A Hybrid Chemical Reaction Optimization Scheme for Task Scheduling on Heterogeneous Computing Systems

Yuming Xu, Kenli Li, *Member, IEEE*, Ligang He, *Member, IEEE*, Longxin Zhang, and Keqin Li, *Fellow, IEEE*

Abstract—Scheduling for *directed acyclic graph* (DAG) tasks with the objective of minimizing makespan has become an important problem in a variety of applications on heterogeneous computing platforms, which involves making decisions about the execution order of tasks and task-to-processor mapping. Recently, the *chemical reaction optimization* (CRO) method has proved to be very effective in many fields. In this paper, an improved hybrid version of the CRO method called HCRO (hybrid CRO) is developed for solving the DAG-based task scheduling problem. In HCRO, the CRO method is integrated with the novel heuristic approaches, and a new selection strategy is proposed. More specifically, the following contributions are made in this paper. (1) A Gaussian random walk approach is proposed to search for optimal local candidate solutions. (2) A left or right rotating shift method based on the theory of maximum Hamming distance is used to guarantee that our HCRO algorithm can escape from local optima. (3) A novel selection strategy based on the normal distribution and a pseudo-random shuffle approach are developed to keep the molecular diversity. Moreover, an exclusive-OR (XOR) operator between two strings is introduced to reduce the chance of cloning before new molecules are generated. Both simulation and real-life experiments have been conducted in this paper to verify the effectiveness of HCRO. The results show that the HCRO algorithm schedules the DAG tasks much better than the existing algorithms in terms of makespan and speed of convergence.

Index Terms—Chemical reaction optimization, hamming distance, hybrid scheduling, normal distribution, pseudo random shuffle

1 Introduction

An application consisting of a group of tasks can be represented by a node- and edge-weighted *directed acyclic graph* (DAG), in which the vertices represent the computations and the directed edges represent the data dependencies as well as the communication times between the vertices. DAGs have been shown to be expressive for a large number of and a variety of applications. Task scheduling is one of the most thought-provoking NP-hard problems in general cases, and polynomial time algorithms are known only for a few restricted cases [1]. Hence, it is a challenge on heterogeneous computing systems to develop task scheduling algorithms that assign the tasks of an application to processors in order to minimize makespan without violating precedence constraints. Therefore, many researchers have proposed a variety of approaches to solving the DAG task scheduling problem. These methods are basically classified into two major categories: dynamic scheduling and static scheduling. In dynamic scheduling, the information, such as a task's relation, execution time, and communication time, are all not previously known. The scheduler has to make decisions in real time. In static scheduling, all information about tasks is known before hand. Static scheduling algorithms by using different techniques to find a near optimal solution are universally classified into two major groups: heuristic scheduling and meta-heuristic scheduling.

Heuristic scheduling algorithms are based on the specific characteristics of an application, and the quality of the solutions obtained by these algorithms is heavily dependent upon the effectiveness of the heuristics. These algorithms give near-optimal solutions but with low time complexity (polynomial time) and acceptable performance, since the attempted solutions are narrowed down by greedy heuristics to a very small portion of the entire solution space. Therefore, it is not likely to produce consistent results on a wide range of problems, especially when the DAG scheduling problem becomes complex.

Meta-heuristic scheduling techniques use guided search strategies to explore the search space more effectively. They often focus on some promising regions of the search space, which generate optimal solutions with exponential time complexity. Meta-heuristic methods begin with a set of initial solutions or an initial population, and then, examine step by step a sequence of solutions to reach a desirable solution. These methods have demonstrated their potential and effectiveness in solving a wide variety of optimization problems than many traditional algorithms. Well-known

- *Y. Xu, K. Li, and L. Zhang are with the College of Computer Science and Electronic Engineering, Hunan University, and with the National Supercomputing Center in Changsha, Hunan, Changsha 410082, China. E-mail: lkl@hnu.edu.cn.*
- *L. He is with the Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom, and also with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz NY 12561, the College of Computer Science and Electronic Engineering, Hunan University, and the National Supercomputing Center in Changsha, Hunan Changsha 410082, China.*

Manuscript received 27 June 2014; revised 26 Oct. 2014; accepted 15 Dec. 2014. Date of publication 22 Dec. 2014; date of current version 6 Nov. 2015. Recommended for acceptance by G. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2385698

examples of meta-heuristic scheduling techniques include *ant colony optimization* (ACO) [2], *genetic algorithm* (GA) [3], [4], *simulated annealing* (SA) [5], etc.

Very recently, a *chemical reaction optimization* (CRO) method has been proposed. The CRO algorithms are a kind of the population based meta-heuristic algorithms developed by Lam and Li in 2010 [6]. The method has been applied to solve the task scheduling problems, which encodes the solutions as molecules, and mimics the interactions of molecules in chemical reactions to search the optimal solutions. To date, the CRO method has only been used to encode the scheduling of independent tasks on heterogeneous computing platforms. Scheduling independent tasks involves mapping tasks to heterogeneous computing processors. Scheduling DAG tasks on heterogeneous computing platforms is more complicated, which involves making decisions about both execution order of tasks (i.e., task priority) and task-to-processor mapping. It is a challenge of applying CRO to handle DAG scheduling problems [7]. In order to solve the dependent task scheduling problem, we have been investigating the problem of applying CRO to address the DAG scheduling. Our previous work presented in [8] applied the conventional CRO framework to address the DAG scheduling. The DAG scheduling algorithm, in the works of both [8] and this paper, is divided into two phases: 1) determining the execution order of the tasks in a DAG, and 2) mapping the tasks to computing processors. In [8], we developed a *double molecular structure-based CRO* (DMSCRO) method, in which a set of CRO operations are designed and applied to both above phases of the DAG scheduling.

Although the developed DMSCRO is able to consistently produce good scheduling solutions, the time overhead, i.e., time spent in finding a good solution, is high. This paper aims to develop a CRO-based DAG scheduling algorithm with similar performance but reduced overhead. In this paper, we develop a hybrid CRO scheduling algorithm, called HCRO, by integrating the CRO algorithm with some heuristic approaches.

Hybrid algorithms have received significant interest in recent years and are being increasingly used to solve real-world problems [9], [10], [11], [12], [13], [14], [15], [16]. A hybrid algorithm is an algorithm that combines two or more other algorithms that solve the same problem, either choosing one (depending on the data), or switching between them over the course of the algorithm. This is generally done to combine desired features of each, so that the overall algorithm is better than the individual components [17].

In the HCRO algorithm presented in this paper, the CRO operations are applied to the first phase of the DAG task scheduling, i.e., determining the execution order of the tasks, while heuristic algorithms are designed and applied to the second phase, i.e., mapping the tasks to processors. A careful balance is struck between makespan and speed of convergence in the hybrid CRO. As a result, the hybrid CRO can achieve similar scheduling performance in terms of makespan while reducing the scheduling overhead, compared with the existing guided meta-heuristic algorithms. Moreover, it can also achieve better performance than heuristic algorithms.

We conducted both simulation experiments over a large set of randomly generated graphs and real experiments by using two well-known real applications: Gaussian elimination and molecular dynamics application. We compared the proposed HCRO with two well-known heuristic algorithms and a pure meta-heuristic method. The theoretical and experimental results show that our proposed HCRO can achieve better performance than the heuristic algorithms, and can achieve a very close performance to that of the pure meta-heuristic algorithm with much faster convergence speed (therefore with much lower overhead).

Further, we investigated the CRO parameters which affect the efficiency of the DMSCRO algorithm presented in our previous work [8], such as the size of the initial population and the particular elementary reactions. Hence, in this paper, a new and novel way is designed to encode the scheduling solutions of the initial molecular population. Different reactions are developed to be performed on the encoded solutions and generate increasingly better solutions.

In summary, the major contributions of this paper are listed below.

- We adopt some novel approaches to generating an initial molecular population and to accelerating the convergence of searching for good scheduling solutions. Because a high-quality solution, obtained from a heuristic technique, can help the HCRO to find better solutions faster than it can from a random start, and with a good uniform coverage. The molecules are well spread out to cover a whole feasible solution space. Moreover, the molecular diversity of the population can help the HCRO to reach a feasible part of the solution space as large as possible.
- We present a Gaussian random walk approach to searching for local optimal candidate solutions in the operator of on-wall ineffective collision. On the other hand, for the purpose of obtaining a global optimum or a near-optimal solution, we employ a left or right rotating shift strategy according to the theory of maximum Hamming distance, aiming to help the operator of decomposition to escape from local optima.
- We propose a novel selection approach based on the normal distribution, and use an exclusive-OR (XOR) operator between two strings to reduce a chance of cloning before new molecules are generated. Then a pseudo-random shuffle approach is employed to generate new molecules to help the operator of inter-molecular ineffective collision to keep the molecular diversity, and on the contrary, to realize the operator of synthesis to eliminate close relatives of the molecules.
- We demonstrate through the experiments over a large set of randomly generated graphs as well as the graphs for real-world problems with various characteristics that our proposed HCRO algorithm outperforms several related heuristic-based list scheduling algorithms and meta-heuristic algorithms in terms of the schedule quality.

TABLE 1
The THM Heterogeneous Computing System

Notation	Definition
The amounts of computing power available at each node	0.1-2.0
The maximum number of processors	32
The level (degree) of heterogeneity of the systems	$\frac{1+h\%}{1-h\%}$

The remainder of this paper is organized as follows. Section 2 describes the system and workload model. In Section 3, a heuristic chemical reaction optimization algorithm is presented for DAG scheduling, aiming to minimize the *makespan* on heterogeneous computing systems. Section 4 compares the performance of the proposed algorithm with three existing algorithms. Finally, Section 5 concludes the paper.

Due to space limitation, a review of related work is presented in Section 1 of the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2385698 available online.

2 Models and Workload

This section first presents the system model and workload considered in this paper, and then presents an example and motivations of our devised algorithm. In Section 2.1, a heterogeneous computing system consisting of a set P of m heterogeneous processors is presented, which are fully interconnected with a high-speed network. In Section 2.2, we introduce an application which is represented by a DAG graph, with the graph vertexes representing tasks and edges between vertexes representing execution precedence between tasks. Then a workload formula is presented. In Section 2.3, we explain a motivation to develop a hybrid approach by integrating CRO with heuristic approaches and make a trade-off between the quality of makespan and the speed of convergence. A list of notations and their definitions used in the paper is summarized in Appendix A of the supplemental file, available online.

2.1 System Model

The heterogeneity model of a computing system can be divided into two categories, i.e., *processor-based heterogeneity model* (PHM) and *task-based heterogeneity model* (THM). In a PHM model, a processor executes the tasks at the same speed, regardless of the type of the tasks. In a THM model, how fast a processor executes a task depends on how well the heterogeneous processor architecture matches the task requirements and features.

In this paper, we assume a THM model, where the main characteristics are given in Table 1. The heterogeneous computing system consists of a set P of m heterogeneous processors, P_1, P_2, ..., P_m, which are fully interconnected with a high-speed network. Each task in a DAG application can only be executed on a processor. The communication time between two dependent tasks should be taken into account if they are assigned to different processors.

Note that when the level of heterogeneity is 1 ($h = 0$), the computing system is homogeneous. Furthermore, as we change the value of h, i.e., the level of heterogeneity, the average computing speed remains unchanged.

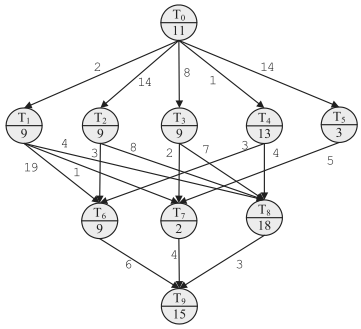


Fig. 1. A simple DAG application model containing 10 tasks.

We also assume that a static computing model in which the dependency relations and the execution orders of tasks are known *a priori* and do not change over the process of the scheduling. In addition, all processors are fully available to the computation on the time slots they are assigned to.

2.2 Application Model

In this work, an application is represented by a DAG graph, with the graph vertexes representing tasks and edges between vertexes representing execution precedence between tasks [6]. $Pred(T_i)$ and $Succ(T_i)$ denote the set of predecessor tasks and successor tasks of task T_i, respectively. The entry task T_{entry} is the starting task of the application without any predecessors, while the exit task T_{exit} is the final task with no successors. The vertex weight, denoted as $W_d(T_i)$, represents the amount of data to be processed in the task T_i, while the edge weight, denoted as $C_d(T_i, T_j)$, represents the amount of communication between task T_i and task T_j. The DAG topology of an exemplar application model is shown in Fig. 1.

In this paper, the execution speeds of the processors in the heterogeneous computing system are represented by a two-dimensional matrix, S, in which an element $S(T_i, P_k)$ represents the speed at which processor P_k to execute task T_i. The computation cost of task T_i running on processor P_k, denoted as $W(T_i, P_k)$, can be calculated by Eq. (1):

$$W(T_i, P_k) = \frac{W_d(T_i)}{S(T_i, P_k)}. \tag{1}$$

The average computation cost of task T_i, denoted as $\overline{W(T_i)}$, can be calculated by Eq. (2):

$$\overline{W(T_i)} = \frac{1}{m}\sum_{k=1}^{m} W(T_i, P_k). \tag{2}$$

The communication bandwidths between heterogeneous processors are represented by a two-dimensional matrix B. The communication startup costs of the processors are represented by an array, C_s, in which the element $C_s(P_k)$ is the startup cost of processor P_k. The communication cost $C(T_i, T_j)$ of $edge(T_i, T_j)$, which is the time spent in transferring data from task T_i (scheduled on P_k) to task T_j (scheduled on P_l), can be calculated by $C(T_i, T_j) = C_s(P_k) +$

TABLE 2
Processor Heterogeneity and Computation Costs

		Speed			$W(T_i, P_j)$ Cost			Ave. Cost
T_i	$W(T_i)$	P_0	P_1	P_2	P_0	P_1	P_2	$\overline{W(T_i)}$.
0	11	1.10	1.00	1.00	10.0	11.0	11.0	10.67
1	9	1.00	0.90	1.13	9.0	10.0	8.0	9.00
2	9	1.13	1.50	1.13	8.0	6.0	8.0	7.33
3	9	0.90	0.90	1.00	10.0	10.0	9.0	9.67
4	13	1.00	1.08	1.00	13.0	12.0	13.0	12.67
5	3	1.00	1.50	0.75	3.0	2.0	4.00	3.0
6	9	0.90	1.13	1.00	10.0	8.0	9.00	9.0
7	2	1.00	1.00	1.00	2.0	2.0	2.00	2.0
8	18	1.00	1.06	1.13	18.0	17.0	16.0	17.00
9	15	1.00	1.07	1.07	15.0	14.0	14.0	14.43

$\frac{C_d(T_i,T_j)}{B(P_k,P_l)}$ and $\overline{C(T_i,T_j)} = \overline{C_s} + \frac{C_d(T_i,T_j)}{\overline{B}}$, where $\overline{C_s} = \frac{1}{m}\sum_{k=1}^{m} C_s(P_k)$ is the average communication startup cost over all processors, and $\overline{B} = \frac{1}{m^2}\sum_{k=1}^{m}\sum_{l=1}^{m} B(P_k, P_l)$ is the average communication cost per transferred unit over all processors [6]. When T_i and T_j are scheduled on the same processor, the communication cost is regarded as 0. That is to say, a communication cost is only required when two tasks are assigned to different heterogeneous processors. It is assumed that the inter-computing-node communications are performed at the same speed (i.e., with the same bandwidths) on all links. For simplicity, we assume $B(P_k, P_l) = 1$ and $C_s(P_k) = 0$ in our DAG task scheduling model.

Assume that the DAG has the topology as shown in Fig. 1. An example of the processor heterogeneity and the computation costs of the tasks are shown in Table 2. Note that there are two numbers in each vertex in Fig. 1. The number at the top is the task id and the one at the bottom is the average computation cost as calculated in Table 2.

2.3 Example and Motivation

In Figs. 2 and 3, we can observe that the DMSCRO can get a better makespan than the HCRO for the simple DAG application in Fig. 1. In Fig. 4, we can also observe that the convergence of HCRO is faster than DMSCRO while maintaining a good makespan of the best individual and the average makespan of the population. Therefore, in this paper, we try to find a trade-off between the quality of makespan and the speed of convergence. We develop a hybrid approach by integrating CRO with heuristic

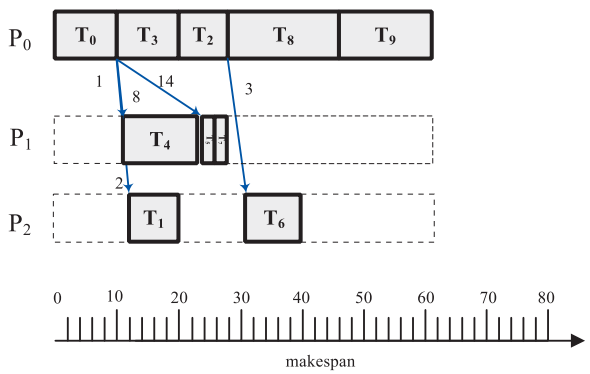


Fig. 2. A case study of the HCRO algorithm (the number of processors = 3, makespan = 61).

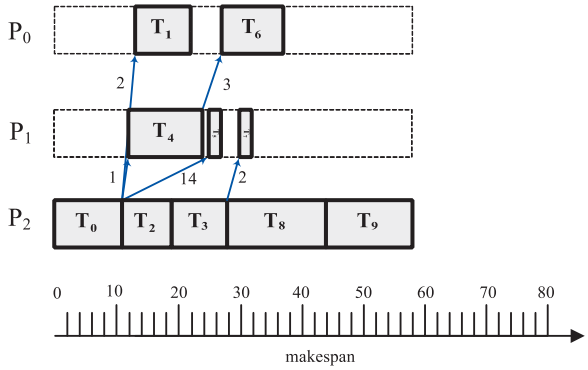


Fig. 3. A case study of the DMSCRO algorithm (the number of processors = 3, makespan = 58).

approaches. We find that being compared with the overhead of pure meta-heuristic algorithms, such as the DMSCRO method, this hybrid approach can achieve similar performance in terms of makespan for DAG scheduling while reducing the scheduling overhead. We also find that the hybrid approach can achieve better performance than heuristic algorithms. All these benefits will be shown in the experiment section.

3 Hybrid Chemical Reaction Optimization

In this paper, a method of integrating CRO and the heuristic approach, called HCRO, is proposed to schedule DAG tasks on heterogeneous computing systems. The idea of this method is to exploit the advantages of both CRO and heuristic algorithms, while avoiding their drawbacks. In HCRO, the CRO technique is used to search the execution order of the tasks, while a heuristic algorithm is used to determine a suitable task-to-processor mapping. In this section, an important factor for the priority queues of task scheduling is first introduced in Section 3.1. Namely, not every permutation of n tasks forms a legal schedule due to the precedence relations in the DAG application graph. Second, the encoding mechanism of our task scheduling algorithm is presented in Section 3.2 to represent the search nodes as molecules. It is desirable that any molecule can determine a unique schedule. Third, four molecular operators are proposed

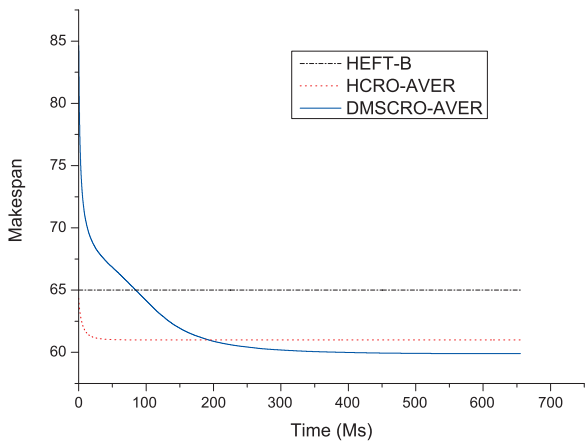


Fig. 4. The convergence trace of the average makespan of the population for the simple DAG task graph (the number of processors = 3, 100 independent runs).

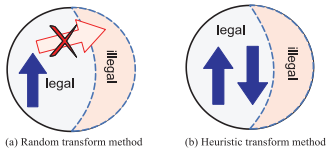


Fig. 5. Illustration of two different transformational schemes for the DAG applications with precedence constraints: (a) Both legal and illegal molecules (b) Only legal molecules.

for our task scheduling algorithm in detail in Section 3.3. Fourth, the heuristic method is proposed in Section 3.4, which tries to map each task in the solution (in the order of its position in the queue) to the computing processor that can provide the *earliest finish time*. Finally, the time and space complexity of HCRO is analyzed in Section 3.5. Section 2 of the supplemental file, available online, introduces the background information of CRO.

3.1 Fundamental Characteristics of Priority Queues

Our aim is to use CRO to find a good execution order of the tasks in a DAG application. Therefore, the execution order is the solution that should be optimized by CRO. A solution ω of the execution order is encoded as an integer queue and an integer represents a task id, i.e., $\omega = \{T_1, T_2, \ldots, T_i, \ldots, T_n\}$. Here, ω corresponds to a molecule, a task corresponds to an atom in the molecule, and the execution order of the tasks corresponds to the molecular structure.

Theorem 1. *The size of the solution space (i.e., the number of possible solutions) of an application containing n tasks without dependency is $n! = n \times (n-1) \times \cdots \times 1$.*

Proof. In the task scheduling, the solution of task scheduling is the execution order (without consideration of the task-to-processor mapping). The number of possible execution order of n tasks is $n!$. Therefore, the size of the solution space is $n!$. □

When an application contains n tasks with dependency, the size of the solution space will be much smaller than $n!$. This is because a solution of task scheduling is an execution order which is a linear order (topological order) of tasks based on their dependencies. In a DAG application graph, the tasks are represented by vertices, which may represent tasks to be performed, and the edges may represent constraints so that one task must be performed before another. Then, a topological sort is just a valid sequence of the tasks and gives an order for performing the tasks. For a DAG application containing n tasks, it is apparent that the number of all topological sorts is much less than $n!$. Therefore, the size of the solution space is much smaller than $n!$. Moreover, an important factor is that every priority queue in a search space is represented uniquely. That is, each solution corresponds to a legal priority queue of a DAG task scheduling. An intermediary encoded representation of the schedule and a decoder are used that always yield the legal solutions to the problem.

Note that not every permutation of n tasks corresponds to a legal schedule because of the precedence relations. This representation of DAG task scheduling falls into the category that molecule (the priority queue) is not in one-to-one correspondence with the search space. The representation of the DAG task scheduling must accommodate the precedence relations between the computational tasks, as shown in Fig. 5.

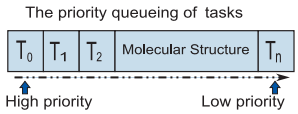


Fig. 6. The demonstration of molecular structure for DAG tasks.

3.2 Molecular Representation and Initial Population

In this section, first, the encoding mechanism of our task scheduling algorithm is proposed for representing search nodes as molecules. It is desirable that any molecule can determine a schedule uniquely. Second, the quality of initial molecular population is introduced to enhance the possibility of obtaining a globally optimal solution. Finally, the initial algorithm for molecular population is proposed.

A solution ω of the execution order is encoded as an integer queue, in which an integer represents a task id, i.e., $\omega = \{T_1, T_2, \ldots, T_i, \ldots, T_n\}$, as shown in Fig. 6.

For the DAG task scheduling problem in this paper, the molecule representation needs to consider the precedence relations between the tasks of DAG applications, and a legal priority queue (a schedule) is one that satisfies the following conditions.

- The precedence relations among the tasks are satisfied.
- Every task appears only once in the priority queue (completeness and uniqueness).

The initial molecular population consists of *PopSize* individuals (molecules), the size of the initial molecular population has been investigated from several theoretical points of view, although the underlying idea is always a trade-off between efficiency and effectiveness. Intuitively, it would seem that there should be some individuals with "good" fitness values for a given molecular structure length, on the grounds that a too small molecular population would not allow sufficient room for exploring the search space effectively, while a too large molecular population would so impair the efficiency of the method that no solution could be expected in a reasonable amount of time. In our study, we adopt a novel approach to generating a good "seeding", good uniform coverage, and molecular diversity for the initial population.

In this paper, a method to generate good seeding, with which there is a higher possibility to generate good solutions, is proposed. A high-quality solution obtained from another heuristic technique can help the CRO to find better solutions more quickly than it can from a random start. In order to achieve a good "seeding" for the task scheduling problem, we take advantage of the heuristic rank policies [18], which are mostly used by traditional list scheduling approaches for estimating the priority of each subtask. The individuals have a good uniform coverage and a molecular diversity if they do not form clusters and leave relatively large areas of the feasible unexplored solution space [19], [20]. In order to achieve

a good uniform coverage, the priority queues are generated by selecting from left to right an atom in the priority queues for these molecules. In a molecule, an atom is selected and inserted into an appropriate position without violating the precedence constraints, aiming to make the largest Hamming distance of two molecules. If the condition $PopN < PopSize$ holds, then we select a molecule randomly from the initial molecular population and adopt a left rotating shift strategy according to the theory of maximum Hamming distance, aiming to have a good uniform coverage and a molecular diversity. Section 4 of the supplemental file, available online, presents the experimental results relevant to these approaches.

A detailed description for generating the initial molecular population is given in Algorithm 1.

Algorithm 1. Initial molecular population

Input:
Seed molecules generated by the heuristic rank policies.
Output:
The initial molecular population.
1: $PopN = 1$;
2: **for** each atom i in the molecule ω **do**
3: Find the first successor $Succ(i)$, namely T_j from T_i to the end;
4: $temp = T_i$;
5: **for** $k = i$ **to** $j - 2$ **do**
6: $T_k \leftarrow T_{k+1}$;
7: **end for**
8: $T_{j-1} = temp$;
9: Generate a new molecule and add to the population;
10: $PopN = PopN + 1$;
11: **end for**
12: **while** $PopN < PopSize$ **do**
13: Select a molecule ω in molecular population randomly;
14: Generate a new individual using the maximum code distance and the left rotating shift approach;
15: Add the new molecule to the population;
16: $PopN = PopN + 1$;
17: **end while**
18: **return** $Population$.

3.3 Hybrid Heuristic Operators

The size of search space of the heuristic CRO method proposed in this paper is much smaller than $n!$, and in order to find increasingly better solutions (i.e., the molecular structures with less energy), the operations simulating the four types of chemical reactions have to be performed over the solutions by using the heuristic transformational approach. In this section, we propose four molecular operators in detail for our task scheduling algorithm. First, in Section 3.3.1, we present a random walk approach, which follows the normal (or Gaussian) distribution. It could help the operator of on-wall ineffective collision to search for a local optimization candidate solution. Second, in Section 3.3.2, for the purpose of obtaining a global optimum or a near-optimal solution, we employ a left or right rotating shift strategy according to the theory of maximum Hamming distance, aiming to help the operator of decomposition to escape from local optima. Third, in Section 3.3.3, before the operator of inter-molecular ineffective collision, a novel selection approach based on the normal distribution is adopted, and an exclusive-OR (XOR) operator between two strings is used to reduce the chance of cloning before new molecules are generated. Then a pseudo-random shuffle approach is employed to generate new molecules to help the operator of inter-molecular ineffective collision to keep the molecular diversity. Fourth, in Section 3.3.4, in the operator of synthesis, we adopt a new scheme to realize the molecules to eliminate close relatives of the molecules according to the results calculated by an exclusive-OR (XOR) operator.

3.3.1 On-Wall Ineffective Collision

On-wall ineffective collision is an uni-molecular reaction, whose reactant involves only one molecule. When a molecule ω collides onto the wall of the closed container, it is allowed to change to another molecule ω' in order to search for a local optimization, if the condition $PE_{\omega'} \leq PE_{\omega}$ or $PE_{\omega'} \leq PE_{\omega} + KE_{\omega}$ holds, where *Potential energy* and *Kinetic energy (KE)* are two key properties attached to a molecular structure. The former corresponds to the *fitness* value of the solution, while the latter is used to control the acceptance of new solutions with worse fitness. After collision, the *KE* energy will be re-distributed. A certain portion of *KE* of the new molecule will be withdrawn to the central energy buffer (i.e., environment) [6]. In other words, it means that on-wall ineffective collision is a random search operator which searches a new and better molecule ω' around the molecule ω without violating the condition $0 \leq PE_{\omega'} \leq PE_{\omega} + KE_{\omega}$.

In this paper, a neighboring molecule ω' is generated, by adopting a random walk of one step approach in every on-wall ineffective collision, namely, Gaussian random walk of one step. A random walk having a step size that varies according to a normal (Gaussian) distribution is used as a model for helping the operator of on-wall ineffective collision to search for a local optimization candidate solution and to speed-up the search. Here, the step size has the normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where X is a normally distributed random variable, μ and σ are the mean and standard deviation of the normal distribution, respectively. We adopt the Box Muller transform [21] to obtain a normally distributed random variable X with any μ and σ.

Suppose U_1 and U_2 are independent random variables that are uniformly distributed in the interval $(0, 1]$. Let stochastic variable $X \sim \mathcal{N}(\mu, \sigma^2)$, with mean μ and variance σ^2, be given as

$$X = \mu + \sigma\sqrt{-2\ln(1 - U_1)}\cos(2\pi U_2), \tag{3}$$

where X is independent random variables following a standard normal distribution with any μ and σ. According to statistics, the $68 - 95 - 99.7$ rule states that nearly all values lie within three standard deviations of the mean in a normal distribution [22]. The operation is illustrated in Fig. 7.

Therefore, in on-wall ineffective collision operator, first, a position i in the queue ω is randomly selected. The task corresponding to position i is denoted as T_i. Second, from position i to the beginning of the queue, we identify the last predecessor of T_j, assuming its position in the queue is j.

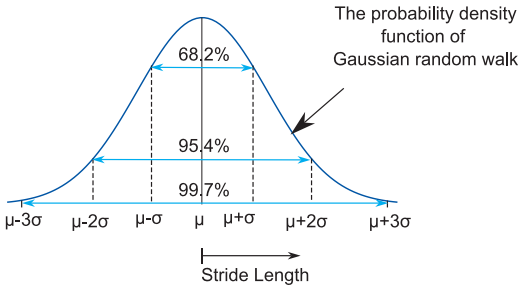


Fig. 7. Illustration of a probability density function of Gaussian random walk.

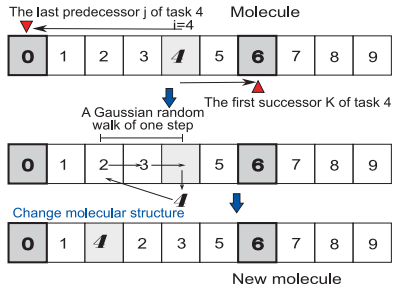


Fig. 8. Illustration of on-wall ineffective collision.

Third, from position i to the end of the queue, we identify the first successor of T_k, assuming its position in the queue is k. Then according to Eq. (3), namely the half-normal distribution, we can calculate the independent random variable X with mean $\mu = 0$ and variance $\sigma^2 = 1$, and a random walk $x = \frac{\max(i-j,k-i)\times|X|}{3}$, where $|X|$ is the absolute value of X.

If the condition $i - j < k - i$ and $i + x < k$ holds, then first we put T_i to the temporary variable $Temp$. Second all atoms, from the position $i + 1$ to $i + x$, shift one position to the left. Third, the temporary variable $Temp$ is placed at the position $i + x$. Otherwise, we put T_i to the temporary variable $Temp$. All atoms, from the position $i - 1$ downto $i - x$, shift one position to the right one by one. Last, the temporary variable $Temp$ is placed at the position $i - x$.

By this operator, a new module ω' is generated. The operation is illustrated in Fig. 8.

A detailed description of the on-wall ineffective collision operator is given in Algorithm 2.

Algorithm 2. On-wall ineffective collision operator

Input:
A molecule ω randomly selected from the current Pop.
Output:
A new molecule ω'.
1: Choose randomly an atom T_i, where $i \in [1, n-1]$;
2: Calculate the last predecessor T_j of the atom T_i;
3: Calculate the first successor T_k of the atom T_i;
4: Calculate the independent random variable X with $\sigma = 1$ and $\nu = 0$ using Eq. (3);
5: Calculate the random walk $x = \frac{\max(i-j,k-i)\times|X|}{3}$;
6: Generate a new molecule ω' copied from the molecule ω;
7: **if** $i - j < k - i$ and $i + x < k$ **then**
8: $Temp \leftarrow T'_i$;
9: **for** $l = i$ **to** $i + x - 1$ **do**
10: $T'_l \leftarrow T'_{l+1}$;
11: **end for**
12: $T'_{i+x} \leftarrow Temp$;
13: **else**
14: $Temp \leftarrow T'_i$;
15: **for** $l = i$ **downto** $i - x + 1$ **do**
16: $T'_l \leftarrow T'_{l-1}$;
17: **end for**
18: $T'_{i-x} \leftarrow Temp$;
19: **end if**
20: **return** a new molecule ω'.

3.3.2 Decomposition

In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different [23]. It can be formally defined as follow,

Definition. *The Hamming distance d(x, y) between two vectors* $x, y \in F_m^{(n)}$ *is the number of coefficients in which they differ.*

For example, $\mathbb{F}_2^{(5)}$ $d(00111, 11001) = 4$, and $\mathbb{F}_{10}^{(10)}$ $d(0431256789, 0412563789) = 5$. Notice that the subscript m denote the different base number system, and the superscript n denote the length of strings.

The decomposition operator is also an uni-molecular reaction. A molecule ω can decompose into two new molecules, ω'_1 and ω'_2, if the condition $PE_\omega + KE_\omega + buffer \geq PE_{\omega'_1} + PE_{\omega'_2}$ holds [6], where $buffer$ denotes the energy stored in the central buffer, which plays an important role to help the HCRO to obtain a global optimum or a near-optimal solution.

In other words, it means that the decomposition operator can generate the new molecules, ω'_1 and ω'_2. If the condition $PE_{\omega'_1} \leq PE_\omega$ or $PE_{\omega'_2} \leq PE_\omega$ holds, the decomposition operator gets a better solution. Otherwise, if the condition $PE_{\omega'_1} \geq PE_\omega$ or $PE_{\omega'_2} \geq PE_\omega$ holds, the decomposition operator generates a worse solution randomly, but may help HCRO to escape from local optima when the on-wall ineffective collision operator executes a certain number of times and doesn't get any better solution. Therefore, in this operator, the further the molecule ω'_1 and ω'_2 jump from the molecule ω, the more the decomposition operator can help HCRO to obtain a global optimum or a near-optimal solution.

In this paper, for the purpose of obtaining a global optimum or a near-optimal solution, we employ a left or right rotating shift strategy according to the theory of maximum Hamming distance, aiming at helping the operator of decomposition to escape from local optima. The key feature of a left or right rotating shift strategy is that it provides a mechanism to escape local optima by allowing moves which worsen the objective function value. For instance, in this operation, a solution ω is decomposed into two new solutions ω'_1 and ω'_2. The solution ω'_1 is generated in the following steps: 1) We randomly select a position i in ω (the corresponding task is T_i); 2) From the position i to the beginning of the queue, we identify the last predecessor of T_j, assume its position in the queue is j, and from the position i to the ending of the queue, we identify the first successor of T_k, assume its position in the queue is k; 3) If the condition $i - j > k - i$ holds, then T_i is stored in the temporary variable $Temp$, and then from the positions $i - 1$ to $j + 1$, we

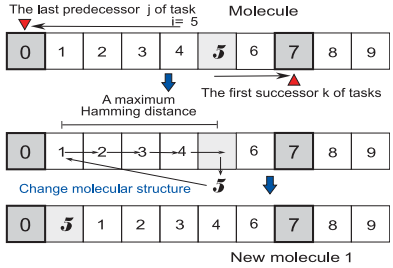


Fig. 9. Illustration of decomposition.

shift one position to the right one by one, and generate an empty slot for the next atom. Last, the temporary variable $Temp$ is placed at the position $j+1$. Otherwise, T_i is stored in the temporary variable $Temp$, and from the position $i+1$ to $k-1$, we shift one place to the left. Then the temporary variable $Temp$ is placed at the position $k-1$.

The approach can effectively help the HCRO operator of decomposition to escape from local optima, by keeping the maximum hamming distance between the old molecule ω and the new molecule ω_1'. From another point of view, it can effectively improve uniform coverage of the population too. We can generate another new solution ω_2' in the similar fashion.

Note that because of the decomposition (and the synthesis operator discussed later) operator, the number of solutions in the CRO process may be varied during the reactions, which is a feature of CRO that is different from genetic algorithms. This operation is illustrated in Fig. 9.

A detailed description of the decomposition operator is given in Algorithm 3.

Algorithm 3. Decomposition operator

Input:
A molecule ω selected randomly from the current Pop.
Output:
Two new molecules ω_1' and ω_2'.

while the decomposition operator is not finished **do**
  Choose randomly a atom from the molecule ω, where $i \in [1, n-1]$;
  Generate a new molecule ω' which is copied from the molecule ω;
  Find the last predecessor T_j of the atom T_i from the position i to the beginning of queue;
  Find the first successor T_k of the atom T_i from the position i to the ending of queue;
  if $i-j < k-i$ **then**
    $Temp \leftarrow T_i'$;
    for $l = i$ **to** $k-1$ **do**
      $T_l' \leftarrow T_{l+1}'$;
    end for
    $T_{k-1}' \leftarrow Temp$;
  else
    $Temp \leftarrow T_i'$;
    for $l = i$ **downto** $j+1$ **do**
      $T_l' \leftarrow T_{l-1}'$;
    end for
    $T_{j+1}' \leftarrow Temp$;
  end if
  Copy ω' as a new molecule ω_1' or ω_2';
end while
return two new molecules ω_1' and ω_2'.

TABLE 3
XOR String Operator

	Position of atoms										Hamming distance
	0	1	2	3	4	5	6	7	8	9	
ω_1	0	1	2	3	4	5	6	7	8	9	
ω_2	0	1	2	3	5	4	8	6	7	9	
Result	0	0	0	0	1	1	1	1	1	0	**5**

3.3.3 *Inter-Molecular Ineffective Collision*

The inter-molecular ineffective collision operator is an inter-molecular reaction, whose reactants involve two molecules. When two molecules, ω_1 and ω_2, collide into each other, they can change into two new molecules, ω_1' and ω_2', if the condition $PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega_1'} + PE_{\omega_2'}$ holds [6]. It means that the molecules ω_1' and ω_2' can have a wide range of random search space than those of on-wall ineffective collision operator can do. In other words, the inter-molecular ineffective collision operator can generate more different molecules to keep a molecular diversity and to search a wider range of solution space. In this section, a novel selection approach based on the normal distribution, an exclusive-OR (XOR) operator between two strings, and a pseudo-random shuffle approach are employed to generate new molecules to help the operator of inter-molecular ineffective collision to keep the molecular diversity.

For the presence of the task scheduling problem, the operator of inter-molecular ineffective collision is combining two valid molecules, whose subtasks are ordered topologically, to generate two new molecules which are also valid. Booker [24] suggested that before applying crossover, we should examine the selected parents to find suitable crossover points. This entails computing an exclusive-OR between the parents, so that only positions between the outermost $1s$ of the XOR string (the reduced surrogate) should be considered as crossover points.

For instance, two task priority queues, ω_1 and ω_2, are shown in Table 3.

They will generate only clones if the crossover point is anyone of the first three positions. Therefore, as previously stated, only crossover points from four to eight will give rise to a different string.

We apply the same principles to the HCRO algorithm. In order to keep the molecular diversity, the operator of inter-molecular ineffective collision adopts a novel approach when new molecules are generated. In this paper, the operator of inter-molecular ineffective collision consists of two stages: a selection stage and a reaction stage.

In the first stage, two candidate molecules are selected in the following steps. First, one molecule ω_1 is selected from the molecular population. Second, some molecules are selected randomly as the candidates, and the Hamming distances are computed between the molecule ω_1 and the candidates. Third, according to the Hamming distance, the candidates are sorted in descending order. Fourth, according to Eq. (3), we can calculate the independent standard normal random variable X with mean $\mu = 0$ and variance $\sigma^2 = 1$. Last, we select one molecule as ω_2, whose index is $x = \frac{the\ number\ of\ candidates \times |X|}{3}$, from the candidates.

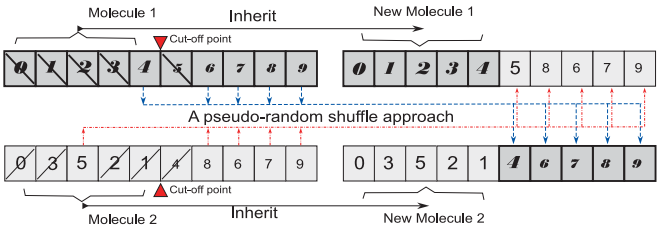


Fig. 10. Illustration of inter-molecular ineffective collision.

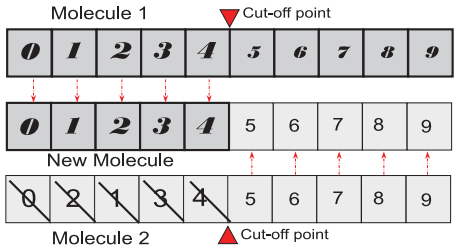


Fig. 11. Illustration of synthesis.

In the second stage, the operations are used to generate two new solutions ω_1' and ω_2' from two existing candidate solutions ω_1 and ω_2. The new solutions are generated by using a pseudo-random shuffle method as shown in Fig. 10. 1) The suitable crossover point i is chosen randomly between the outermost $1s$ of the XOR string and n, and the crossover point cuts the priority queues of the pair of molecules (ω_1 and ω_2) into left and right segments. 2) The left segments of ω_1' and ω_2' are inherited from the left segments of ω_1 and ω_2, respectively. 3) Each task in the right segment of ω_1' comes from the tasks in ω_2 that do not appear in the left segment of ω_1. This can guarantee that a task will not appear more than once in a new solution. Similarly, each task in the right segment of ω_2' comes from the tasks in ω_1 that do not appear in the left segment of ω_2. The operation is illustrated in Fig. 10.

A detailed description of the inter-molecular ineffective collision operator is given in Algorithm 4.

Algorithm 4. Inter-molecular ineffective collision operator

Input:
The current Pop.
Output:
Two new molecules ω_1' and ω_2'.
1: Choose randomly a molecules ω_1 from the current population Pop;
2: Choose some molecules as candidates from the population;
3: Calculate the Hamming distance between the molecules ω_1 and the candidates;
4: Sort the candidates in descending order by the Hamming distance;
5: Calculate the independent random variable X with $\sigma = 1$ and $\mu = 0$ using Eq. (3);
6: Calculate $x = \frac{the\ number\ of\ candidates \times |X|}{3}$;
7: Select one molecule as ω_2, whose index is x, from the candidates;
8: Choose randomly a suitable cut point i according to an exclusive-OR result;
9: Inherit from the left segments of ω_1 and ω_2 to generate the left segments of ω_1' and ω_2', respectively;
10: Generate each task in the right segments of ω_1', which comes from the tasks in ω_2 that do not appear in the left segment of ω_1
11: Generate each task in the right segments of ω_2', which comes from the tasks in ω_1 that do not appear in the left segment of ω_2;
12: **return** two new molecules ω_1' and ω_2'.

3.3.4 Synthesis

The synthesis operator is also an inter-molecular reaction. When two molecules, ω_1 and ω_2, collide into each other, they can be combined to generate a new molecule, ω', if the condition $PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega'}$ holds [6]. It means that the synthesis operator is a random search operator too, which searches a new better or worse molecule ω' randomly, and satisfies the condition $0 \leq PE_{\omega'} \leq PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2}$. In other words, the synthesis operator can generate a new and better or worse molecule. However, in this paper, a new scheme is added into the synthesis operator to eliminate close relatives and to keep the molecular diversity.

The synthesis operator consists of two stages: a selection stage and a synthesis stage. In the first stage, two candidate molecules are selected in the following steps. First, one molecule ω_1 is selected from the molecular population randomly. Second, some molecules are selected if they have equal PE to ω_1, and the Hamming distances, which denote close relatives, are computed between the molecule ω_1 and candidates. Third, according to the Hamming distance, the candidates are sorted in ascending or descending order, respectively. Then we can choose one order randomly. The ascending order denotes that the synthesis operator is to eliminate close relatives and to keep the molecular diversity, and a descending order denotes that the synthesis operator is to escape from local optima. Fourth, according to Eq. (3), we can calculate the independent standard normal random variable X with mean $\mu = 0$ and variance $\sigma^2 = 1$. Last, we select one molecule as ω_2, whose index is $x = \frac{the\ number\ of\ candidates \times |X|}{3}$, from the candidates.

In the second stage, we generate a new solution ω' from two existing solutions ω_1 and ω_2, which have close or distant relatives. The new solution is generated in the following steps. 1) If the molecules ω_1 and ω_2 are not the same, then a suitable crossover point i is chosen randomly between the outermost $1s$ of the XOR string and n, and the crossover point cuts the priority queues of the pair of molecules (ω_1 and ω_2) into left and right segments; else the molecule ω_2 is deleted directly, and the operator is over. 2) The left segment of ω_{t1} is inherited from the corresponding segment of ω_1, and each task in the right segment of ω_{t1} comes from the ω_2 and it does not appear in the left segment of ω_1. In contrast, the left segment of ω_{t2} is inherited from the corresponding segment of ω_2, and each task in the right segment of ω_{t2} comes from the ω_1 that it does not appear in the left segment of ω_2. 3) We select a better molecule between the ω_{t1} and the ω_{t2}, and copy it to the ω'.

This operation is illustrated in Fig. 11.

A detailed description of the synthesis operator is given in Algorithm 5.

Algorithm 5. Synthesis operator

Input:
The current population Pop.
Output:
One new molecule ω'.
1: Choose randomly a molecules ω_1 from the current population Pop;
2: Choose some molecules as candidates from the population;
3: Calculate the Hamming distance between the molecules ω_1 and the candidates;
4: **if** $a\ random\ number > 0.5$ **then**
5: Sort the candidates in ascending order by the Hamming distance;
6: **else**
7: Sort the candidates in descending order by the Hamming distance;
8: **end if**
9: Calculate the independent random variable X with $\sigma = 1$ and $\mu = 0$ using Eq. (3);
10: Calculate $x = \frac{the\ number\ of\ candidates \times |X|}{3}$;
11: Select one molecule as ω_2, whose index is x, from the candidates;
12: **if** the molecules ω_1 and ω_2 are the same **then**
13: Delete ω_2 from the current population;
14: **else**
15: Choose randomly a suitable cut point i according to an exclusive-OR result;
16: Inherit from the left segments of ω_1 and ω_2, and generate the left segments of ω_{t1} and ω_{t2}, respectively;
17: Copy from the tasks in ω_2 to the right segments of ω_{t1} that do not appear in the left segment of ω_1;
18: Copy from the tasks in ω_1 to the right segments of ω_{t2} that do not appear in the left segment of ω_2;
19: **if** $PE_{\omega_{t1}} > PE_{\omega_{t2}}$ **then**
20: Copy the molecule ω_{t2} to the new molecule ω';
21: **else**
22: Copy the molecule ω_{t1} to the new molecule ω';
23: **end if**
24: **end if**
25: **return** the new molecule ω'.

3.4 Heuristic Mapping and Potential Energy Value Computing

Given a solution of the execution order obtained in Section 3, the heuristic method tries to map each task in the solution (in the order of its position in the queue) to a computing processor that can provide the *earliest finish time*. For the task scheduling problem, the goal is to obtain a task assignment that ensures the minimum makespan and guarantees the precedence constraints among all tasks. The earliest finish time of the exit task will be the makespan of the DAG application. The scheduling length, namely makespan, or potential energy is defined as:

$$PE = makespan = AFT(T_{exit}), \tag{4}$$

where $AFT(T_{exit})$ is the actual finishing time of the exit task T_{exit}. The overall schedule length of the entire DAG is the latest finish time among all tasks, which is equivalent to the actual finish time of the exit task T_{exit}.

In this paper, in order to accelerate convergence speed of the HCRO algorithm, a heuristic-based *heterogeneous earliest finish time* (HEFT) algorithm is applied to realize the performance-effective and low-complexity approach [18], which avoids less effective task-to-processor mapping [25]. The HEFT is proposed to search for a solution in order to minimize $makespan$ without violating precedence constraints. Section 3 of the supplemental file, available online, presents the entire framework of combining CRO and the heuristic method in the hybrid CRO algorithm.

3.5 Time and Space Complexity Analysis

The time complexity of the HCRO is analyzed as follows. The time is mainly spent in running the searching loop in the proposed HCRO. In each iteration of the loop, the algorithm may need to execute a fitness evaluation function, on-wall ineffective collision, decomposition operator, inter-molecular ineffective collision, and synthesis operator. The time complexity of a fitness evaluation function is $O(e \times m)$ [18], where e is the number of edges in the DAG and m is the number of heterogeneous computing processors. The worst case time complexity of on-wall ineffective collision, decomposition operator, inter-molecular ineffective collision, and synthesis operator is $O(n^2)$. Therefore, the time complexity of the HCRO is $O(iters \times (e \times m + n^2))$, where $iters$ is the number of iterations performed by the HCRO. For a dense graph where the number of edges is proportional to $O(n^2)$ (n is the number of subtasks), the time complexity is thus $O(iters \times n^2 \times m)$.

The space complexity of the HCRO can be analyzed as follows. In the HCRO, for each molecule we need an array of size $(2 \times n)$ to store it. There are *PopSize* molecules in the initial *Pop*. Therefore, the space complexity of the HCRO is $O(PopSize \times n)$.

4 Experimental Studies

In this section, we conduct the experiments to verify the effectiveness of our proposed HCRO. Since the proposed HCRO combines CRO with the heuristic methods, we compare our HCRO with two popular existing heuristic methods HEFT-B, CPOP [18], and the DMSCRO method, which is a pure meta-heuristic method developed in [8] to schedule DAG tasks. HEFT-B uses the length of the longest path, i.e., from the exit task of the DAG application to task T_i, to determine T_i's execution order, while CPOP adopts different mapping strategies for the nodes in the critical path and the non-critical path nodes. A CP processor is defined as the processor that minimizes the overall execution time of the critical path assuming all nodes in the critical path are mapped onto the CP processor. If the selected node is in the critical path, it is mapped onto the CP processor. Otherwise, it is mapped onto a processor that minimizes its EFT (similar as in the HEFT algorithm). Since the DMSCRO algorithm developed in [8] is a pure meta-heuristic method, it spends much longer time than the proposed HCRO to find a desirable scheduling solution, which has been analyzed in Theorem 1 and will also be demonstrated in the experiments.

The performance metric used for comparison is makespan. We use the *Communication to Computation Ratio* (CCR) of a DAG to represent the characteristic of the graph, which

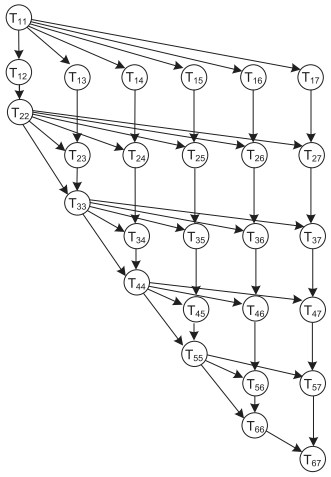


Fig. 12. Gaussian elimination of problem size 7.

is communication-intensive or computation-intensive. The higher the value of CCR is, the more communication-intensive the DAG graph is. The lower the value of CCR is, the more computation-intensive the DAG is. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target computing system.

The HCRO is programmed in C#. A DAG task in the program is represented by a class, whose members include an array of subtasks, a matrix of the speed at which each subtask runs on each computing processor, and a matrix of communication data between every pair of subtasks. A subtask in the program is also represented by a class, whose members include an array of predecessors of the subtask, an array of successors of the subtask, the in-degree of the subtask, the out-degree of the subtask, and the computational data of the subtask. The simulations are performed on a computer with an Intel Core 2 Duo-E6700 @ 2.66 GHz CPU and 2 GB RAM.

In the experiments, based on the results of preliminary studies, the values of the parameters to be used in the HCRO are determined as follows:

- $InitialKE = 10{,}000$; $buffer = 2{,}000$;
- $MoleColl = 0.2$; $KE_{LossRate} = 0.5$;
- $\alpha = 200$; $\beta = 100$;

where *InitialKE* is the initial energy associated to molecules, *buffer* is the initial energy in the energy buffer, *MoleColl* is the rate of which determines whether to perform an uni-molecular or an inter-molecular operation, $KE_{LossRate}$ is the loss rate of the KE energy. α and β are thresholds defined for the conditions for decomposition and synthesis, respectively. These values are deduced from [6]. In our experiments, α and β are dynamically adjusted to keep the population size. The algorithms are terminated when the value converges to a relatively stable state (i.e., the makespan remains unchanged) for a preset number of consecutive iterations in the search loop (in the experiments, it is 10,000).

TABLE 4
Matrix Size and Number of Subtasks

No.	Matrix size	Number of subtasks
0	7	27
1	10	54
2	15	119
3	20	209

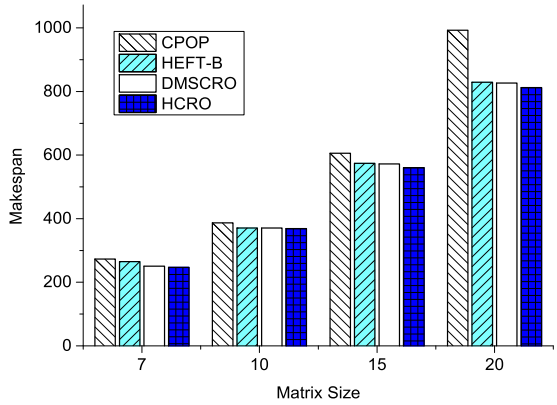


Fig. 13. Average makespan of Gaussian elimination with different matrix size (the number of processors = 8, CCR = 1.0, 100 independent runs).

4.1 Real World Application Graphs

This experiment uses the task graphs from two real world applications: Gaussian elimination [26] and Molecular dynamics code [27].

4.1.1 Gaussian Elimination

Gaussian elimination is used to calculate the solution of a set of linear equations. In this experiment, a Gaussian elimination algorithm with the problem size of seven (i.e., the number of linear equations or the size of the coefficient matrix) was used to evaluate the HCRO. Its underlying DAG is shown in Fig. 12. The total number of tasks in the graph is 27, and the largest number of tasks at the same level is six (i.e., degree of parallelism in the DAG). Moreover, because of the feature of the Gaussian elimination application, each task in the graph contains the same operations, and the same amount of data is communicated from a task to its children tasks.

The computational complexity of Gaussian elimination is $O(n^3)$; that is, the number of operations required is (approximately) proportional to n^3 for a matrix of size $n \times n$. The DAG for the Gaussian elimination algorithm for $n = 7$, $n = 10$, $n = 15$ and $n = 20$ is shown in Table 4 where n is the size of the matrix.

Since the structure of the graph is fixed, the total number of subtasks in a graph can be calculated according to the size of the matrix of Gaussian elimination. In the experiments, the number of heterogeneous processors and the matrix size values are varied. CCR value is equal to 1.0 in the experiments.

Fig. 13 shows the performance of these algorithms under the increasing problem size of the Gaussian elimination application. As the number of tasks increases, the figure shows that the HCRO outperforms HEFT-B and CPOP, which is to be expected. The reason may lie in that when the number of tasks becomes bigger, the problem becomes more complicated and it becomes increasingly difficult for the heuristic algorithms to find good solutions.

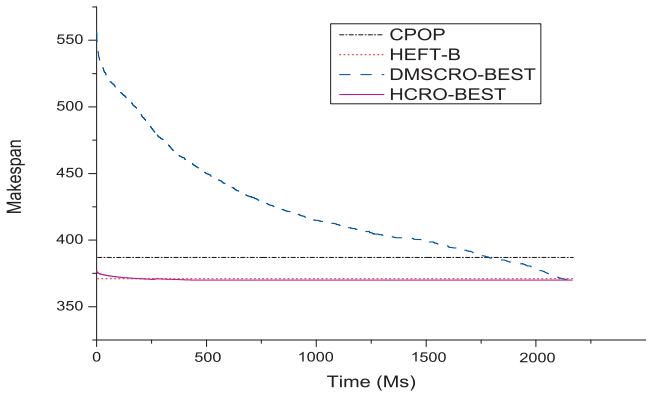


Fig. 14. The convergence trace of the average makespan of the best individual for Gaussian elimination (the number of processors = 8, CCR = 1.0, the matrix of size 10, 100 independent runs).

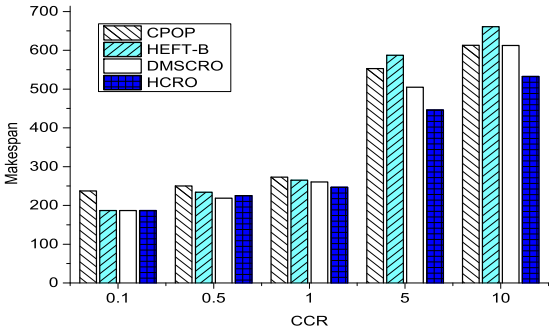


Fig. 15. Average makespan of Gaussian elimination with different CCR (the number of processors = 8, the matrix of size 7, 100 independent runs).

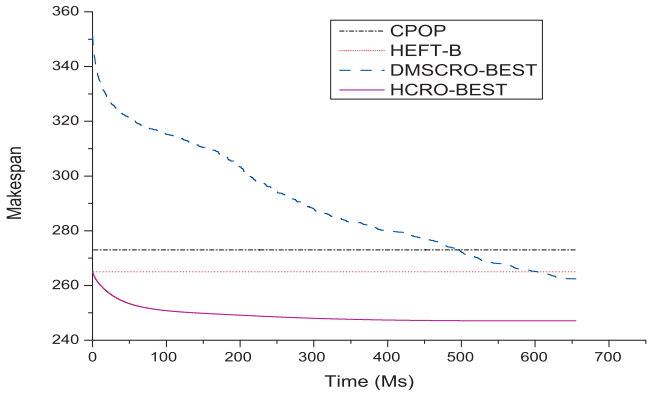


Fig. 16. The convergence trace of the average makespan of the best individual for Gaussian elimination (the number of processors = 8, CCR = 1.0, the matrix of size 7, 100 independent runs).

Fig. 15 shows the performance of these four algorithms as CCR increases. It can be observed that the average *makespan* increases rapidly with the increasing of CCR. This may be because when CCR increases, the application becomes more communication-intensive, and consequently, the computing processors remain idle for longer periods of time. Fig. 15 shows that the HCRO outperforms HEFT-B and CPOP, and achieves better performance than DMSCRO. We can observe that the advantage becomes more obvious when CCR becomes bigger.

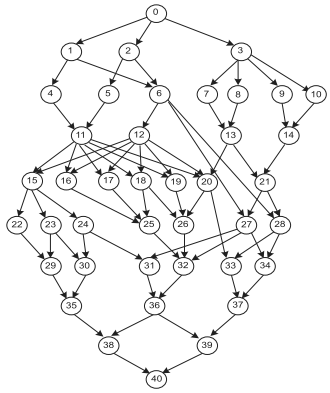


Fig. 17. A molecular dynamics code.

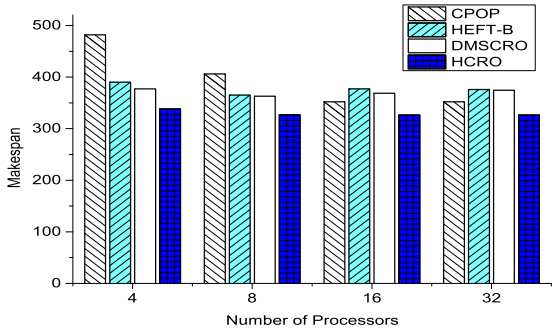


Fig. 18. Average makespan for the molecular dynamics code (the number of subtasks = 41, CCR = 1.0, 100 independent runs).

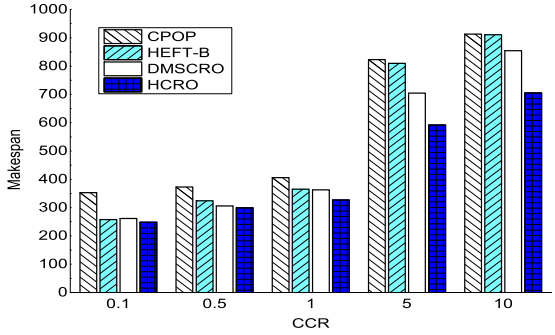


Fig. 19. Average makespan for the molecular dynamics code with different CCR (the number of subtasks = 41, the number of processors = 8, 100 independent runs).

Figs. 14 and 16 show the final makespan achieved by the HCRO and the DMSCRO after the stopping criteria are satisfied. In this case, the fact that the HCRO converges faster than the DMSCRO means that the makespan obtained by the HCRO could be much better than that of the DMSCRO when the algorithms stop.

4.1.2 *Molecular Dynamics Code*

This experiment used a DAG extracted from the molecular dynamics code presented in [27] to evaluate the performance of the scheduling algorithms. The underlying DAG is shown in Fig. 17.

Fig. 19 shows the performance of these four algorithms as CCR increases. These results suggest that the heuristic algorithms HEFT-B and CPOP perform less effectively for

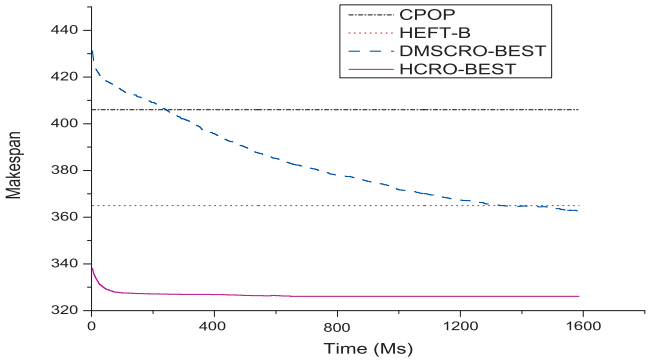


Fig. 20. The convergence trace of the average makespan of the best individual for the molecular dynamics code (the number of subtasks = 41, the number of processors = 8, CCR = 1.0, 100 independent runs).

TABLE 5
The Different Parameters of Random Task Graphs

Tasks	Levels	Maximum In-degree	Maximum Out-degree	Average In-degree
10	3-11	11	11	3.63
20	4-21	21	21	5.46
50	6-44	42	44	5.86
100	7-57	47	51	7.81
200	7-77	181	188	30.72

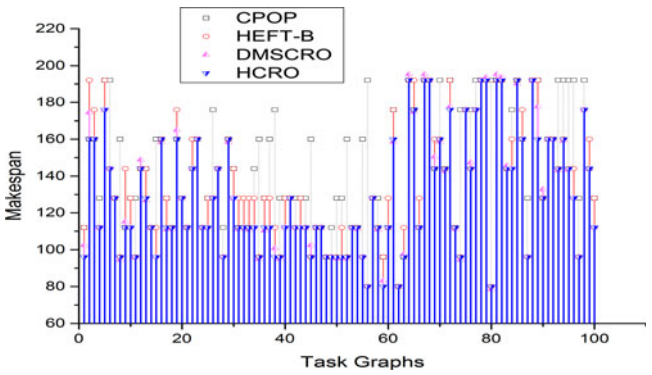


Fig. 21. The average makespan of task graphs with different characteristics (the size of the task graphs = 10, the number of task graphs = 100, the number of processors = 16, 100 independent runs).

communication-intensive applications, while the HCRO and the DMSCRO can deliver more consistent performance in a wide range of scheduling scenarios.

Fig. 18 shows the average *makespan* achieved by these four algorithms under the increasing number of computing processors. In Fig. 18, the makespan decreases with the increasing number of computing processors, which is to be expected. The trend observed in these figures is consistent with those seen in our simulation experiments. The results observed in all these figures show that the HCRO proposed in this paper outperforms HEFT-B, CPOP, and the DMSCRO in most cases.

This figure also shows that when the number of computing processors increases, the average makespan decreases, which is to be expected. Furthermore, the decreasing trend of the average makespan tails off as the number of computing processors continues to increase. This is because the degree of parallelism is limited in the DAG application. When the

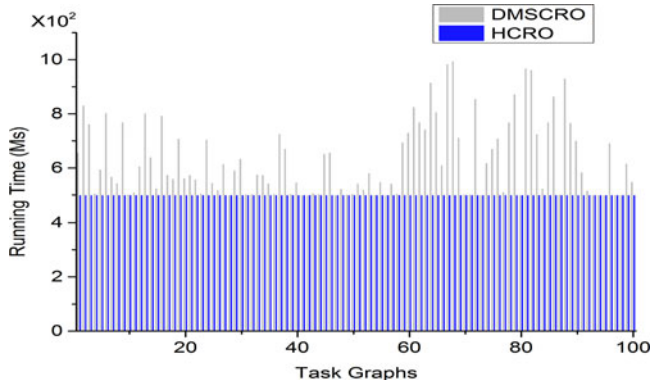


Fig. 22. The average running time of task graphs with different characteristics (the size of the task graphs = 10, the number of task graphs = 100, the number of processors = 16, 100 independent runs).

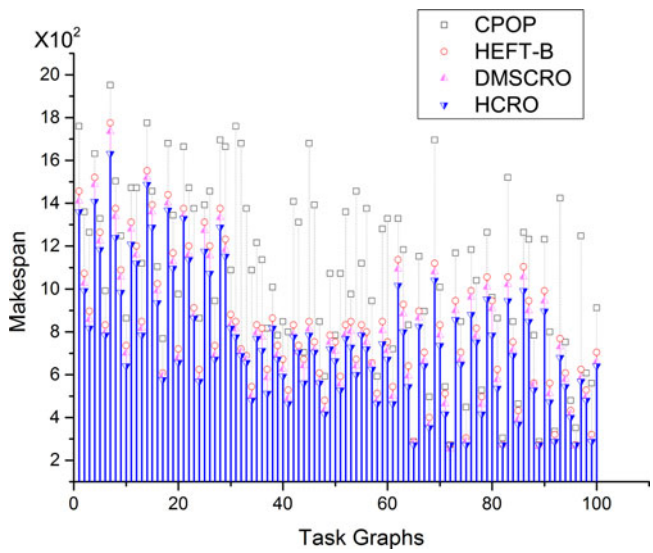


Fig. 23. The average makespan of task graphs with different characteristics (the size of the task graphs = 200, the number of task graphs = 100, the number of processors = 16, 100 independent runs).

number of computing processors approaches the degree of parallelism, further increasing the number of computing processors will be of little help in reducing the makespan.

Fig. 20 shows the final makespan achieved by the HCRO and the DMSCRO after the stopping criteria are satisfied. It can be seen that the HCRO can obtain a better average makespan performance than the DMSCRO. This result once again verifies that the HCRO is able to strike a good balance between performance and overhead.

4.2 Randomly Generated Application Graphs

In these experiments, we use randomly generated task graphs to evaluate the performance. In order to generate random graphs, we implemented a random graph generator which allows the user to generate a variety of random graphs with different characteristics. Input parameters of the generator are CCR, the number of instructions in a task (representing the computation workload), the levels of the graphs, and the number of tasks in a graph. We have generated a large set of random task graphs with different characteristics, and scheduled these task graphs on a heterogeneous computing system. We have evaluated the

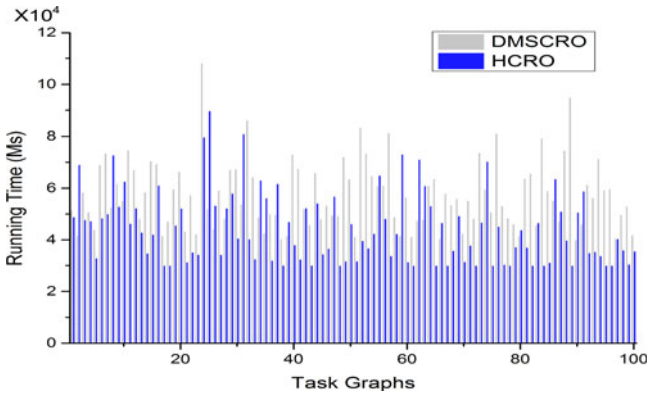


Fig. 24. The average running time of task graphs with different characteristics (the size of the task graphs = 200, the number of task graphs = 100, the number of processors = 16, 100 independent runs).

performance of the algorithms under different parameters, including numbers of tasks, different numbers of computing processors, and different CCR values. The performance of the HCRO is compared with those of other algorithms. Each value plotted in the graphs is the result averaged over 100 different random DAG graphs.

The following is the values of parameters used in the simulation experiments, unless otherwise stated. The number of tasks generated in a DAG is randomly selected from 10, 20, 50, 100, and 200. The number of instructions in the tasks varies randomly between 1 and 80. The number of successors that a task can have is a random number. The random DAG task graphs have the following characteristics shown in Table 5.

Figs. 21 and 23 show that the HCRO outperforms HEFT-B and CPOP. This may be because the problem becomes more complicated and it becomes increasingly difficult for the heuristic algorithms to find good solutions when the number of tasks becomes bigger. It can also be seen from these figures that the HCRO is able to achieve better average performance than the DMSCRO in all cases.

Figs. 22 and 24 show the final makespan achieved by the HCRO and the DMSCRO after the stopping criteria are satisfied.

Our experiments show that the HCRO spends much less overhead to find a desirable solution than the DMSCRO. The results indicate that although the HCRO costs much less than the DMSCRO to find a sub-optimal solution, it can

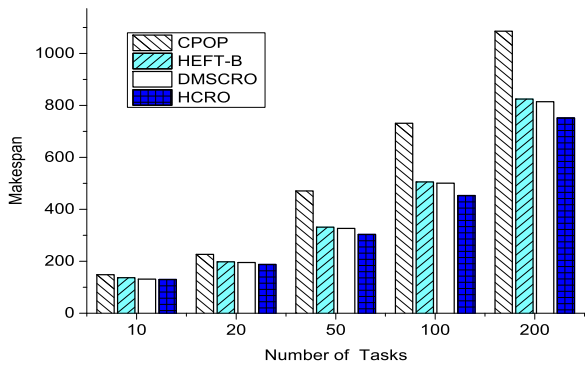


Fig. 25. The average makespan of task graphs with different size (the number of task graphs = 100, the number of processors = 16, 100 independent runs).

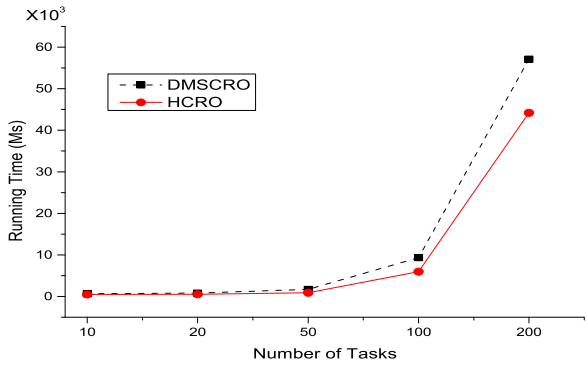


Fig. 26. The average running time of task graphs with different size (the number of task graphs = 100, the number of processors = 16, 100 independent runs).

achieve similar or better performance. Our experimental results show that the HCRO can obtain a better average makespan performance than the DMSCRO.

Figs. 25 and 26 show that the proposed HCRO algorithm outperforms HEFT-B and CPOP, and it can achieve a better average performance than the DMSCRO with lower overhead. In this experiment, the algorithms are stopped the search when the performance stabilizes (i.e., the makespan remains unchanged) for a preset number of consecutive iterations in the search loop (in the experiments, it is 10,000).

5 Conclusions

In this paper, a hybrid chemical reaction optimization approach is proposed for DAG scheduling on heterogeneous computing systems. The algorithm incorporates the CRO technique to search the execution order of tasks while using a heuristic method to map tasks to computing processors. By doing so, our proposed HCRO scheduling algorithm can achieve a good performance without incurring a high scheduling overhead. In the experiments, the proposed HCRO is compared with two heuristic algorithms (HEFT-B and CPOP) and a pure meta-heuristic method (DMSCRO). The results show that our proposed HCRO algorithm outperforms HEFT-B and CPOP, and it can achieve a better average performance than the DMSCRO with lower overhead.

Acknowledgments

The authors are very grateful to the three anonymous reviewers for their constructive comments which have helped to improve the manuscript. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124, 61202109, 61472126). K. Li is the corresponding author.

References

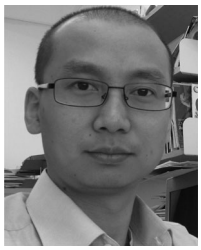
[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NPCompleteness*. New York, NY, USA: Freeman, 1979.

[2] F. Ferrandi, P. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.

[3] R. Correa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 825–837, Aug. 1999.
[4] A. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: Comparative studies and performance issues," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 795–812, Aug. 1999.
[5] M. Kashani and M. Jahanshahi, "Using simulated annealing for task scheduling in distributed systems," in *Proc. Int. Conf. Comput. Intell., Model. Simul.*, sep. 2009, pp. 265–269.
[6] A. Lam and V. Li, "Chemical-reaction-inspired metaheuristic for optimization," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 381–399, Jun. 2010.
[7] J. Xu, A. Lam, and V. Li, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1624–1631, Oct. 2011.
[8] 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," *J. Parallel Distrib. Comput.*, vol. 73, no. 9, pp. 1306–1322, 2013.
[9] S. Santander-Jimenez and M. Vega-Rodriguez, "Parallel multiobjective metaheuristics for inferring phylogenies on multicore clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. PP, no. 99, pp. 1–1, May. 2014.
[10] Y. Bai, S. Xiao, C. Liu, and B. Wang, "A hybrid iwo/pso algorithm for pattern synthesis of conformal phased arrays," *IEEE Trans. Antennas Propagation*, vol. 61, no. 4, pp. 2328–2332, Apr. 2013.
[11] Y. Tominaga, Y. Okamoto, S. Wakao, and S. Sato, "Binarybased topology optimization of magnetostatic shielding by a hybrid evolutionary algorithm combining genetic algorithm and extended compact genetic algorithm," *IEEE Trans. Magnetics*, vol. 49, no. 5, pp. 2093–2096, May. 2013.
[12] J. Yang, W. Li, X. Shi, L. Xin, and J. Yu, "A hybrid abcde algorithm and its application for timemodulated arrays pattern synthesis," *IEEE Trans. Antennas Propagation*, vol. 61, no. 11, pp. 5485–5495, Nov. 2013.
[13] D. Fodorean, L. Idoumghar, and L. Szabo, "Motorization for an electric scooter by using permanentmagnet machines optimized based on a hybrid metaheuristic algorithm," *IEEE Trans. Veh. Technol.*, vol. 62, no. 1, pp. 39–49, Jan. 2013.
[14] M. Li, H. Kang, and P. Zhou, "Hybrid optimization algorithm based on chaos, cloud and particle swarm optimization algorithm," *Syst. Eng. Electron., J.*, vol. 24, no. 2, pp. 324–334, Apr. 2013.
[15] X. Li and Y. Zhang, "Adaptive hybrid algorithms for the sequencedependent setup time permutation flow shop scheduling problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 3, pp. 578–595, Jul. 2012.
[16] H. Cheng, I. Chung, and W. Chen, "Thermal chip placement in mcms using a novel hybrid optimization algorithm," *IEEE Trans. Components, Packaging Manufacturing Technol.*, vol. 2, no. 5, pp. 764–774, May 2012.
[17] Hybrid_algorithm. (2014). [Online]. Available: http://en.wikipedia.org/wiki/Hybrid_algorithm.htm
[18] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performanceeffective and lowcomplexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[19] A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith, " Solving the graphical Steiner tree problem using genetic algorithms," *J. Oper. Res. Soc.*, vol. 44, no. 4, pp. 397–406, 1993.
[20] D. Levine, "Commentary genetic algorithms: A practitioner's view," *INFORMS J. Comput.*, vol. 9, no. 3, pp. 256–259, 1997.
[21] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Statist.*, vol. 29, no. 2, pp. 610–611, Jun. 1958.
[22] Wikipedia, 68-95-99.7 rule. (Jun., 2014). [Online]. Available: http://en.wikipedia.org/wiki/68-80-95.7_rule
[23] Wikipedia, Hamming distance. (Mar., 2014). [Online]. Available: http://en.wikipedia.org/wiki/Hamming_distance
[24] L. Davis, *Genetic Algorithms and Simulated Annealing*. San Francisco, CA, USA: Morgan Kaufmann, 1987.
[25] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inform. Sci.*, vol. 270, no. 0, pp. 255–287, 2014.
[26] M.-Y. Wu and D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 330–343, Jul. 1990.
[27] S. Kim and J. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in *Proc. Int. Conf. Parallel Process.*, 1988, pp. 1–8.

Yuming Xu is currently working toward the PhD degree at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include evolutionary computation, modeling and scheduling for distributed computing systems, parallel algorithms, grid and cloud computing.

Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University and a deputy director of National Supercomputing Center in Changsha. His major research includes parallel computing, cloud computing, and DNA computing. He has published more than 100 papers in international conferences and journals, such as *IEEE-TC, IEEE-TPDS, IEEE-TSP, JPDC, ICPP, CCGrid, FGCS*. He is an outstanding member of CCF, and a member of the IEEE and serves on the editorial board of IEEE Transactions on Computers.



Ligang He received the bachelors and masters degrees from the Huazhong University of Science and Technology, Wuhan, China, and received the PhD degree in computer science from the University of Warwick, United Kingdom. He was also a post-doctoral researcher at the University of Cambridge, United Kingdom. In 2006, he joined the Department of Computer Science at the University of Warwick as an assistant professor, and then became an associate professor. His areas of interest are parallel and distributed computing,high performance computing and cloud computing.

Longxin Zhang is working towards the PhD degree at the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include real-time systems, power aware computing and fault-tolerant systems, modeling and scheduling for distributed computing systems, distributed system reliability, and parallel algorithms.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multi-core computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 320 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*. He is an IEEE fellow.