend(p_i);
(3) **end for**
(4) Choose the processor $p_{\min}$ with the smallest Tend($p_{\min}$);
(5) Set the candidate set, *cand*, empty;
(6) **for** each task v_i in the set exec($p_{\min}$) **do**
(7) Compute the set predecessor(v_i);
(8) Update predecessor(v_i) with predecessor(v_i) = predecessor(v_i) − exec($p_{\min}$);
(9) *cand* = *cand* + predecessor(v_i);
(10) **end for**
(11) Randomly choose a task v_{random} from *cand*;
(12) Reallocate v_{random} to the processor $p_{\min}$;
(13) Encode and reschedule the changed solution $\boldsymbol{\omega}'$;
(14) **return** $\boldsymbol{\omega}'$;

Algorithm 6: GenNeighborhoodofCommReduction($\boldsymbol{\omega}$).

(1) choose randomly a tuple (v_i, f_i, p_i) in $\mathbf{m}_1$ where $f_i = 0$;
(2) exchange the positions of (v_i, f_i, p_i) and $(v_{i-1}, f_{i-1}, p_{i-1})$;
(3) modify f_{i-1}, f_i and f_{i+1} in $\mathbf{m}_1$ as defined in the last paragraph of Section 4.2.1;
(4) generate a new molecule $\mathbf{m'}_1$ = GenLBNeighborhood($\mathbf{m}_1$);
(5) choose randomly a tuple (v_i, f_i, p_i) in $\mathbf{m}_2$ where $f_j = 0$;
(6) exchange the positions of (v_i, f_i, p_i) and $(v_{j-1}, f_{j-1}, p_{j-1})$;
(7) modify f_{j-1}, f_j and f_{j+1} in $\mathbf{m}_2$ as defined in the last paragraph of Section 4.2.1;
(8) generate a new molecule $\mathbf{m'}_2$ = GenLBNeighborhood($\mathbf{m}_2$);

Algorithm 7: IntermoleGO($\mathbf{m}_1$, $\mathbf{m}_2$).

Table 3: Execution cost of DAG tasks by each processor.

Tasks	p_1	p_2	p_3
v_1	7	8	9
v_2	12	14	16
v_3	14	15	16
v_4	13	3	14

before, to promote the intensification capability of DRSCRO and avoid the function duplication.

The operator, IntermoleGO (i.e., the ineffective collision operator), is used to generate new molecules $\mathbf{m'}_1$ and $\mathbf{m'}_2$ from given molecules $\mathbf{m}_1$ and $\mathbf{m}_2$. This operator first uses the steps in OnWallGO to generate $\mathbf{m'}_1$ from $\mathbf{m}_1$, and then the operator generate the other new molecule $\mathbf{m'}_2$ from $\mathbf{m}_2$ in the similar fashion. In the end, the operator generates two new molecules $\mathbf{m'}_1$ and $\mathbf{m'}_2$ from $\mathbf{m}_1$ and $\mathbf{m}_2$ as an intensification search. The detailed executions are presented in Algorithm 7. Figure 7 shows the example of the IntermoleGO, in which the molecules correspond to the DAG as shown in Figure 1(a).

4.5. Illustrative Example. Consider the example shown in Figure 1(a). Its edges are labeled with the communication costs, whereas the execution costs are shown in Table 3.

Initially, the path (v_1, v_2, v_4, v_3) is found based on the upward rank value of each task in DAG, and the first molecule, $\mathbf{m} = ((v_1, 0, p_1), (v_2, 1, p_1), (v_4, 0, p_1)$, and $(v_3, 1, p_1))$, can be obtained. Algorithm 2, InitMoleSMS, is then executed to generate the initial population with 10 elements (i.e., *PopSize* is set as 10) for super molecule selection phase. The initial population molecules are operated during the iterations in the super molecule selection phase as presented in the framework of DRSCRO in Section 4.1, and the super molecule, SMole = $((v_1, 0, p_3), (v_2, 1, p_1), (v_4, 0, p_2)$, and $(v_3, 1, p_1))$, can be obtained.

In the global optimization phase, Algorithm 4, InitMoleGOVNS, is then executed to generate (or to update) the initial population after each iteration as presented in Section 4.1. The molecules are operated during the iterations in the global optimization phase as presented in the framework of DRSCRO in Section 4.1, and the global minimal *makespan* = 40 is finally obtained, for which the corresponding solution (i.e., molecule) is $((v_1, 0, p_2), (v_4, 1, p_2), (v_2, 0, p_2)$, and $(v_3, 1, p_2))$.

4.6. Analysis of DRSCRO. As a new metaheuristic strategy, the CRO-based methods for DAG scheduling which is proposed very recently have demonstrated the capability for solving this kind of NP-hard optimization problems. By analyzing the framework, molecular structure, chemical reaction operators, and the operational environment in DRSCRO, it can be shown to some extent that DRSCRO scheme has the advantage of three points in comparison with other CRO-based algorithms for DAG scheduling.

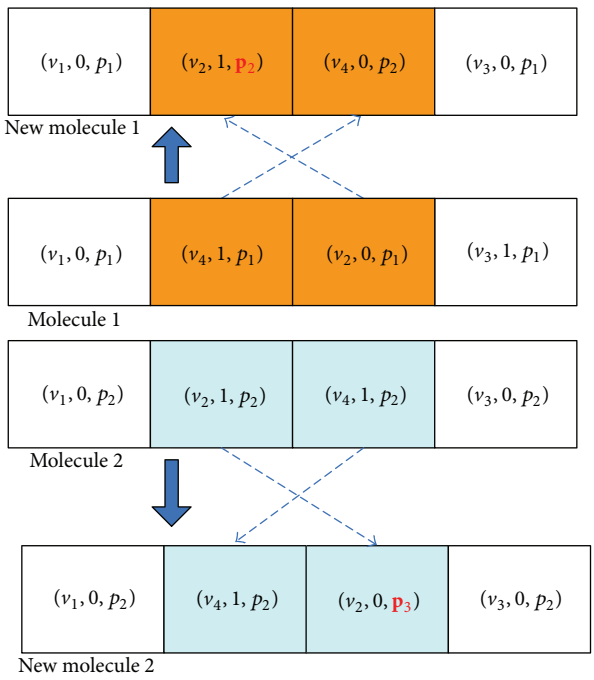

Figure 7: Example of IntermoleGO.

First, to some degree, super molecule in DRSCRO is similar to *InitS* in TMSCRO [30] or the "elite" in GA [31]. However, the "elite" in GA is usually generated from two chromosomes, while super molecule is approached by executing the first phase of DRSCRO. Moreover, in comparison with TMSCRO, DRSCRO uses a metaheuristic strategy (CRO) to get a better super molecule. It is because, as intelligent random-search algorithm, CRO used in the phase of DRSCRO for super molecule selection searches a wider area of the solution space than CEFT applied in TMSCRO, which narrow the search down to a very small portion of the solution space. As a result, a better super molecule may contribute to a better global optimum solution and accelerates convergence. Second, DAG scheduling problem has two complex aspects including task sequence optimization and processor assignment optimization, which lead to a very large and complicated solution space. So, for a better capability of intensification search than other CRO-based algorithms for DAG scheduling on heterogeneous systems, DRSCRO applied VNS algorithm as the initialization of the global optimization phase, which is also as a local search process during the running of DRSCRO, and one of the neighborhood structures of VNS is also utilized in the ineffective reaction operator. Moreover, during the running of DRSCRO, the task priority is changeable in our adopted VNS algorithm and a new model for processor selection is also utilized in the neighborhood structures for promoting efficiency of VNS, different from the VNS proposed in [22]. All of three advantages as previously mentioned enhance the ability to get better rapidity of convergence and better search result in the whole solution space, which is demonstrated by the experimental results in Sections 5.3 and 5.4. The time complexity of DRSCRO is $O(\text{iter} \times (2 \times |V| + 4 \times |E| \times |P| + d_{\max} \times subpopNum))$.

5. Experimental Details

In this section, the simulation experiment and comparative evaluation of HEFT, DMSCRO, TMSCRO, and proposed DRSCRO are presented. As presented in [27, 30], by theory analysis and experimental results, TMSCRO and DMSCRO proved to have better performance than GA; therefore, our work is the further study of CRO-based algorithms for DAG scheduling on heterogeneous systems, and, for DRSCRO as a metaheuristic algorithm, we focus on the performance of our proposed algorithm itself and the comparison between DRSCRO and other similar kinds of algorithms.

First, two extensive sets of graphs as the test beds for comparative study are described. Next, the parameter settings which are used in the simulation experiments are presented. The results and analysis of the experiment, including *makespan* test and convergence rate test, are given in the final part.

5.1. Test Bed. As presented in [27, 30], two extensive sets of DAGs, real-world application and randomly generated application graphs, are considered as the test beds in the experiments to enhance the comparability of various algorithms. The first extensive test bed is two real-world problem DAGs, molecular dynamics code [38] and Gaussian elimination [8]. Molecular dynamics are a computer simulation of physical movements of the molecules and atoms, which are allowed to interact for a period of time, in the context of N-body simulation. Molecular dynamics code DAG is shown in Figure 8. Gaussian elimination is used to calculate the solution for a linear equation system, which is applied systematically to convert row operations on a set of linear equations to the upper triangular form. As shown in Figure 9, the total number of tasks in the Gaussian elimination DAG with the matrix size of 7 is 27, and the largest task number at the same level is 6. The reason of the utilization of these two application graphs as a test bed is not only to enhance the comparability of various algorithms but also to show the function application of our proposed algorithm as an illustrative demonstration without loss of generality. The second extensive test bed for comparative study is the DAGs of random graphs. A random graph generator presented in [39] is implemented to generate random graphs in the simulation experiment. It allows the user to generate a variety of random graphs with different characteristics, such as CCR, the amount of calculation of a task, the successor number of a task, and the total number of tasks in a random graph. It is also assumed that all tasks and communication links have the same computation cost and communication cost, respectively.

As shown in Figure 2, the next phase criteria and stopping criteria of DRSCRO are that the *makespan* stays unchanged for 5000 consecutive iterations in the search loop. And the stopping criterion of TMSCRO and DMSCRO is that the *makespan* remains the same for 10000 consecutive iterations.

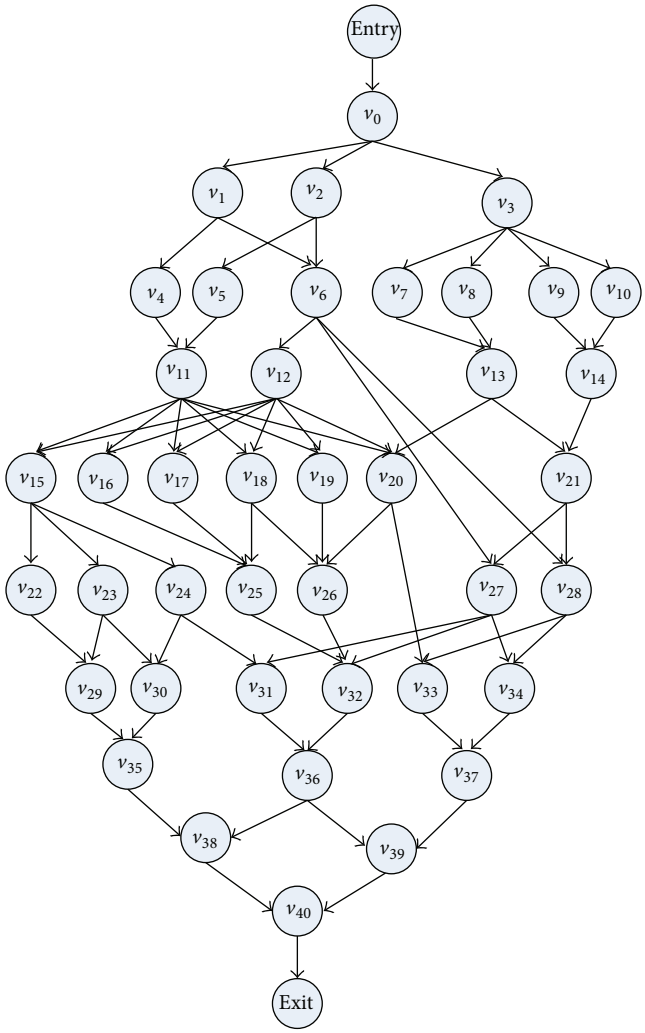

Figure 8: A molecular dynamics code DAG.

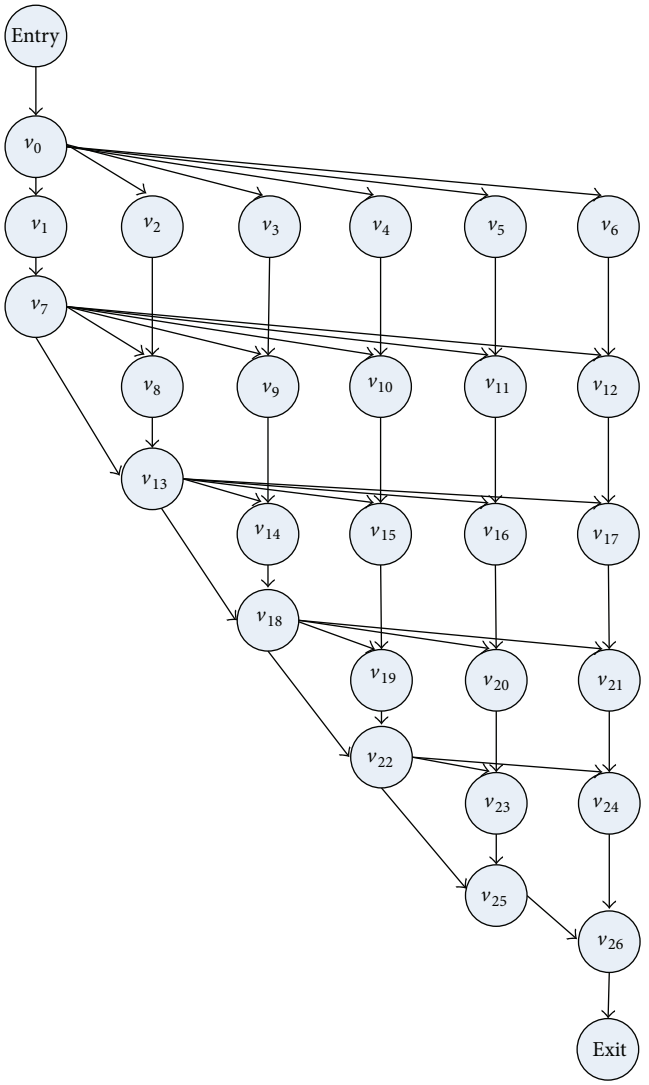

Figure 9: A Gaussian elimination DAG for a matrix of size 7.

5.2. Parameter Setting. In the experiments, a parameter hl is set to represent the heterogeneity level as presented in the first paragraph of Section 3. It complies with the MHM model assumption and results in the fact that speeds of a computing processor are different for different tasks. In doing so, the heterogeneity level $(1 + \text{hl}\%)/(1 - \text{hl}\%)$ is equal to the biggest possible ratio of the best processor speed to the worst processor speed for each task. hl is set as the value to make the heterogeneity level 2 unless otherwise specified in this paper.

The details of parameter setting are shown in Table 4. The parameters 6–12, which are the CRO-based algorithms tested in the simulation, are set as presented in [25].

5.3. Makespan Tests. The performance of the proposed algorithm is compared with two state-of-the-art CRO-based scheduling algorithms, DMSCRO and TMSCRO, and a heuristic algorithm HEFT. Each *makespan* value plotted in the graphs is the average value of a number of independent runs. In the first extensive test bed, the *makespan* is averaged over 10 independent runs (HEFT is run only once as a deterministic algorithm.), while in the second extensive test bed the *makespan* is averaged over 30 different random graph running instances. Moreover, to prove the robustness

Table 4: Parameter values for simulation experiment.

Parameter	Value
CCR	{0.1, 0.2, 1, 2, 5}
Number of processors	{4, 8, 16, 32}
hl	0.333
Successor number of a task in a random graph	{1, 2, 3, 4}
Total number of tasks in a random graph	{10, 20, 50}
InitialKE	1000
θ	500
ϑ	10
Buffer	200
KELossRate	0.2
MoleColl	0.2
PopSize	10

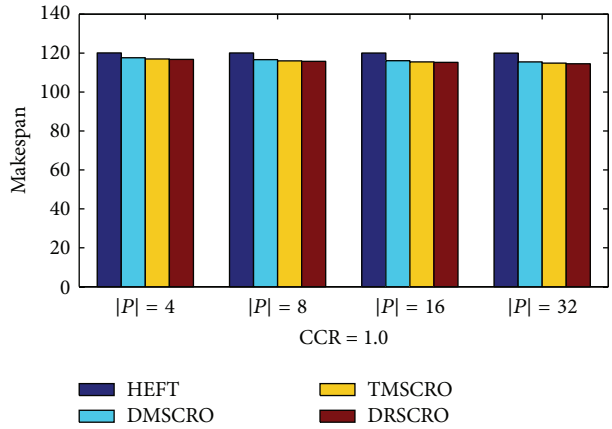

Figure 10: Average makespan for the molecular dynamics code, CCR = 1.0.

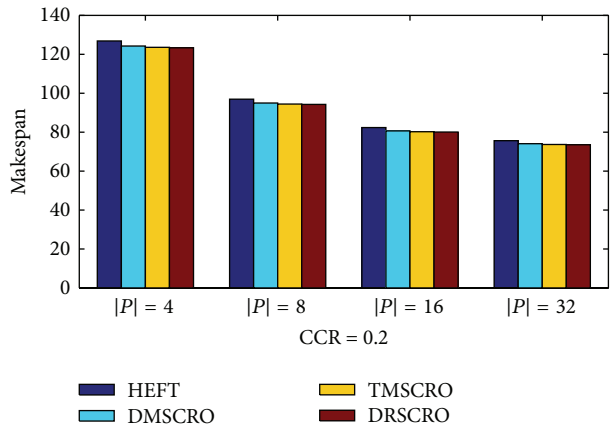

Figure 11: Average makespan for Gaussian elimination, CCR = 0.2.

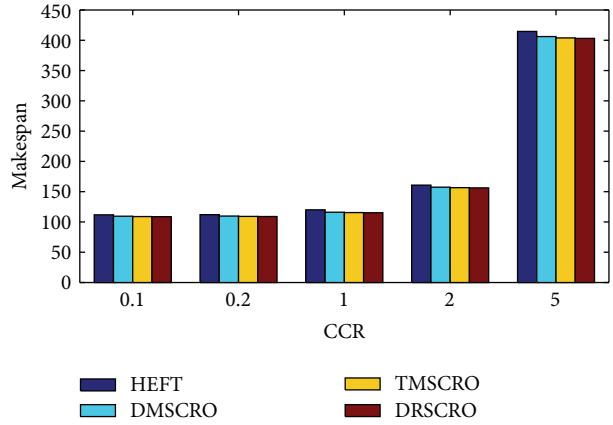

Figure 12: Average makespan for the molecular dynamics code; the processors number is 16.

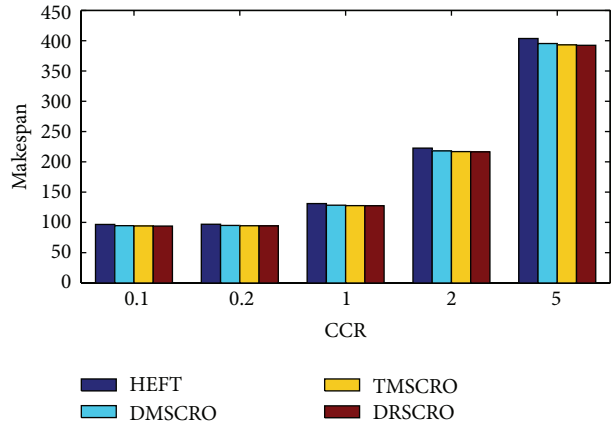

Figure 13: Average makespan for Gaussian elimination; the processors number is 8.

of DRSCRO, the best final value achieved in all these runs, the worst final value, and the related standard deviation or variance are also presented.

5.3.1. Real-World Application Graphs. Figures 10–13 show the simulation experiment results of DRSCRO, DMSCRO, TMSCRO, and HEFT on the real-world application graphs, and Tables 4–7 list the detail of the experimental results.

As shown in Figures 10 and 11, it can be observed that the average *makespan* decreases as the processor number increases. The results also show that DRSCRO, TMSCRO, and DMSCRO achieve very similar performance, which are all metaheuristic methods. It is because, according to No-Free-Lunch Theorem, all well-designed metaheuristic methods have the same performance on searching for optimal solutions when averaged over all possible fitness functions. The TMSCRO and DMSCRO used in the simulation are well-designed and taken from the literature. Therefore it proved that DRSCRO developed in our work is also well-designed.

A close observation of the results in Tables 10 and 11 shows that DRSCRO outperforms TMSCRO and DMSCRO on average slightly. The reason is only because DRSCRO has better capability of intensification search by applying VNS and the utilization of one of its neighborhood structures in the ineffective reaction operator, as presented in the last paragraph of Section 4.6. Therefore the performance of the average results obtained by DRSCRO is better than that obtained by TMSCRO and DMSCRO, when the stopping criterion is satisfied. Moreover, DRSCRO, TMSCRO, and DMSCRO typically outperform HEFT because they search a wider area of the solution space as metaheuristic methods, while the search of HEFT is narrowed down to a very smaller portion by means of the heuristics.

Figures 12 and 13 and the results in Tables 7 and 8 show the performance of the experimental results of these four algorithms with CCR value increasing. It can be seen that the *makespan* on average increases with the CCR value increasing. It is because the heterogeneous processors are in the idle state for longer, as a result of the DAGs becoming more communication-intensive. It also can be observed that DRSCRO, TMSCRO, and DMSCRO outperform HEFT and the advantage becomes more significant with the value of CCR increasing, which suggest that heuristic algorithm like HEFT has less consistent performance in a wide scheduling

Table 5: Experiment results for the molecular dynamics code, CCR = 1.0.

The number of processors	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
4	120.086	117.624	116.993	116.759	114.365	119.589	2.487
8	120.074	116.633	116.007	115.775	113.402	118.582	2.466
16	120.031	116.101	115.478	115.247	112.885	118.041	2.455
32	119.993	115.476	114.856	114.529	112.182	117.306	2.440

Table 6: Experiment results for Gaussian elimination graph, CCR = 0.2.

The number of processors	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
4	126.852	124.252	123.585	123.337	122.042	126.327	1.769
8	96.952	94.965	94.455	94.266	92.333	96.551	2.008
16	82.356	80.668	80.235	80.074	78.433	82.015	1.706
32	75.630	74.080	73.682	73.535	72.027	75.317	1.566

Table 7: Experiment results for the molecular dynamics code; the processors number is 16.

The value of CCR	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
0.1	111.782	109.491	108.903	108.685	106.457	111.320	2.315
0.2	112.034	109.737	109.148	108.930	106.697	111.571	2.320
1	120.031	116.101	115.478	115.247	112.885	118.041	2.455
2	160.760	157.465	156.619	156.306	153.102	158.532	3.034
5	414.581	406.082	403.902	403.095	400.878	404.805	2.125

Table 8: Experiment results for Gaussian elimination graph; the processors number is 8.

The value of CCR	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
0.1	96.612	94.632	94.124	93.935	92.010	96.212	2.001
0.2	96.952	94.965	94.455	94.266	92.333	96.551	2.008
1	131.094	128.407	127.717	127.462	124.849	130.552	2.715
2	222.635	218.071	216.900	216.467	214.194	219.550	2.456
5	403.600	395.327	393.204	392.418	390.260	394.083	2.069

scenario range, and metaheuristic algorithm performs more effectively for communication-intensive DAGs.

5.3.2. Randomly Generated Application Graphs. As shown in Figures 14–16, randomly generated DAGs are used to evaluate the performance of DRSCRO, TMSCRO, DMSCRO, and HEFT in these experiments. And the details of these experimental results are listed in Tables 9–11.

Figure 14 shows the performance on the experimental results of these four algorithms with the processor number increasing. As shown in Figure 14, DRSCRO always outperforms TMSCRO, DMSCRO, and HEFT as the number of processors increases. Figure 15 shows that DMSCRO has better performance than the other three algorithms as the task number increases. The reasons for these are similar to those explained in the third paragraph of Section 5.3.1. Figure 16 shows the *makespan* on average with CCR values increasing. It can be seem that the average *makespan* increases rapidly with the increasing of the value of CCR. As shown in Figure 16, the *makespan* on average increases rapidly when the value of CCR rises. It is the fact that the DAG becomes more communication-intensive with CCR increasing which leads to the processors staying in the idle state for longer.

5.4. Convergence Tests. In this section, the convergence experiments are conducted to show the change of *makespan*

Table 9: Experiment results for random graphs under different processor numbers; task number is 50 and CCR = 0.2.

The number of processors	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
4	149.400	146.337	145.552	145.261	142.283	148.782	3.094
8	119.511	117.061	116.433	116.200	113.818	119.017	2.475
16	119.473	117.024	116.396	116.163	113.782	118.979	2.474
32	119.468	115.550	114.929	114.700	112.348	117.480	2.443

Table 10: Experiment results for random graphs under different task numbers; processors number is 32 and CCR = 10.

The number of tasks	HEFT (average *makespan*)	DMSCRO (average *makespan*)	TMSCRO (average *makespan*)	DRSCRO (average *makespan*)	DRSCRO (best *makespan*)	DRSCRO (worst *makespan*)	DRSCRO (variance)
10	714.484	706.149	699.100	690.708	688.982	693.638	2.025
20	1100.918	1089.685	1080.428	1069.082	1066.410	1071.479	2.619
50	1666.373	1662.543	1649.988	1634.230	1633.415	1636.261	1.165

Table 11: Experiment results of DRSCRO for the random graph under different CCRs; the task number is 50.

Value of CCR	Processors number is 4	Processors number is 8	Processors number is 16	Processors number is 32
0.1	145.093	116.068	116.058	114.592
0.2	145.261	116.200	116.163	114.700
1	153.248	120.228	119.511	118.534
2	194.401	166.014	164.246	162.292
5	508.759	498.427	492.049	487.830

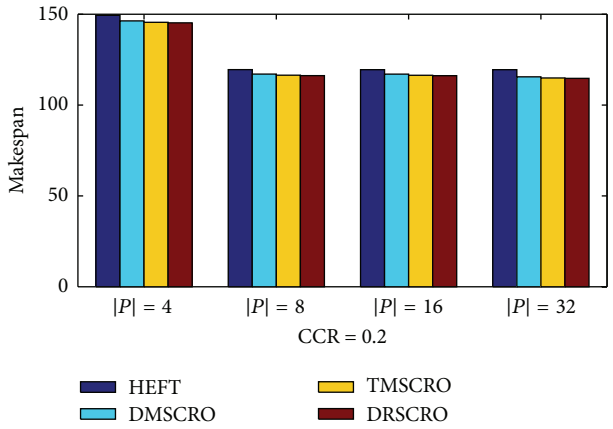

Figure 14: Average makespan for random graphs under different processor numbers; task number is 50 and CCR = 0.2.

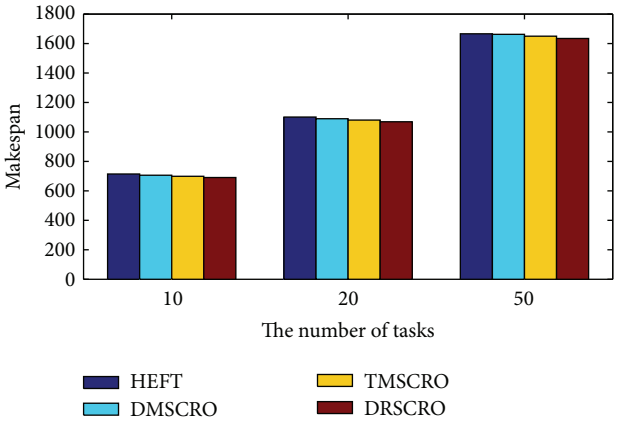

Figure 15: Average makespan for random graphs under different task numbers; processors number is 32 and CCR = 10.

among DRSCRO, TMSCRO, and DMSCRO. The convergence traces and significant tests are to further reveal the differences between DRSCRO and the other two algorithms. In these experiments, as suggested in [27], the stopping criteria of these three algorithms are that the total running time reaches a setting value (e.g., 180 s). Under the consideration of comparability, the beginning of the time counting of DRSCRO is set as the start of the global optimization phase processing. In the first extensive test bed, the *makespan* is averaged over 10 independent runs, while in the second extensive test bed the *makespan* is averaged over 30 different random graph running instances.

5.4.1. Convergence Trace. The convergence traces of DRSCRO, TMSCRO, and DMSCRO for processing the molecular dynamics code and Gaussian elimination are plotted in Figures 17 and 18, respectively. Figures 19–21 show the convergence traces when processing the randomly generated DAG sets, of which each contains 10, 20, and 50 tasks, respectively. As shown in Figures 17–21, it can be observed that the convergence traces of these three algorithms have obvious differences. And the DRSCRO converges faster than the other two algorithms in every case. The reason for the better rate of convergence of DRSCRO is as

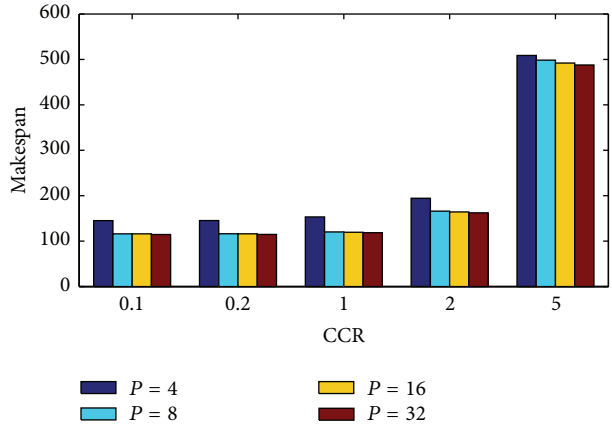

Figure 16: Average makespan of DRSCRO for the random graph under different CCRs; the task number is 50.

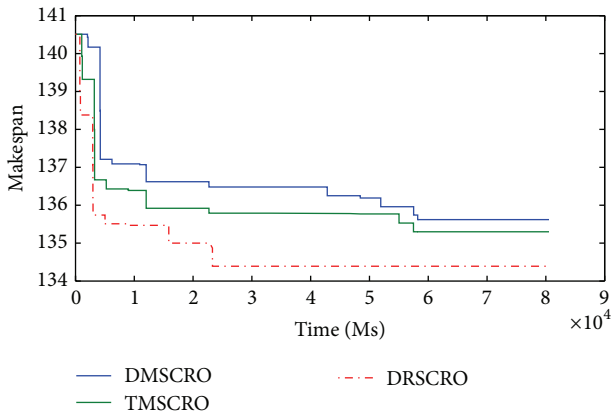

Figure 17: Convergence trace for the molecular dynamics code; CCR = 1 and the number of processors is 16.

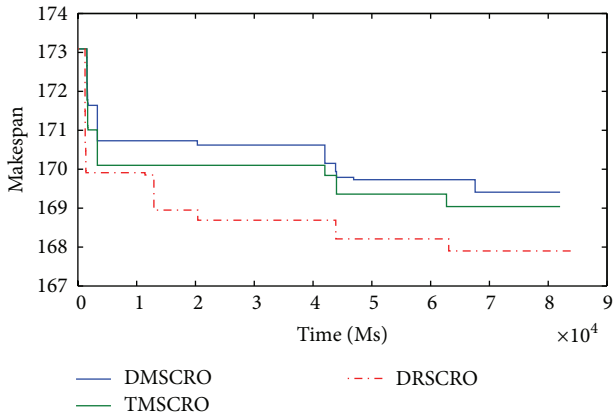

Figure 18: Convergence trace for Gaussian elimination; CCR = 0.2 and the number of processors is 8.

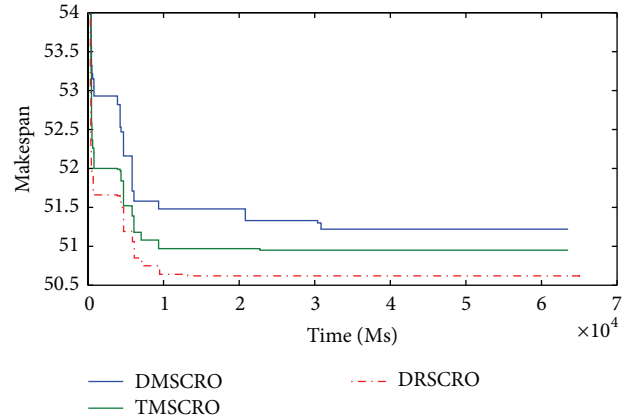

Figure 19: Convergence trace for the set of the randomly generated DAGs with 10 tasks.

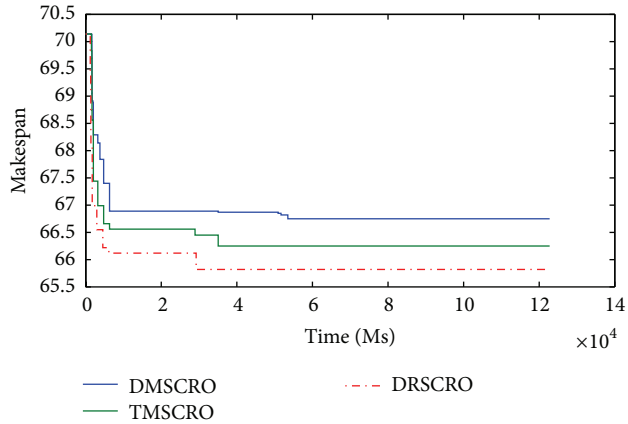

Figure 20: Convergence trace for the set of the randomly generated DAGs with 20 tasks.

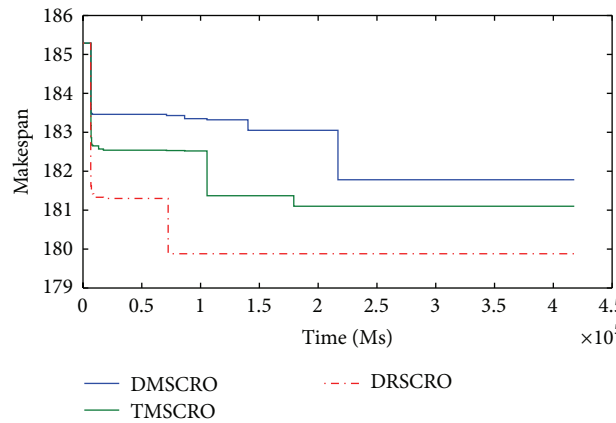

Figure 21: Convergence trace for the set of the randomly generated DAGs with 50 tasks.

presented in the last paragraph of Section 4.6 (i.e., DRSCRO takes the advantage of its double-reaction structure to obtain a better super molecule for accelerating convergence). Even though the VNS algorithm adds the time cost in each iteration, the enhanced optimization capability of DRSCRO also makes it obtain a better coverage rate than TMSCRO and DMSCRO. The simulation experimental results show that DRSCRO converges faster than TMSCRO by 19.4% on average (by 29.3% in the best case) and faster than DMSCRO by 33.9% on average (by 41.2% in the best case).

Moreover, the statistical analysis based on the average values achieved is also presented in Section 5.4.2, to prove

Table 12: Results of Friedman tests, $\alpha = 0.05$.

Method	Algorithm	p value	Hypothesis
Friedman test	DRSCRO, DMSCRO	$2.53E-02$	Rejected
	DRSCRO, TMSCRO	$2.53E-02$	Rejected
	DRSCRO, DMSCRO, TMSCRO	$6.70E-03$	Rejected

Table 13: Results of Quade tests, $\alpha = 0.05$.

Method	Algorithm	p value	Hypothesis
Quade test	DRSCRO, DMSCRO	$1.32E-02$	Rejected
	DRSCRO, TMSCRO	$1.32E-02$	Rejected
	DRSCRO, DMSCRO, TMSCRO	$1.09E-03$	Rejected

that DRSCRO outperforms the other CRO-based algorithms for DAG scheduling from a statistical point of view.

5.4.2. Significant Tests. Statistical analysis is necessary for the average coverage rates obtained in all cases by DRSCRO, TMSCRO, and DMSCRO, which are metaheuristic methods, in order to find significant differences among these results. Nonparametric tests according to the recommendations in [40] are specifically considered to be used, since the experimental results may present neither normal distribution nor variance homogeneity. Therefore, the Friedman test and the Quade test are applied to check whether significant differences exist in the performance between these three algorithms. A significance level $\alpha = 0.05$ is used in all statistical tests.

Tables 12 and 13, respectively, list the test results of the Friedman test and the Quade test, which both reject the null hypothesis of equivalent performance. In both of these two tests, our proposed DRSCRO is not only compared against all the algorithms but also compared against the remaining ones as the control method. The results in Tables 12 and 13 validate the significant differences α in the performance of DRSCRO, TMSCRO, and DMSCRO.

In sum, it could be concluded that DRSCRO, which is the control algorithm, statistically outperforms the other CRO-based DAG scheduling algorithm on coverage rate with a significant level of 0.05.

6. Discussion

The experimental results of *makespan* tests show that the performance of DRSCRO is very similar to the other similar kinds of metaheuristic algorithms because when averaged over all possible fitness functions, each well-designed metaheuristic algorithm has the same performance for searching optimal solutions, according to No-Free-Lunch Theorem. However, the proposed DRSCRO can achieve better performance and find good solutions faster than the other similar kinds of metaheuristic algorithms as the experimental results of convergence tests, and the reason for it, as the analysis in the last paragraph in Section 4.6, is that DRSCRO has a better super molecule creation by metaheuristic method, and under the consideration of the optimization of scheduling order and processor assignment, DRSCRO takes the advantages of VNS algorithm in the global optimization phase to improve the optimization capability. A load balance neighborhood structure is also applied in the ineffective reaction operator for a better intensification capability. The new processor selection model utilized in the neighborhood structures also promotes the efficiency of VNS algorithm.

7. Conclusion and Future Study

An algorithm named Double-Reaction-Structured CRO (DRSCRO) is developed for DAG scheduling on heterogeneous systems in this paper. DRSCRO includes two reaction phases, one for super molecule selection and another for global optimization. The phase of super molecule selection is used to obtain a super molecule by the metaheuristic method for better convergence rate, different from other CRO-based algorithms for DAG scheduling on heterogeneous systems. In addition, to promote the intersection capability of DRSCRO, the VNS algorithm, which is with a new model for processor selection utilized in the neighborhood structures, is used as the initialization of global optimization phase, and the load balance neighborhood structure of VNS is also applied in the ineffective reaction operator. The experimental results show that DRSCRO can also achieve a higher speedup than the other CRO-based algorithms as far as we know. And DRSCRO algorithm can also obtain better performance on average *makespan* in some cases.

In future work, we will analyze the parameter sensitivity of DRSCRO for promoting its activeness. Moreover, to make the proposed algorithm more practical, DRSCRO will be also extended to aim at two main objectives, such as (1) minimization of schedule length (time domain) and (2) minimization of number of used processors (resource domain).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is financially supported by the National Natural Science Foundation of China (Grant no. 61462073) and

the National High Technology Research and Development Program of China (863 Program) (Grant no. 2015AA020107).

References

[1] T. Lewis and H. Elrewini, *Introduction to Parallel Computing*, Prentice-Hall, New York, NY, USA, 1992.

[2] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[3] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[4] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[5] F. Suter, F. Desprez, and H. Casanova, "From heterogeneous task scheduling to heterogeneous mixed parallel scheduling," in *Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference, Pisa, Italy, August 31- September 3, 2004. Proceedings*, vol. 3149 of *Lecture Notes in Computer Science*, pp. 230–237, Springer, Berlin, Germany, 2004.

[6] C.-Y. Lee, J. J. Hwang, Y.-C. Chow, and F. D. Anger, "Multiprocessor scheduling with interprocessor communication delays," *Operations Research Letters*, vol. 7, no. 3, pp. 141–147, 1988.

[7] M. Maheswaran and H. J. Siegel, "A dynamic matching and scheduling algorithm for heterogeneous computing systems," in *Proceedings of the 7th Heterogeneous Computing Workshop*, pp. 57–69, IEEE Computer Society, Orlando, Fla, USA, March 1998.

[8] M.-Y. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330–343, 1990.

[9] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

[10] G. C. Sih and E. A. Lee, "Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175–187, 1993.

[11] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," *IEEE Software*, vol. 5, no. 1, pp. 23–32, 1988.

[12] M. A. Al-Mouhamed, "Lower bound on the number of processors and time for scheduling precedence graphs with communication costs," *IEEE Transactions on Software Engineering*, vol. 16, no. 12, pp. 1390–1401, 1990.

[13] A. Tumeo, C. Pilato, F. Ferrandi, D. Sciuto, and P. L. Lanzi, "Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '08)*, pp. 142–149, IEEE Computer Society, Samos, Greece, June 2008.

[14] I. Ahmad and M. K. Dhodhi, "Multiprocessor scheduling in a genetic paradigm," *Parallel Computing*, vol. 22, no. 3, pp. 395–406, 1996.

[15] M. R. Bonyadi and M. E. Moghaddam, "A bipartite genetic algorithm for multi-processor task scheduling," *International Journal of Parallel Programming*, vol. 37, no. 5, pp. 462–487, 2009.

[16] R. C. Corrêa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 825–837, 1999.

[17] E. S. H. Hou, N. Ansari, and H. Ren, "Genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, 1994.

[18] F. A. Omara and M. M. Arafa, "Genetic algorithms for task scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 70, no. 1, pp. 13–22, 2010.

[19] A. Singh, M. Sevaux, and A. Rossi, "A hybrid grouping genetic algorithm for multiprocessor scheduling," in *Contemporary Computing: Second International Conference, IC3 2009, Noida, India, August 17–19, 2009. Proceedings*, vol. 40 of *Communications in Computer and Information Science*, pp. 1–7, Springer, Berlin, Germany, 2009.

[20] A. S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824–834, 2004.

[21] Y. Wen, H. Xu, and J. Yang, "A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system," *Information Sciences*, vol. 181, no. 3, pp. 567–581, 2011.

[22] R. Shanmugapriya, S. Padmavathi, and S. Shalinie, "Contention awareness in task scheduling using Tabu search," in *Proceedings of the IEEE International Advance Computing Conference*, pp. 272–277, IEEE Computer Society, Patiala, India, March 2009.

[23] S. C. Porto and C. C. Ribeiro, "A tabu search approach to task scheduling on heterogeneous processors under precedence constraints," *International Journal of High Speed Computing*, vol. 7, no. 1, pp. 45–72, 1995.

[24] A. V. Kalashnikov and V. A. Kostenko, "A parallel algorithm of simulated annealing for multiprocessor scheduling," *Journal of Computer and Systems Sciences International*, vol. 47, no. 3, pp. 455–463, 2008.

[25] A. Y. S. Lam and V. O. K. Li, "Chemical-reaction-inspired metaheuristic for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 381–399, 2010.

[26] P. Choudhury, R. Kumar, and P. P. Chakrabarti, "Hybrid scheduling of dynamic task graphs with selective duplication for multiprocessors under memory and time constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 7, pp. 967–980, 2008.

[27] Y. Xu, K. Li, L. He, and T. K. Truong, "A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[28] J. Xu and A. Lam, "Chemical reaction optimization for the grid scheduling problem," in *Proceedings of the IEEE International Conference on Communications*, pp. 1–5, IEEE Computer Society, Cape Town, South Africa, May 2010.

[29] B. Varghese, G. McKee, and V. Alexandrov, "Can agent intelligence be used to achieve fault tolerant parallel computing systems?" *Parallel Processing Letters*, vol. 21, no. 4, pp. 379–396, 2011.

[30] Y. Jiang, Z. Shao, and Y. Guo, "A DAG scheduling scheme on heterogeneous computing systems using tuple-based chemical reaction optimization," *Scientific World Journal*, vol. 2014, Article ID 404375, 23 pages, 2014.

[31] J. Xu, A.Y. S. Lam, and V. O. K. Li, "Stock portfolio selection using chemical reaction optimization," in *Proceedings of the International Conference on Operations Research and Financial Engineering*, pp. 458–463, WASET, Paris, France, June 2011.

[32] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[33] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2867–2876, 2014.

[34] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[35] M. A. Khan, "Scheduling for heterogeneous Systems using constrained critical paths," *Parallel Computing*, vol. 38, no. 4-5, pp. 175–193, 2012.

[36] P. Hansen, N. Mladenovic, and J. M. Perez, "Variable neighborhood search: methods and applications," *4OR-A Quarterly Journal of Operations Research*, vol. 6, no. 4, pp. 319–360, 2008.

[37] F. Glover and G. Kochenberger, *Handbook of Metaheuristics*, Springer, New York, NY, USA, 2003.

[38] S. Kim and J. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in *Proceedings of the International Conference on Parallel Processing*, pp. 1–8, Pennsylvania State University, Pennsylvania State University Press, University Park, Pa, USA, August 1988.

[39] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Arabe, and D. A. Menasce, "Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 683–691, IEEE Computer Society, Minneapolis, Minn, USA, November 1992.

[40] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.